



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA TELEMÁTICA

DASHBOARD PARA LA GESTIÓN Y MONITORIZACIÓN DE CUENTAS DEL SERVICIO CLOUD AWS

Autor: Carlos Santamaría Barroso

Director: Atilano Fernández-Pacheco Sánchez-Migallón

Madrid

Julio 2017

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título
“Dashboard para la gestión y monitorización de cuentas del servicio cloud AWS”
en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el
curso académico 2016/17 es de mi autoría, original e inédito y
no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido
tomada de otros documentos está debidamente referenciada.

Fdo.: Carlos Santamaría Barroso

Fecha: 10 / 07 / 2017

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Atilano Fernández-Pacheco Sánchez-Migallón

Fecha: 10 / 07 / 2017

Vº Bº del Coordinador de Proyectos

Fdo.: David Contreras Bárcena

Fecha: 10 / 07 / 2017

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESIS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Carlos Santamaría Barroso DECLARA ser el titular de los derechos de propiedad intelectual de la obra: “Dashboard para la gestión y monitorización de cuentas del servicio cloud AWS”, que esta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

- a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar “marcas de agua” o cualquier otro sistema de seguridad o de protección.
- b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
- c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
- d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
- e) Asignar por defecto a estos trabajos una licencia Creative Commons.
- f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente*).

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

- a) Que la Universidad identifique claramente su nombre como autor de la misma
- b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
- c) Solicitar la retirada de la obra del repositorio por causa justificada.
- d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

- El autor se compromete a:
 - a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
 - b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
 - c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.
 - d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción

de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusiva del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 07 de Julio de 2017.

ACEPTA

Fdo.....

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA TELEMÁTICA

DASHBOARD PARA LA GESTIÓN Y MONITORIZACIÓN DE CUENTAS DEL SERVICIO CLOUD AWS

Autor: Carlos Santamaría Barroso

Director: Atilano Fernández-Pacheco Sánchez-Migallón

Madrid

Julio 2017

Agradecimientos

A mis padres, por educarme, apoyarme, e intentar sacar siempre lo mejor de mí.

A mi hermano, por ser una pieza fundamental y un ejemplo a constante a seguir.

A mis abuelos, por enseñarme el valor de las cosas importantes en la vida.

A Ángela, por ayudarme a crecer como persona y haber estado a mi lado durante todo este camino.

A todos los profesores que han formado parte de mi vida, por saber enseñarme y guiarme para el futuro.

También me gustaría agradecer a todos los integrantes del BBVA que me han acompañado en este trayecto, su acogida, su ayuda constante para hacerme sentir uno más, y hacerme crecer profesional, y personalmente. En particular, a Atilano, por su buen hacer como director, su apoyo constante durante la realización del proyecto y su confianza depositada en mí desde el principio.

DASHBOARD PARA LA GESTIÓN Y MONITORIZACIÓN DE CUENTAS DEL SERVICIO CLOUD AWS.

Autor: Santamaría Barroso, Carlos.

Director: Fernández-Pacheco Sanchez-Migallón, Atilano.

Entidad Colaboradora: Banco Bilbao Vizcaya Argentaria, BBVA.

RESUMEN DEL PROYECTO

En este proyecto se ha desarrollado una aplicación web destinada a la monitorización de todos los posibles servicios *cloud* que tenga la entidad colaboradora, centrándonos solo en uno (AWS), con el fin de facilitar ciertas tareas de gestión, así como a ayudar a tener una idea más concreta y precisa de todos los datos de importancia dentro de estos servicios.

Palabras clave: AWS, *dashboard*, *cloud*, *monitorización*, *pago por uso*.

1. Introducción

La informática en la nube es la entrega bajo demanda de potencia informática, almacenamiento en bases de datos, aplicaciones y otros recursos de TI (Tecnologías de la información) a través de Internet con un sistema de precios basado en el consumo realizado.

Muchas empresas están migrando gran parte de sus aplicaciones a servicios *cloud*, debido a la gran funcionalidad y facilidad que ofrecen. Para ello no se necesita realizar grandes inversiones en cuanto a la adquisición y a la administración de equipos. En lugar de todo eso, podemos aprovisionar exactamente el tipo y el tamaño de recursos informáticos que necesitemos para realizar nuestros proyectos, pagando únicamente cuando los utilicemos. Este aspecto se denomina, pago por uso. Dichos servicios, además de almacenamiento, nos permiten niveles mucho más altos de programación con una complejidad extremadamente reducida. A muy alto nivel, podemos decir que ofrecen 3 tipos de servicios, aunque realmente son muchos más. Una base de datos (o servicios de centro de datos), una plataforma de infraestructura para el desarrollo de aplicaciones y el mantenimiento para que todo funcione. Es rápido, flexible y fiable.

A día de hoy, hay varias empresas que ofrecen este tipo de servicios, pero el periodo de adaptación a ellos ha sido bastante lento, y, de hecho, a día de hoy sigue siéndolo, en gran parte debido a un entorno regulatorio muy estricto, sobre todo en el mundo de la banca. Únicamente por el hecho de estar localizados en Europa, no podemos tener nada almacenado fuera de Europa. Esto puede suponer un problema si hay sedes en otros continentes que necesitan de nuestra información. También hay que decir que en algunos países no se permite el almacenamiento de información en la nube. Estamos hablando de un concepto relativamente nuevo, pero muy potente, que facilita mucho todo el trabajo de desarrollo, como hemos comentado anteriormente. En este punto de adaptación a las nuevas tecnologías, surge la idea de globalización, es decir, cada vez hay más proyectos o recursos a los que tienen que acceder un número alto de personas o, porque no, de máquinas. Es aquí donde el concepto “*cloud*” aparece con mucha fuerza. Evidentemente, las empresas son reacias a cambiar una metodología de trabajo que les ha estado

funcionando durante mucho tiempo, pero poco a poco, van viendo y entendiendo la utilidad de formar un ecosistema “en la nube” alrededor de ella. En nuestro caso, vamos a centrarnos en Amazon Web Services (AWS).

Además, todas las grandes empresas buscan hacer que su dinero sirva para inversión en lugar de ser un coste fijo. En el apartado Parte I4.1 comentaremos más este aspecto, pero básicamente es hacer que un dinero gastado en algo no se pierda. Por ejemplo, cuando una empresa compra un servidor para cualquier necesidad y después ya no lo usa, es un aparato que no sirve de nada, y, sin embargo, tiene que estar en algún sitio físicamente, tiene un coste de mantenimiento, etc...

Con AWS, este problema desaparecería, ya que el servidor es totalmente virtual, y en el momento de no necesitarlo se da de baja de la cuenta sin ningún compromiso, y de esta manera, todo el dinero gastado en dicha máquina ha ido únicamente al desarrollo de la funcionalidad correspondiente. En términos económicos, es lo que se denomina trasladar un coste OPEX a CAPEX.

2. Definición del proyecto

Para la gestión de todos los usuarios y todas las máquinas que tenemos contratadas, AWS está provisto de una serie de *dashboards* independientes que nos ofrecen cierta información y funcionalidad muy útil para ver y gestionar todo lo que tenemos contratado, como por ejemplo crear o eliminar usuarios, parar o levantar máquinas durante cierto tiempo, ver el consumo actual, precio, etc...

El objetivo de este proyecto es integrar todos esos *dashboards* en un único *front-end*, creando una aplicación web que monitorice al mismo tiempo todos los datos necesarios para gestionar las cuentas de AWS de una manera clara y sencilla, y aquellos otros datos necesarios para llevar un correcto seguimiento de los perfiles, sin tener que estar cambiando de ventana en el navegador. También, la aplicación debe quedar preparada para monitorizar la información de otros servicios *cloud* como Google cloud Platform, y así, poder tener un sistema de monitorización completo.

El proyecto se encuentra situado dentro del departamento de desarrollo de la entidad colaboradora y se ha realizado durante el transcurso de unas prácticas de seis meses de duración. Durante el proyecto, y a través de los *sprints* propios de una metodología ágil como SCRUM, se pretenden cubrir los siguientes objetivos:

- Selección de las herramientas adecuadas para que la aplicación web posea las funcionalidades buscadas y que sean capaces de integrarse correctamente.
- El objetivo principal del proyecto es tener una aplicación visual que monitorice datos, sin embargo, la parte dónde se ha invertido más tiempo es en el diseño de todo el desarrollo “*back-end*”, que coge todos los datos que después vamos a mostrar.
- Se utilizará la API que proporciona AWS para poder crear nuestras propias aplicaciones, y basaremos todo nuestro código “*back-end*” en ella.

- Una vez se tenga todo el código back, lo conectaremos con nuestro “*front-end*” (basado en Angular 2) a través de una API-REST.
- Todo el proyecto se desarrollará con integración continua, utilizando un repositorio de control de versiones para tener un buen control del mismo.
- Por último, se identifican futuras aplicaciones y mejoras, de cara a tener una visión general de todos los servicios *cloud* que el banco pueda tener contratados, así como mostrar más datos del servicio actual (AWS) que no se muestren en nuestro diseño.

3. Descripción de la herramienta

La aplicación estará desarrollada siguiendo el esquema que se muestra a continuación



Figura 1. Esquema de desarrollo de la herramienta

Como podemos observar en la Figura 1, el código en que se desarrollaremos toda la parte interna de la herramienta, con la que accederemos a todos los datos necesarios, será Java. La mayor parte de las grandes empresas ofrecen librerías para trabajar con ellas de manera sencilla, y poder crear nosotros nuestras propias aplicaciones teniendo una funcionalidad añadida sobre lo que dichas empresas ya nos ofrecen. En nuestro caso utilizaremos en SDK de AWS para Java.

Una vez finalizado el código que haga toda nuestra lógica, pasaremos a crear nuestra API-REST, que es un protocolo cliente/servidor para el desarrollo web sin estado, es decir, cada petición http contiene toda la información necesaria para ejecutarse. Los objetos en **REST** siempre se manipulan a partir de la **URI**. Es la URI y ningún otro elemento el identificador único de cada recurso de ese sistema REST. La URI nos facilita acceder a la información para hacer con ella lo que la aplicación requiera. Al ingresar

una URI en el navegador, indicando dónde está levantada la aplicación y el puerto de conexión, accedemos a una parte de nuestra lógica, que previamente hemos dicho nosotros que se ejecute al escribir dicha URI en el navegador.

Finalmente, atacamos a nuestra API-REST desde Angular 2 para mostrar de manera amigable todos los datos que tengamos. Angular es un *framework* web desarrollado por Google. Está basado en *TypeScript*, que es como *Javascript* pero tipado, y podremos realizar todo nuestro *front-end* con él, además de utilizar html y css.

4. Resultados

A la hora de mostrar los resultados de una web, es muy difícil de hacer en una sola imagen, pero la que se muestra a continuación es la que mejor define lo que se ha trabajado en este proyecto, y que compone parte de él.

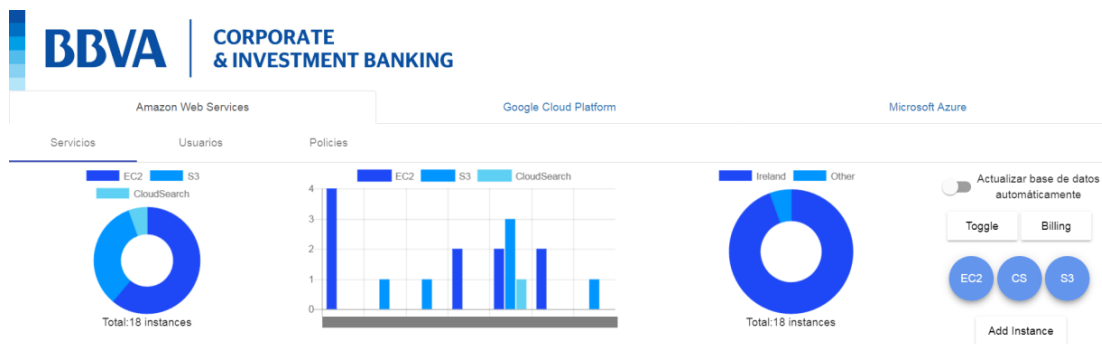


Figura 2: Visión principal de la aplicación web

Por temas de confidencialidad, algunos de los datos no se pueden mostrar. En este caso, los id de los usuarios.

Somos capaces de mostrar información de suma importancia de manera sencilla e intuitiva, como se pueden ver en los gráficos. Así mismo, se obtienen tablas de datos que son capaces de mostrar toda la información relevante que se buscaba en un inicio.

Además, la aplicación queda preparada para monitorizar los posibles futuros servicios *cloud* que el banco tenga contratados.

Los objetivos se han cumplido de manera satisfactoria, no solo por los resultados, si no por las tecnologías aplicadas, ya que son muy utilizadas en el mercado actual, lo que hace que la aplicación no vaya a quedar obsoleta en poco tiempo.

5. Conclusiones

Tras 6 meses de trabajo se ha creado una aplicación web capaz de monitorizar datos de un servicio *cloud* (AWS) y que ha quedado preparada para monitorizar otros. Esta aplicación ha estado destinada a resolver una necesidad básica dentro de las empresas, y es tener un mayor control de los servicios en la nube que tienen contratados. Esta necesidad surge del gran impacto que están teniendo estos servicios en el desarrollo software actual, por lo que la utilidad resultará muy satisfactoria.

DASHBOARD FOR MANAGEMENT AND MONITORING OF ACCOUNTS OF SERVICE CLOUD AWS.

Author: Santamaría Barroso, Carlos.

Supervisor: Fernández-Pacheco Sánchez-Migallón, Atilano.

Collaborating Entity: Banco Bilbao Vizcaya Argentaria, BBVA

ABSTRACT

This project has developed a web application for monitoring all possible *cloud* services that the partner entity has, focusing on only one (AWS), in order to facilitate certain areas of management, as well as helping to have a more precise and precise idea of all the important data within these services.

Keywords: AWS, *dashboard*, *cloud*, *monitoring*, *pay-as-you-go*.

1. Introduction

Cloud computing is the on-demand delivery of computing power, storage in databases, applications and other IT resources through Internet based in a system of prices as you go.

Many companies are migrating much of their applications to cloud services because of the great functionality and ease they offer. This does not require large investments in the acquisition and management of equipment. Instead, we can provision exactly the type and size of computing resources we need to carry out our projects, paying only when we use them. Such services, in addition to storage, allow us much higher levels of programming with extremely reduced complexity.

At a very high level, we can say that they offer 3 types of services, although there really are many more. A database (or data center services), an infrastructure platform for application development and maintenance to make everything work. It is fast, flexible and reliable. Today, there are several companies that offer this type of services, but the period of adaptation to them, has been quite slow, and, in fact, continues to be so. We are talking about a relatively new concept, but very powerful, which greatly facilitates all the development work, as we have discussed above. At this point of adaptation to the new technologies, the idea of globalization arises, that is to say, there are more and more projects or resources that have to be accessed by a large number of people or, why not, of machines. This is where the concept "*cloud*" appears with great force.

Obviously, companies are reluctant to change a working methodology that has been working for a long time, but little by little, they are seeing and understanding the utility of forming an ecosystem "in the *cloud*" around it. In our case, we will focus on *Amazon* Web Services (AWS).

In addition, all large companies seek to make their money serve for investment rather than a fixed cost. In section 4.1 we will comment more on this aspect, but basically is not to lose the money spent on something. For example, when a company buys a server

for any need and in a few time, it isn't used, it is a section that is of no use, and yet it has to be somewhere physically, has a cost of maintenance, etc ...

With AWS, this problem would disappear, since the server is totally virtual, and at the moment of not needing it, it is removed from the account without any commitment, and in this way, all the money spent on this machine has gone solely to the development Of the corresponding functionality. In economic terms, this is called transferring an OPEX cost to CAPEX.

2. Project definition

For the management of all users and all the machines we have contracted, AWS is provided with a series of independent dashboards that offer us some useful information and functionality to view and manage everything we have contracted, such as creating or deleting users, stop or lift machines for a certain time, see current consumption, price, etc...

The goal of this project is to integrate all these dashboards into a single front-end, creating a web application that simultaneously monitors all the data needed to manage the AWS accounts in a clear and simple way, and those other data necessary to carry a correct tracking of the profiles, without having to be changing window in the browser.

The project is located within the development department of the collaborating entity and has been carried out during the course of six-month practices. During the project, and through the sprints of an agile methodology such as SCRUM, the following objectives are sought:

- Selection of the appropriate tools for the web application to have the desired features and to be able to integrate correctly.
- The main objective of the project is to have a visible application that monitors data, however, the part where you have invested more time is in the design of all the code that goes below and that catches all the data that we are going to show later.
- The API provided by AWS will be used to create our own applications, and we will base all our "back-end" code on it.
- Once you have all the code back, we will connect it with our front (based on Angular) through a Rest API.
- The entire project will be developed with continuous integration, using a version control repository to have a good control of the same.
- Lastly, future applications and improvements are identified, in order to have an overview of all the *cloud* services that the bank can have contracted, as well as to show more data of the current service (AWS) that are not shown in our design.

3. Tool description

The application will be developed following the scheme shown below



Figure 1: Tool development scheme

As we can see in figure 1, the code in which we will develop the entire internal part of the tool, with which we will access all the necessary data, will be Java. Most large companies offer libraries to work with them in a simple way, and we can create our own applications with added functionality on what these companies already offer us.

In our case we will use in AWS SDK for Java. Once the code has completed all our logic, we will create our API Rest, which is a client / server protocol for stateless web development, that is, each http request contains all the information needed to run. Objects in REST are always manipulated from the URI. It is the URI and no other element the unique identifier of each resource of that REST system. The URI allows us to access the information to do with it what the application requires.

When entering a URI in the browser, indicating where the application is raised and the connection port, we access a part of our logic, which we previously told us to execute when writing the URI in the browser. Finally, we attacked our Rest API from Angular 2 to display in a friendly way all the data we have. Angular is a web framework developed by Google. It is based on TypeScript, which is like Javascript but typed, and we can do all our front-end with, in addition to using html and css.

4. Results

At the time of showing the results of a web, it is very difficult to do in a single image, but the one that is shown below is the one that best defines what has been worked on this project, and that composes part of it.

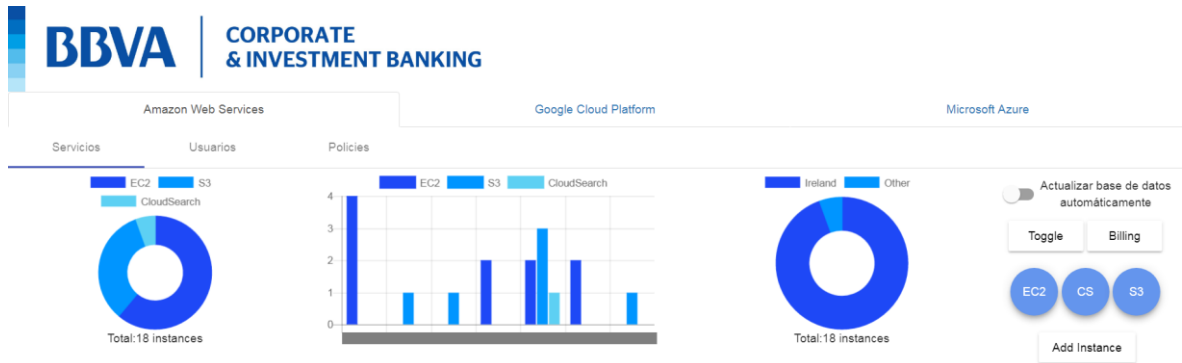


Figure 2: Main view of the web application

We are able to display important information in a simple and intuitive way, as you can see in the graphics. Also, you can get data tables that are able to display all the relevant information that was sought at a start.

In addition, the application is prepared to monitor possible future *cloud* services that the bank has contracted.

The objectives have been fulfilled in a satisfactory way, not only by the results, but by the applied technologies, since they are widely used in the current market, which means that the application will not be obsolete in a short time.

5. Conclusions

After 6 months of work, a web application has been created in order to monitoring data from one *cloud* service (AWS) and has been created and it has been prepared to monitor others too. This application has been designed to solve a basic need within companies, and is to have greater control of the services in the *cloud* that they have contracted. This need arises from the great impact these services are having on current software development, so the utility will be very satisfactory.

Índice de la memoria

Capítulo 1. Introducción	7
1.1 Motivación del proyecto.....	9
Capítulo 2. Descripción de las Tecnologías.....	10
2.1 API de AWS.....	10
2.2 IntelliJ.....	11
2.3 Maven.....	12
2.4 Angular.....	13
2.5 Spring Boot	14
2.6 Git.....	15
2.7 Bootstrap	16
2.8 Angular Material	17
2.9 Postman	18
Capítulo 3. Estado de la Cuestión	19
Capítulo 4. Definición del Trabajo	23
4.1 Justificación.....	23
4.2 Objetivos	25
4.3 Metodología.....	28
4.4 Planificación y Estimación Económica.....	30
Capítulo 5. Sistema de Desarrollo.....	32
5.1 Arquitectura lógica de la aplicación.....	32
5.1.1 Diagrama a alto nivel de la aplicación.....	32
5.2 Diseño.....	33
5.2.1 Servidor / Código back-end.....	35
5.2.2 API-REST	36
5.2.3 Cliente Angular 2	38
Capítulo 6. Implementación.....	54
6.1 Servidor – código back-end.....	54
6.1.1 Creación de proyecto Maven con Intelli-J	54

6.1.2 Inyección de dependencias	56
6.1.3 Integración continua	58
6.1.4 Credenciales de AWS	64
6.1.5 Servicios	67
6.1.6 Gestión de alarmas ante situaciones no deseadas	82
6.1.7 Usuarios, permisos y grupos	86
6.1.8 Creación de la base de datos.....	89
6.2 API-REST	99
6.3 Cliente Angular 2	103
6.3.1 Componente cabecera	107
6.3.2 Componente cuerpo.....	111
Capítulo 7. Análisis de Resultados.....	124
Capítulo 8. Conclusiones y Trabajos Futuros.....	129
8.1 Trabajos futuros.....	130
8.1.1 Billing	130
8.1.2 Mayor número de componentes.....	131
Capítulo 9. Bibliografía.....	132
ANEXO A: Guía de instalación	133
1.1 Instalación de Java.....	133
1.2 Instalación del entorno de desarrollo IntelliJ.....	136
1.3 Node.js y NPM.....	142
1.4 Instalación de las credenciales de AWS.....	146
ANEXO B: Manual de usuario.....	148

Índice de figuras

Figura 1. Esquema de desarrollo de la herramienta.....	12
Figura 2: Visión principal de la aplicación web	13
Figura 3: Tareas del desarrollo establecidas en trello.....	29
Figura 4. Cronograma del proyecto.....	30
Figura 5: Esquema de la aplicación a alto nivel	32
Figura 6: Esquema más detallado de la aplicación.....	33
Figura 7: Diagrama de caso de uso.....	34
Figura 8: Inicio de la aplicación	39
Figura 9: Parte de la web en desarrollo.	40
Figura 10: Gráfico uno. Instancias por tipo.....	41
Figura 11: Gráfico dos. Instancias levantas por usuario.....	41
Figura 12: Gráfico tres. Cantidad de instancias en Irlanda y fuera de ella.....	41
Figura 13: Gráfico uno con datos suprimidos.	42
Figura 14: Gráfico dos con datos suprimidos.....	42
Figura 15: Gráfico tres con datos suprimidos.....	42
Figura 16: Menú de los servicios.....	43
Figura 17: Comparación gráficos circulares.....	43
Figura 18: Dashboard oficial de billing en AWS.	44
Figura 19: Tabla de instancias EC2.....	45
Figura 20: Tabla de instancias S3.....	45
Figura 21: Tabla de instancias CloudSearch.	45
Figura 22: Botón de actualización manual desactivado.	46
Figura 23: Tabla EC2 actualizándose.....	46
Figura 24: Tabla S3 actualizándose.....	47
Figura 25: Correos de alerta	47
Figura 26: Advertencia de región incorrecta	47
Figura 27: Advertencia de instancia EC2 con Red Hat.	48
Figura 28: Visualización del formulario para añadir una instancia.....	48

Figura 29: Formulario mal rellenado.....	48
Figura 30: Formulario correcto.....	49
Figura 31: Pestaña de usuarios	49
Figura 32: Tabla de usuarios	50
Figura 33: Formulario de entrada para grupos mal rellenado	50
Figura 34: Rellenamos el formulario para los grupos correctamente.....	50
Figura 35: Tablas de usuarios y policias	51
Figura 36: Filtro en tablas.....	52
Figura 37: Checkbox de las tablas	52
Figura 38: PDF de policias.....	53
Figura 39: Ejemplo de creación de proyecto Maven.....	55
Figura 40: Búsqueda de dependencias	57
Figura 41: Dependencia de Maven repository.....	57
Figura 42: Comprobación de instalación de Git.....	59
Figura 43: Repositorios creados	59
Figura 44: Menú de TortoiseGit para Git	62
Figura 45: Commit-Push sin cambios.	62
Figura 46: Carpeta con cambios sin notificar en el repositorio.....	63
Figura 47: Ejemplo de carpeta con cambios no notificados.....	63
Figura 48: Menú de la aplicación por consola.....	64
Figura 49: Variables de entorno para AWS.....	66
Figura 50: Servicios de AWS. Parte 1	67
Figura 51: Servicios de AWS. Parte 2.....	67
Figura 52: Instancias S3 por consola.....	73
Figura 53: Instancias EC2 por consola.....	80
Figura 54: Instancias CloudSearch por consola.....	82
Figura 55: Conjunto de permisos de un mismo servicio	87
Figura 56: Denegación de acción por falta de permisos.....	88
Figura 57: Tabla instancias-usuarios	90
Figura 58: Clases Spring para la base de datos.....	94

Figura 59: Acceso a la base de datos	97
Figura 60: Interfaz de la base de datos con su contenido.	98
Figura 61: Visualización de una tabla de la base de datos	99
Figura 62: Ejemplo de petición en Postman.....	102
Figura 63: creación de variable de ejecución	104
Figura 64: Configuración de la variable de start npm	105
Figura 65: División de un componente	107
Figura 66: Header de la aplicación web	111
Figura 67: Visión de las variables desde la consola del navegador.....	117
Figura 68: Gráfico de instancias por tipo	121
Figura 69: Diagrama de actividad con petición a AWS.	123

Índice de tablas

Tabla 1. Estimación económica de la aplicación web	31
Tabla 2: Regiones de AWS	69

Capítulo 1. INTRODUCCIÓN

En la actualidad, la mayor parte de las empresas crean sus propias aplicaciones y las mantienen ellos mismos. Las desarrollan en sus propias máquinas y las almacenan en sus propios servidores, lo que en terminología anglosajona se denomina “*on-premise*”.

Cloud computing es un término que tiene su origen en los años sesenta, propuesto por JCR Licklider, cuya visión era que todo el mundo pudiera acceder a distintos recursos desde cualquier lugar a través de la red, y estar interconectado con otros usuarios a través de ella. Sin embargo, obtener potencia de cómputo a través de la red ha sido una ardua tarea hasta hace relativamente poco, debido al escaso ancho de banda que existía en aquellos años. Alrededor de los años 2000, Internet dotaba de un ancho de banda bastante mayor, y fue en 2002 cuando se desarrolló Amazon Web Services, que prevé un conjunto de servicios basados en la nube, incluyendo almacenamiento, pero no fue hasta 2006 cuando introdujo la computación, o, incluso, la inteligencia humana a través del *Amazon Mechanical Turk*.

En definitiva, *Amazon* fue una de las primeras empresas en proporcionar servicios de infraestructura de TI para empresas en forma de servicios web, más conocido hoy como *cloud computing*.

Todos los avances en el mundo de la tecnología buscan lo mismo, reducir costes o facilitar el trabajo al cliente. El *cloud computing* es un término que está cogiendo mucha fuerza porque lidia con los dos. Hoy en día existen las condiciones suficientes para tener aplicaciones almacenadas o corriendo en cualquier servidor externo a nosotros y que funcione de manera óptima, por lo que cada vez más, las empresas están pensando en migrar todos sus desarrollos a la nube. Uno de los principales beneficios que ofrece el *cloud computing* es la oportunidad de reemplazar importantes gastos anticipados en infraestructura con costos variables reducidos que se escalan con el negocio. Gracias a la *cloud*, las empresas ya no tienen que planificar ni adquirir servidores u otras infraestructuras de TI con semanas o meses de antelación. Pueden disponer en cuestión de minutos de cientos o de miles de

servidores y ofrecer resultados más rápidamente. En otras palabras, las empresas no tienen que gastarse el dinero en comprar las máquinas, administrarlas ni mantenerlas, si no que las alquilan a un proveedor de servicios, se ahorran todos esos costes, y pagan sólo por lo que utilizan, es decir, si una empresa alquila una instancia de cómputo a AWS y la tiene funcionando, pagará por ella el coste por hora que corresponda, pero, aunque dicha instancia siga perteneciendo a la empresa que la alquila, si está sin funcionar, no generará ningún coste. Por lo general las aplicaciones deben estar funcionando continuamente, pero puede haber otras que no tengan que estar las 24 horas del día activas, y puedan apagarse de vez en cuando, para lo cual, esto también es muy útil.

Hoy en día, Amazon Web Services proporciona una plataforma de infraestructura escalable, de confianza y de bajo costo en la *cloud*, pero evidentemente, con los grandes modelos de negocio, siempre hay competencia. Las grandes empresas como Google o Microsoft tienen su propio negocio de servicios *cloud*, como son Cloud Platform o Azure.

El negocio que ofrecen todas ellas es similar, y puede haber distintas razones para contratar a cualquier de ellos, o, porque no, a todos. La contratación de cada uno de estos servicios puede variar en muchas cosas, acuerdos más antiguos de la empresa en la que trabajamos con cualquiera de ellos, o distintos precios, o una mejor experiencia con otros.

El BBVA lleva varios años despuntando en el ámbito tecnológico con su departamento de desarrollo, y hace poco tiempo que decidieron apostar por este tipo de tecnología. En este tiempo ya se han dado cuenta del acierto que supuso y ya tienen varias aplicaciones corriendo en los servidores que tienen contratados de Amazon Web Services.

Al formar parte de AWS, trabajaremos con dicha cuenta para realizar el proyecto, pero es bastante probable que en el futuro el banco, o cualquier otra empresa quiera ampliar sus servicios *cloud* a otro proveedor. Es en este punto donde surge la necesidad de este proyecto.

1.1 MOTIVACIÓN DEL PROYECTO

Dentro de la empresa, se me ofreció la opción de elegir una serie de proyectos muy diferentes entre sí, en los que se veían distintas tecnologías dominantes en el mercado actual.

La decisión de escoger este proyecto en concreto fue porque hoy en día, es el que ofrece aprender un abanico más amplio de herramientas muy utilizadas en el ámbito de desarrollo actual, y que se van a verse desarrolladas de cara al futuro. Además, resuelve una necesidad básica dentro de la empresa, como es la gestión de perfiles del banco en AWS, y, sobre todo, tiene una gran capacidad de ampliación, haciendo que en un futuro dicho proyecto pueda ampliarse con una funcionalidad similar a otros servicios *cloud* contratados por la empresa, como puedan ser Azure, Google Cloud Platform, etc....

Hoy en día, todo el desarrollo web y desarrollo de aplicaciones móviles, está teniendo un auge increíble dentro del mundo IT, porque todas las personas y todas las futuras generaciones van a trabajar junto a este tipo de tecnologías, dando igual el ámbito de su trabajo. Todo se está informatizando y tener conocimientos más amplios de este tipo de desarrollos permite tener un plus en la formación personal.

En cuanto a la motivación empresarial, hay que decir que, aunque los servicios *cloud* sean una tecnología relativamente nueva, todos los expertos apuntan a que van a ser cruciales en el desarrollo en los próximos años. La entidad ya los tiene contratados y trabaja actualmente con ellos. Es cierto que un banco no puede almacenar todos sus datos ahí por temas de seguridad, ya que alberga datos de alta confidencialidad, pero para el resto de ámbitos son una herramienta realmente útil. Hay varias empresas actualmente que ofrecen este tipo de servicios y cada uno de ellos tiene sus ventajas concretas. Por ello, el banco desea tener una aplicación web de monitorización que sea capaz de mostrar datos de todos ellos a la vez, para llevar a cabo una gestión más fácilmente y poder tomar decisiones más rápidamente, ya que analizar los datos de esta manera es mucho más intuitivo.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

En este punto, para conocer más a fondo el desarrollo de todo el proyecto es necesario explicar las herramientas para llevarlo a cabo.

2.1 API DE AWS



AWS, como muchas otras herramientas, proporciona un interfaz para desarrolladores que nos permite hacer nuestras propias aplicaciones de esta tecnología. Para poder acceder a él, vamos a utilizar lo que se conoce como SDK, que es un kit de desarrollo software que nos ofrece unos paquetes de software muy concretos, que comentaremos a continuación, para poder acceder de manera más sencilla a los servicios de la aplicación que queramos. En nuestro caso, cómo toda la lógica del proyecto estará desarrollada en java, el SDK utilizado será el *AWS SDK for Java*, aunque AWS proporciona el mismo interfaz para muchos más lenguajes de programación, como C++ o Python.

Amazon Web Services proporciona ciertos servicios, dando la opción al cliente de alquilar instancias concretas para cada tipo de servicio. La API que nos proporciona AWS, sigue una estructura organizada por tipo de instancia, y dentro de cada una de ellas, los paquetes correspondientes. De esta manera, es muy fácil identificar dónde se encuentra la funcionalidad que deseamos implementar.

2.2 INTELLIJ



Hoy en día, el mundo de la programación está muy extendido y hay muchas herramientas disponibles para los desarrolladores, como Eclipse, NetBeans, Sublime, VisualStudio, etc...

En nuestro caso, el IDE escogido ha sido el IntelliJ, ya que, dentro de la entidad, y en concreto dentro de mi departamento, era el entorno de desarrollo más usado. Es propiedad de la empresa JetBrains, especializada en desarrollar herramientas para programadores. Es relativamente nuevo y está cogiendo mucho auge entre los programadores, gracias a la interfaz bastante sencilla que posee y a la funcionalidad que proporciona.

Actualmente, cuenta con dos versiones, la *Community Edition* y la *Ultimate Edition*. La diferencia entre ambas es la funcionalidad extra que proporciona la segunda, permitiendo hacer desarrollos de html, css, JavaScript, TypeScript, etc...

Como el proyecto va a constar de una parte *back-end* desarrollada en java y otra *front-end* basada en angular, que usa TypeScript, la versión *ultimate* es perfecta para el desarrollo. Es una versión de pago, pero para estudiantes es gratuita durante un año, así que no hay costes adicionales por usar esta versión. Si hubiera habido, habríamos utilizado el WebStorm, otro IDE de JetBrains para desarrollos web.

2.3 MAVEN



Maven es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl, de Sonatype, en 2002. Tiene un modelo de configuración de construcción más simple, basado en un formato XML. Maven utiliza un *Project Object Model* (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado [1].

Dentro del proyecto, tenemos un archivo llamado POM, que es literalmente es archivo xml que define las dependencias externas que tiene nuestro proyecto. Es una de las herramientas más útiles en la actualidad en el desarrollo de proyectos software, ya que la incorporación de librerías de manera manual es un trabajo muy arduo.

Maven posee un repositorio donde se almacenan todas las dependencias de las librerías para incluirlas en nuestro proyecto. Se accede a él a través de internet, se busca la librería que necesitamos y la incluimos en nuestro archivo *pom*. Dicho repositorio también te da las dependencias correspondientes a otras herramientas similares a Maven como, por ejemplo, Gradle.

2.4 ANGULAR



Angular es un *framework* de Google para el desarrollo de clientes web. Actualmente, tiene dos versiones muy distintas: Angular JS y Angular 2. Estas versiones sólo se parecen en el nombre, pues tienen una funcionalidad muy diferente. Angular JS fue el primer *framework* en aparecer, y, por tanto, ha sido el más utilizado hasta ahora, pero tiene dos problemas bastante incómodos. Primero, el lenguaje. Está basado en JavaScript, que es un lenguaje asíncrono y nada modular, que necesita de mucho control para aplicaciones de negocio, banca, o administración. Y segundo, la escalabilidad. Las aplicaciones empresariales son muy pesadas en lógica y datos. La falta de un cargador dinámico decente hacía muy difícil reducir el tiempo antes del primer impacto. Una vez lanzadas las aplicaciones iban razonablemente bien, hasta que alguien creaba informes editables. Con miles de datos de los que preocuparse la técnica del doble *binding* saturaba los *watchers* [2].

Para solucionar este tipo de problemas, apareció Angular 2 basado en TypeScript. Las características de este lenguaje son todas aquellas del más avanzado JS, con la diferencia de que es tipado. Esto significa que las variables que se utilizan son de un tipo concreto, no como en JavaScript, que se definen simplemente como “var”. Estas características hacen que los tests sean mucho más sencillos, así como que la curva de aprendizaje sea mucho más lenta.

También es importante comentar que a diferencia de *Angular JS*, Angular 2 trabaja con componentes, que son los que conforman la web. Cada componente está formado por su estructura visual (html, css) y por su lógica (TypeScript). La gran ventaja de trabajar de esta manera es que cada componente es totalmente independiente de otros, es decir, se ejecuta de manera independiente, si necesita un refresco se hace de manera independiente, etc...

Evidentemente, se pueden inyectar componentes dentro de otros componentes para simplificar la lógica o la estructura de nuestro proyecto, pero aquí entra la habilidad de cada trabajador como programador. Para poder trabajar con Angular 2, lo único que hay que tener instalado en el puesto local, es NPM y Node.js. NPM es la herramienta que gestiona las librerías y Node.js es la herramienta que nos permite programar en TypeScript.

2.5 *SPRING BOOT*



Spring es una herramienta para desarrolladores que intenta facilitar el trabajo a un desarrollador. Dentro de Spring, existen muchos *frameworks* con distintas funcionalidades. En función de lo que queramos realizar en nuestro proyecto, utilizaremos uno u otro. Entre ellos se encuentran: Spring Io, Spring Boot, Spring Framework, Spring *Cloud*, Spring Data, etc...

Nosotros nos vamos a centrar en Spring Boot. La principal razón de usar este *framework* es que simplifica bastante el desarrollo de cierta funcionalidad dentro de la aplicación, como la creación de la base de datos, la creación de una API REST para realizar PoCs (prueba de concepto) de funcionamiento, etc...

Es posible que la aplicación acabe corriendo en un servidor Spring, por lo que el uso de este tipo de herramientas es muy útil.

2.6 GIT



Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente [3].

Este proyecto, va a estar desarrollado con integración continua, lo que significa que se va a utilizar este tipo de herramientas, teniendo un control de versiones actualizado cada semana.

Los repositorios, son los lugares en el *cloud* donde se almacena nuestro código, y al que puede acceder cualquier que esté trabajando en el proyecto. Es una herramienta muy útil a la hora de trabajar con varias personas que están físicamente lejos, porque todos acceden al proyecto a través de internet, y cada uno puede indicar que cambios ha realizado, aceptarlo o rechazarlo, volver a una versión anterior, etc...

En nuestro caso, como el desarrollo del proyecto lo he llevado a cabo yo solo, no aprovecharemos esta ventaja y simplemente lo utilizaremos como control de versiones.

Hay muchas clases de repositorios actualmente, aunque todos ellos funcionan igual. El más famoso de todos ellos es GitHub, el más extendido mundialmente por ser el primero de ellos, sin embargo, nosotros usaremos GitLab, ya que posee ciertas ventajas muy útiles, como un interfaz desde el que se puede acceder a todos los proyectos. Además, los ajustes permiten controlar si un repositorio es público o privado, tiene ramas protegidas y se puede dar acceso de lectura/escritura [4].

2.7 *BOOTSTRAP*



Bootstrap es uno de los *frameworks* HTML, CSS y JavaScripts más populares en el mundo del desarrollo web, ya que permite el desarrollo de proyectos web *responsive* y *mobile first*.

Al desarrollar un proyecto en Angular 2, no vale solo con importar el link en tu proyecto que te da acceso a Bootstrap, sino que hay que importar la librería que te da acceso a Bootstrap en unión con angular 2, de manera que podamos trabajar con los *frameworks* a la vez.

Para importar librerías en un proyecto Angular, se hace a través de NPM. Haciendo una pequeña metáfora, NPM es a Angular, lo que es Maven a Java. Para instalar la librería, el código ejecutado desde la Shell de Windows en la carpeta del proyecto es el siguiente:

```
npm install ng2-bootstrap --save
```

Con el comando anterior descargamos la librería y la incluimos en el proyecto y también gracias al parámetro `--save` incluimos dicha librería como una dependencia de nuestro proyecto Angular 2 en el fichero `package.json`.

2.8 ANGULAR MATERIAL



Angular Material es un *framework* de html y css proporcionado por Google para realizar aplicaciones web y aplicaciones móviles.

Es muy parecido a Bootstrap y su manejo es muy similar. Todos los componentes que permite incorporar a las aplicaciones se encuentran disponibles en la página web. Es importante recordar que, de igual manera que en Bootstrap (incorporar en el html un link haciendo referencia a sus estilos), en Angular Material hay que incorporar el componente correspondiente a nuestro fichero.

Lo primero que hay que hacer es instalarlo en nuestro proyecto, en el terminal, mediante el siguiente código:

```
npm install --save @angular/material
```

Una vez instalado, solo tenemos que cargar en nuestro archivo de configuración `app.module.ts`, los componentes que este paquete nos provee y que son necesarios para el correcto funcionamiento de los módulos y las animaciones que este *framework* nos provee. En el capítulo de desarrollo se explicará más en detalle la configuración de dicho archivo.

A partir de aquí, ya podemos comenzar a usar elementos de Angular Material en nuestro proyecto Angular.

2.9 POSTMAN



Postman es una extensión de chrome que sirve para interfaces de programación de aplicaciones (API) directamente y como parte de las pruebas de integración para determinar si cumplen con las expectativas de funcionalidad, fiabilidad, rendimiento y seguridad. Dado que las API carecen de una interfaz gráfica de usuario, las pruebas API se realizan en la capa de mensaje. Las pruebas de las API se consideran críticas para la automatización de las pruebas porque actualmente, sirven de interfaz primaria para la lógica de la aplicación y porque las pruebas GUI son difíciles de mantener con los ciclos de lanzamiento cortos y los cambios frecuentes usados comúnmente con el desarrollo de software Agile y DevOps.

En nuestro caso particular, lo utilizaremos para comprobar el buen funcionamiento de la API-REST creada, ver que realmente se envía la respuesta como deseamos, que las peticiones son del tipo correcto, etc...

Capítulo 3. ESTADO DE LA CUESTIÓN

La importancia de los servicios *cloud* en la sociedad actual se encuentra en constante aumento gracias a la cantidad de funcionalidad y facilidad que proporcionan, aunque hay ciertas barreras que hacen que estos servicios no se desarrollen todo lo que podrían. El principal inconveniente de dicho desarrollo es la regulación, que en nuestro caso particular, dentro del sector bancario, es aún más compleja. Hay varias normas que impiden la total integración de estas herramientas en la forma de trabajar actual, como la limitación física. Al estar en Europa, no podemos almacenar nada fuera del continente ya que se supone que la información bancaria es de carácter confidencial. Podemos concluir pues, que hay ciertos aspectos que hacen que, a pesar de que esta tecnología se esté desarrollando rápidamente, no se haga en su total capacidad de evolución.

En cuanto al tema económico, para muchas de las grandes empresas que tienen necesidad de crear aplicaciones nuevas o almacenar cualquier tipo de información, los servicios *cloud* son una gran apuesta. Hasta ahora, cada vez que se tenía que almacenar algo, se tenían que comprar físicamente los servidores y dejarlos en algún sitio del edificio, generalmente una habitación dedicada solo a tener hardware puro y duro. Esto supone un problema ya que muchas veces ciertas aplicaciones no son “para siempre” y vendría bien deshacerse de todo el material comprado. Esto supone una pérdida cuantiosa de dinero que podrían ahorrarse contratando los servicios *cloud*. Gracias a ellos puede pagarse solo por lo que se consume, y como no son servidores que se compran de forma física, la empresa puede dejar de tener los servicios de dichos servidores en cuanto no sean útiles.

Evidentemente, todas las aplicaciones que se encuentren en servidores físicos van a seguir permaneciendo en ellos, ya que se suponen aplicaciones duraderas en el tiempo, además de que migrarlas a servidores virtuales puede costar más dinero del que se necesita para mantener los servidores donde estén almacenadas. Con el tiempo, las máquinas se irán sustituyendo conforme estas se vayan amortizando.

Por lo tanto, es importante volver a insistir en la capacidad de los servicios *cloud* para proporcionar escalabilidad y ahorro en el coste de las aplicaciones. Es evidente, que, al tener almacenada información en la nube, el acceso a ella es mucho más sencillo. Podemos obviar todas las barreras físicas con otros países dentro del continente o todos los problemas de conexión con los servidores particulares. Hoy en día, en una empresa grande, hacer compra de ciertos aparatos, supone una ardua tarea. Generalmente se habla de servidores, porque son lo que van a contener la información y las aplicaciones.

Hablamos de una tarea complicada ya que la compra de dichos aparatos supone realizar un estudio de cuales la mejor opción, comunicar al responsable cual es la decisión tomada, que se comunique al departamento de cuentas de la empresa, y así, un largo etcétera. En resumen, entre que se establece la necesidad de adquirir una máquina, hasta que físicamente está en la empresa para su uso, pueden pasar fácilmente varios meses. Con AWS solucionamos estos problemas, ya que el alquiler de unos servidores es cuestión de minutos. Incluso hay ocasiones, en las que ciertas aplicaciones requieren de muchos recursos, que AWS te garantiza el crecimiento horizontal automático para el correcto funcionamiento de aplicaciones. Hablamos del tiempo que se tarda en acceder a la página web y levantar una instancia del tipo correspondiente, por tanto, es temas de escalabilidad, lo servicios *cloud* no tienen competidor.

Por otro lado, también es importante destacar el tema del coste, donde la computación en la nube también es un competidor digno muy difícil de superar, ya que el hecho de pagar solo por tiempo de uso supone ahorrarte costes fijos que se tienen si no tienes contratados los servicios *cloud* de algún proveedor.

En la actualidad, la entidad tiene contratados los servicios *cloud* de *Amazon*, llamados *Amazon Web Services*. Los servicios que AWS proporciona son muy variados, desde simple almacenamiento, hasta desarrollos para IoT (Internet Of Things) o indexación de archivos. AWS, desde su página web, permite ver la información de la cuenta del banco, como lista de instancias alquiladas, usuarios registrados, gastos, etc...

Desde hace algún tiempo, el departamento de arquitectura tiene pensado ampliar dichos servicios *cloud*, combinándolos con la funcionalidad que ofrece Google *Cloud*. Es importante destacar que mucha de dicha funcionalidad es parecida en todas las empresas que ofrecen computación en la nube, sin embargo, hay otro tipo de servicios que son diferentes, o que se gestionan de manera más óptima. Esta es la razón por la que se decide contratar los servicios de otra empresa de computación *cloud*.

Cada empresa proporciona gráficos y tablas con la información propia del cliente, pero llega un punto donde es necesaria una agrupación global de la información para tener un control de los servicios que se tienen contratados, y es aquí dónde surge este proyecto.

Actualmente no hay aplicaciones que proporcionen una información conjunta de todos los servicios *cloud* que una empresa pueda tener. Nos referimos al hecho de que haya aplicaciones de libre distribución para que cualquier usuario o empresa particular pueda usar. Evidentemente no se puede saber si alguna compañía de manera privada ha realizado su propia aplicación para su funcionamiento interno, y aunque así fuera, no nos serviría de mucho ya que no podríamos hacer uso de ella. Hay algunas que dan una información algo más detallada de los costes que se tiene al tener contratados ciertos servicios de AWS, pero no monitorizan más información. Evidentemente, esto ocurre debido a la novedad que tienen los servicios *cloud*. Si se llevaran utilizando más tiempo, prácticamente seguro que existiría una aplicación como la que estamos comentando, aunque evidentemente, muchas veces surgen proyectos a partir de las nuevas tecnologías, como es nuestro caso. No estamos hablando de un proyecto crítico cuya necesidad sea máxima, pero si es cierto que una aplicación como esta va a ser muy útil en el futuro cercano, ya que los servicios *cloud* van a ser una herramienta muy demandada.

Es importante comentar que la aplicación desarrollada va a ser un sistema de monitorización que tendrá funcionalidad limitada. Al fin y al cabo, se busca una aplicación que muestre datos, pero una aplicación que realice ciertas funciones sería muy útil para la entidad y para la gestión de la cuenta dentro de la misma.

Con funcionalidad limitada nos referimos a la gestión de permisos dentro de la aplicación, y a la comprobación de funcionalidad no deseada. Es importante destacar que el departamento de seguridad del banco no permite asignar directamente los permisos a nuestro departamento, por lo que desde la aplicación se podrá asignar los permisos correspondientes a cualquier usuario y generar un archivo PDF que los indique. Dicho archivo se enviará automáticamente al responsable de la cuenta para verificar la información y para que pueda ser enviado al departamento encargado de gestionar los permisos implícitamente.

Por otro lado, de manera interna la aplicación podrá realizar comprobaciones en la cuenta para controlar si está pasando algo no deseado. Por norma interna, todas las instancias deben levantarse en servidores localizados físicamente en Irlanda. Si por alguna razón, al monitorizar instancias, se observa que alguna no está levantada allí, la aplicación dará un aviso. De igual manera, si se comprueba que el sistema operativo de la máquina que es alquila no es Linux, también dará un aviso, ya que cualquiera que no lo sea es un sistema más caro. Evidentemente, es puramente informativo. Si el responsable cree que el hecho de tener una instancia más cara es necesario para correr alguna aplicación más pesada de lo normal, la instancia quedará levantada sin problemas. Lo único que se busca es tener un mayor control y saber si se ha levantado una instancia extraña. A partir de ahí es decisión del responsable continuar con ella o darla de baja.

Por todas estas razones, podemos concluir que nuestra aplicación será bastante novedosa y resolverá una necesidad que prácticamente seguro surgirá en el futuro más cercano, cuando los servicios *cloud* supongan la manera más cotidiana de trabajar. Por ejemplo, dentro del departamento de desarrollo del banco, CIB, se tienen más de 500 máquinas *on-premise*, que son muchas más que las se tienen *cloud*, 17. Este ejemplo representa la lenta transición que están teniendo en el sector bancario, pero sin duda representarán la manera más cotidiana de trabajar en el futuro.

Capítulo 4. DEFINICIÓN DEL TRABAJO

En este apartado explicaremos algunos de los puntos de vital necesidad para la realización del proyecto en cuestión. Hablaremos de la justificación, los objetivos, y la planificación llevada a cabo. Para finalizar, haremos una pequeña aproximación económica de cuánto dinero cuesta finalizar un proyecto como este.

4.1 JUSTIFICACIÓN

En el apartado anterior ya se comentó el hecho de que no hay ninguna aplicación que monitorice datos de varios servicios en la nube, pero no es la única razón por la que una empresa debería lanzarse a realizar un proyecto de este tipo.

Generalmente, todas las entidades financieras buscan trasladar todos los costes fijos que tienen en inversiones. En este punto es muy importante definir dos conceptos económicos, que más adelante relacionaremos con nuestro proyecto. Un *OPEX*, del inglés "*Operating expense*", es un coste permanente para el funcionamiento de un producto, negocio o sistema. Puede traducirse como gasto de funcionamiento, gastos operativos, o gastos operacionales. Su contraparte, el gasto de capital (*CAPEX*), es el coste de desarrollo o el suministro de componentes no consumibles para el producto o sistema [5].

Hasta hace unos años, todas las empresas tenían una instalación "*on-premise*", es decir, disponían de varios cientos de servidores y máquinas para almacenar toda la información necesaria para su funcionamiento dentro su propia infraestructura. Esto quiere decir que una empresa, era la que tenía que gastarse parte de su capital para la adquisición de equipos, almacenamiento de los mismos en un lugar físico y en su mantenimiento. En términos económicos podríamos llamar a este hecho, tener gastos *OPEX*, es decir, un coste permanente.

DEFINICIÓN DEL TRABAJO

Lógicamente, todas las reducciones de estos costes son aspectos muy interesantes y que pueden suponer un buen beneficio a las empresas. Parte de esas reducciones pueden conseguirse con la computación en la nube. Gracias a esta forma de trabajar, las empresas pueden alquilar cientos de máquinas al instante y pagar únicamente por su consumo, por lo tanto, todos los costes de almacenamiento y mantenimiento desaparecerían.

Por otro lado, también son muy útiles a la hora de llevar a cabo un proyecto temporal, para el cuál, podría alquilarse una máquina durante el tiempo necesario, y, después de la finalización del proyecto, dejar de pagar por dicha máquina, en vez de tener que dejarla sin funcionar en un almacén. Es evidente, que en ciertas ocasiones los servidores que se compran para llevar a cabo un proyecto suelen servir también para llevar a cabo otros, pero, en cualquier caso, los servicios *cloud* siguen suponiendo un ahorro importante de dinero.

Por dichas razones, una empresa al contratar servicios *cloud*, estaría trasladando varios de sus gastos de tipo ‘OPEX’ a tipo ‘CAPEX’, invirtiendo dinero únicamente en lo que el servidor externo necesita, sin tener en cuenta mantenimiento ni almacenamiento. Por lo tanto, lo más probable en el futuro es que la mayoría de las empresas acaben llevando a cabo esta manera de trabajar para la mayoría de sus proyectos.

De esta manera, las empresas tendrían contratados varios servicios *cloud* de manera independiente, cada uno para la funcionalidad correspondiente. Todos mostrarían los datos de su propia cuenta de manera individual, y no dispondrían de una aplicación dónde poder visualizar toda la información de útil de todos los servicios *cloud* que tienen.

Aquí es donde entra en juego nuestro proyecto, que va a consistir en monitorizar cierta información y poder desarrollar cierta funcionalidad sobre la cuenta AWS que tiene el banco. Como he dicho, la aplicación web solo estará desarrollada sobre AWS, pero quedará terminada con el propósito de ampliarla a los próximos servicios que tenga el banco para poder monitorizar toda la información necesaria de todos los servicios *cloud* contratados para poder tener un mejor control de los usuarios y las cuentas de las que se dispongan. Actualmente ya cuenta con los servicios de Amazon Web Services y Google Cloud, y es

bastante probable que acabe contratando también los de Microsoft Azure. Al fin y al cabo, un banco tan grande que opera en tantos países tiene mucho trabajo de desarrollo y mejoras.

4.2 OBJETIVOS

En este apartado se comentan los principales objetivos del proyecto. Como se ve a continuación, no todos los objetivos son puramente funcionales. Tendremos un amplio abanico de ellos, pasando por la más básica elección de herramientas, hasta la funcionalidad deseada y el desarrollo amigable de la aplicación.

Para comenzar, al desarrollar una aplicación basada a su vez en otra aplicación usada mundialmente, se suele utilizar la API facilitada por el proveedor, porque de lo contrario, desarrollar cierta funcionalidad sería prácticamente imposible. El banco es muy conocido por estar a la vanguardia en tecnología de desarrollo software, por lo que la aplicación deberá estar desarrollada con las herramientas más actuales en programación web. La lógica deberá estar desarrollada en Java, ya que es el lenguaje de programación más usado mundialmente, utilizando el uso de *frameworks* como Spring para facilitar el trabajo. Por otro lado, el interfaz estará desarrollado con html y css, haciendo uso de Angular 2 y Bootstrap, para que quede un entorno amigable y propio de nuestro tiempo.

Además, el proyecto se realizará de manera profesional, utilizando un entorno de integración continua, usando Git, teniendo control sobre todas las versiones que vayan surgiendo según se vaya realizando el proyecto. Por recomendación de la entidad, se utilizarán dos repositorios privados en GitLab, un entorno para trabajar con Git. Se creará uno para llevar el control del servidor, es decir, de toda la lógica que va a contener nuestra página web, y otro para el cliente Angular 2, que tendrá la misma función que el anterior.

A partir de ahí, podemos dividir los objetivos principales en cuatro grandes módulos: Usuarios/Grupos/Permisos, Servicios de AWS, *Billing* y la gestión de alarmas en determinadas situaciones.

A la cabeza de este grupo de objetivos funcionales, se encuentran los relacionados con los usuarios. Se deben poder mostrar todos los usuarios del banco que estén registrados en AWS,

DEFINICIÓN DEL TRABAJO

junto con ciertas características que *Amazon* provee, como la fecha de registro, la última fecha de conexión, o a que grupo pertenece. Dentro de AWS, se pueden crear distintos grupos de usuarios, para diferenciarlos por jerarquías, permisos o cualquier otra razón que el cliente crea razonable.

También se deben poder crear grupos para que, posteriormente, el responsable de la cuenta pueda reagrupar o diferenciar a los usuarios como el considere. Es importante destacar aquí, que debido a los permisos que la cuenta utilizada tiene, no se permite asignar usuarios a grupos, ya que esta tarea debe realizarla otro departamento de la entidad, por lo que solo nos ocuparemos de la creación de grupos. La asignación de usuarios se deberá comunicar al departamento correspondiente.

Amazon Web Services, también tiene una larga lista de *policies* (permisos) para asignar a todos sus usuarios. Dentro de una cuenta perteneciente a una gran entidad, lo normal sería que el responsable tuviera todos los permisos disponibles para hacer lo que considere necesario, mientras que los trabajadores que estén a su cargo tuvieran una funcionalidad más limitada, haciendo que no puedan realizar todo tipo de acciones. Es básicamente una forma de controlar la actividad dentro de la cuenta. De nuevo por los permisos dentro del perfil de AWS, no se permite asignar permisos a usuarios, por lo que se realizará lo siguiente. Desde la web, se deberán mostrar todas las *policies* y todos los usuarios con un *checkbox* (cuadro de selección) para poder generar automáticamente un archivo PDF donde se indican que permisos se quieren asignar a ciertos usuarios. Dicho archivo PDF se enviará al departamento correspondiente de asignar permisos para que lo hagan ellos, y los permisos queden correctamente asignados.

En segundo lugar, tenemos los servicios de AWS. El banco tiene varios servicios contratados a AWS. Básicamente, son los siguientes. *Amazon Simple Storage Service (Amazon S3)*, que es almacenamiento, *Amazon Elastic Compute Cloud (EC2)*, que es capacidad de cómputo y *Amazon Cloudsearch*, que es un servicio administrado en la *cloud* de AWS que facilita la configuración, la administración y el escalado rentables de una solución de búsqueda para su sitio web o aplicación [6].

Dichos servicios tienen características propias. La más importante de ellas es la región en la que los servidores de dichos servicios están físicamente localizados, ya que, por política de la

DEFINICIÓN DEL TRABAJO

entidad, todas las instancias deben levantarse en Irlanda, por lo que, si hay alguna fuera de dicho país, se debe emitir un comunicado. Aún más grave sería tener levantada una instancia fuera de la UE, ya que el BCE (Banco Central Europeo) ha comunicado un requerimiento regulatorio que indica que todos los bancos europeos deben mover todos los servicios en la nube dentro de Europa. Se deben mostrar todas las máquinas alquiladas de todos los servicios, junto con sus características correspondientes.

Otra característica muy útil, sería saber que usuario ha levantado cada instancia, pero AWS no facilita dicha información, simplemente asigna la instancia a la cuenta global del banco. Para solucionar este aspecto, se debe crear una base de datos en la que se indique dicha información. La base de datos se deberá rellenar de forma manual cada vez que se levante una instancia de cualquier servicio, mediante un formulario en la página web.

También se deben crear gráficos que muestren información de forma intuitiva. Uno de ellos mostrará el número de instancias de cada tipo que posee la entidad, otro de ellos mostrará el número de instancias en la región correcta (Irlanda) y cuántas no. Y finalmente, el último gráfico mostrará, de la tabla de la base de datos comentada en el párrafo anterior, las instancias que ha levantado cada usuario de la entidad dentro de AWS.

Por otro lado, tenemos el “*billing*”, que es el análisis del coste de la cuenta. Evidentemente, para este tipo de aplicaciones, el hecho de llevar un control sobre el coste que las instancias están causando sobre la entidad, sería muy útil. Desgraciadamente, la API de AWS no proporciona esta clase de información, y por lo tanto, no podemos acceder al coste individual por instancia. Hay otras formas de poder acceder a los costes. AWS proporciona un documento Excel de extensión .csv con los costes de cada operación en cada máquina. Sería bastante complicado de leer y de obtener información de él, pero, en cualquier caso, el departamento de seguridad no nos permite el acceso a este tipo de archivos confidenciales, por lo que, de momento, no podemos mostrar los análisis de los costes que nos gustaría.

Y, por último, pero no por ello menos importante, la gestión de alarmas para situaciones no deseadas. Estableceremos como situaciones no deseadas el alquiler de ciertas máquinas en regiones que no sean Irlanda, independientemente de la clase de instancia que sea y la contratación de una instancia EC2 con un sistema operativo que no sea *Amazon Linux*. Al

contratar una instancia de este tipo, AWS da la opción de que funcione con distintos sistemas, entre ellos varios tipos de Windows, Red Hat o varios tipos de Linux. Todos ellos suponen un coste mayor que el de *Amazon Linux*, por lo que, si se detecta que se ha creado una instancia con cualquier de ellos, debe notificarse mediante correo electrónico al responsable, para que evalúe si es necesario mantener la instancia o eliminarla.

4.3 METODOLOGÍA

La metodología escogida para llevar a cabo el proyecto será SCRUM. SCRUM es el nombre con el que se denomina a los marcos de desarrollo ágiles caracterizados por adoptar una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto, basar la calidad del resultado más en el conocimiento tácito de las personas en equipos auto organizados, que, en la calidad de los procesos empleados, y por el solapamiento de las diferentes fases del desarrollo, en lugar de realizar una tras otra en un ciclo secuencial o en cascada [7].

Seguiremos un desarrollo iterativo e incremental, donde los requisitos se definirán con el paso del tiempo y la evolución del proyecto. Se ha escogido esta manera de trabajar ya que es bastante adecuada para la realización de las distintas partes en las que consta el proyecto, que se definirán más adelante en el cronograma. Para llevar un control de las tareas, se ha utilizado Trello, una herramienta online que permite definir tareas con fechas de entrega. Como podemos ver en la siguiente imagen, se han definido cuatro ramas principales. La primera de ellas representa el total de tareas a realizar, y el resto representan que tareas se están realizando, cuáles se han terminado, y cuáles están paradas. Las actividades paradas pueden estarlo por cualquier razón, porque no se sepa aún la mejor forma de realizarlo, porque se necesite respuesta de otro equipo, o porque simplemente no funciona y se ha dejado en segundo plano para reformarla en otro momento...

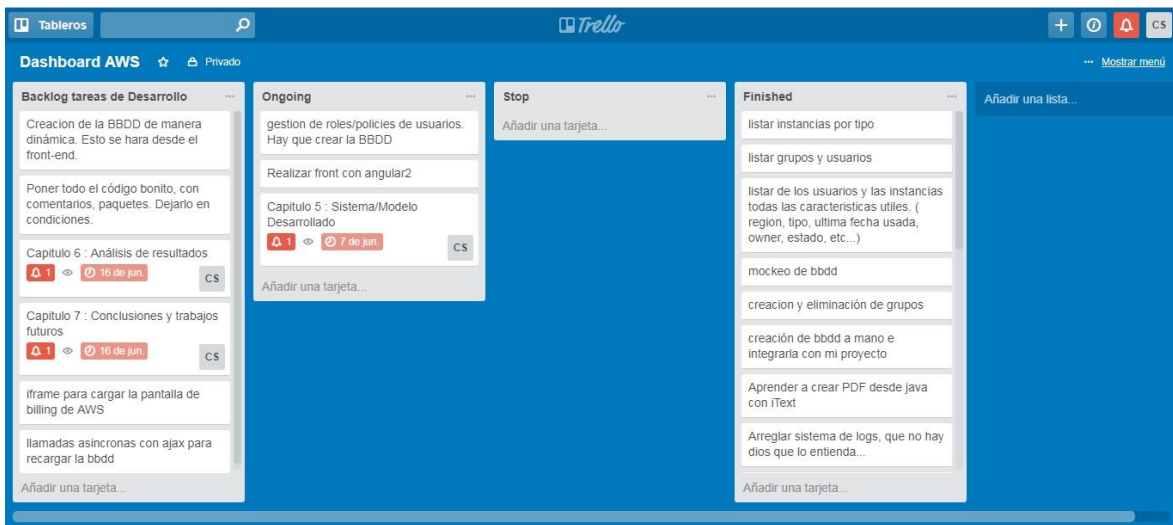


Figura 3: Tareas del desarrollo establecidas en trello.

Se realizarán revisiones periódicas con el director (Sprint) cada dos semanas para ver y comprobar el desarrollo del proyecto y definir los nuevos objetivos de funcionalidad que se pueden seguir implementando. Es importante decir que, en el inicio del proyecto, parte de la funcionalidad de la aplicación web es desconocida, por lo que esta metodología es bastante útil, ya que permite realizar cambios en el proyecto sin provocar grandes cambios en el proyecto.

Para aclarar un poco las fases que va a tener el proyecto, realizamos un cronograma para visualizar mejor todas las tareas que hemos definido para organizar el proyecto, pero la metodología, será ágil. Como durante el primer mes de la realización de prácticas estuve rotando por los distintos departamentos de la entidad, no se dedicó tiempo al proyecto, por lo que realmente el proyecto se comenzó a mediados de febrero. Mostramos el cronograma a continuación.

4.4 PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA



Figura 4. Cronograma del proyecto.

Como podemos observar en la Figura 4, el proyecto constará de tres partes principales. La primera consistirá en el estudio de la tecnología de AWS y conocer cómo funciona, así como definir los requisitos que la aplicación web tendrá. Posteriormente, comenzaremos a desarrollar toda la lógica que llevará el proyecto. Como podemos observar, supondrá la mayor parte del tiempo total de desarrollo. Este aspecto se debe a que los conocimientos previos sobre la API de AWS son totalmente nulos, y, por tanto, se tendrá que dedicar mucho más tiempo a conocer cómo funcionan las clases y los métodos que tiene, para después aplicarlo a la funcionalidad que nosotros queremos. Además, se deberá conocer la forma de trabajar de manera profesional, usando un IDE y utilizando el repositorio de librerías Maven, que no se habían utilizado previamente nunca.

A mediados de mayo, se deberá tener toda la lógica desarrollada para empezar a desarrollar la parte visual de la web. Tendremos que conectar mediante API-REST con nuestra lógica, así como desarrollar un entorno amigable y fácil de usar, utilizando *frameworks* como Angular 2 o Bootstrap.

En teoría para finales de junio, el proyecto debe estar acabado para poder ser presentado a principios de Julio en la universidad.

Como se ha podido observar, nuestro proyecto tiene una duración de aproximadamente 5 meses. Para la consecución del mismo, se producen una serie de costes que analizaremos en este apartado y que suponen la estimación económica de la aplicación web. Los analizamos la Tabla 2.

DEFINICIÓN DEL TRABAJO

<i>Concepto</i>	<i>Justificación</i>	<i>Coste</i>
Equipo	Equipo portátil de 8 GB de memoria RAM y 500 GB de disco duro	400 €
Herramientas	Incluyendo: IntelliJ, Java, Spring, Angular 2, Bootstrap	0 €
Cuenta en AWS	Cuenta de registro en el servicio <i>cloud</i> de Amazon	0 €
Coste de la implementación	Coste de un desarrollador (50€/h) multiplicado por un total de 400 horas (5 meses de trabajo a media jornada)	20000 €
Total		20400 €

Tabla 1. Estimación económica de la aplicación web

Como podemos observar, hemos dividido los costes en 4 apartados. El equipo de trabajo HP con el que estamos desarrollando la aplicación, que cuesta aproximadamente unos 400 €. Todas las herramientas son gratuitas, ya que la versión *ultimate* del IDE IntelliJ no cuesta nada para estudiantes, y todos los *frameworks* son de libre distribución. Tener una cuenta en AWS es totalmente gratuito, únicamente se paga si haces uso de alguno de sus servicios, y en algunos casos, si se hace un uso escaso, también es gratuito. Por último, hay que analizar el coste del desarrollador. Un ingeniero tiene un coste para la empresa de aproximadamente 50 €/hora, por lo que el coste aproximado de tener a una persona haciendo este proyecto son 20000€. En total, tendríamos un coste de 20400 €.

Capítulo 5. SISTEMA DE DESARROLLO

En este capítulo es donde el alumno debe describir su proyecto. En función del tipo de proyecto la estructura interna variará. El título del mismo, así como sus apartados, son sólo una sugerencia que cada alumno deberá adaptar particularmente a su proyecto.

5.1 ARQUITECTURA LÓGICA DE LA APLICACIÓN

A continuación, se describe en profundidad las distintas partes de la arquitectura y las herramientas que la componen.

5.1.1 DIAGRAMA A ALTO NIVEL DE LA APLICACIÓN

El funcionamiento de la aplicación a alto nivel se describe en la siguiente figura. Más adelante explicaremos en más profundidad cómo trabaja la aplicación.



Figura 5: Esquema de la aplicación a alto nivel

El proyecto está compuesto en dos partes muy diferenciadas:

- **Código *back-end*:** Es toda la lógica de la aplicación web que se va a desarrollar. Deberá realizar todas las peticiones AWS a través de la API para obtener toda la información que sea necesaria.
- **Código *front-end*:** Una vez se haya realizado toda la lógica necesaria, pasaremos a realizar el interfaz de la web, es decir, la parte visual de nuestra aplicación.
- **Conexión a través de la API-REST:** Es la parte encargada de conectar todo nuestro código *back-end* al interfaz para poder monitorizar toda la información obtenida.

5.2 DISEÑO

A continuación, mostraremos y explicaremos el diseño de la aplicación entrando un poco más en detalle.

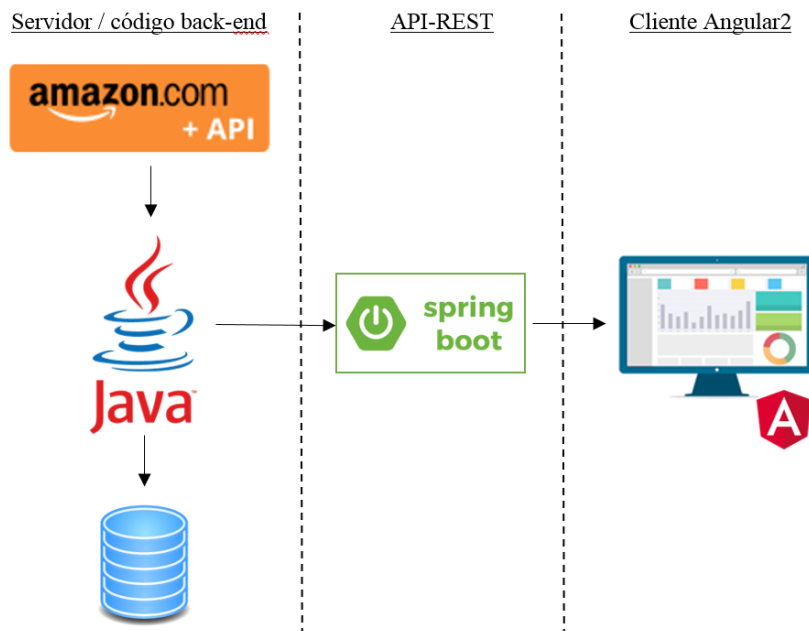


Figura 6: Esquema más detallado de la aplicación

En este apartado definiremos todas las partes del proyecto, dando un repaso por encima.

Muchas aplicaciones hoy en día se basan en dos partes principales, el servidor, que es dónde se almacenará toda nuestra lógica, y el cliente, que mediante peticiones http obtendrá la información correspondiente del servidor. Para conectarlos, necesitamos un servicio REST, que a través de URI's (cadena de caracteres que identifica los recursos de una red de forma unívoca) se nos proporciona esta conexión. La principal diferencia respecto a una URL (localizador de recursos uniforme) es que estos últimos hacen referencia a recursos que, de forma general, pueden variar en el tiempo [8]. Todos los detalles del desarrollo se explicarán en el apartado de implementación. Para facilitar el entendimiento de las tareas que va a poder realizar el usuario que utilice la aplicación, mostramos los casos de uso posibles:

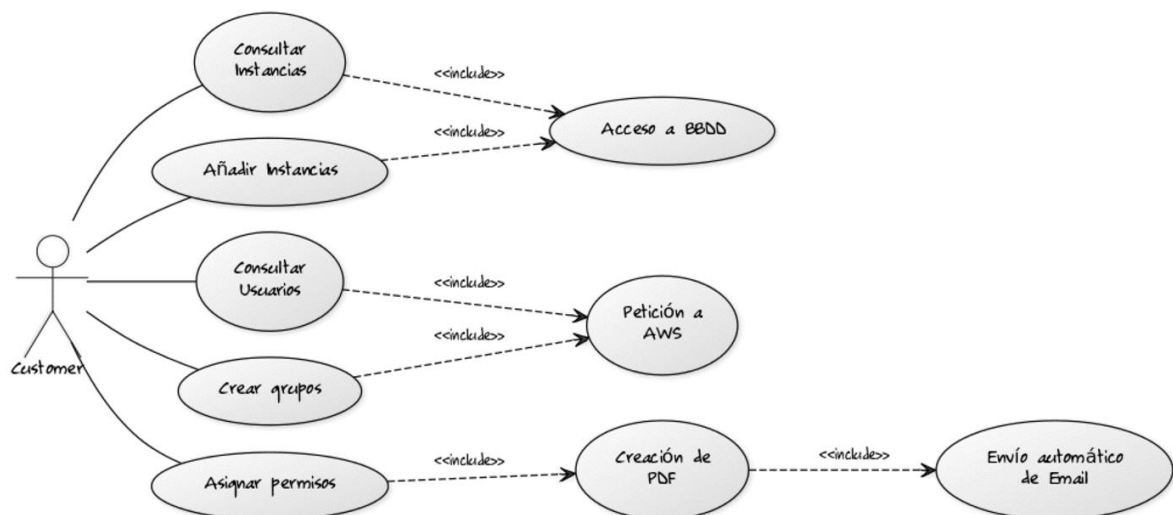


Figura 7: Diagrama de caso de uso.

5.2.1 SERVIDOR / CÓDIGO BACK-END

El primer paso es tener un usuario dentro de la cuenta de la entidad en AWS. El segundo paso es entender cómo funciona la API de AWS para hacer las correspondientes peticiones. Dicha API está dividida en paquetes que representan cada uno de los servicios y funcionalidades que AWS proporciona. Dentro de cada paquete, se encuentran las clases correspondientes para conseguir la funcionalidad que se desea. Es importante comentar, que hay cierta funcionalidad útil que la API no te permite realizar, cómo consultar el coste de cada instancia levantada. Dentro de la cuenta de AWS se pueden consultar los gastos globales por tipo de instancias, pero desde fuera no se permite acceder a dichos datos. Más adelante lo comentaremos.

Para acceder a la API de AWS, necesitamos cargar en nuestro proyecto el “*AWS SDK for Java*”, que es la herramienta que nos da acceso a todas las clases y métodos. La manera de hacerlo es igual que incorporar cualquier librería en un proyecto Java. En nuestro caso, como tenemos un proyecto Maven, podemos hacerlo fácilmente.

El servidor va a contener toda nuestra lógica, por tanto, antes de desarrollar cualquier tipo de funcionalidad, es necesario obtener un *token* de autenticación de AWS para hacer que todas nuestras peticiones a la API lleven un identificador, demostrando quienes somos y a qué entidad pertenecemos, en nuestro caso, al BBVA. Esta parte en el desarrollo es clave, ya que, si no se posee ningún tipo de credencial de autenticación, AWS no permite ninguna llamada a su API. La seguridad que tiene es evidente, ya que, si no fuera así, cualquiera podría acceder a todos los datos de cualquier cuenta abierta con dichos servicios.

En este punto ya estamos preparados para realizar todas las peticiones necesarias a AWS para desarrollar la funcionalidad que deseamos.

Como nuestro sistema de monitorización se va a basar únicamente en los servicios que tenemos levantados (EC2, S3 y CloudSearch), deberemos centrarnos en dichos paquetes y en sus clases para obtener la información que deseamos. También deberemos prestar especial

atención a todos aquellos paquetes relacionados con los usuarios y los permisos. Recordemos que todos estos pasos se explicarán mejor en el tema de implementación.

Crearemos, además, una base de datos persistente a partir de Spring, un *framework* para Java que facilita muchas tareas, como la inserción o consulta de datos a través de API REST.

5.2.2 API-REST

REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Es una alternativa en auge a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol). En nuestra aplicación, leeremos del servidor objetos JSON desde el cliente.

Sus principales características son las siguientes:

- **Protocolo cliente/servidor sin estado:** cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla. Aun así, algunas aplicaciones HTTP incorporan memoria caché. En estos casos, se configura el llamado protocolo cliente-caché-servidor sin estado. En dicho protocolo, existe la posibilidad de definir algunas respuestas a peticiones HTTP concretas como “cacheables”, con el objetivo de que el cliente pueda ejecutar en un futuro la misma respuesta para peticiones idénticas, leyendo en la memoria caché, sin necesidad de volver a enviar la petición. En cualquier caso, que exista la posibilidad no significa que sea lo más recomendable.

- Las **operaciones** más importantes y comunes en relación con los datos en cualquier sistema REST y la especificación HTTP son cuatro:

- POST (crear)
- GET (leer y consultar)
- PUT (editar)

- DELETE (eliminar).

En nuestro caso, como lo que deseamos es consultar toda la información desarrollada en el servidor, realizaremos peticiones GET.

- Los **objetos** en **REST** siempre se manipulan a partir de la URI. Es la URI y ningún otro elemento el identificador único de cada recurso de dicho sistema REST. La URI nos permite acceder a la información para su modificación o borrado, o para compartir su ubicación exacta con terceros.
- **Interfaz uniforme:** para la transferencia de datos en un sistema REST, este aplica operaciones concretas como las mencionadas anteriormente sobre los recursos, siempre y cuando estén identificados con una URI. Esto facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información.
- **HATEOAS** (Hypermedia As The Engine Of Application State - Hipermedia Como Motor del Estado de la Aplicación). Este principio es de uso obligatorio para cualquier sistema API REST. Es el que define que cada vez que se hace una petición al servidor y este devuelve una respuesta, parte de la información que contendrá serán los hipervínculos de navegación asociada a otros recursos del cliente [9].

En nuestro proyecto, para realizar la API REST, también haremos uso de Spring. Crearemos una clase llamada 'Rest_Controller' que tendrá toda la lógica necesaria para definir en una 'URI' toda la funcionalidad que se desea realizar. Recordamos que, en nuestro cliente, accederemos a cualquier información a través de la URI correspondiente.

5.2.3 CLIENTE ANGULAR 2

Nuestro cliente, que es dónde mostraremos todos los datos necesarios, será Angular 2. Angular 2 es uno de los últimos *frameworks* desarrollados para aplicaciones web, creado por Google. Angular 2 se basa en componentes independientes. Crear un componente es como crear una etiqueta html nueva y propia. Al incorporar dicha etiqueta al ‘index.html’ de nuestro proyecto, cargamos el componente. Cada componente tiene su propia vista y su propia lógica.

Nosotros desarrollaremos dos componentes principales, el “header” y el “body”. En el *header* se encontrará la cabecera de la página web, con el logo del banco y el departamento de desarrollo, mientras que en el *body* se mostrará toda la información correspondiente. Como la aplicación debe estar preparada para futuros servicios *cloud*, y no solo para AWS, tendremos un sistema de pestañas para diferenciarlos. Serán tres principales: AWS, Google *Cloud*, y Microsoft Azure. Estas dos últimas se dejarán en la web para el futuro desarrollo de la misma, nuestra tarea será completar lo máximo posible la pestaña de AWS.

Dentro de dicha pestaña, tendremos un sub-menú con los principales puntos a mostrar. Los principales objetivos ya fueron explicados anteriormente, pero los volvemos a resumir brevemente para facilitar la comprensión del apartado. Los principales objetivos estaban separados en 4 puntos: Usuarios, Servicios, Permisos, y alarmas. El aspecto económico no se ha podido desarrollar ya que, dentro de la empresa, el departamento que tiene acceso a datos de índole económica no puede ceder los permisos para acceder a este tipo de información, con lo que el desarrollo de esta parte no es posible.

Las pestañas de nuestro sub-menú dentro de AWS, se dividirán pues, en Servicios, Usuarios y *policies*. También se debería incluir una pestaña de “*Billing*”, pero por temas internos a la empresa, no se puede realizar, por lo que no incluiremos ninguna pestaña para albergar datos económicos. Lo explicaremos en el tema de trabajos futuros. Cómo las alarmas se ejecutan en segundo plano, no será necesario incluir una pestaña sólo para ellas. En servicios, se mostrarán una serie de gráficos representando datos de manera intuitiva. Además, se dará la posibilidad de mostrar todas las instancias de cada servicio junto con sus características más

representativas. En la pestaña de usuarios se mostrarán todos los usuarios pertenecientes a nuestra cuenta de AWS. También se dará la posibilidad de crear un grupo para poder asignarle unos determinados usuarios posteriormente. Y finalmente en *policias*, aparecerán dos tablas. La primera con todos los usuarios actuales que haya activos, y la otra con todas las *policias* aplicables a los usuarios. De esta manera, se podrán marcar todos aquellos permisos que se deseen aplicar a los usuarios correspondientes. Se generará automáticamente un PDF que se enviará al responsable de la cuenta para que asigne dichos permisos. Los permisos no se asignan directamente desde la web por falta de permisos dentro de la entidad. Como esta sección trata la parte visible de la aplicación, mostramos una serie de imágenes para explicar el diseño de la misma.

A continuación, mostramos ciertas imágenes del proyecto ya terminado para explicar cómo se ha llevado a cabo el diseño de la interfaz de la aplicación.

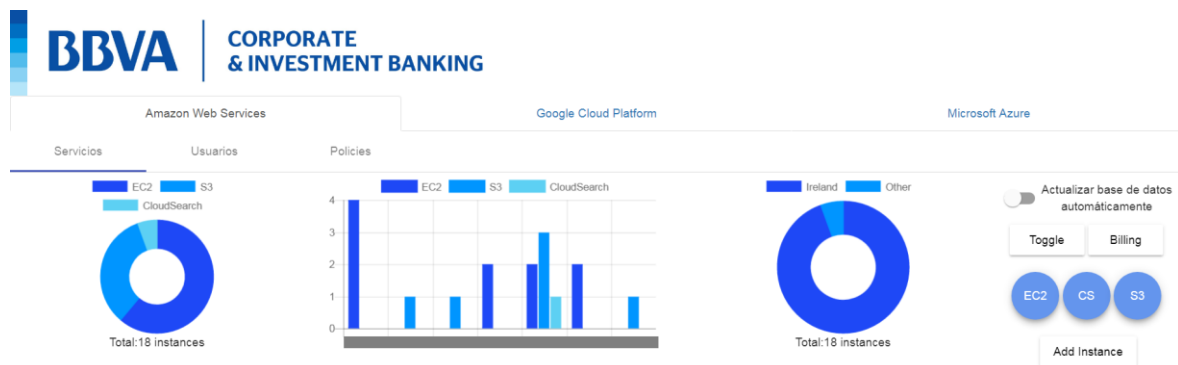


Figura 8: Inicio de la aplicación

Por temas de confidencialidad, cierta información propia del banco no se puede mostrar en la memoria, por lo que ocultaremos dichos datos. En este caso, los id de usuarios aparecen ocultos. Se encuentran tras el rectángulo gris. Como podemos observar, en la parte superior se encuentra el logo del departamento de desarrollo de la entidad. Esta zona se corresponde con el “*header*” de la página web. A su vez, el resto de la página compone el “*body*” de la aplicación.

Se quiere monitorizar los datos de los servicios *cloud* posibles que tiene el banco en activo. Actualmente se tienen máquinas alquiladas en AWS y en Google *Cloud*, sin embargo, para

Microsoft, habrá que esperar. En cualquier caso, la página debe quedar preparada, y, como podemos ver, así ha sido. Como el proyecto consiste únicamente en monitorizar los datos de AWS, habrá dos pestañas que no contendrán ningún dato, esperando a ser completadas. En ellas, aparecerá la siguiente imagen, indicando que el sitio está en construcción.



Figura 9: Parte de la web en desarrollo.

Se puede observar, que el menú principal está dividido en tres pestañas diferentes, cada una indicando un tema a monitorizar diferente, que son, “Servicios”, “Usuarios” y “Policies”.

En primer lugar, en lo alto de la pestaña de AWS, podemos observar 3 gráficos, con el fin de obtener una información de manera rápida y que sea útil para hacernos una idea de lo que se tiene contratado.

El primero de ellos muestra el número de instancias que se tienen alquiladas de cada tipo. Sólo se estudian tres tipos de instancias, por lo que cómo máximo se tendrán tres distinciones diferentes

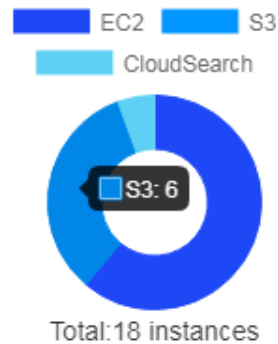


Figura 10: Gráfico uno. Instancias por tipo.

Como podemos ver, podemos mostrar de manera gráfica cuántas instancias se tienen de cada tipo. El segundo de ellos es un gráfico de barras, que indica las instancias que ha levantado cada usuario.

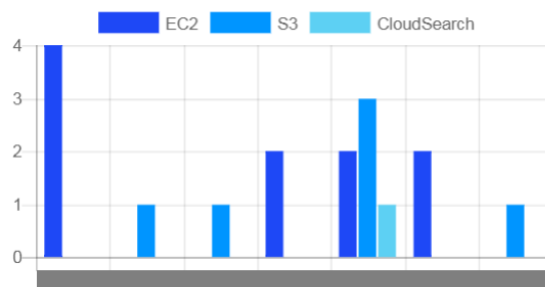


Figura 11: Gráfico dos. Instancias levantas por usuario

El último de ellos simplemente muestra cuántas instancias hay levantas en Irlanda y fuera de ella.

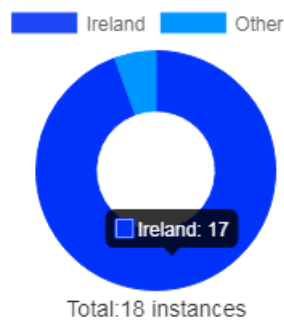


Figura 12: Gráfico tres. Cantidad de instancias en Irlanda y fuera de ella.

Dentro de la entidad, es bastante crítico saber en que región se ha levantado cada instancia y llevar un control sobre ello. Gracias al gráfico anterior, lo conseguimos.

Todos los gráficos van a permitir seleccionar que información mostrar. Es decir, si en la leyenda tachamos alguno de los datos, la información relativa a este elemento desaparecerá automáticamente. Evidentemente, aunque en el gráfico se quite información, el total de instancias seguirá siendo el mismo.



Figura 13: Gráfico uno con datos suprimidos.

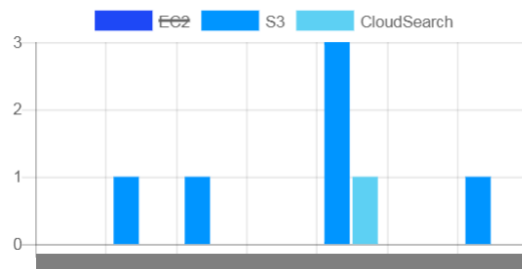


Figura 14: Gráfico dos con datos suprimidos.

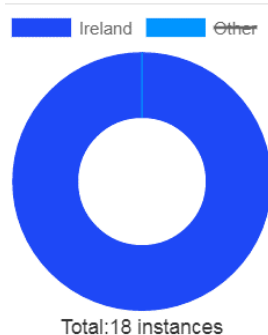


Figura 15: Gráfico tres con datos suprimidos.

Así, la parte de gráficos queda cerrada.

En la parte derecha de la aplicación, podremos ver un pequeño menú, compuesto por distintos botones y que tienen una funcionalidad concreta. Lo mostramos a continuación.



Figura 16: Menú de los servicios.

El botón de “*Toggle*”, que en español significa conmutar, cambia literalmente el tipo de gráfico de los gráficos uno y tres. Pasan de ser una corona circular a un gráfico circular puro y duro. Es un elemento puramente estético.

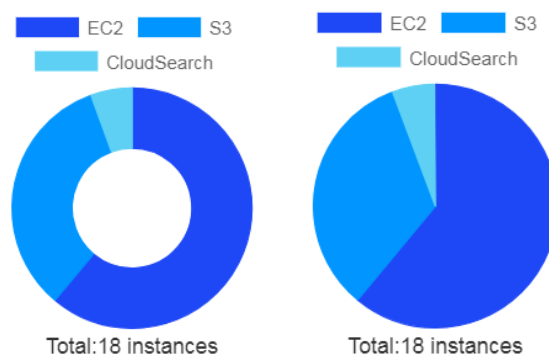


Figura 17: Comparación gráficos circulares

El botón de “*Billing*” nos abre una pestaña nueva en el navegador con los costes oficiales que debe asumir la entidad con la cuenta de AWS. Aquí podemos ver los costes globales por tipo de instancias, pero no por instancia individual o por usuarios, que era lo que se deseaba, pero debido a la negación de permisos, esta tarea no se pudo realizar. A continuación,

mostramos una imagen del *dashboard* de costes que proporciona AWS. Como la imagen está tomada a principios de mes, el coste es bastante bajo.

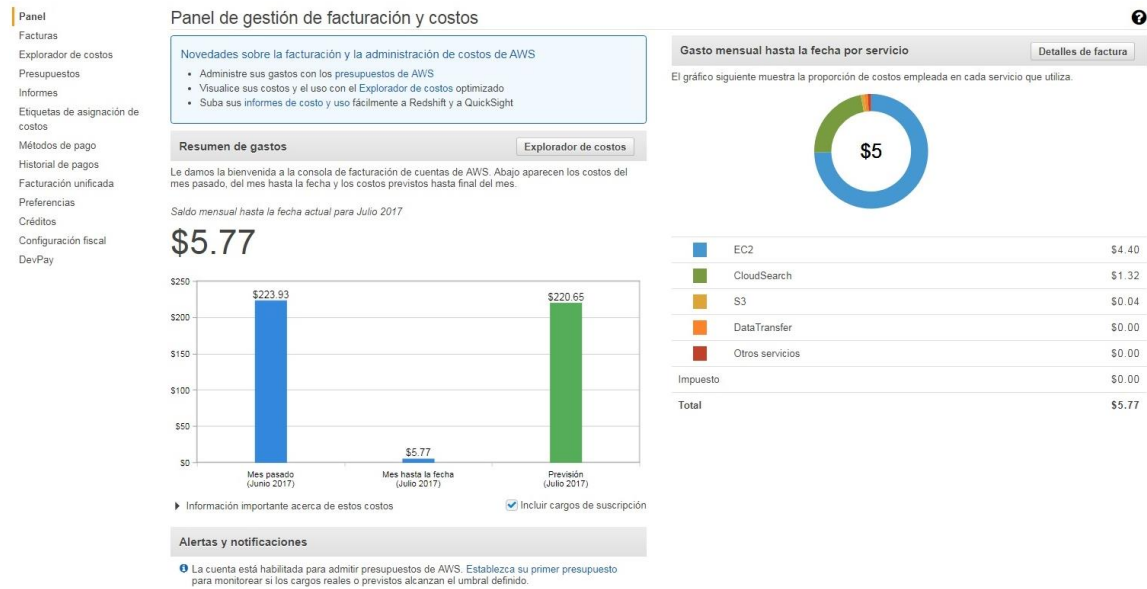




Figura 18: Dashboard oficial de billing en AWS.



Los tres botones que están a continuación, con forma redonda y de color azul, son los que van a activar las tablas de instancias. Evidentemente, cada uno mostrará las del tipo que les corresponda, haciendo un “*scroll*” automático a la tabla correspondiente.

A continuación, mostramos las distintas tablas que se pueden mostrar, monitorizando todas las instancias actuales en AWS, junto con sus características más representativas, como se indicaba en los objetivos. De nuevo, por temas de confidencialidad, los nombres de las instancias no pueden ser mostrados.

 EC2 Instances 



Name	Id	LaunchDate	Region	State	Type	Owner	Dns	Platform
[REDACTED]	i-0704da293804c948c	2017-04-09 09:35:18	eu-west-1b	running	t2.micro		ip-172-31-39-191	Other Linux
[REDACTED]	i-0d473d8afd5b1ad9	2017-06-29 16:02:08	eu-west-1b	stopped	t2.micro		ip-172-31-44-178	Red Hat
[REDACTED]	i-0625a23ef5686a230	2017-01-11 14:27:29	eu-west-1b	stopped	t2.micro		ip-172-31-43-50	Amazon Linux
[REDACTED]	i-01b55ea35904b050e	2017-06-02 10:09:54	eu-west-1b	stopped	t2.micro		ip-172-31-39-10	Other Linux
[REDACTED]	i-0f24174dc0f948c66	2017-01-25 15:33:26	eu-west-1a	stopped	m4.xlarge		ip-172-31-19-149	Other Linux
[REDACTED]	i-0ff243d2a4f2fab5	2017-05-24 07:21:40	eu-west-1a	running	t2.medium		ip-172-31-20-12	Other Linux
[REDACTED]	i-0992f8dea8d246c9	2017-04-26 13:30:07	eu-west-1b	running	t2.micro		ip-172-31-41-35	Amazon Linux
[REDACTED]	i-0755d2dc499e38606	2017-06-26 15:15:28	eu-west-1b	stopped	m4.2xlarge		ip-172-31-36-19	Other Linux
[REDACTED]	i-02513240c1b045127	2017-03-02 16:09:04	eu-west-1b	running	t2.micro		ip-172-31-35-111	Other Linux
[REDACTED]	i-07210b9e9f48145c2	2017-04-11 12:45:04	eu-west-1b	running	t2.nano		ip-172-31-32-162	Amazon Linux
[REDACTED]	i-01a2c100e99c3c383	2016-10-20 13:48:56	eu-west-1b	running	m4.2xlarge	CIB_TA	ip-172-31-44-166	Other Linux

Figura 19: Tabla de instancias EC2

 S3 Instances 

Name	Region	Size - GB	Owner	CreationDate
[REDACTED]	EU-WEST-1	0,0001	ambitodigital+ResearchCIB.dev	2017-06-14 12:22:36
[REDACTED]	EU-WEST-1	0,0134	ambitodigital+ResearchCIB.dev	2017-03-08 08:46:09
[REDACTED]	EU-WEST-1	37,0959	ambitodigital+ResearchCIB.dev	2017-06-26 13:43:49
[REDACTED]	EU-WEST-1	0	ambitodigital+ResearchCIB.dev	2017-01-27 08:47:35
[REDACTED]	EU-WEST-1	0,0071	ambitodigital+ResearchCIB.dev	2017-03-23 16:32:22
[REDACTED]	EU-CENTRAL-1	0,2331	Unknown	2017-01-27 08:47:02

Figura 20: Tabla de instancias S3

 CloudSearch 

Name	Region	Type	Number of Index Fields	Partitions
[REDACTED]	eu-west-1	search.m1.small	18	1

Figura 21: Tabla de instancias CloudSearch.

Como se puede ver, las tres tablas tienen una estructura similar. Cada una de ellas se compone de su logo, el título, y un botón de actualizar. Como los datos de la tabla se leen de la base de datos, este botón permite actualizar dicha base de datos cuando se desee.

Las columnas que componen cada tabla son atributos referentes a cada tipo de instancia, y, por tanto, no son los mismos en todas ellas.

El hecho comentado anteriormente de que el usuario pueda actualizar manualmente la base de datos es una funcionalidad añadida pero no es realmente útil. Si se ha creado una instancia

en AWS, lo correcto sería que la aplicación lo detectara automáticamente y los datos se actualizarán solos. Es aquí donde entra en juego el botón de actualizar la base de datos automáticamente. Al activar este botón, se permite la actualización automática de la base de datos, y, la manual se deniega (los botones no se pueden pulsar). Cada tabla se podrá actualizar de manera independiente. La tabla de instancias S3 se actualiza cada 2 minutos. La tabla de instancias EC2, cada 1, y la tabla de CloudSearch, cada 3 minutos.

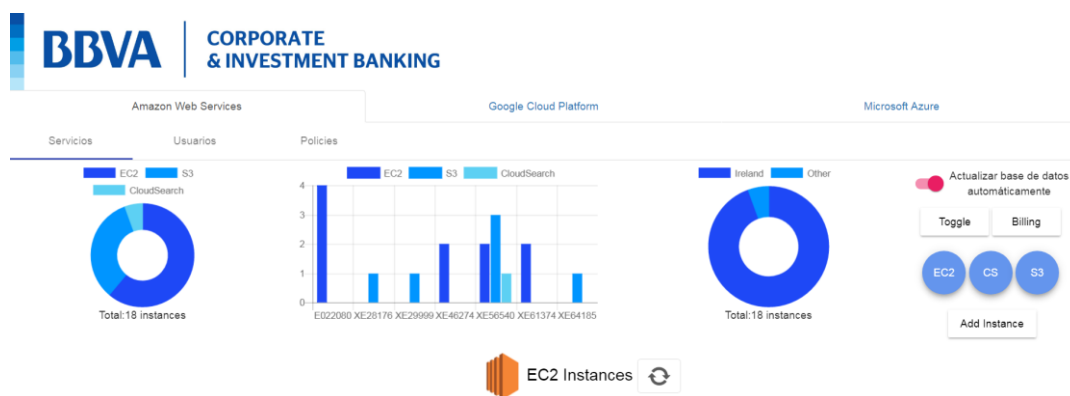


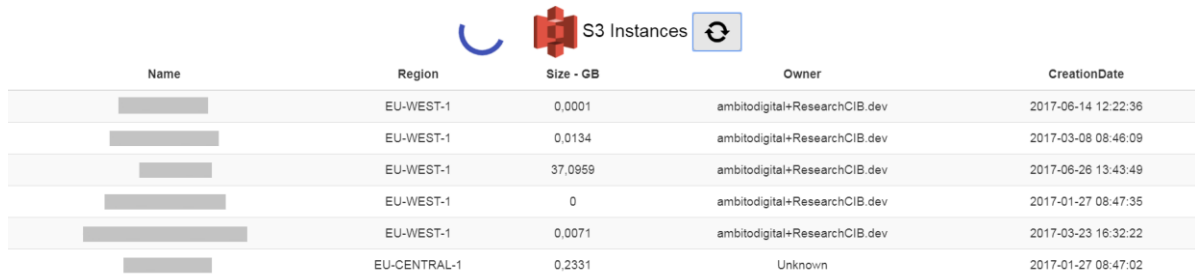
Figura 22: Botón de actualización manual desactivado.

La manera de saber que la tabla correspondiente se está actualizando, es mediante un “spinner” de progreso. Cada vez que una tabla se actualice, este dibujo aparecerá al lado del logo de cada tabla, indicando así que se está actualizando, y para que, por tanto, el usuario sea consciente de eso.

The screenshot shows a table titled 'EC2 Instances' with a spinner icon. The table contains the following data:

Name	Id	LaunchDate	Region	State	Type	Owner	Dns	Platform
[Redacted]	i-0704da293804c948c	2017-04-09 09:35:18	eu-west-1b	running	t2.micro		ip-172-31-39-191	Other Linux
[Redacted]	i-0d473d8afd5bf1ad9	2017-06-29 16:02:08	eu-west-1b	stopped	t2.micro		ip-172-31-44-178	Red Hat
[Redacted]	i-0625a23ef5686a230	2017-01-11 14:27:29	eu-west-1b	stopped	t2.micro		ip-172-31-43-80	Amazon Linux
[Redacted]	i-01b55ea35904b050e	2017-06-02 10:09:54	eu-west-1b	stopped	t2.micro		ip-172-31-39-10	Other Linux
[Redacted]	i-0f24174dc0f946c66	2017-01-25 15:33:26	eu-west-1a	stopped	m4.xlarge		ip-172-31-19-149	Other Linux
[Redacted]	i-0ff243d2a4f2feb5	2017-05-24 07:21:40	eu-west-1a	running	t2.medium		ip-172-31-20-12	Other Linux
[Redacted]	i-0992f8dea88d246c9	2017-04-26 13:30:07	eu-west-1b	running	t2.micro		ip-172-31-41-35	Amazon Linux
[Redacted]	i-0755d2dc499e38606	2017-06-26 15:15:28	eu-west-1b	stopped	m4.2xlarge		ip-172-31-36-19	Other Linux
[Redacted]	i-02513240c1b045127	2017-03-02 16:09:04	eu-west-1b	running	t2.micro		ip-172-31-35-111	Other Linux
[Redacted]	i-07210b9e9f48145c2	2017-04-11 12:45:04	eu-west-1b	running	t2.nano		ip-172-31-32-162	Amazon Linux
[Redacted]	i-01a2c100e99c3c383	2016-10-20 13:48:56	eu-west-1b	running	m4.2xlarge	CIB_TA	ip-172-31-44-166	Other Linux

Figura 23: Tabla EC2 actualizándose.



Name	Region	Size - GB	Owner	CreationDate
[Redacted]	EU-WEST-1	0,0001	ambitodigital+ResearchCIB.dev	2017-06-14 12:22:36
[Redacted]	EU-WEST-1	0,0134	ambitodigital+ResearchCIB.dev	2017-03-08 08:46:09
[Redacted]	EU-WEST-1	37,0959	ambitodigital+ResearchCIB.dev	2017-08-26 13:43:49
[Redacted]	EU-WEST-1	0	ambitodigital+ResearchCIB.dev	2017-01-27 08:47:35
[Redacted]	EU-WEST-1	0,0071	ambitodigital+ResearchCIB.dev	2017-03-23 16:32:22
[Redacted]	EU-CENTRAL-1	0,2331	Unknown	2017-01-27 08:47:02

Figura 24: Tabla S3 actualizándose.

Si hay algún cambio en alguna tabla, porque realmente se ha creado alguna instancia en AWS, los datos se actualizarán automáticamente cada vez que la actualización de la base de datos detecte esta instancia nueva. Los gráficos circulares se activarán y se actualizarán, indicando el número nuevo de instancias. Antes de hablar de la gestión del gráfico dos para saber que usuario ha levantado cada instancia, vamos a comentar el diseño de las alertas. Como ya dijimos, se debería mandar un correo cada vez que se detectara que una instancia estaba levantada fuera de Irlanda o si una instancia EC2, esta levantada con un sistema operativo Red Hat. Gráficamente no ocurrirá nada en el interfaz, simplemente el responsable obtendrá un correo de alerta. Esto es así ya que una alerta no tiene porque indicar una funcionalidad que debe ser eliminada.

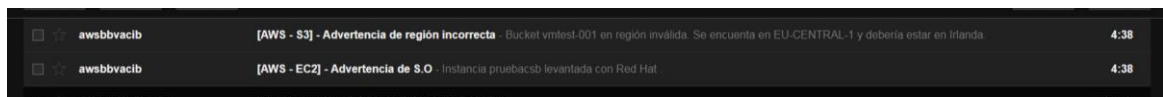


Figura 25: Correos de alerta

[AWS - S3] - Advertencia de región incorrecta Recibidos x



awsbbvacib@gmail.com

para mí

Bucket vntest-001 en región inválida. Se encuentra en EU-CENTRAL-1 y debería estar en Irlanda.

Figura 26: Advertencia de región incorrecta

[AWS - EC2] - Advertencia de S.O Recibidos x

awsbbvacib@gmail.com
 para mí ▼
 Instancia prueba**csb** levantada con Red Hat .

Figura 27: Advertencia de instancia EC2 con Red Hat.

El deber del usuario será introducir esta instancia nueva en la base de datos para actualizar el gráfico de barras. Para ello, está el último botón del pequeño menú que estamos comentando. El botón “*Add Instance*” es el que permitirá hacerlo. Al pulsarlo, aparecerá un formulario de entrada con unos determinados parámetros, que serán los mínimos necesarios para crear una instancia en la base de datos, y que, por tanto, se muestre en el gráfico de barras.

The screenshot shows the BBVA Corporate & Investment Banking dashboard. It features three main sections for cloud services: Amazon Web Services, Google Cloud Platform, and Microsoft Azure. Under Amazon Web Services, there are three donut charts: 'Servicios' (showing EC2 and S3), 'Usuarios' (showing EC2, S3, and CloudSearch), and 'Policies' (showing EC2, S3, and CloudSearch). A central bar chart displays usage across different regions. On the right, there are buttons for 'Toggle', 'Billing', and 'Add Instance'. A green 'Añadir Instancia' button is visible at the bottom right of the dashboard.

Figura 28: Visualización del formulario para añadir una instancia.

El formulario está comprobado, es decir, si no se rellenan todos los campos, no se realizará ninguna acción y se mostrará un “*warning*”. En el ejemplo que se muestra a continuación, hemos dejado sin completar la región.

The screenshot shows the 'Add Instance' form with the following fields: 'Nombre de la instancia' (Prueba), 'Usuario' (Carlos), 'Grupo del usuario' (Admins), 'Región' (empty), and 'Tipo' (EC2). A green 'Añadir Instancia' button is present. A red error message box at the bottom right states: "¡Error! Por favor, rellena todos los campos."

Figura 29: Formulario mal rellenado.

En el caso contrario, si todo se realiza correctamente, saldrá un aviso diciendo que la creación de la nueva instancia en la base de datos ha sido correcta, y, por tanto, debería actualizarse el gráfico de barras.



Figura 30: Formulario correcto.

Con esta explicación, se termina el diseño de la primera pestaña de AWS, llamada “Servicios”, y pasamos a explicar la pestaña de “Usuarios”.

Como su propio nombre indica, la pestaña “*Usuarios*” gestionará todo lo relativo a los usuarios. Tendremos dos botones que aparecerán en la parte superior de la ventana.

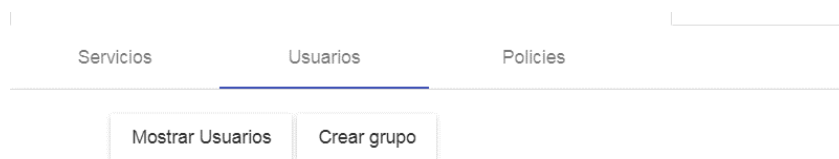




Figura 31: Pestaña de usuarios

El primero de ellos se llamará “Mostrar usuarios”, y mostrará los usuarios dados de alta en AWS dentro de la cuenta del banco en el momento actual, es decir, estos datos no se leen de ninguna base de datos previamente creada. El proceso es el siguiente: Al pulsar el botón, se hace una petición a AWS sobre los usuarios actuales. En la “*response*” se encuentra toda la información requerida. Únicamente la gestionamos, y hacemos que se muestre en una tabla, al igual que en la tabla de instancias. Volvemos a ocultar la información confidencial, los id del banco y de AWS.


 Usuarios 

idBvva	id	Última conexión	Fecha de alta
█	█	2017-06-06 06:54:34	2016-10-19 16:54:28
█	█	2017-04-20 09:38:43	2016-08-26 10:31:04
█	█	2017-04-28 09:06:37	2016-08-26 10:31:04
█	█	Nunca conectado	2016-09-06 08:08:29
█	█	Nunca conectado	2016-10-19 16:54:28
█	█	2017-02-17 07:10:12	2017-01-09 17:09:20
█	█	2017-06-01 20:16:27	2017-01-09 17:09:20
█	█	2017-01-16 09:17:20	2017-01-11 07:23:51
█	█	2016-10-20 09:34:39	2016-10-19 16:54:28
█	█	Nunca conectado	2016-10-19 16:55:06
█	█	2017-06-30 07:31:05	2016-10-19 16:54:28
█	█	Nunca conectado	2016-10-19 16:55:06
█	█	2017-07-04 07:12:41	2016-10-19 16:54:28
█	█	2017-07-03 14:34:26	2016-10-19 16:55:06
█	█	2017-07-05 07:57:37	2017-02-14 16:14:30
█	█	2017-04-26 11:26:52	2017-01-30 15:08:12

Figura 32: Tabla de usuarios

El siguiente botón, “crear grupo”, es el que debe ser pulsado si se desea crear un grupo, para, posteriormente, desde la página oficial, asignarles los usuarios correspondientes. Al pulsarlo, aparece un pequeño formulario de entrada para introducir el nombre del nuevo grupo. De nuevo, salta un error si el formulario no se completa.

Nombre del grupo



Error Debes escribir un nombre de grupo para que la acción se realice correctamente. ✕

Figura 33: Formulario de entrada para grupos mal rellenado

Rellenamos el formulario.

Nombre del grupo



Grupo "Prueba" creado correctamente. ✕

Figura 34: Rellenamos el formulario para los grupos correctamente

Por último, queda por explicar la última de las pestañas de AWS, la pestaña de “Policies”, o también llamados, permisos. Uno de los objetivos principales del proyecto era poder asignar ciertos permisos a los usuarios que se desee. De nuevo, no podemos hacerlo directamente por no disponer de los permisos necesarios, así que se optó por la siguiente solución. Crear un PDF dinámico que indique los usuarios y los permisos que se quieren asignar.

En la pestaña aparecen dos tablas, una de ellas mostrando los usuarios actuales en AWS vinculados al banco, y, en la otra, todas las *policias* disponibles para asignar. Cada una de las tablas tiene en un parte superior un buscador, para facilitar el hecho de acceder a cualquier usuario o cualquier permiso de manera rápida y cómoda, ya que, si no, buscar uno en concreto puede ser una ardua tarea.

Mostramos las tablas completas.

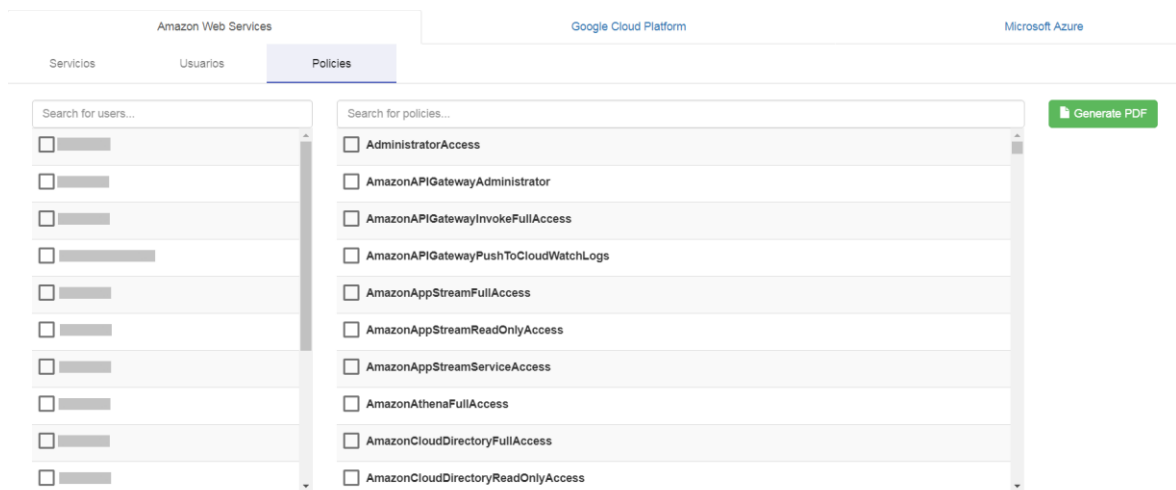


Figura 35: Tablas de usuarios y policias

A continuación, mostramos una captura de cómo funcionan los buscadores. En concreto, buscaremos los usuarios cuyo id comience por “XE2” y todas las *policias* que contengan en su nombre “EC2”, es decir, que sean para este tipo de instancias.

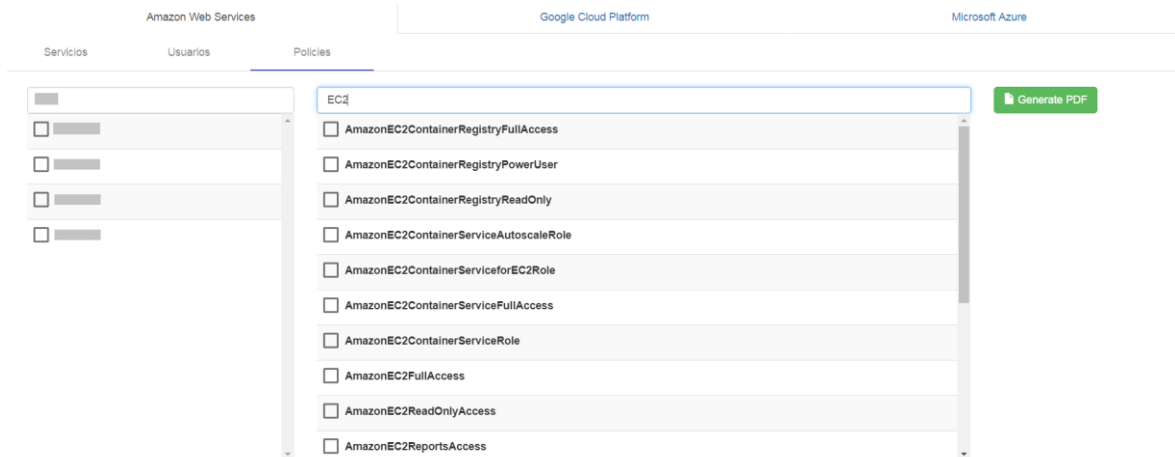


Figura 36: Filtro en tablas

Como podemos ver, los filtros funcionan correctamente, mostrando únicamente cuatro usuarios (no mostrados por confidencialidad) y las *policias* que son de tipo EC2.

Así mismo, todos los elementos de la tabla tienen un elemento de selección para poder ser seleccionados. Es un simple *checkbox*, que, al pulsarlo, se ilumina indicando que elemento se ha seleccionado.

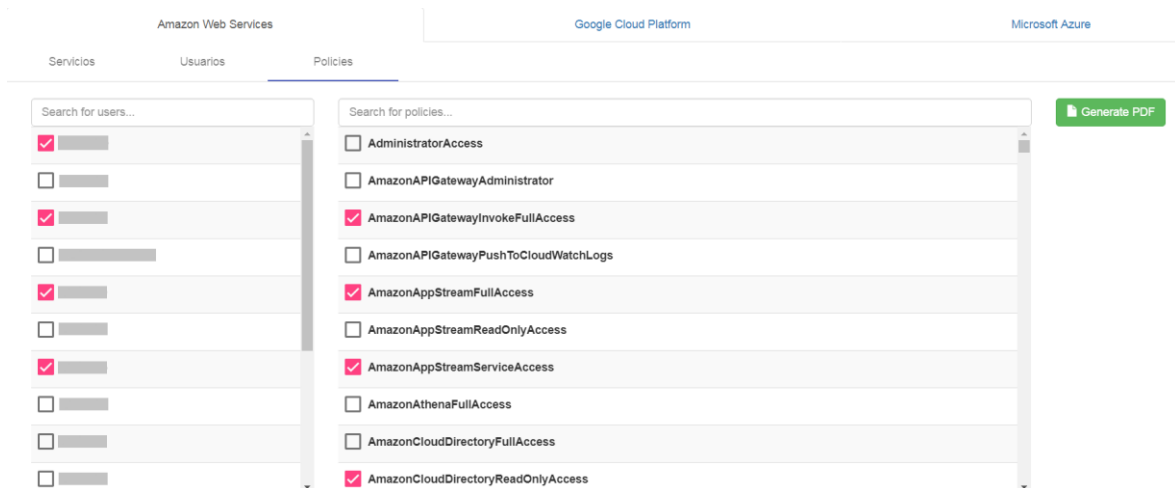


Figura 37: Checkbox de las tablas

De esta manera, podemos indicar fácilmente que permisos queremos asignar a un usuario o usuarios determinados. Por tema de permisos internos en la entidad, no podemos asignar directamente los permisos que se deseen, por lo que se ha llevado a cabo una solución intermedia. La creación de un archivo en formato PDF indicando los usuarios y las *policies* que se desean asignar y que se enviará automáticamente vía email al departamento correspondiente de seguridad para que ellos se encarguen.

Esta acción se realiza pulsando el botón de la derecha, llamado “*Generate PDF*”, que coge la información de las tablas, genera el archivo y se lo envía al departamento correspondiente. Lo mostramos a continuación:



Figura 38: PDF de policies.

Como podemos observar, el PDF se genera con los usuarios y las *policies* que hemos seleccionado en las tablas, cumpliendo satisfactoriamente con uno de los objetivos más importantes del proyecto.

Capítulo 6. IMPLEMENTACIÓN

En este apartado se describe de forma detallada todo el proceso de realización de la aplicación web. Dicha creación, será explicada siguiendo todos los procesos y pequeños adelantos que se han ido haciendo hasta llegar al final de lo que deseamos.

Todos los subapartados que estén aquí redactados supondrán distintas partes del proyecto que merecen ser explicadas al detalle para entender como se ha realizado y cómo funciona la aplicación web. Aunque se aclaren y detallen gran cantidad de procesos en las siguientes páginas, se ha evitado profundizar en estos al límite, por ejemplo, omitiendo el código de los módulos programados o sin mostrar algunos procedimientos.

6.1 SERVIDOR – CÓDIGO BACK-END

6.1.1 CREACIÓN DE PROYECTO MAVEN CON INTELLI-J

Antes de comenzar a hacer cualquier cosa, lo primero es crearnos un proyecto en el IDE correspondiente para poder trabajar. Dicho IDE, en nuestro caso, será el IntelliJ, desarrollado por JetBrains. Las razones de dicha elección son varias, entre ellas, la gran funcionalidad que proporciona su versión ‘*ultimate*’, gratuita para estudiantes, o el interfaz moderno que proporciona. Dentro de dicha funcionalidad, es muy útil que desde la versión *ultimate*, se puedan desarrollar también proyectos web, pudiendo crear archivos html o typescript, algo que desde la versión normal no se podría.

El proyecto, deberá crearse como proyecto Maven, y no como un proyecto Java normal. La ventaja de crearlo de esta manera es que la inyección de dependencias en el proyecto es muy sencilla. En nuestro caso, como vamos a trabajar con Spring, el SDK de AWS, y algunas librerías concretas para Java, es muy útil trabajar con una herramienta como Maven, para añadir dichas librerías de manera muy sencilla.

Para crear el proyecto de esta manera, deberemos seguir la siguiente ruta: File - new – Project – Maven. En el campo “proyect SDK”, se deberá indicar el jdk de java instalado en la máquina, en nuestro caso, el 8. (Evidentemente Java debe estar instalado).

A continuación, mostramos un ejemplo:

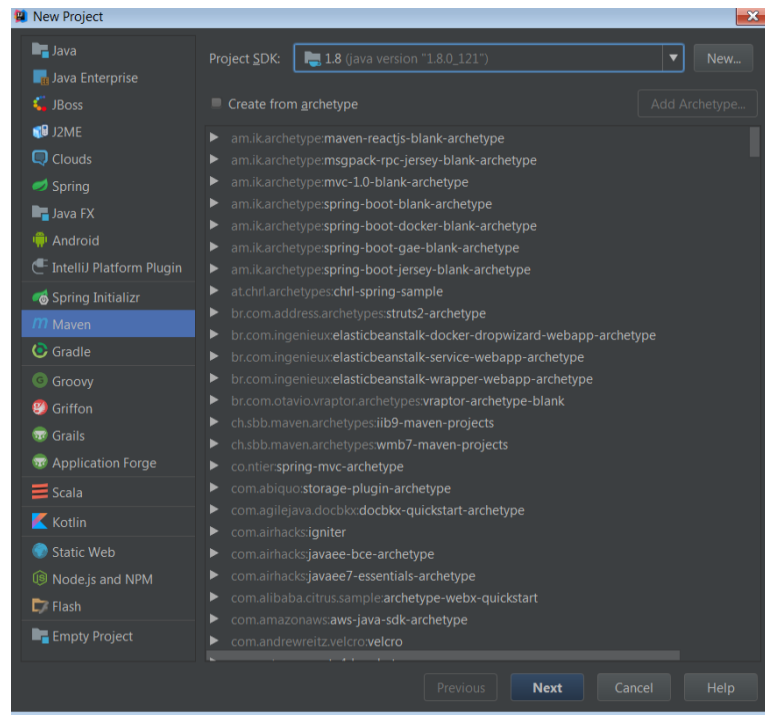


Figura 39: Ejemplo de creación de proyecto Maven.

Después, se deberá indicar el ‘GroupId’ y el ‘ArtifactId’ al proyecto. Estos dos campos no son más que identificativos del proyecto. El ‘GroupId’ de manera predeterminada debe ser el nombre del proyecto, pero si existe un proyecto Maven padre, esta variable puede heredarse de dicho proyecto, indicándolo, y pulsando “inherit”. El ‘ArtifactId’ también es el nombre del nuevo módulo. En nuestro caso los llamaremos “AWSMonitorización”. Posteriormente, indicaremos el nombre del proyecto, que se llamará de igual forma.

A partir de este punto, ya podemos empezar a trabajar en nuestro nuevo proyecto. Como hemos indicado que utilizaremos Java 8, podremos utilizarlo, pero para realizar toda la funcionalidad que deseamos, deberemos añadir varias librerías a nuestro proyecto, por lo que tendremos que aprender a añadir dependencias con Maven. Lo vemos en el siguiente apartado.

6.1.2 INYECCIÓN DE DEPENDENCIAS

La inyección de dependencias con Maven en un proyecto Java no es más que incorporar librerías para su posterior uso de manera muy sencilla. Al crear un proyecto Maven, se genera dentro del mismo un archivo tipo xml que contiene etiquetas con configuraciones indicando que dependencias existen dentro del proyecto. Dicho archivo se denomina POM.xml y cualquier dependencia que queramos incluir, deberá ser indicada en dicho archivo aquí.

Maven tiene un repositorio web dónde están almacenadas todas las dependencias que tiene. Para incluir una de ellas, es tan sencillo como acceder a él, buscar la dependencia que deseamos, copiar el texto que nos facilita el repositorio, y copiarlo dentro de nuestro archivo POM. Todas las dependencias deberán ir dentro de las etiquetas “<dependencies>” y “</dependencies>”.

En nuestro caso, como la principal librería que queremos añadir es la de AWS para Java, lo que tenemos que hacer es ir al repositorio de Maven en Internet, buscar la dependencia y añadirla a nuestro POM. A continuación, mostramos unas imágenes explicativas.

The screenshot shows a web browser window with the URL <https://mvnrepository.com/search?q=aws+sdk+for+java>. The search results page displays the following information:


- Indexed Artifacts (6.52M)**: A line graph showing the number of indexed artifacts from 2004 to 2017, with values ranging from 0 to 6512k.
- Popular Categories**: A list of categories including Aspect Oriented, Actor Frameworks, Application Metrics, Build Tools, Bytecode Libraries, Command Line Parsers, Cache Implementations, Cloud Computing, Code Analyzers, Collections, Configuration Libraries, and Core Utilities.
- Found 71587 results**: The search results are sorted by **relevance**, with options for **popular** and **newest**.
- 1. AWS SDK For Java**: 595 usages. The artifact is `com.amazonaws:aws-java-sdk`. The description states: "The Amazon Web Services SDK for Java provides Java APIs for building software on AWS' cost-effective, scalable, and reliable infrastructure products. The AWS Java SDK allows developers to code against APIs for all of Amazon's infrastructure web services (Amazon S3, Amazon EC2, Amazon SQS, Amazon Relational Database Service, Amazon AutoScaling, etc)." The last release was on Jun 2, 2017.
- 2. AWS SDK For Java Core**: 276 usages. The artifact is `com.amazonaws:aws-java-sdk-core`. The description states: "The AWS SDK for Java - Core module holds the classes that are used by the individual service clients to interact with Amazon Web Services. Users need to depend on aws-java-sdk artifact for accessing individual client classes." The last release was on Jun 2, 2017.

Figura 40: Búsqueda de dependencias

Normalmente, el primero que aparece suele ser la dependencia correcta, ya que es la que más se ha utilizado entre todos los usuarios que han requerido utilizar la librería de AWS.

Una vez seleccionada la dependencia deseada, nos aparecerán todas las versiones que existen de la misma. Elegimos una reciente y copiamos el texto que se nos facilita.

Home » com.amazonaws » aws-java-sdk » 1.11.140

 **AWS SDK For Java » 1.11.140**

The Amazon Web Services SDK for Java provides Java APIs for building software on AWS' cost-effective, scalable, and reliable infrastructure products. The AWS Java SDK allows developers to code against APIs for all of Amazon's infrastructure web services (Amazon S3, Amazon EC2, Amazon SQS, Amazon Relational Database Service, Amazon AutoScaling, etc).

License	Apache 2.0
Categories	Cloud Computing
HomePage	https://aws.amazon.com/sdkforjava
Date	(Jun 02, 2017)
Files	Download (JAR) (3 KB)
Repositories	Central
Used By	595 artifacts

Maven | Gradle | SBT | Ivy | Grape | Leiningen | Buildr

```
<!-- https://mvnrepository.com/artifact/com.amazonaws/aws-
java-sdk -->
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-java-sdk</artifactId>
  <version>1.11.140</version>
</dependency>
```

Include comment with link to declaration

Figura 41: Dependencia de Maven repository.

En nuestro archivo POM quedaría de la siguiente manera:

```
<dependencies>
<!-- https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk -->
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-java-sdk</artifactId>
  <version>1.11.89</version>
</dependency>
<!--Aquí irían el resto de dependencias -->
</dependencies>
```

Al añadir esto a nuestro archivo POM, automáticamente se indexan todas las librerías que contiene al proyecto para poder usarlas. Como podemos observar, existen dos etiquetas

iguales a las que aparecen al crear un proyecto con Maven. El ‘GroupId’ de la dependencia significa que hereda lo que se necesite heredar de ‘com.amazonaws’, mientras que el ‘ArtifactId’ indica que dicho paquete se llama *aws-java-sdk*. Por último, la etiqueta “versión” indica la versión del mismo.

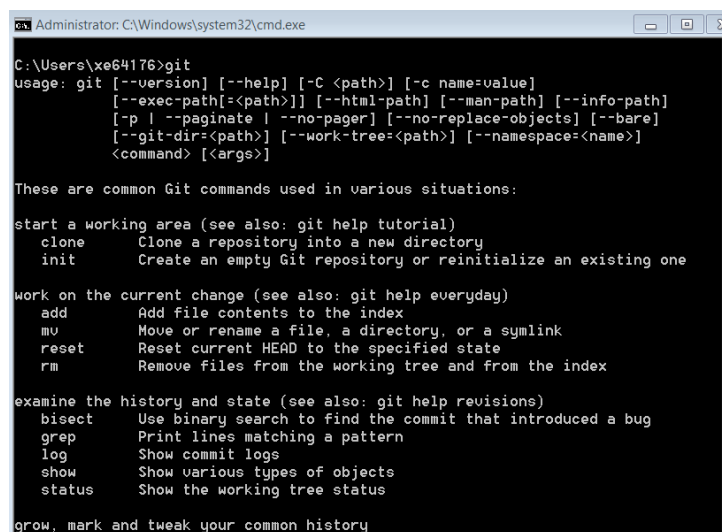
Para cualquier otra dependencia que se quiera añadir, se debe seguir el mismo proceso.

6.1.3 INTEGRACIÓN CONTINUA

Todo el desarrollo del proyecto se realizará con integración continua, lo que significa ir llevando un control de las versiones cada vez que haya cambios en el proyecto.

Para ello, la herramienta más famosa es ‘GitHub’, pero en nuestro caso, hemos utilizado ‘GitLab’, ya que es la herramienta más usada dentro de la entidad.

Lo primero que se debe hacer, es instalar Git en nuestra máquina. Curiosamente, el archivo ejecutable para la instalación está almacenado, a su vez, en un repositorio Git. Lo descargamos y lo instalamos. Para comprobar que la instalación ha sido correcta, nos vamos a la ventana de comandos (en nuestro caso de Windows) y ejecutamos el comando ‘git’. Si aparece una imagen como la que se muestra a continuación, es que la instalación ha ido correctamente, en caso contrario, se debe revisar la instalación o volver a realizarla porque algo ha ido mal.



```
Administrator: C:\Windows\system32\cmd.exe
C:\Users\xe64176>git
usage: git [--version] [--help] [-C <path>] [-c name=value]
       [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
       [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
       [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
       <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv         Move or rename a file, a directory, or a symlink
  reset      Reset current HEAD to the specified state
  rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
  bisect     Use binary search to find the commit that introduced a bug
  grep       Print lines matching a pattern
  log        Show commit logs
  show       Show various types of objects
  status     Show the working tree status

grow, mark and tweak your common history
```

Figura 42: Comprobación de instalación de Git

Una vez instalado ‘Git’ en nuestra máquina, el siguiente paso es crearnos una cuenta en GitLab, después, ya tendremos capacidad para crearnos repositorios para almacenar nuestras versiones. Para crear un repositorio nuevo, lo único que tenemos que hacer es ir a “New project” dentro de nuestra cuenta, e indicar el nombre, si queremos importar algo de otro proyecto, y, por último, indicar el nivel de visibilidad, es decir, si será privado o público.

Como ya sabemos que vamos a tener que crear mínimo dos proyectos (cliente y servidor), creamos dos repositorios. También se creará otro para almacenar versiones de esta memoria. El repositorio para almacenar las versiones del servidor se llamará “awsboard”, mientras que el repositorio para almacenar las versiones del cliente será “awsclient”. El de esta memoria será “MemoriaTFG”.

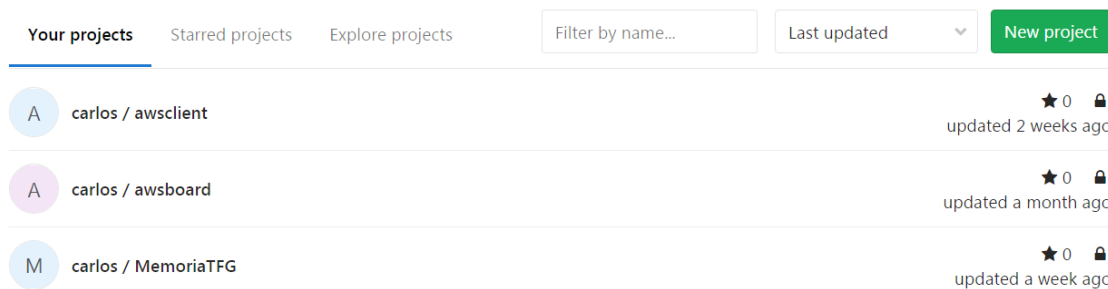


Figura 43: Repositorios creados

Para vincular nuestro proyecto con nuestro repositorio, debemos clonar nuestro repositorio en una carpeta, y meter nuestro proyecto en dicha carpeta. Para clonar un repositorio, se puede hacer de dos maneras, con ssh y clave privada, o con Https. En nuestro caso lo haremos con Https. Se hace desde la cmd, de la siguiente manera:

```
git clone https://carsanbar@gitlab.com/carsanbar/awsclient.git
```

Esa url es la que GitLab te proporciona para acceder al repositorio, siendo “carsanbar” el nombre de la cuenta, y “awsclient” el nombre del repositorio. De esta manera, se crea en la carpeta principal del usuario, una carpeta llamada igual que el repositorio que está conectada

con el mismo. Si se quiere clonar en otra carpeta solo se debe incluir la ruta en la url, o mover la carpeta donde se desee una vez creada.

A veces, por la configuración de Git, esta url da problemas de autenticación. Para solucionarlo, únicamente se debe añadir la contraseña de la cuenta de Git a continuación del nombre de la cuenta precedido por dos puntos. Ejemplo:

```
git clone https://carsanbar:password@gitlab.com/carsanbar/awsclient.git
```

Git tiene muchas funciones asociadas, pero no vamos a explicar todas, simplemente vamos a comentar las usadas aquí, que son las básicas para realizar integración continua con un proyecto.

Las 4 principales acciones que debemos tener en cuenta son las siguientes:

- Add
- Commit
- Push
- Pull

El 'add', lo que hace es preparar el contenido indicado para el siguiente 'commit'. En nuestro caso, sería el proyecto entero. El 'commit' confirma que todos los cambios realizados en la nueva versión, son correctos y deseados, pero hasta aquí, todo está en nuestra máquina local, no hemos vinculado la información con el repositorio. Esto es precisamente lo que hace el 'push', subir todos esos cambios confirmados en el 'commit' al repositorio, generando así las distintas versiones. El 'pull' es el comando correspondiente para bajar la última versión del repositorio a nuestra máquina local. Está pensado para proyectos en los que interviene mucha gente, o proyectos que se desarrollan en dos sitios físicos distintos. En este caso, por ejemplo, el proyecto se ha realizado en la empresa y en casa, por lo que ha habido versiones subidas desde distintos sitios. Para continuar con el proyecto en el último punto, si dicho último punto ha sido desarrollado en un lugar distinto al actual, se debe realizar un 'pull'.

En nuestro caso vamos a realizar versiones cada vez que se realicen cambios grandes e importantes en el proyecto, y no cada día. Si hay días en los que no se producen avances o son muy pequeños, no se realizarán versiones. En cualquier caso, en el ordenador de la entidad se encuentra el proyecto hasta el último punto desarrollado.

Hay otro punto importante que comentar. Si se quiere que haya archivos ‘ignorados’ a la hora de realizar cambios (versiones) en el repositorio, git proporciona un archivo denominado ‘gitignore’, donde se indican aquellos archivos que no se van a subir al repositorio.

Esto es muy útil, ya que hay archivos propios de los IDE de desarrollo, por ejemplo, que no es necesario subir al repositorio, ya que, si se quiere abrir dicho proyecto en otro IDE diferente, estos archivos no valen para nada. Lo esencial son los archivos con código útil dentro del proyecto.

Cómo realizar todo esto desde la Shell de Windows cada vez que se quiera notificar algún cambio es una tarea ardua, hemos instalado un interfaz de git llamado ‘TortoiseGit’ que facilita todas estas operaciones.

Ahora simplemente podemos crear una carpeta, y con el botón derecho, seleccionado TortoiseGit, y dentro tenemos todas las opciones de git. Lo primero que debemos realizar es el ‘add’. Una vez hecho, el commit y el push pueden hacerse en un solo paso si se desea. Al seleccionar dicha opción, aparecen todos los archivos del proyecto que se ha añadido que tienen cambios. Es recomendable indicar que número de versión es junto con algún comentario explicativo de los cambios que se ha realizado. A continuación, mostramos dos imágenes explicativas.

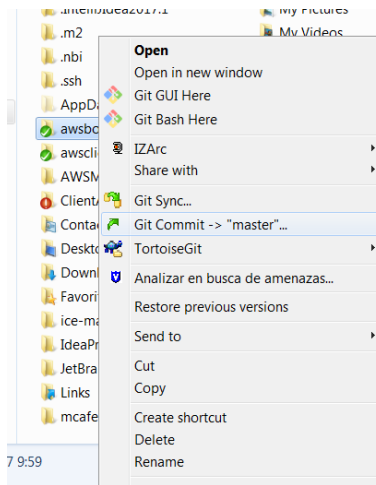


Figura 44: Menú de TortoiseGit para Git

Si en nuestra carpeta aparece un símbolo pequeño, verde y con un *tick*, significa que nuestra carpeta está actualizada en el repositorio y que no hay algún cambio que no se haya notificado. En este caso, podemos ver que es así. En caso contrario, aparecerá un símbolo rojo con una exclamación, indicando que algún cambio no se ha subido al repositorio.

En nuestro caso, como está todo actualizado, al intentar hacer un ‘commit-push’, nos va a dar un error porque no hay ningún archivo con algún cambio. Lo mostramos a continuación.

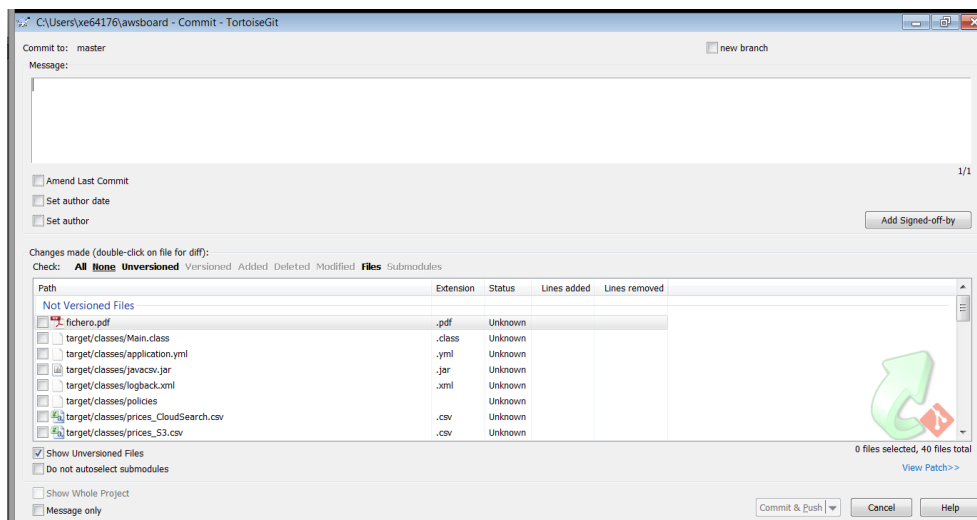


Figura 45: Commit-Push sin cambios.

A continuación, mostramos un ejemplo de cómo sería un ‘commit-push’ con cambios.

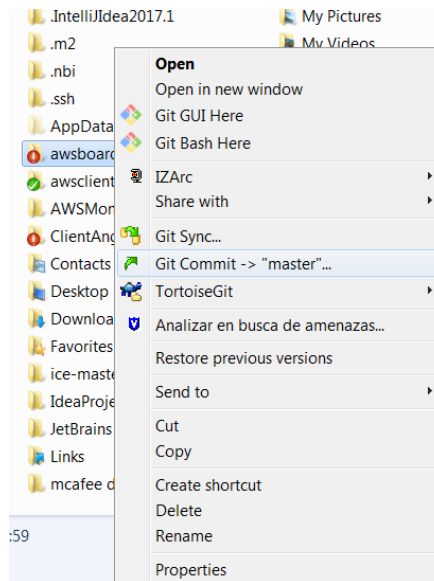


Figura 46: Carpeta con cambios sin notificar en el repositorio.

Podemos observar como en la carpeta ‘awsboard’, con el símbolo rojo, se nos indica que ha habido cambios que no se han notificado al repositorio. Si pulsamos en Git Commit -> master, nos deben aparecer los archivos con cambios.

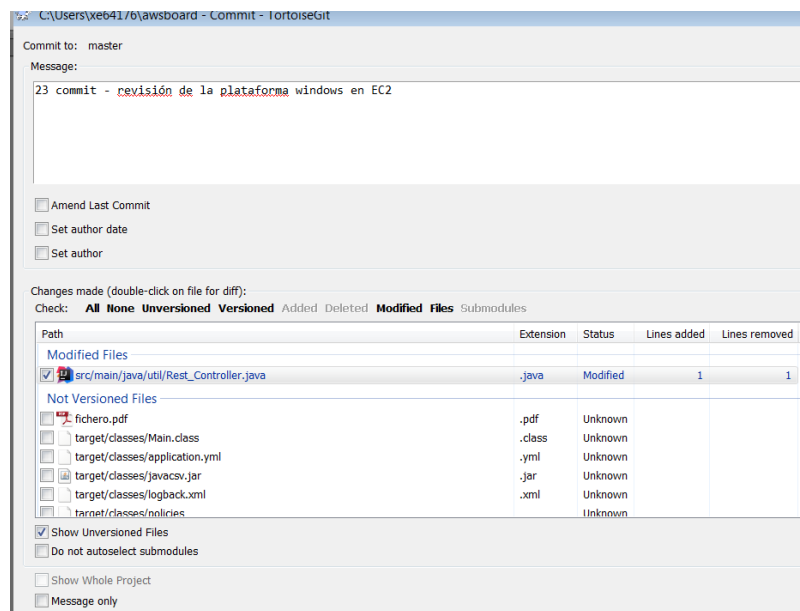


Figura 47: Ejemplo de carpeta con cambios no notificados

Como podemos observar, efectivamente ha habido cambios. En “Modified Files” se encuentran los archivos que han sido modificados y no han sido subidos al repositorio. En nuestro caso, el Rest_Controller.java. Escribimos un mensaje con el número de la versión y una explicación y realizamos el *push*. De esta manera subimos versiones al repositorio.

A partir de ahora, comenzamos con la explicación del código.

6.1.4 CREDENCIALES DE AWS

Para el desarrollo del proyecto, primeramente, se va a realizar un menú por consola con opciones a escoger para desarrollar la funcionalidad correspondiente, mostrando la información obtenida de AWS por consola. Evidentemente, esta manera de trabajar es para comprobar que lo que se está haciendo funciona, pero en el futuro, dicha información tendrá que enviarse al cliente Angular 2 vía API REST para mostrarse en la web.

Las opciones que se manejan son los servicios (EC2, S3 y CloudSearch), usuarios, *billing*, base de datos y creación de PDF (para los permisos).

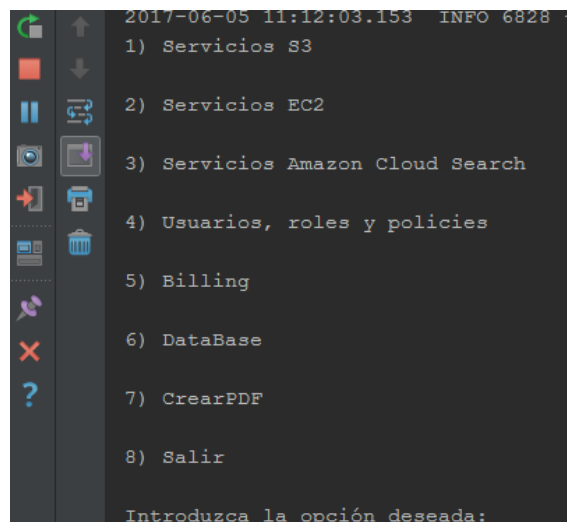


Figura 48: Menú de la aplicación por consola

Para obtener cualquier tipo de información de AWS, no vale únicamente con realizar peticiones a través de su API. Si lo hacemos así, nos saltarán errores de autenticación.

Evidentemente, si no nos autentificamos de alguna manera, AWS no sabe quién está realizando peticiones y, por tanto, devuelve errores.

AWS tiene diferentes maneras de poder implantar un *token* de seguridad para demostrar quién eres en una máquina local.

Una de las maneras es establecer en nuestro perfil local las credenciales, creando una carpeta en el raíz llamada “.aws”, y dentro de ella, crear un archivo de texto denominado “credentials”. Dentro de él, debe estar escrito lo siguiente:

```
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key
```

Evidentemente, cada usuario de AWS tiene un identificador y una clave únicos. Para acceder a ellos, debemos ir a nuestro perfil IAM en AWS, acceder a los usuarios y seleccionar el nuestro. En el apartado “*security credentials*” se pueden crear como máximo dos id para nuestro usuario. La clave se muestra en el momento de la creación y después es imposible acceder a ella, por lo que, si se pierde, deberíamos borrar uno de los identificadores y crear uno nuevo. Para crear uno nuevo, simplemente pulsar en “*Create access key*”.

En este apartado, al hacer estas acciones, estamos generando un token de seguridad para nuestro usuario. Lo único que tenemos que hacer es copiar nuestro id y nuestra clave en nuestro archivo de credenciales. También se puede meter la región dónde está creada nuestra cuenta (cada cuenta tiene asociada una región física), pero más adelante explicaremos la razón de porque no es beneficioso crearse una región por defecto.

Otra de las maneras para generar un token de seguridad, es seguir los mismos pasos que en el caso anterior, pero en vez de crear una carpeta con un archivo de credenciales, debemos establecer el id y la contraseña como variables de entorno en nuestro sistema. En nuestro caso, esta es la manera que hemos elegido para trabajar. Lo mostramos a continuación:

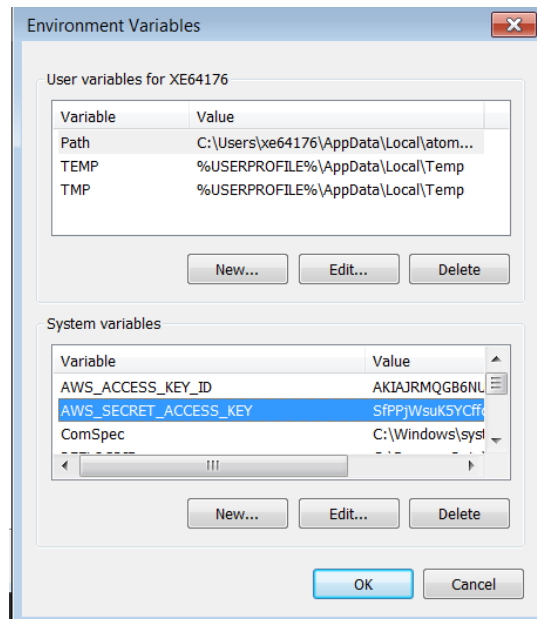


Figura 49: Variables de entorno para AWS

La foto está cortada para no mostrar el id y la contraseña del perfil de manera completa.

Cada vez que se intenta hacer una petición a AWS, este lanza una cadena de comprobación de credenciales. Primero mira las variables de entorno de la máquina de desarrollo, por esta razón escogimos esta manera de trabajar. Si no hay variables de entorno definidas, busca si java tiene algunas propiedades definidas para AWS. En caso negativo, busca en la carpeta “.aws”. Si todas estas opciones son fallidas, AWS no permite lanzar peticiones.

Cómo nosotros hemos elegido usar las variables de entorno, esta comprobación será lo más rápida posible. Cuando AWS comprueba quiénes somos, ya sabe que pertenezco al BBVA y que, por tanto, quiero acceder a dicha información. Por tanto, en este punto, ya tenemos todo en orden para empezar a hacer peticiones a los servicios de AWS y obtener la información que deseamos.

6.1.5 SERVICIOS

AWS proporciona una cantidad de servicios realmente grande, que permite a los usuarios realizar proyectos prácticamente de cualquier tipo. En las imágenes que se muestran a continuación podemos verlo.

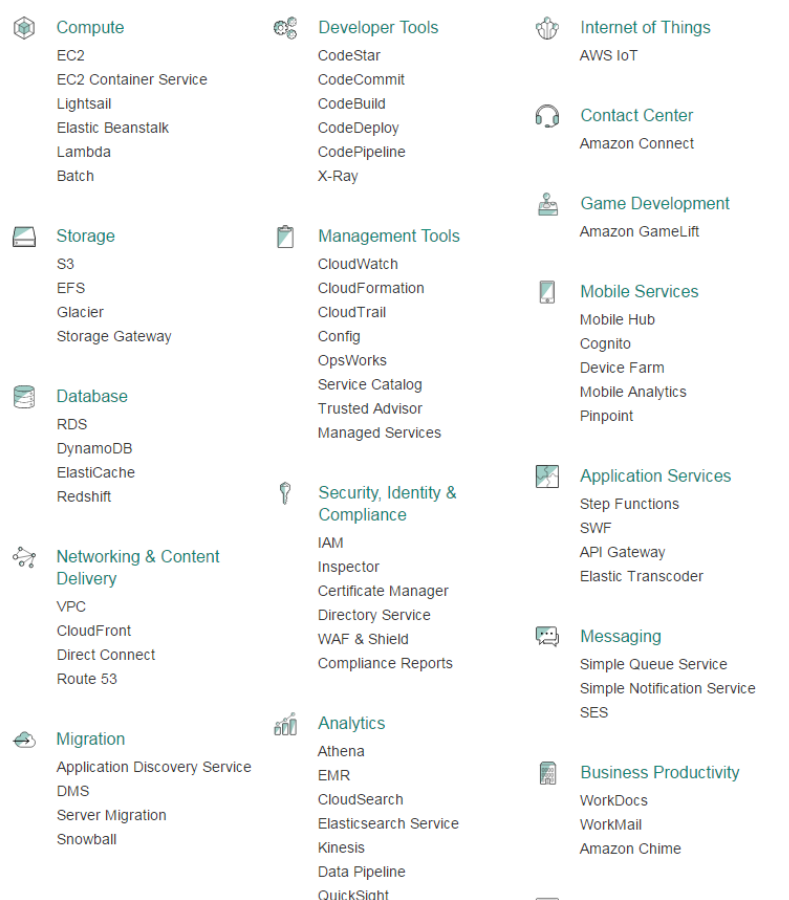


Figura 50: Servicios de AWS. Parte 1

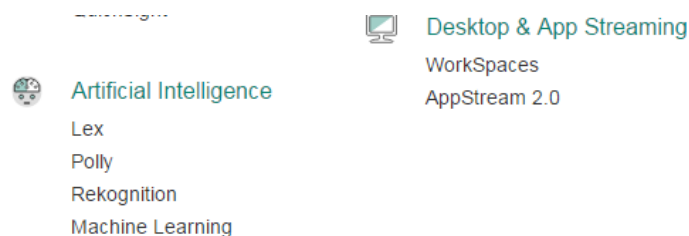


Figura 51: Servicios de AWS. Parte 2

Nosotros explicaremos con detalle como accedemos a la información de los servicios que tenemos dados de alta, que son, EC2, S3 y CloudSearch.

Para acceder a cualquier tipo de servicio, debe iniciarse un proceso común, que es la creación de un cliente para poder realizar peticiones. Hay dos maneras básicas para hacerlo. La primera, consiste en crear un cliente con las configuraciones por defecto. Este carga los datos introducidos en las variables de entorno o en la carpeta “.aws”, y lo crea. Dichos datos deben ser como mínimo, el id y la *password* del perfil, junto con una localización. Dicha localización suele ser dónde la empresa tenga levantados la mayoría de sus servicios. Si no están definidos estos datos, no se creará el cliente, y, por tanto, se producirá un error. El código de ejemplo para un cliente S3 sería de la siguiente manera:

```
AmazonS3 s3Client = AmazonS3ClientBuilder.defaultClient()
```

La segunda manera, consiste en crearse un cliente definiendo manualmente alguno de los parámetros, generalmente la región que se desea tener.

```
AmazonEC2 ec2Client = AmazonEC2ClientBuilder.standard()  
    .withRegion(Regions.EU_WEST_1)  
    .build();
```

En nuestro caso, definiremos todos los clientes como en el ejemplo anterior, cogiendo el id y la contraseña del perfil de AWS por defecto e indicando la región que deseamos de manera manual (EU_WEST_1 es Irlanda). Explicaremos el motivo de esta decisión en el siguiente apartado.

Antes de continuar, mostramos una tabla con todas las regiones físicas que ofrece AWS. Cada una de ellas se corresponde con un código para identificarla. Alguna de ellas, a su vez, puede tener varias regiones interiores, como Irlanda, que, dentro del país, tiene 3 diferentes localizaciones.

<i>Nombre de la región</i>	<i>Código de región de AWS</i>
Asia Pacífico (Seúl)	ap-northeast-2
Asia Pacífico (Singapur)	ap-southeast-1
Asia Pacífico (Sídney)	ap-southeast-2
Asia Pacífico (Tokio)	ap-northeast-1
UE (Fráncfort)	eu-central-1
UE (Irlanda)	eu-west-1
América del Sur (São Paulo)	sa-east-1
US East (N. Virginia)	us-east-1
EE.UU. Oeste (Norte de California)	us-west-1
EE.UU. Oeste (Oregón)	us-west-2
China (Beijing). Norte	cn-north-1
EE.UU (Government)	us-gov-west-1

Tabla 2: Regiones de AWS

Hay que comentar que las dos últimas son un tanto especiales. Desde nuestra región física no se nos da acceso a ninguna de las dos. No se nos deja acceder a China. Tampoco a la región de EE. UU únicamente destinada para el gobierno, por lo que, en nuestro proyecto, obviaremos estas dos regiones.

6.1.5.1 Servicios S3

Amazon *Simple Storage Service* (Amazon S3) es un almacenamiento de objetos con una sencilla interfaz de servicios web para almacenar y recuperar la cantidad de datos que desee desde cualquier ubicación de la web. Se ha diseñado para ofrecer una durabilidad de 99,999999999% y escalar más allá de billones de objetos en todo el mundo.

Los clientes utilizan S3 como almacenamiento principal para aplicaciones nativas en la nube; como repositorio masivo, o como "conjunto de datos" para el análisis; como destino de *backups* y recuperación y para la recuperación de desastres; y con la informática sin servidores.

Resulta sencillo introducir o extraer grandes volúmenes de datos de Amazon S3 con opciones de migración de datos de la nube de Amazon. Una vez que los datos están almacenados en S3, se pueden ordenar por niveles automáticamente en clases de almacenamiento en la nube a largo plazo y menor costo como S3 Estándar [10].

Dentro de todos los servicios, es bastante crítico obtener todas las instancias levantadas en las regiones dónde se encuentren, ya que la política de la empresa así lo requiere. Si no creamos un único cliente, solo podremos acceder a las instancias en dicha región. Para el banco, es necesario que salte una alarma si hay alguna instancia levantada en una región incorrecta. En nuestro caso, la región adecuada es Irlanda, por lo que, si hay alguna instancia levantada en otra localización, se debe comunicar.

Para ello, debemos crearnos clientes con todas las regiones e ir comprobando cada región para ver si hay algo contratado allí. AWS proporciona un enumerado con todas las regiones (Regions), por lo que crearnos un cliente en cada región es una tarea sencilla, únicamente tendremos que hacer un bucle 'for' pasando por cada región. Como comentamos, deberemos omitir China y la parte de Estados Unidos orientada al gobierno.

Se haría de la siguiente manera:

```
Regions[] arrayRegions = Regions.values();

for( Regions r: arrayRegions ) {
    if (r != Regions.GovCloud && r!= Regions.CN_NORTH_1) {
        AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
            .withRegion(r)
            .build();
        // Lógica a desarrollar
    }
}
```

Las instancias de tipo S3, comúnmente se denominan “*buckets*”, por tanto, así las buscaremos dentro de la API de AWS para realizar cualquier cosa con ellas. La clase ‘AmazonS3’ proporciona un método que devuelve un ‘array’ con todos los *buckets* de la cuenta en cuestión.

Para obtener mucha de la información asociada a una instancia S3, es suficiente con utilizar la clase “*Bucket*”.

Para obtener otro tipo de información adicional como el tamaño, o los archivos que contiene, debe utilizarse la clase la clase “*ObjectListing*”. Para poder utilizar esta clase, es necesario crearse un objeto utilizando la propiedad de java ‘*with*’. Dicho método facilita la creación de un objeto teniendo en cuenta una característica concreta, en nuestro caso, el nombre del *bucket*. La creación, sería de la siguiente manera:

```
ObjectListing objectListing = s3Client.listObjects(new ListObjectsRequest()
    .withBucketName(b.getName())
);
```

Como estamos dentro de un bucle ‘for’ recorriendo cada *bucket* en particular, la ‘b’ que se muestra en el código representa un objeto de la clase “*Bucket*”, y el método “*getName()*” devuelve el nombre, por tanto, lo que estamos haciendo es crear un objeto de tipo “*ObjectListing*” del *bucket* correspondiente.

La información básica que se va a recoger al inicio del proyecto recogerá una serie de parámetros que será imprescindible mostrar en la web, como son el nombre, la localización,

el tamaño, el propietario y la fecha de creación. También se recogerán las carpetas interiores y los archivos de cada *bucket* para una mejora futura, pero no se mostrarán en la aplicación web.

Para la información adicional que nombramos anteriormente, se necesita de una clase adicional que se obtiene a partir de la clase “*ObjectListing*” que mencionamos unas líneas atrás. Hablamos de la clase ‘*S3ObjectSummary*’, que nos permitirá acceder a las carpetas y archivos, así como al tamaño global del *bucket*. Es importante comentar que el tamaño no es una característica que se pueda obtener directamente del *bucket*, por lo que aquí entra la utilidad de obtener los archivos almacenados en la instancia.

De los archivos, sí que se puede obtener su tamaño, por lo que lo único que se debe hacer es sumar el tamaño de todos ellos para obtener el tamaño final del *bucket*. El tamaño que devuelve la API esta medido en *bytes*, por lo que habrá que convertirlo a *gigabytes* para que sea un tamaño intuitivo.

Cómo hay demasiados decimales, y en una web no queda muy profesional, truncaremos el resultado a 4 decimales como máximo. Se hace del siguiente modo:

```
DecimalFormat df = new DecimalFormat("#.####"); // Formato para 4 decimales
totalsize = totalsize / 1073741824; // Para ponerlo en GB

df.setRoundingMode(RoundingMode.CEILING);
Number n = totalsize;
Double finalTotalSize = n.doubleValue(); /*finalTotalSize es el totalsize
truncado a 4 decimales. finalTotalSize es un string*/

String finalSize = df.format(finalTotalSize);
System.out.println(finalSize);
```

Para la muestra de datos por consola, simplemente nos hemos creado una clase llamada ‘*S3Instance*’ cuyos parámetros son aquellos necesarios para definir una instancia, como el nombre, la región, el tamaño, el propietario, la fecha de creación, las subcarpetas y los archivos. Aplicando el método *toString()*, mostramos todo lo obtenido y podemos ver que todo es correcto.

Todas las pruebas se hacen por consola ya que es mucho más rápido y fácil hacer cambios así que directamente desde la web, por eso al inicio del proyecto todo se desarrolla de esta manera.

A continuación, mostramos un ejemplo. Pasamos por todas las regiones y cuando llegamos a las regiones donde tenemos instancias levantadas se muestra lo siguiente. De nuevo, ocultamos los datos confidenciales propios a la entidad.

```
region bucket: EU-WEST-1
*
Bucket location = EU-WEST-1
Owner -->
Data created --> 2017-06-14 12:22:36
 ( Last modified : Wed Jun 14 16:56:59 CEST 2017 | Size = 33513 bytes | Storage class --> STANDARD )
S3File(name=' ', size=33513, storageClass='STANDARD', lastModified=Wed Jun 14 16:56:59 CEST 2017)
0,0001
Total size: 3.121141344308853E-5 bytes --> 3.121141344308853E-5 Gb 0,0001 finalsize

S3InstanceTable(name=' ', location='EU-WEST-1', size=0,0001, folder=[], owner=' ', creationDate=2017-06-14 12:22:36,
region bucket: EU-WEST-1
*
Bucket location = EU-WEST-1
Owner -->
Data created --> 2017-03-08 08:46:09
Folder: documents
Files:
- .pptx ( Last modified : Fri Mar 31 13:32:31 CEST 2017 | Size = 264235 bytes | Storage class --> STANDARD )
S3File(name=' .pptx', size=264235, storageClass='STANDARD', lastModified=Fri Mar 31 13:32:31 CEST 2017)
- .pdf ( Last modified : Mon Mar 27 11:55:33 CEST 2017 | Size = 611393 bytes | Storage class --> STANDARD )
S3File(name=' .pdf', size=611393, storageClass='STANDARD', lastModified=Mon Mar 27 11:55:33 CEST 2017)
- PSM_ParteActividadmes 01082016 (BBVA).xls ( Last modified : Thu May 04 11:19:33 CEST 2017 | Size = 150016 bytes | Storage class --> STANDARD )
```

Figura 52: Instancias S3 por consola

6.1.5.2 Servicios EC2

Amazon *Elastic Compute Cloud* (Amazon EC2) es un servicio web que proporciona capacidad informática en la nube segura y de tamaño modificable. Está diseñado para facilitar a los desarrolladores el uso de la informática en la nube a escala de la Web.

La sencilla interfaz de servicios web de Amazon EC2 permite obtener y configurar la capacidad con una fricción mínima. Proporciona un control completo sobre los recursos informáticos y puede ejecutarse en el entorno informático acreditado de Amazon. Amazon EC2 reduce el tiempo necesario para obtener y arrancar nuevas instancias de servidor en cuestión de minutos, lo que permite escalar rápidamente la capacidad, ya sea aumentándola o reduciéndola, según cambien sus necesidades. Amazon EC2 cambia el modelo económico de la informática, ya que solo tendrá que pagar por la capacidad que realmente utilice.

Amazon EC2 les brinda a los desarrolladores las herramientas necesarias para crear aplicaciones resistentes a errores y para aislarlas de los casos de error comunes [11].

Para dichos servicios, se seguirá una lógica parecida a la aplicada en el caso anterior. A alto nivel, podemos decir que este tipo de instancias ofrecen potencia de cálculo, por lo que habrá una serie de características distintas respecto a las instancias S3. Nuestro deber será sacar todas aquellas que sean necesarias para tener un control óptimo sobre ellas. El primer paso, igual que en cualquier servicio, será crear un cliente AmazonEC2 para poder realizar las peticiones oportunas. Debemos comprobar cada región para comprobar que no hay ninguna instancia levantada en una localización no deseada. Debemos volver a ignorar China y a la parte de EE. UU orientada únicamente al gobierno.

```
for (Regions r : arrayRegions) {
    String region = r.name();
    System.out.println("Region " + region);

    if (r != Regions.GovCloud && r != Regions.CN_NORTH_1) {
        AmazonEC2 ec2Client = AmazonEC2ClientBuilder.standard()
            .withRegion(r)
            .build();
        //Lógica a desarrollar
    }
}
```

Las características propias de estos servicios y que debemos obtener son las siguientes: el nombre, el id, la fecha de creación, la región, el estado, el tipo, el propietario, el dns que tiene y la plataforma sobre la que corre.

Algunos de ellos son triviales y fáciles de entender, pero algunos merecen una explicación sencilla. El estado de una instancia de tipo EC2, es la característica que indica si la máquina está corriendo o está parada. La potencia de cómputo puede estar haciendo que una aplicación se esté ejecutando continuamente porque así se desea, o por el contrario, se puede querer que se ejecute solamente en un momento dado. Los servicios *cloud* se pagan según el consumo, por lo que esta característica ayuda a entender bastante bien lo que significa. Si la instancia está parada, no se paga por ella.

El tipo particular de las instancias EC2 pueden ser varios. Al ser potencia de cómputo, es evidente que puede haber máquinas mucho más potentes y preparadas que otras. La contratación de una máquina más potente u otra depende del peso de la aplicación que se quiera correr dentro de la máquina. Si una aplicación es muy pesada y tiene mucha lógica, deberá estar alojada en una máquina que ofrezca las garantías necesarias para que funcione con fluidez. Por otro lado, si la aplicación en cuestión no es tan grande, bastará con alquilar una máquina menos potente.

Hay gran variedad de tipos de instancias para dar al cliente una capacidad óptima de potencia. A continuación, las describimos.

Dentro del **uso general**, hay distintos tipos.

- T2

Las instancias T2 son instancias de desempeño con ráfagas que proporcionan un nivel base de desempeño de la CPU con la posibilidad de alcanzar ráfagas por encima del nivel básico. El desempeño de referencia y la capacidad de alcanzar ráfagas se rigen por los créditos de la CPU. Cada instancia T2 recibe créditos de CPU continuamente a un nivel establecido dependiendo del tamaño de la instancia. Las instancias T2 acumulan créditos de la CPU cuando están inactivas y los utilizan cuando están activas. Las instancias T2 son una buena opción para cargas de trabajo que no usan la CPU por completo, a menudo o de manera constante, pero que de vez en cuando tienen que alcanzar ráfagas (por ejemplo, servidores web, entornos para desarrolladores y bases de datos).

- M3 y M4

Las instancias M3 y M4 son la última generación de instancias de uso general. Esta familia proporciona un equilibrio de recursos informáticos, de memoria y red, por lo que constituye una buena opción para muchas aplicaciones.

Dentro de la **optimización informática**:

- C3 y C4

Las instancias C3 y C4 son la última generación de instancias con optimización informática, que disponen de los procesadores de mejor desempeño y ofrecen la mejor relación precio/desempeño informático de EC2.

Dentro de las **optimizadas para memoria**:

- X1

Las instancias X1 están optimizadas para aplicaciones en la memoria a larga escala de clase empresarial y ofrecen el costo más bajo por GiB de RAM entre los tipos de instancias de Amazon EC2.

- R4

Las instancias R4 están optimizadas para aplicaciones de uso intensivo de memoria y ofrecen un precio mejor por GiB de RAM que las instancias R3.

- R3

Las instancias R3 están optimizadas para aplicaciones de uso intensivo de memoria y ofrecen un costo más bajo por GiB de RAM.

Instancias de **informática acelerada**:

- P2

Las instancias P2 están destinadas a aplicaciones de informática GPU de uso general.

- G2

Las instancias G2 están optimizadas para aplicaciones con uso intensivo de gráficos.

- F1

Las instancias F1 ofrecen aceleración de hardware personalizable con matrices de puertas programables en campo (FPGA).

Instancias optimizadas para almacenamiento:

- I3 – Instancias de E/S de alto desempeño

Esta familia incluye instancias de gran capacidad de almacenamiento que ofrecen almacenamiento de instancias respaldado por SSD en memoria no volátil expres (NVMe) optimizado para baja latencia, desempeño de E/S aleatoria muy alta, desempeño de lectura secuencial alto y ofrecen IOPS altas a bajo costo.

- D2 – instancias de alta densidad de almacenamiento

Las instancias D2 ofrecen hasta 48 TB de almacenamiento local basado en HDD, proporcionan un alto desempeño de disco y están disponibles al precio más bajo por desempeño de disco en Amazon EC2 [12].

Evidentemente, lo que hace que unas instancias sean más potentes que otras, son el número de CPU's virtuales que se posean, la memoria, el almacenamiento, el procesador, la frecuencia de trabajo, etc...

Todos estos tipos vienen acompañados de un sub-tipo que indica la capacidad de la instancia dentro del tipo principal, que puede ser cualquiera de los que acabamos de explicar. Dichos sub-tipos, son los siguientes: nano, micro, small, medium, large, xlarge, 2xlarge, 4xlarge, 8xlarge, 10xlarge, y 16xlarge. Nano es el más básico en cuanto a las características mencionadas en el párrafo anterior, y según se va avanzando, las características van siendo mejores, y, por lo tanto, más caras.

Por otro lado, a la hora de levantar este tipo de instancias, se puede indicar un propietario, que suele ser el departamento dentro de la entidad al que va a pertenecer. Si no se introduce nada, este campo aparecerá en blanco.

Otro punto bastante importante a explicar, son las plataformas en las que corren. Dichas plataformas no son más que los sistemas operativos que tienen. El método que devuelve un 'string' con dicha plataforma no funciona de manera óptima, ya que sólo devuelve información en caso de que la plataforma se haya levantado con Windows. En caso contrario,

devolverá un objeto ‘null’, lo cual no es muy útil teniendo en cuenta que muchas instancias se levantan en Linux.

Dentro de los sistemas operativos más famosos que se pueden encontrar, están el propio Linux que proporciona Amazon, Windows o Red Hat, juntos con todas sus correspondientes versiones. Además, AWS también permite levantar instancias con una plataforma propia privada a las que AWS no tiene acceso, a las que, por defecto, AWS denomina ‘Other Linux’.

Para solucionar este pequeño problema, deberemos acceder a otro campo de las instancias que se denomina ‘Image’. Este campo es una pequeña descripción de la máquina en cuestión, y, en cuyo nombre, se encuentra también el nombre de la plataforma sobre la que corre la máquina, por lo que desde java es bastante sencillo acceder a la plataforma.

Una vez explicados todos los conceptos importantes de este tipo de servicios, retomamos la explicación del desarrollo. La clase AmazonEC2 (clase del cliente creado) posee un método llamado ‘*describeInstances()*’ que devuelve un objeto de la clase “DescribeInstancesResult”. Todas las clases necesarias para obtener información de las instancias EC2 son las creadas por AWS para realizar el proceso. Creemos que podría ser más sencillo, pero no podemos modificar la API a nuestro gusto, por lo que tendremos que trabajar con la herramienta proporcionada. Del objeto de esta última clase obtenemos un ‘array’ de objetos de clase “Reservation”, y finalmente de esta clase podemos obtener un ‘array’ de la clase “Instance” que ya son las instancias que tenemos reservadas. El código sería el siguiente:

```
DescribeInstancesResult result = ec2Client.describeInstances();
List<Reservation> reservations = result.getReservations();
    Set<Instance> instances = new HashSet<Instance>();
for (Reservation reservation : reservations) {
    instances.addAll(reservation.getInstances());
}

for (Instance instance : instances) {
    //Lógica a desarrollar
}
```

Dentro de la lógica a desarrollar, hay que comentar que el parámetro “nombre” no se obtiene de igual modo que el “nombre” de otro tipo de instancias, si no que en este, es una etiqueta. Realmente sí que se puede obtener un nombre igual que en otras instancias, pero suele ser un nombre poco intuitivo, y para ello se crearon las etiquetas, donde se puede indicar un nombre mucho descriptivo, por lo que accederemos a este campo. Las etiquetas se obtienen con el método “*getKey()*”. El resto de parámetros se pueden obtener con métodos de la clase “Instance” muy intuitivos, como “*getAvailabilityZone()*”, “*getState()*” o “*getType()*”.

Sin embargo, como comentamos anteriormente, la obtención de la plataforma no es algo tan trivial. Debemos recurrir a dos clases diferentes, llamada “DescribeImagesRequest” y “DescribeImagesResult”.

La primera de ellas sirve para crear un mini-cliente con una instancia en particular, indicándole el id correspondiente.

```
DescribeImagesRequest imagerequest = new DescribeImagesRequest()  
    .withImageIds(instance.getImageId()); //IdImage de cada instancia
```

Después, obtenemos las “images” de la siguiente manera:

```
DescribeImagesResult imagesResult = ec2Client.describeImages(imagerequest);  
List<Image> images = imagesResult.getImages();
```

Posteriormente, únicamente hay que obtener la descripción y estudiar dicha cadena mediante un ‘if-else’ para ver qué tipo de plataforma es la establecida.

También se estudian todas las regiones posibles. Mostramos a continuación, un ejemplo del resultado por consola.


```

Name: [redacted] | Owner: [redacted] | Instance ID = [redacted] | Availability zone: eu-west-1a | Private DNS = [redacted] | Instance state: stopped | Instance Type: m4.xlarge | Ami: [redacted] | Image
[redacted]
AWS-VMImport service: Linux - Ubuntu 14.04.5 LTS \n \1 - 3.13.0-35-generic
Platform : Other Linux

Name: [redacted] | Owner: [redacted] | Instance ID = [redacted] | Availability zone: eu-west-1a | Private DNS = [redacted] | Instance state: running | Instance Type: t2.medium | Ami: [redacted] | Image
Canonical, Ubuntu, 16.04 LTS, amd64 xenial image build on 2017-04-14
Platform : Other Linux

Name: [redacted] | Owner: [redacted] | Instance ID = [redacted] | Availability zone: eu-west-1b | Private DNS = [redacted] | Instance state: running | Instance Type: t2.micro | Ami: [redacted] | Image
Amazon Linux AMI 2016.09.1.20161221 x86_64 HVM GP2
Platform : Amazon Linux

Name: [redacted] | Owner: [redacted] | Instance ID = [redacted] | Availability zone: eu-west-1b | Private DNS = [redacted] | Instance state: stopped | Instance Type: m4.2xlarge | Ami: [redacted] | Image
AWS-VMImport service: Linux - CentOS release 6.7 (Final) - 2.6.32-573.el6.x86_64
Platform : Other Linux

Name: [redacted] | Owner: [redacted] | Instance ID = [redacted] | Availability zone: eu-west-1b | Private DNS = [redacted] | Instance state: running | Instance Type: t2.micro | Ami: [redacted] | Image
Chatbots_Platform_v.0
Platform : Other Linux

```

Figura 53: Instancias EC2 por consola.

La información se muestra de manera correcta, aunque no podamos visualizarla por confidencialidad.

6.1.5.3 Servicios CloudSearch

Amazon CloudSearch es un servicio administrado en la *cloud* de AWS que facilita la configuración, la administración y el escalado rentables de una solución de búsqueda para su sitio web o aplicación.

Amazon CloudSearch soporta 34 idiomas y características de búsqueda populares, como resaltar, autocompletar y la búsqueda geoespacial [6].

Básicamente es el servicio de búsqueda de archivos en la cuenta AWS de la entidad. Permite a través de un formulario de búsqueda, obtener todos los documentos que contengan la palabra introducida en algún parámetro del archivo. Trabajaremos de la misma manera que en los casos anteriores, revisando todas las regiones y creando un cliente para cada una de ellas.

```

for (Regions r : arrayRegions) {
    if (r != Regions.GovCloud && r != Regions.CN_NORTH_1) {
        AmazonCloudSearch ACSCClient = AmazonCloudSearchClientBuilder.standard()
            .withRegion(r)
            .build();
        //lógica a desarrollar
    }
}

```

Cada instancia de CloudSearch, puede denominarse dominio. Puede ser un símil a *bucket* con las instancias de tipo S3. Lo que debemos hacer, es obtener una lista de todos los dominios creados de la siguiente manera:

```
DescribeDomainsResult res = ACSClient.describeDomains();  
List<DomainStatus> stat = res.getDomainStatusList();
```

A partir de la clase “DomainStatus” podemos obtener gran parte de la información que deseamos, como la región, el nombre del dominio o el tipo. El tipo se entiende de la misma manera que se entienden los tipos en las instancias EC2.

Para obtener la región, no podemos hacerlo de manera trivial. No existe un método que te la devuelva. Sin embargo, todos estos dominios poseen una cadena explicativa denominada ‘ARN’, dónde de forma continuada se detallan ciertas características del dominio en cuestión.

Para sacar la localización, simplemente tenemos que analizar dicha cadena y comprobar si tiene el *string* perteneciente a la región deseada, que es Irlanda, cuya identificación en AWS es “EU_WEST_1”. Con el método ‘contains’ es bastante sencillo.

Otros campos de interés son el número de particiones que tiene el dominio, así como los campos de búsqueda que posee. Para ello, debemos crearnos un sub-cliente de igual manera que hicimos en EC2. Las clases a utilizar en este caso son “DescribeIndexFieldsRequest” y “DescribeIndexFieldsResult”.

```
DescribeIndexFieldsRequest indexFieldsRequest = new DescribeIndexFieldsRequest()  
    .withDomainName(d.getDomainName());  
DescribeIndexFieldsResult indexFieldsResult =  
ACSClient.describeIndexFields(indexFieldsRequest);  
List<IndexFieldStatus> indexfiledstatus = indexFieldsResult.getIndexFields();
```

Así, nos creamos un sub-cliente con el nombre del dominio estudiado, y posteriormente obtenemos los campos de búsqueda que este utiliza para indexar información y buscar

archivos. A partir del ‘array’ del tipo “IndexFieldStatus” podemos hacerlo, con métodos como `getOptions().getIndexFieldName()` o `getOptions().getIndexFieldType()`.

Los parámetros de búsqueda pueden modificarse desde la página oficial de AWS accediendo al dominio deseado. Hay infinidad de parámetros, pero no suele ser óptimo seleccionarlos todos, ya que muchas veces se suele querer que una búsqueda sea concreta.

A continuación, mostramos una imagen con la información obtenida por pantalla. Podemos ver que obtenemos toda la información deseada

```
Type: search.ml.small ARN: arn:aws:cloudsearch:eu-west-1:[:redacted]:domain/[:redacted] Partitioncount: 1
author: text
company: text
content: text
content_encoding: text
content_type: text
created: text
creation_date: date
creator: text
filename: text
keyword: text
last_modified: date
owner: text
producer: text
s3_path: text
source_modified: text
subject: text
title: text
xmptpg_npages: int
```

Figura 54: Instancias CloudSearch por consola.

Una vez explicados todos los servicios estudiados en este proyecto, el siguiente paso es explicar la gestión de alarmas ante funcionalidad no deseada.

6.1.6 GESTIÓN DE ALARMAS ANTE SITUACIONES NO DESEADAS

La gestión de alarmas es una parte bastante importante dentro de este proyecto. Tendrán relación con todos los servicios aquí presentes.

Los problemas que pueden aparecer y que deben ser notificados, son los siguientes:

- Instancia de cualquier tipo en región no deseada
- Instancia EC2 con una plataforma Red Hat.

Las razones de estas dos normas son muy sencillas. Por localización física de la entidad y por política de la misma, todas las instancias deben estar, al menos a día de hoy, levantadas en Irlanda. Si se detecta al monitorizar las máquinas alquiladas que hay alguna fuera de dicho país, debe notificarse al responsable.

Por otro lado, particularmente en las instancias de tipo EC2, debe mirarse con detalle la plataforma sobre la que está corriendo la instancia. Este tipo de servicios suelen ser más caros que el resto y en función de la plataforma, el precio puede subir más, o no. Dentro de la entidad el coste se asume si la plataforma es Linux o Windows, pero si es ‘Red Hat’ debe notificarse, ya que se debe comprobar si realmente se necesita una máquina con este sistema operativo, o no. Una vez se notifique al responsable la existencia de una instancia con esta plataforma, este debe comprobar si es necesario seguir manteniendo una máquina de este tipo, o si, por el contrario, se ha levantado por error con dicha plataforma y la instancia debe ser eliminada.

La manera de notificar estas situaciones será a través de correo electrónico. Para ello, nos hemos creado una cuenta de correo llamada “awsbbvacib”, en referencia al departamento de desarrollo del banco y al tema estudiado en cuestión, ‘AWS’.

Para poder mandar un correo electrónico desde Java, hay que realizar varios pasos. El primero de ellos, es incorporar la librería ‘javax.mail’ al proyecto. Para ello, nos vamos al repositorio de Maven y la buscamos, cogemos el código que nos proporciona, y lo añadimos a nuestro archivo POM de configuración. El código es el siguiente:

```
<!-- https://mvnrepository.com/artifact/javax.mail/mail -->
<dependency>
  <groupId> javax.mail </groupId>
  <artifactId> mail </artifactId>
  <version> 1.4.7 </version>
</dependency>
```

Una vez añadida la librería al Proyecto, debemos entender el protocolo utilizado a la hora de mandar un correo. Dicho protocolo es SMTP (Simple Mail Transfer Protocol). El Simple Mail Transfer Protocol (SMTP) o “protocolo para transferencia simple de correo”, es un

protocolo de red utilizado para el intercambio de mensajes de correo electrónico entre computadoras u otros dispositivos (PDA, teléfonos móviles, impresoras, etc...) [13].

Como la cuenta de correo que hemos utilizado es de tipo Gmail, es decir, de Google, vamos a utilizar el smtp que proporciona.

Para organizar la lógica en el envío de e-mails en el proyecto, nos hemos creado una clase Java con toda la lógica para que únicamente llamemos a un método desde nuestro 'Main', indicando a quien va dirigido el correo, el asunto del mismo, y el contenido.

Dicha clase se llama "EmailSenderService", y básicamente, coge las propiedades del sistema, y modifica una serie de parámetros para que todo funcione. Mostramos el código a continuación:

```
Properties props = System.getProperties();  
String host = "smtp.gmail.com";  
props.put("mail.smtp.starttls.enable", "true");  
props.put("mail.smtp.host", host);  
props.put("mail.smtp.user", from);  
props.put("mail.smtp.password", pass);  
props.put("mail.smtp.port", "587");  
props.put("mail.smtp.auth", "true");
```

Después únicamente hay que iniciar sesión con las características anteriores y configurar el mensaje.

```
message.setFrom(new InternetAddress(from));  
InternetAddress[] toAddress = new InternetAddress[to.length];  
  
// To get the array of addresses  
for( int i = 0; i < to.length; i++ ) {  
    toAddress[i] = new InternetAddress(to[i]);  
}  
  
for( int i = 0; i < toAddress.length; i++) {  
    message.addRecipient(Message.RecipientType.TO, toAddress[i]);  
}  
  
message.setSubject(subject);  
message.setText(body);  
Transport transport = session.getTransport("smtp");  
transport.connect(host, from, pass);
```

```
transport.sendMessage(message, message.getAllRecipients());  
transport.close();
```

Evidentemente, los parámetros que nosotros debemos introducir son: el correo de salida junto con su *password*, el correo de destino, el asunto, y el contenido.

El método creado se denomina `sendFromGMail()`, y llamarlo es tan sencillo como:

```
EmailSenderService.sendFromGMail(from, pass, to, subject, body);
```

Por temas de confidencialidad, no se muestran todos los valores de dichos parámetros. En el caso de la comprobación de la región, en todos los servicios la lógica será común, ya que el proceso es el mismo. Mostramos un ejemplo de los servicios S3.

```
if (!region.equals("EU_WEST_1")) {  
    String from = "awsbbvacib@gmail.com";  
    String pass = "password de email de salida";  
    String[] to = {"emails de llegada"}; // list of recipient email  
addresses  
    String subject = "[AWS] - Región incorrecta";  
    String body = "Bucket " + bucket + " en región inválida. Se encuentra en "  
+ bucketLocation + " y debería estar en " + correctRegion + ".";  
    EmailSenderService.sendFromGMail(from, pass, to, subject, body);  
}
```

En el caso de que lo que estemos comprobando sea la plataforma en la que se ha levantado una instancia, la lógica a desarrollar sería la siguiente:

```
if (i.getDescription().contains("Red Hat") &&  
!instance.getState().getName().equals("stopped")) {  
  
    String from = "awsbbvacib@gmail.com";  
    String pass = "password de email de salida";  
    String[] to = {"emails de llegada"}; // list of recipient email addresses  
    String subject = "[AWS] - Prueba_EC2";  
    String body = "Instance " + i.getName() + " levantada con " + " Red Hat " +  
".";  
    EmailSenderService.sendFromGMail(from, pass, to, subject, body);  
}
```

Evidentemente, en este tipo de instancias, hay que comprobar la plataforma sobre la que se levanta, a la vez que se comprueba si está o no activa, ya que, si está desactivada, no se cobraría nada por dicha instancia, y la plataforma sobre la que está levantada sería irrelevante.

6.1.7 USUARIOS, PERMISOS Y GRUPOS

La aplicación también debe gestionar ciertos aspectos respecto a los usuarios dentro de la cuenta en AWS.

Para comenzar a explicar el desarrollo de esta parte del proyecto, es necesario comentar antes el concepto 'IAM' de AWS.

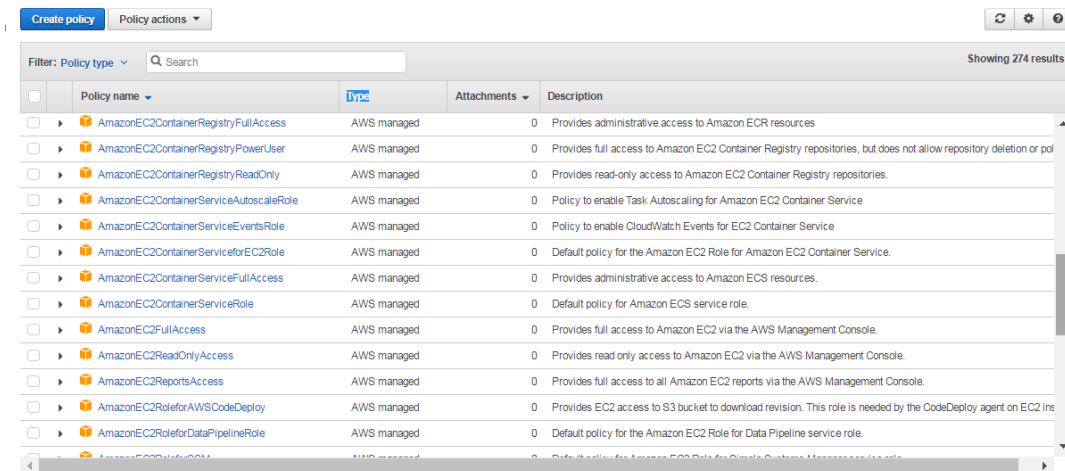
AWS Identity and Access Management (IAM) permite controlar de forma segura el acceso de los usuarios a servicios y recursos de AWS. Con IAM puede crear y administrar usuarios y grupos de AWS, así como utilizar permisos para permitir o denegar el acceso de estos a los recursos de AWS.

IAM es una característica de su cuenta de AWS que se ofrece sin cargos adicionales. Solo se le cobrará por la utilización de los demás servicios de AWS por parte de sus usuarios [14].

En resumen, IAM es la parte de AWS que permite gestionar los integrantes de una cuenta de AWS, así como crear grupos, asignar o quitar roles o permisos, etc...

Un rol no es más que un conjunto de permisos agrupados. Este aspecto surgió con la necesidad de asignar varios permisos comunes a un cierto servicio. A un mismo servicio pertenecen varios permisos, como lectura, lectura o escritura, creación, borrado...

A continuación, mostramos una imagen con posibles diferentes permisos (*policies*) de un mismo servicio:



Policy name	Type	Attachments	Description
AmazonEC2ContainerRegistryFullAccess	AWS managed	0	Provides administrative access to Amazon ECR resources.
AmazonEC2ContainerRegistryPowerUser	AWS managed	0	Provides full access to Amazon EC2 Container Registry repositories, but does not allow repository deletion or pol...
AmazonEC2ContainerRegistryReadOnly	AWS managed	0	Provides read-only access to Amazon EC2 Container Registry repositories.
AmazonEC2ContainerServiceAutoscaleRole	AWS managed	0	Policy to enable Task Autoscaling for Amazon EC2 Container Service.
AmazonEC2ContainerServiceEventsRole	AWS managed	0	Policy to enable CloudWatch Events for EC2 Container Service.
AmazonEC2ContainerServiceforEC2Role	AWS managed	0	Default policy for the Amazon EC2 Role for Amazon EC2 Container Service.
AmazonEC2ContainerServiceFullAccess	AWS managed	0	Provides administrative access to Amazon ECS resources.
AmazonEC2ContainerServiceRole	AWS managed	0	Default policy for Amazon ECS service role.
AmazonEC2FullAccess	AWS managed	0	Provides full access to Amazon EC2 via the AWS Management Console.
AmazonEC2ReadOnlyAccess	AWS managed	0	Provides read only access to Amazon EC2 via the AWS Management Console.
AmazonEC2ReportsAccess	AWS managed	0	Provides full access to all Amazon EC2 reports via the AWS Management Console.
AmazonEC2RoleforAWSCodeDeploy	AWS managed	0	Provides EC2 access to S3 bucket to download revision. This role is needed by the CodeDeploy agent on EC2 ins...
AmazonEC2RoleforDataPipelineRole	AWS managed	0	Default policy for the Amazon EC2 Role for Data Pipeline service role.

Figura 55: Conjunto de permisos de un mismo servicio

Lo primero que tenemos que realizar, igual que en el caso de los servicios, es crearnos un cliente ‘iam’ en el proyecto para realizar peticiones a esta parte de nuestra cuenta. Se hace de la siguiente manera:

```
AmazonIdentityManagement aim = AmazonIdentityManagementClientBuilder.standard()
    .withRegion(Regions.EU_WEST_1)
    .build();
```

Cogemos el id y la contraseña de mi perfil de las variables de entorno y asignamos la región de estudio en Irlanda. Como queremos listar todos los usuarios activos en la cuenta de la entidad, debemos buscar un método que nos devuelva un ‘array’ con todos ellos. La clase “AmazonIdentityManagement” provee un método que devuelve un ‘array’ de clase “User” que se llama *listUsers()*.

De la clase ‘User’, ya podemos obtener todos los datos que necesitamos, como el nombre, el id, la fecha de la última conexión o la fecha en que se dio de alta. En nuestro caso, el nombre de dicho usuario será su identificador dentro del banco.

Respecto a los grupos, se deben poder crear desde la aplicación indicando el nombre en un formulario de entrada.

Es lógico pensar que, una vez creados los grupos, sería recomendable poder acceder a una lista de usuarios para asignárselos a dicho grupo, así como determinados permisos.

En el banco, hay un departamento concreto de seguridad que se encarga de realizar todas estas gestiones, por lo que ni mi perfil, ni el de mi responsable dentro del proyecto tienen los permisos suficientes para poder asignar usuarios a grupos ni permisos concretos a usuarios, por lo que desde la aplicación dejaremos la funcionalidad respecto a los grupos únicamente en el paso de creación.

El proceso para crear el grupo es el siguiente:

```
CreateGroupRequest newgroup = new CreateGroupRequest ();  
newgroup.setGroupName(name);  
CreateGroupResult group = aim.createGroup(newgroup);
```

Lógicamente, a estas alturas del proyecto la variable ‘name’ será introducida por consola, pero en un futuro será leído de un formulario.

En cuanto a la asignación de permisos a usuarios concretos, tenemos el mismo problema que anteriormente. Nuestro perfil no está dotado de los permisos suficientes para poder realizarlo, y desde el departamento correspondiente no aceptan cedérmolos. Si lo intentamos, sucede lo siguiente:

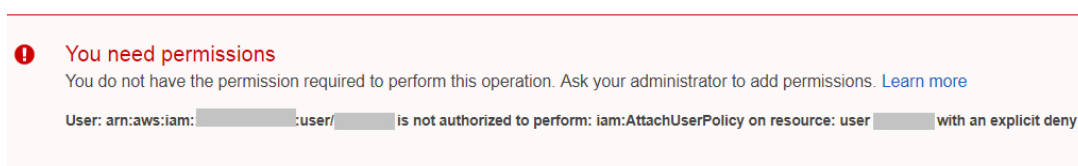


Figura 56: Denegación de acción por falta de permisos

Cómo la asignación de permisos si va a ser una parte bastante crítica del proyecto, tendremos que buscar una posible solución.

Dicha solución es crear un archivo en formato PDF desde Java, pasándole ciertos parámetros, como los permisos que se desean asignar y a quién.

Para la creación de archivos PDF desde Java, debemos cargar la librería correspondiente en Maven. En este caso, usaremos la librería denominada “iText”.

```
<!-- https://mvnrepository.com/artifact/com.lowagie/itext -->
<dependency>
  <groupId>com.lowagie</groupId>
  <artifactId>itext</artifactId>
  <version>4.2.1</version>
</dependency>
```

El código utilizado para crear el archivo correspondiente se muestra en el anexo A.

6.1.8 CREACIÓN DE LA BASE DE DATOS

Dentro de la funcionalidad que debe tener la aplicación en cuestión, hay una que todavía no se ha mencionado. Cuando una instancia es levantada en AWS, el propietario que se asigna es la cuenta a la que pertenece el usuario que la ha contratado, pero no el usuario en sí.

Lo que se desea es saber literalmente que usuario ha levantado cada instancia, pero como AWS no proporciona esa información, se ha decidido crear una base de datos con una tabla que lo indique. La tabla tendrá varios campos, como el nombre de la instancia, el usuario creador, la región, el grupo al que pertenece el usuario y el tipo de instancia. A continuación, mostramos una imagen de la tabla. Los valores de los usuarios están ocultos por temas de confidencialidad, así como las instancias. La región puede ser directamente Irlanda o poner el calificativo que le da AWS (EU-WEST-1), ya que el programa valora ambas. Mostramos algunos de ellos.

INSTANCE_CREATOR_USER_NAME	INSTANCE_GROUP	INSTANCE_NAME	INSTANCE_REGION	INSTANCE_TYPE
[oculto]	Admins	[oculto]	eu-west-1b	EC2
[oculto]	Admins	[oculto]	eu-west-1b	EC2
[oculto]	Admins	[oculto]	eu-west-1b	EC2
[oculto]	Admins	[oculto]	eu-west-1b	EC2
[oculto]	Admins	[oculto]	eu-west-1b	EC2
[oculto]	Admins	[oculto]	eu-west-1b	EC2
[oculto]	Admins	[oculto]	eu-west-1a	EC2
[oculto]	Admins	[oculto]	Irlanda	S3
[oculto]	Admins	[oculto]	Irlanda	S3
[oculto]	Admins	[oculto]	Irlanda	S3

Figura 57: Tabla instancias-usuarios

Evidentemente, esta tabla no se puede rellenar automáticamente ya que AWS no permite obtener la información que deseamos, por lo que será obligación del responsable rellenar esta tabla a través de un pequeño formulario cada vez que se levante una máquina nueva.

Para la creación de la base de datos, se ha hecho uso del *framework* Spring, en concreto Spring Boot. El primer paso para la utilización de Spring, es añadir la librería correspondiente. Hay varias dependencias de Maven que deben ser incluidas:

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.2.RELEASE</version>
</parent>

<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>1.7.25</version>
</dependency>

<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
</dependency>

<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.16.8</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-web</artifactId>
  <version>4.3.7.RELEASE</version>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
```

```
<groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
  <version>1.5.2.RELEASE</version>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
</dependency>

<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.2.2</version>
</dependency>

<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-core</artifactId>
  <version>1.2.2</version>
</dependency>

<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

Cuando se usa Spring para desarrollar cierta funcionalidad, este importa todo lo que necesita a través de etiquetas precedidas de un @. Por ejemplo, si en una clase se quieren importar los ‘getter’ y ‘setter’ de los atributos, simplemente se podría al inicio un ‘@Getter’ y un ‘@Setter’.

Todas estas etiquetas son las que se necesitan y se importan en todas las librerías anteriores.

La base de datos va a ser de tipo 'H2', que son basadas en Java. Otra opción podría haber sido crearla de tipo 'mysql', pero cómo todo el proyecto está desarrollado en Java, se decidió hacerla 'H2'. Además, otro elemento que se tuvo en cuenta para la decisión del tipo de base de datos es que no se requieren tiempos exigentes de respuesta ni el manejo de volúmenes grandes de información, características con las que H2 funciona realmente bien.

Cuando se crea una base de datos con Spring, se deben crear 3 cosas por cada tabla que tengamos. Hasta este punto del proyecto, solo tenemos una, por lo que solo debemos crear dichos elementos una vez. Más adelante se explicará una ampliación de la base de datos.

Dichos elementos son:

- La **clase principal**, también llamada **entidad**, que va a definir la tabla
- Un **interfaz**, que será el **repositorio** de la clase
- Una **clase de servicios** que será dónde definiremos la funcionalidad que queramos.

Spring, para el uso de base datos, hace uso del *framework* de Java JPA (Java Persistence API) que nos permite establecer una correlación entre una base de datos relacional y un sistema orientado a objetos. Esta correlación es llamada ORM (Object Relational Mapping), la cual genera anotaciones sobre Entidades. JPA establece una interface común que es implementada por un proveedor de persistencia de nuestra elección

Una entidad es una clase POJO que debe proporcionar un método constructor por defecto, no debe ser final, y debe implementar "Serializable" para accesos remotos. Todas las entidades tienen que poseer una identidad única (conocido como 'Primary Key' en bases de datos relacionales) por lo que deben tener una propiedad marcada con la anotación @Id y "@GeneratedValue" (para la generación de la 'primary key' por el proveedor de persistencia) [15].

Como podemos ver, a una entidad hay que añadirle bastantes etiquetas para indicar todos los aspectos que deseamos. Con la etiqueta '@Data' importamos todos los 'getters' y 'setters' de los atributos indicados en la clase, con la '@ToString', importamos el método toString() con todos los atributos.

Para indicar que dicha clase es una entidad, debemos incorporar la etiqueta '@Entity', para comunicar a nuestro proyecto que esta clase va a ser una entidad de JPA, y que, por tanto, servirá para la base de datos. Para definir el nombre de la tabla que queremos crear, hacemos uso de la etiqueta '@Table', con la que podemos especificar dicho nombre.

Por otro lado, el repositorio de la entidad, para tener las funciones que proporciona JPA, debe extender de él. Como nuestra entidad se ha denominado 'InstanceBbdd', la manera de hacerlo sería la siguiente:

```
public interface RepositoryInstanceBbdd extends JpaRepository<InstanceBbdd, Long>
{
    //Solo nombramos los métodos. Los que lleven Query, se pone por encima.
}
```

Al extender de JPA, automáticamente el repositorio contiene ciertos métodos muy útiles que no hace falta nombrarlos. Los más importantes de ellos en nuestro caso son los de lectura de la tabla, que comienzan por 'find'.

Si queremos acceder a todos los datos de la tabla, el método es 'findAll()', que por dentro ejecuta la siguiente *query*: `select * from InstanceBbdd`, pero al extender de JPA, lo hace automáticamente. Los *frameworks* están para facilitar el trabajo al desarrollador, y como podemos ver, 'Spring' lo hace bastante en cuestión de manejo de bases de datos.

Aparte del 'findAll()', también proporciona métodos de búsqueda según los parámetros de la base de datos. Por ejemplo, si tenemos una columna que se llama 'InstanceName', JPA nos proporciona un método llamado 'findByInstanceName()', y así continuamente con todos los atributos de la tabla en cuestión. Evidentemente se nos proporcionan muchos más atributos, pero no vamos a comentarlos todos aquí.

Por último, en cuanto a la base de datos, explicaremos la funcionalidad de la clase 'Services'. Dicha clase es la que implementa todos los métodos del repositorio y añade más lógica para que se pueda desarrollar la funcionalidad deseada. Para indicar que es una clase de servicios debe indicarse con la etiqueta '@Service'.

Cómo lo que se desea es añadir lógica a los métodos proporcionados por el repositorio, es evidente que en algún momento hay que importarlo. Para ello, Spring utiliza la etiqueta `@Autowired`, que importa el archivo que indiquemos. Ejemplo:

```
@Autowired  
private RepositoryInstanceBbdd repositoryInstanceBbdd;
```

Cómo nuestra mayor funcionalidad respecto a esta tabla en la base de datos va a ser mostrar todas las instancias, así como las instancias por usuarios, definiremos los siguientes métodos en el servicio.

```
public List<InstanceBbdd> allInstances() {  
    List<InstanceBbdd> instanceBbdds = (List<InstanceBbdd>)  
    repositoryInstanceBbdd.findAll();  
    return instanceBbdds;  
}  
  
public List<InstanceBbdd> allInstancesbyUser(String user) {  
    List<InstanceBbdd> instanceBbdds = (List<InstanceBbdd>)  
    repositoryInstanceBbdd.findByInstanceCreatorUserName(user);  
    return instanceBbdds;  
}
```

Para ejecutar estas clases, debemos crear una clase principal, que, a su vez, por debajo, ejecutará las anteriormente explicadas para crear la base de datos. Antes de seguir con la explicación, mostramos un diagrama de clases para ver la relación entre las clases que componen una tabla de la base de datos.

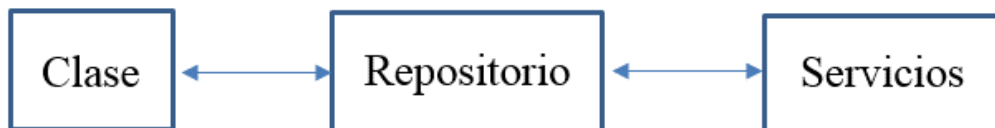


Figura 58: Clases Spring para la base de datos.

La clase, definirá las columnas de tabla, así como su título y los tipos de datos que contendrá. El repositorio de Spring, contiene una serie de métodos ya creados que hacen que el acceso

a los datos sea más sencillo. Y, por último, en los servicios se definen los métodos que nosotros queramos para realizar una lógica particular.

La clase principal mencionada anteriormente la llamaremos ‘*Application*’, y deberá importar el repositorio y el servicio que requiera utilizar, así como todos los paquetes necesarios para su correcto funcionamiento. En este caso, como lo único que queremos es guardar datos en la base de datos, importaremos el repositorio correspondiente. En cuanto a las etiquetas necesarias, tenemos las siguientes:

```
@Slf4j  
@SpringBootApplication  
@ComponentScan(basePackages = {"util"})  
@EntityScan(basePackages = {"util"})  
@EnableJpaRepositories(basePackages = {"util"})
```

La primera es la que activa el sistema de log para poder utilizarlo y mirar que está ocurriendo en cada paso, ‘*@SpringBootApplication*’ simplemente indica que se está desarrollando una aplicación Spring, los ‘*scan*’ escanean todos las entidades y los componentes que hay en el proyecto, y ‘*EnableJpaRepositories*’ activa las funciones de JPA.

Dentro de dicha clase, deberemos introducir el siguiente código para arrancar la aplicación y que empiece a funcionar:

```
public void createbbdd() {  
    SpringApplication.run(Application.class);  
}  
  
@Bean  
public CommandLineRunner demo(RepositoryInstanceBbdd repositoryInstanceBbdd) {  
    return (args) -> {  
  
        // Lógica  
    }  
}
```

Hasta este punto, no hemos guardado ningún dato en nuestra tabla y, por tanto, está vacía. En la lógica a desarrollar, será donde introduzcamos los datos correspondientes. Para ello,

haremos uso del método `save(Object object)`, almacenado en el repositorio. Mostramos un ejemplo:

```
repositoryInstanceBbdd.save(new InstanceBbdd("instanceName", "Type", "Creator",  
"Region", "Group"));
```

Es importante comentar, que Spring da la facilidad de poder crear un archivo de configuración para definir ciertos parámetros. Dicho archivo es de extensión ‘yml’, y permite definir mediante código de definición los puntos necesarios para el buen funcionamiento de la base de datos, como el tipo de la misma, en que puerto va a levantarse, si es persistente o definida en memoria, que driver se va a usar, etc...

A continuación, comentaremos los aspectos más importantes.

La aplicación se levantará para el desarrollo en nuestra máquina local (localhost), pero en un futuro, será almacenada en un servidor para poder acceder remotamente.

El puerto será el 36080. El driver utilizado será ‘jdbc’ y el archivo de almacenamiento de la base de datos se denominará ‘test’. Evidentemente es un archivo binario cuyos datos no pueden ser visualizados correctamente. La plataforma como indicamos anteriormente será de tipo ‘H2’. En cualquier caso, si en algún momento en la futura ampliación de la aplicación, se desea hacer uso de otro tipo de base de datos, se deberían hacer unos cambios mínimos en la programación del proyecto, pero que no implicarían ninguna diferencia en cuanto al funcionamiento de la web. Desde el archivo de configuración también activaremos una consola, que será la interfaz que se proporciona para poder visualizar las tablas de manera más sencilla, poder hacer peticiones para comprobar el funcionamiento y, en definitiva, comprobar que todo funciona según lo deseado. El código de configuración se muestra en el Anexo final de la memoria.

A continuación, mostramos cómo acceder a la consola (mirar archivo de configuración) y varias imágenes de las tablas, y cómo se visualizarían las tablas creadas. En el navegador se debe introducir la siguiente dirección “<http://localhost:36080/console>”. Como la aplicación

esta levantada en local, deberemos acceder a través de localhost. El puerto y la dirección de la consola están definidos en el archivo de configuración, así como el archivo de persistencia y en nombre de usuario.

The screenshot shows a 'Login' dialog box with the following fields and controls:

- Language: English (dropdown)
- Preferences Tools Help (links)
- Saved Settings: Generic H2 (Embedded) (dropdown)
- Setting Name: Generic H2 (Embedded) (text input) with Save and Remove buttons
- Driver Class: org.h2.Driver (text input)
- JDBC URL: jdbc:h2:~/test (text input)
- User Name: sa (text input)
- Password: (empty text input)
- Connect and Test Connection buttons

Figura 59: Acceso a la base de datos

Una vez conseguido el acceso, visualizaremos en la parte superior izquierda el contenido de nuestro archivo persistente, es decir, todas las tablas que tenemos creadas. Hasta ahora sólo hemos explicado la tabla “Instances”, pero más adelante explicaremos la razón del resto de tablas creadas, que cómo podemos ver, corresponden a los servicios que se están monitorizando.

The screenshot shows the H2 database interface. At the top, there is a toolbar with icons for connection, auto-commit, max rows (set to 1000), and auto-complete (set to Off). Below the toolbar is a file tree on the left showing the database structure, including tables like CLOUDSEARCHINSTANCES, EC2INSTANCES, INSTANCES, S3FILE, S3FOLDER, S3INSTANCES, INFORMATION_SCHEMA, Sequences, and Users. The main area is a large text input field for SQL statements, with buttons for 'Run', 'Run Selected', 'Auto complete', and 'Clear'. Below the input field, there are two help sections: 'Important Commands' and 'Sample SQL Script'. The 'Important Commands' section lists shortcuts like Ctrl+Enter for executing the current statement and Ctrl+Space for auto-complete. The 'Sample SQL Script' section provides a list of actions and their corresponding SQL commands, such as 'Delete the table if it exists' leading to 'DROP TABLE IF EXISTS TEST;'.

Important Commands		
		Displays this Help Page
		Shows the Command History
	Ctrl+Enter	Executes the current SQL statement
	Shift+Enter	Executes the SQL statement defined by the text selection
	Ctrl+Space	Auto complete
		Disconnects from the database

Sample SQL Script	
Delete the table if it exists	DROP TABLE IF EXISTS TEST;
Create a new table with ID and NAME columns	CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
Add a new row	INSERT INTO TEST VALUES(1, 'Hello');
Add another row	INSERT INTO TEST VALUES(2, 'World');
Query the table	SELECT * FROM TEST ORDER BY ID;
Change data in a row	UPDATE TEST SET NAME='Hi' WHERE ID=1;
Remove a row	DELETE FROM TEST WHERE ID=2;
Help	HELP ...

Figura 60: Interfaz de la base de datos con su contenido.

Una vez dentro, si queremos comprobar los valores de una de las tablas o realizar alguna ‘query’ en particular, simplemente debemos introducirla en el cuadro de input y pulsar el botón ‘run’ o pulsar el símbolo verde de ‘play’.

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM INSTANCES;

ID	INSTANCE_CREATOR_USER_NAME	INSTANCE_GROUP	INSTANCE_NAME	INSTANCE_REGION	INSTANCE_TYPE
1		Admins		eu-west-1b	EC2
2		Admins		eu-west-1b	EC2
3		Admins		eu-west-1b	EC2
4		Admins		eu-west-1b	EC2
5		Admins		eu-west-1b	EC2
6		Admins		eu-west-1b	EC2
7		Admins		eu-west-1a	EC2
8		Admins		Irlanda	S3
9		Admins		Irlanda	S3
10		Admins		Irlanda	S3
12		Admins		Frankfurt	S3
13		Admins		eu-west-1	CloudSearch
65		Admins		Irlanda	EC2
66		Admins		Irlanda	EC2
97		Admins		EU-WEST-1	EC2
98		Admins		EU-WEST1	S3
159		Admins		Eu-WEST-1	S3

(17 rows, 2 ms)

Edit

Figura 61: Visualización de una tabla de la base de datos

Volvemos a ocultar la información confidencial. Como podemos observar, se visualizan todos los datos correctamente. Es importante decir que, en este punto del desarrollo, los usuarios están “*mockeados*”, pero al final del mismo serán los correctos.

6.2 API-REST

Actualmente las API-REST están cogiendo mucho auge dentro del desarrollo de aplicaciones web. La clave de este éxito tan rotundo está en que un servicio REST no tiene estado (es ‘*stateless*’), lo que quiere decir que, entre dos llamadas cualesquiera, el servicio pierde todos sus datos. Esto es, que no se puede llamar a un servicio REST y pasarle unos datos (p. ej. un usuario y una contraseña) y esperar que “nos recuerde” en la siguiente petición. De ahí el nombre: el estado lo mantiene el cliente y por lo tanto es el cliente quien debe pasar el estado en cada llamada. Si quiero que un servicio REST me recuerde, debo pasarle quien soy en cada llamada. Eso puede ser un usuario y una contraseña, un token o

cualquier otro tipo de credenciales, pero debo pasarlas en cada petición. Y lo mismo aplica para el resto de información [16].

Esta API, nos servirá de puente entre nuestro servidor con la lógica y nuestro cliente, que le hará ciertas peticiones para obtener la información correspondiente. Para ello, volveremos a hacer uso de Spring, el mismo *framework* utilizado para la creación de base de datos. La comunicación con la lógica se realizará a través de pequeñas URI's, que identifica los recursos de la red de forma unívoca.

Para definir cada punto de acceso al servicio, deberemos indicar varias cosas, entre las que se encuentran:

- Uri
- Tipo de petición (Get, Post...)
- Formato en el que se devuelven los datos, JSON o XML.
- Método que se ejecutará.

Todas las peticiones, cómo son de obtención de datos, serán de tipo 'GET', así como el formato de los datos, que se devolverán todos en formato JSON, para poder ser leídos posteriormente con facilidad.

Este aspecto, hace que necesitemos una clase que defina objetos por cada objeto que queramos obtener, por lo que necesitaremos una clase que defina cada tipo de servicio (S3, EC2, y CloudSearch), y otra que defina a los usuarios. Cada una de ellas con sus respectivos parámetros, evidentemente.

Para indicar a nuestra clase Spring que va a ser un punto de acceso *rest* al servicio, debemos indicar con una etiqueta dicha funcionalidad. Esta etiqueta es '@RestController'.

A continuación, solo debemos indicar en los métodos creados toda la lógica que hemos desarrollado anteriormente cuya visión era por consola, con la pequeña diferencia que ahora al obtener todos los datos de peticiones a AWS, debemos guardarlos en una clase previa

creada y devolverla en la response correspondiente. La response es la respuesta que se va a proporcionar cuando un cliente acceda a la URI correspondiente.

Mostramos a continuación la creación de un punto de acceso al servicio para mostrar todos los usuarios:

```
@RequestMapping(value = "/allusers", method = RequestMethod.GET, produces =  
MediaType.APPLICATION_JSON_VALUE)  
public ResponseEntity<List<Usuario>> listAllUsers() {  
  
    //Lógica a desarrollar  
  
    if (usuarios.isEmpty()) {  
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);  
    }  
  
    return new ResponseEntity<>(usuarios, HttpStatus.OK);  
}
```

Evidentemente, ‘usuarios’ es el ‘array’ de la clase Usuario que se va a rellenar con los datos obtenidos de las peticiones a AWS y que posteriormente se enviará como respuesta a la petición correspondiente. Usuario es la clase que nos hemos tenido que crear para el almacenamiento de datos desde AWS.

Para comprobar que todos estos ‘SAP’ (puntos de acceso al servicio) funcionan de manera correcta, tenemos dos opciones. La primera consiste simplemente es meter la URI correspondiente en el navegador y ver la respuesta que devuelve. En caso de que haya ido todo correcto, se mostrarán los datos, pero de manera poco intuitiva. En caso contrario, deberemos ir a la consola del navegador con F12 y ver los errores. La otra opción es hacer uso de Postman, dónde podemos hacer las comprobaciones de nuestra API creada, indicando si queremos la respuesta en tipo JSON o XML, si la petición es GET o POST, etc...

A la hora de este tipo de pruebas, es realmente un interfaz muy útil, ya que es capaz de mostrar todo de manera muy intuitiva y clara, no como el navegador.

A continuación, mostramos una imagen que lo demuestra.

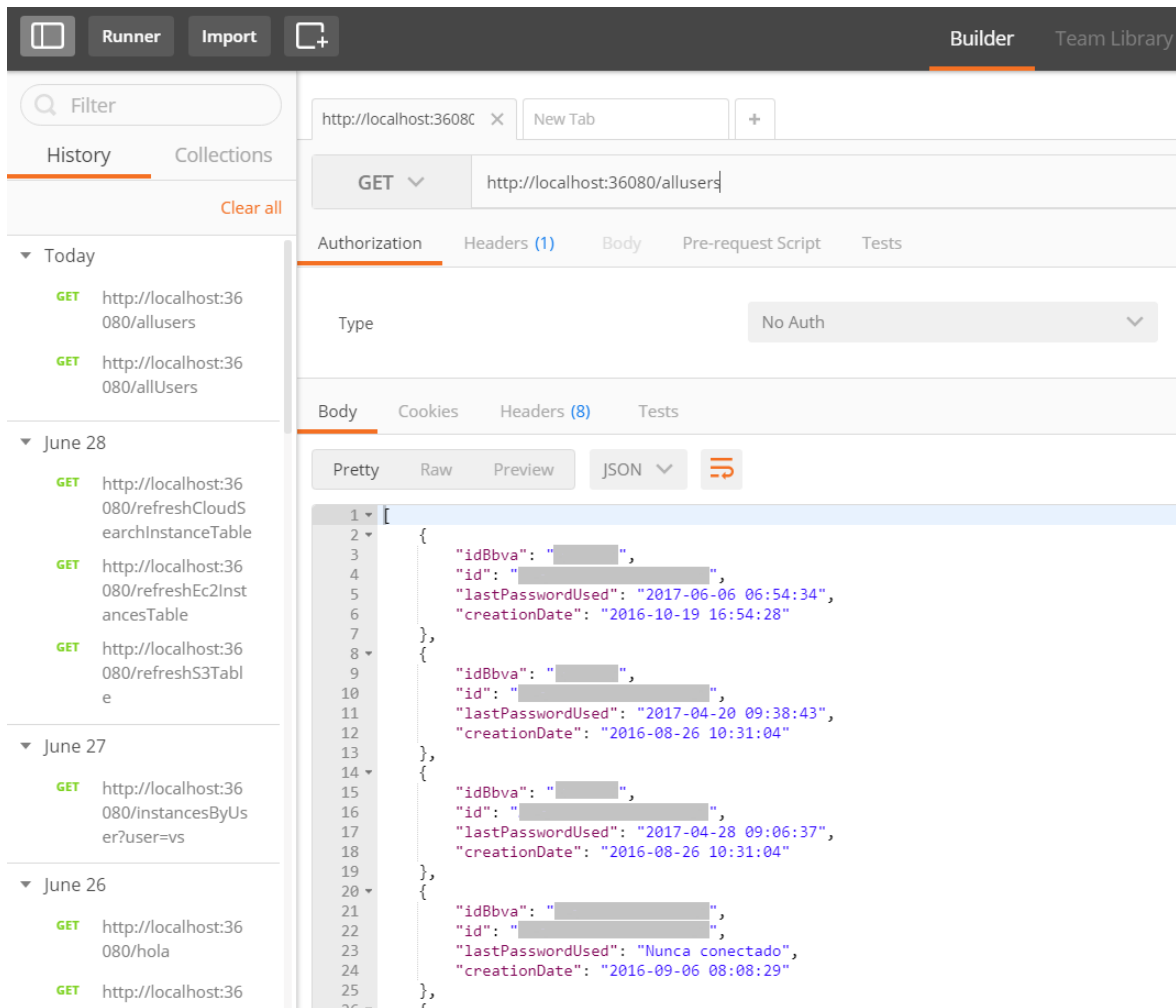


Figura 62: Ejemplo de petición en Postman

Como podemos observar, la petición realizada ha sido la de mostrar todos los usuarios dados de alta en AWS. En este caso, la URI a introducir ha sido “allusers”, el tipo de respuesta debe ser JSON ya que así se ha definido en la creación de la API REST y todas las peticiones son de tipo GET. Como se puede observar, la clase que nos hemos creado tiene los parámetros Id del banco, Id dentro de AWS, la última fecha de conexión y la fecha de alta. La última fecha de conexión se denomina “lastPasswordUsed” porque AWS lo define de esta manera. Es como la última vez que se usó la contraseña para acceder. Se vuelven a ocultar los id por temas de privacidad.

El proceso para comprobar el resto de la funcionalidad es el mismo. Desde el cliente web, para acceder a estos datos, sólo tenemos que acceder a la URI correspondiente y leer el objeto JSON. La manera de mostrarlos ya forma parte del desarrollo *front-end*.

6.3 CLIENTE ANGULAR 2

Como hemos indicado a lo largo del documento, nuestro cliente será desarrollado con unos de los últimos *frameworks* en desarrollo web, Angular 2.

Una de las mayores ventajas que provee este sistema es el “*two way data binding*”. A diferencia de la mayoría de sistemas de ‘*templates*’ y/o *frameworks*, Angular usa un sistema en el que vista y modelo están en relación constante, se considera el modelo como ‘*Single-Source-of-Truth*’. Gracias a esto, se logra que todo cambio visual, se actualice a tiempo real en el modelo y viceversa, evitando que sea el desarrollador el encargado de lograr la sincronía entre modelo y vista, como es el caso de otros *frameworks* [17].

Angular 2 basa sus proyectos en componentes que unidos, van conformando una página web. Se parte de una base principal, el ‘*index.html*’, y en vez de desarrollar toda la vista en él, se añaden los nuevos componentes. Cada componente se basa en una vista propia, que consiste en un html puro, sin importaciones ni etiquetas de configuración, y su lógica, basa en Typescript, que es como JavaScript, pero tipado, es decir, las variables tienen tipo.

Para crear un proyecto Angular 2, se necesitan tener dos cosas previamente instaladas, Node.js y npm.

Node.js es un intérprete Javascript del lado del servidor que cambia la noción de cómo debería trabajar un servidor. Su meta es permitir a un programador construir aplicaciones altamente escalables y escribir código que maneje decenas de miles de conexiones simultáneas en una sólo una máquina física [18].

Por otro lado, npm es el manejador de paquetes por defecto para Node.js. Si queremos importar algún paquete con una funcionalidad concreta, debe realizarse con npm.

Para crear un proyecto Angular 2 se deben seguir los siguientes pasos, después de haber instalado Node.js y npm.

Primero debemos instalar el CLI de Angular, para ayudarnos a introducir comandos de configuración. Evidentemente, desde la ventana de comandos de Windows

```
npm install -g @angular/cli
```

Una vez instalado, crear un proyecto es sumamente sencillo:

```
ng new NombreDelProyecto
```

En nuestro caso, vamos a llamar al proyecto “awsclient”. La creación del mismo es algo tardía ya que este tipo de proyectos requieren de muchos archivos de configuración, y se tarda algo en crearlos todos.

En el inicio del proyecto podemos observar que hay un único componente muy básico. Para ejecutar este tipo de proyectos desde IntelliJ, lo primero que tenemos que hacer es crear una nueva variable de ejecución de tipo npm. En la imagen que se muestra a continuación, podemos ver como se hace.

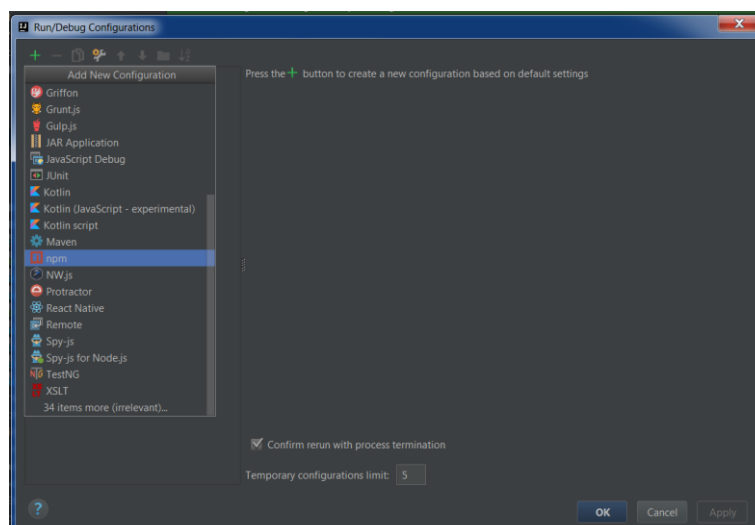


Figura 63: creación de variable de ejecución

Posteriormente, el programa detectará automáticamente que modos posibles hay. Podemos elegir el de construir el proyecto, que será ‘build’, el de compilar y ejecutar, que será ‘start’ entre otros muchos. Nosotros configuraremos el ‘start’ para compilar y ejecutar el programa directamente. Esta variable por debajo ejecuta el comando “ng serve”, que levanta un servidor local con nuestro proyecto.

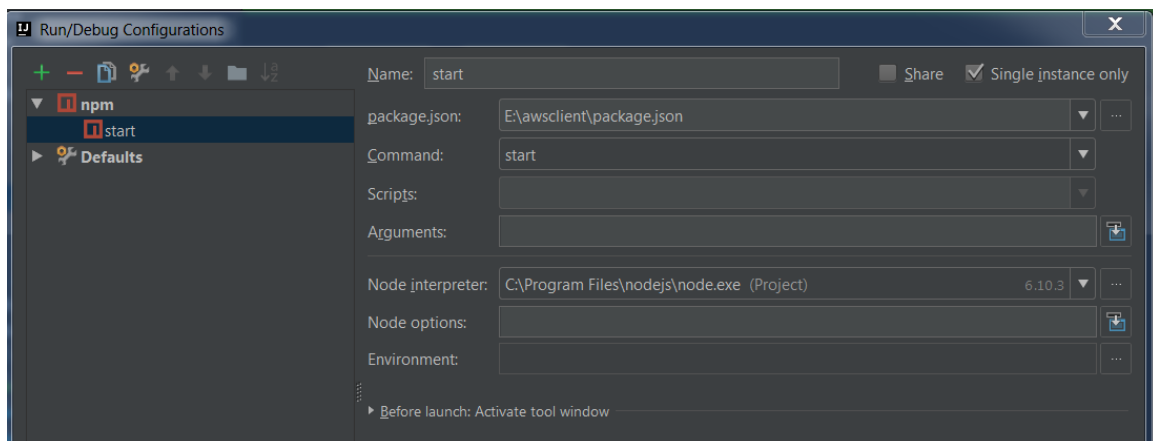


Figura 64: Configuración de la variable de start npm

A la hora de compilar y ejecutar el proyecto, se nos indica por la consola del IDE donde se ha levantado el cliente, es decir, una dirección seguida de un puerto. Lo único que tenemos que hacer es copiar dichos parámetros e introducirlos en el navegador.

Antes de comenzar a explicar el desarrollo del cliente, es importante explicar el archivo de configuración tslint.json, que indica cómo se debe escribir el código dentro de cualquier programa Angular. Algunas reglas son bastante tediosas, ya que uno no está acostumbrado a programar estando obligado a poner espacios, llaves en lugares concretos, comillas simples en vez de dobles, etc...

Por todo esto, vamos a comentar algunos cambios que hemos realizado para hacer la programación algo menos estresante. Para establecer un valor cualquiera a una variable, en typescript se utiliza el comando ‘let’. Por sí sólo no se podría utilizar ya que el archivo de configuración no te lo permite y te obliga a usar otro comando, como ‘const’, que funciona

peor. Para poder habilitarlo, simplemente comentamos la línea dónde el ‘const’ se antepone a cualquier otro comando de definición.

```
/*"prefer-const": true,*/
```

Si quisiéramos admitir las comillas dobles, deberíamos omitir el siguiente código, pero en nuestro caso se ha decidido dejar ya que no supone una ardua tarea escribir comillas simples en vez de dobles.

```
"quotemark": [  
  true,  
  "single"  
]
```

Y, por último, se han omitido los tres ‘=’ seguidos dentro de un comando ‘if’ para comparar dos variables, ya que es algo que se suele hacer con 2 en la mayoría de los casos.

```
"triple-equals": [  
  false, // Antes era true, se cambio...  
  "allow-null-check"  
]
```

Se ha decidido no tocar más este archivo ya que es cierto que las reglas que propone ayudan a tener el código limpio, entendible y bien organizado.

Una vez explicado esto, pasamos a comentar los componentes que van a conformar nuestra aplicación. Normalmente sería lógico dividir una web en muchos componentes pequeños e independientes, pero ya que nosotros vamos a realizar un sistema de monitorización, vamos a meter todos los datos en un único componente denominado “*body*”, mientras que las imágenes corporativas, como el logo del banco o el nombre del departamento de desarrollo estará, en otro componente denominado “*header*”. Podemos concluir pues, que toda la parte grande de nuestro proyecto se centrará en el “*body*” de la web.

Comenzaremos explicando el primer componente. Cómo es bastante básico, ayudará a entender cómo funcionan y como se muestran los componentes en la web.

6.3.1 COMPONENTE CABECERA

Lo primero que se debe saber de los componentes es que para crearlos se deben crear dos archivos, un “html” y un “ts” (vista y lógica). Lo más recomendable es meterlos en paquetes para poder diferenciar fácilmente cuantos componentes hay. Es fundamental llamar a estos archivos por el nombre del componente seguidos de un punto y la palabra “component”, como se muestra a continuación.

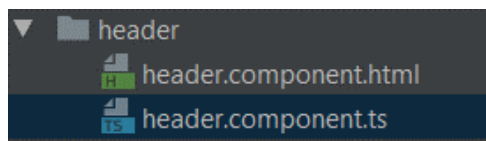


Figura 65: División de un componente

Empezaremos por la parte lógica de nuestro componente. Cada componente se va a declarar en la vista principal como una etiqueta html normal, por lo tanto, debe crearse una para poder referenciar dicha etiqueta con nuestro componente en cuestión.

Para declarar la etiqueta de un componente y cuál va a ser su vista, se hace de la siguiente manera:

```
@Component ({  
  selector: 'app-header',  
  templateUrl: './header.component.html'  
})
```

De esta manera estamos indicando que la etiqueta correspondiente a este componente se va a denominar ‘app-header’ y que la vista del mismo se va a definir en el archivo header.component.html. Evidentemente la etiqueta ‘@Component’ es propia de Angular, por lo que antes de definir estos parámetros, deberemos importarla al inicio del archivo. Se hace de la siguiente manera:

```
import {Component} from '@angular/core';
```

Para declarar la lógica, no se hace de manera muy diferente a una clase Java normal. Se exporta la clase de nuestro componente y se empiezan a definir las variables, el constructor y la lógica correspondientes.

```
export class HeaderComponent {  
  
  imgHeaderBbva: String = this.barraDegradadaCabecera;  
  imgBbvaCIB: String = this.BBVAcib;  
  
  constructor() {  
  
  }  
  
}
```

Cómo podemos observar, en nuestro componente hemos definido dos variables de tipo String que se igualan a otras variables de tipo String que definen las rutas dónde se encuentran las imágenes que vamos a mostrar en la vista. Los puntos en las rutas simplemente son para bajar al nivel del paquete llamado ‘assets’ que es dónde se encuentran las imágenes.

La parte visual del componente únicamente está compuesto por el siguiente código:

```
<div>  
  <img [src]="imgHeaderBbva" width="20" height="100"/>  
  <img [src]="imgBbvaCIB" width="550" height="100"/>  
</div>
```

Como dijimos anteriormente, este html está formado únicamente por contenido, y no por etiquetas de configuración. Importamos solamente las imágenes desde html y les definimos un ancho y un alto.

Nuestra vista principal, denominada ‘index.html’ si es un archivo con etiquetas de configuración, donde se define el head y el *body* del archivo, donde se importan archivos para el correcto funcionamiento de la vista, etc...

Antes de cargar nuestro componente en la vista principal, es necesario explicar un archivo de configuración muy importante en cualquier proyecto Angular 2. Dicho archivo es el que tiene definidos todos los componentes que están siendo utilizados en nuestro proyecto. Se llama *app.module.ts*, aunque se suele denominar simplemente el archivo ‘module’. Dicho

archivo contiene una etiqueta denominada `@NgModule`, con 3 parámetros principales (podría tener más, pero con éstos es suficiente). Dichos parámetros dividen a los componentes en 3 “equipos diferentes”, que son:

- El componente principal (bootstrap)
- Componentes declarados por nosotros (declarations)
- Componentes importados de alguna librería externa (imports)

Por lo tanto, lo que tendremos que hacer es añadir al apartado de ‘declarations’ el componente que nos acabamos de crear, para que el proyecto tenga en cuenta que este componente exista. Si no se declara, el proyecto no tendrá conciencia de este componente y por tanto, no podrá cargarlo.

A continuación, mostramos el contenido de la etiqueta ‘`@NgModule`’, aunque lo iremos explicando poco a poco.

```
@NgModule({
  declarations: [ // CUANTOS COMPONENTES HAY Ó COMPONENTES DECLARADOS
    AppComponent,
    BodyComponent,
    HeaderComponent
  ],
  imports: [ // COMPONENTES IMPORTADOS
    BrowserModule,
    ChartsModule,
    FormsModule,
    HttpClientModule,
    JsonpModule,
    BrowserModule,
    BrowserModule,
    BrowserModule,
    BrowserAnimationsModule,
    FormsModule,
    HttpClientModule,
    MaterialModule,
    MdNativeDateModule,
    ReactiveFormsModule,
    Ng2BootstrapModule,
    TabsModule.forRoot(),
  ],
  bootstrap: [AppComponent] // CUAL ES EL COMPONENTES PRINCIPAL
})
```

Podemos ver cómo el componente principal de nuestro proyecto se llama AppComponent (que será el que contenta nuestros componentes y el que luego se cargará en la vista principal), así como la declaración de los componentes creados en la parte de ‘declarations’. La parte de ‘imports’ la iremos explicando a medida que vamos avanzando en la explicación global del cliente.

Para cargar nuestro componente, simplemente añadimos la etiqueta que nos hemos creado a nuestro componente principal, y, posteriormente, cargamos nuestro componente principal a la vista general del proyecto.

Es decir, el código del componente principal será el siguiente:

```
<app-header></app-header>
```

El código de nuestra vista index.html que será la que se cargue, y esta a su vez cargue todo lo demás sucesivamente, será:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>AwsClient</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

En esta vista principal no se cargan aún los scripts de configuración para que funcionen ciertos *frameworks* como Bootstrap o angular material. Se añadirán más tarde. También podemos ver escrito “loading...” dentro del componente. Es simplemente un el texto que se suele escribir para que se visualice algo antes de cargar el componente.

La visualización de la web hasta este punto sería la siguiente:



Figura 66: Header de la aplicación web

Cómo podemos ver, el cliente se ha levantado en local, en el puerto 4200, y sólo se visualizan las dos imágenes que hemos indicado en el componente.

Ahora que ya hemos definido como funcionan los componentes dentro de Angular, y en concreto, como funciona la cabecera de nuestra aplicación web, vamos a pasar a definir el cuerpo de nuestra aplicación, como funciona su lógica y como hemos definido su vista.

6.3.2 COMPONENTE CUERPO

Ahora pasamos a explicar el desarrollo del componente importante de la aplicación, que será el que visualice todos los gráficos y datos que queremos mostrar.

El modo de crearlo y declararlo es similar al componente anterior, por lo que no incidiremos en esa parte. Indagaremos más en la conexión con nuestra lógica Java, en las etiquetas de Angular 2 dentro del código html o en los componentes importados de Angular Material. Intentaremos explicar la vista a medida que vamos avanzando con la lógica.

Comenzaremos por la división principal de nuestra web. Como dicha aplicación debe ser una página de monitorización de varios servicios *cloud*, aunque en este proyecto únicamente nos centremos en AWS, deberá estar preparada para visualizar los datos correspondientes de Google *Cloud* y Microsoft Azure, por lo que debemos dejar la aplicación preparada.

Esta pequeña división la vamos a realizar con una tabla cogida de *Bootstrap* para Angular 2. Lo bueno de utilizar este *framework*, es que te proporciona componentes ya creados visualmente muy atractivos y quedan un toque bastante moderno a la aplicación. Para poder cargar un componente, debemos realizar varios pasos. Lo primero de todo es asegurarnos de

que tenemos cargado el paquete de *Bootstrap* para Angular 2 en la carpeta “node_modules” de nuestro proyecto.

La carpeta “node_modules” es la que contiene todos los paquetes de Node.js para cargar en nuestro proyecto. Desde el inicio, carga los básicos por defecto, pero muchas veces tenemos que añadir alguno, como el paquete para el uso de Angular Material, o mismamente el de *Bootstrap*.

Para cargarlo en la carpeta, debemos ir a la terminal, y situarnos en nuestro proyecto. IntelliJ facilita mucho esta tarea, ya que podemos hacerlo directamente desde el IDE. En el menú, nos vamos a View - Tool Windows - Terminal, y accedemos a la Shell directamente en nuestro proyecto.

Para agregar un paquete a la carpeta ‘node_modules’, debemos hacerlo con npm, que como dijimos anteriormente, es el que gestiona los paquetes de node. La manera de agregarlo es similar a todos. Mostramos un ejemplo, concretamente para añadir el módulo de Angular material.

```
npm install ngx-bootstrap -save
```

Al ejecutar este comando, automáticamente se nos añade a nuestra carpeta de módulos el paquete que hayamos indicado.

Una vez que tengamos añadido el módulo de bootstrap, es necesario cargar en el archivo ‘module.ts’ todos los componentes del *frameworks* que queramos utilizar. En este caso, el componente a incorporar en el apartado ‘imports’ es el ‘TabsModule’, que nos permitirá incorporar a la web la tabla deseada.

Con este componente en concreto debemos tener cuidado, ya que a diferencia de otros, a este debemos ejecutarle el método *forRoot()* para que funcione correctamente.

Como este componente no tiene lógica agregada, no es necesario cargar nada de lógica en el archivo .ts de nuestro componente.

En este momento, si ejecutáramos el proyecto, no visualizaríamos nada de este componente, ya que, para ello, hace falta añadirlo a nuestro componente principal y además, en la vista que se ejecuta debemos cargar algunos scripts para el correcto funcionamiento de los componentes. Son los siguientes:

```
<!--importamos bootstrap-->
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>
  <script
src="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"></script
>
```

Importamos el paquete de bootstrap normal por si en algún momento queremos utilizarlo, así como los scripts para cargar toda la funcionalidad y estilos correspondientes a los paquetes del *framework*.

Cómo hemos indicado anteriormente, no tenemos lógica agregada con este componente por lo que lo único que tendremos que hacer para visualizar la tabla es añadirla a la vista del componente. Una vez tengamos la división principal de la web, nosotros nos centraremos en la pestaña de AWS, que va a ser la que vamos a desarrollar. Esta, a su vez, deberá dividirse en otras sub-pestañas que dividan todos los datos y funcionalidad que se van a monitorizar.

Dichos apartados se han denominado Servicios, Usuarios y *Policies*. Como es evidente, en el apartado de servicios mostraremos las instancias levantadas de cada servicio, así como algún gráfico explicativo, que indique cuantas instancias de cada tipo hay levantadas o cuantas hay en Irlanda o fuera de ella, sin duda datos que son de suma relevancia.

En el apartado de ‘Usuarios’, se mostrarán todos los usuarios, así como un formulario para crear grupos, donde posteriormente el responsable podrá añadir los usuarios que considere oportunos.

Y por último, el apartado de ‘*Policies*’ será dónde se gestione que permisos añadir a los usuarios que se desee. Aparecerá una tabla con los usuarios y otra con todos los permisos.

Evidentemente deberán ser “clickables” por filas para guardar los eventos y la información que se seleccionen y poder generar el PDF correspondiente para mandárselo al responsable y que el asigne los permisos.

Esta sub-división la vamos a realizar en este caso con una tabla de Angular Material. Para incorporar este componente tenemos que seguir exactamente los mismos pasos que para cargar el componente anterior, por lo que estas acciones empezaremos a omitirlas.

Únicamente debemos comentar que para que los componentes de Angular Material funcionen correctamente, con sus estilos y sus animaciones, debemos cargar el componente ‘*BrowserAnimationsModule*’ en el archivo ‘*module.ts*’ y en la vista principal los siguientes scripts:

```
<!--importamos interfaz de angular material-->  
<link href="https://unpkg.com/@angular/material/prebuilt-themes/indigo-pink.css"  
rel="stylesheet">  
<link href="https://fonts.googleapis.com/icon?family=Material+Icons"  
rel="stylesheet">  
<style>body { font-family: Roboto, Arial, sans-serif; }</style>  
<script  
src="https://cdnjs.cloudflare.com/ajax/libs/hammer.js/2.0.8/hammer.min.js"></scri  
pt>
```

El de estilo se podría omitir, es únicamente definido por gustos propios. Los dos primeros cargan funcionalidad para los componentes de Angular Material, mientras que el segundo soporta los eventos de tocar botones o ciertos componentes que cambian de vista o funcionalidad. Hay que decir que no es del todo necesario importarlo, aunque si no lo hacemos, obtendremos un *warning*.

Para ver todos los errores o *warnings* que obtenemos al desarrollar cualquier aplicación web, es muy útil abrir la ventana de inspección con F12 e ir a la consola, para ver lo que realmente está pasando y poder investigar.

Como ya tenemos la vista un tanto enfocada, pasamos a explicar la recogida de datos de toda nuestra lógica. Para conectar con nuestra lógica a través de la API-REST, deberemos definir cierta lógica en nuestro componente que nos ayude a ello.

Lo primero es definir nuestro constructor con una variable del tipo `Http` por defecto, para indicar que se van a hacer peticiones `Http`.

```
constructor(private http: Http) {  
}
```

En el `'module.ts'`, deberemos incorporar el módulo `'HttpModule'` de Angular 2. Para conectar con la API-REST, lo más recomendable es crear un método que se ejecute al pulsar un botón o al recibir un evento cualquiera con toda la lógica. Lo primero, es definir la URI a la que nos queremos conectar. Para dar un valor a una variable local, en TypeScript se realiza con `'let'`. Ejemplo:

```
let url = 'http://localhost:36080/instancesEC2';
```

Aquí estamos indicando que queremos acceder al servicio de instancias EC2 levantado en local, en el Puerto 36080. A los objetos `Http` se les debe pasar la URI correspondiente y mediante el método `subscribe()`, podremos acceder a la respuesta de dicho link con la `response`. Con un ejemplo lo visualizamos mejor:

```
let url = 'http://localhost:36080/instancesEC2';  
this.http.get(url).subscribe(  
  response => {  
    let data = response.json();  
    this.booleanBarraProgreso = 0;  
  }  
)
```

Podemos ver como en la variable `data` introducimos los datos de la respuesta correspondiente. Ya sabemos que están en forma de JSON, pero debemos indicarlo. Ahora, una vez obtenidos los datos a monitorizar, debemos aprender a mostrarlos correctamente.

IMPLEMENTACIÓN

Angular posee una característica muy buena, y es que dota de la capacidad de saber cuándo una variable está cambiando u guardando información, por lo que el paso a seguir es meter en una variable todos los datos obtenidos y leerlos desde la vista.

Para declararnos dicha variable, únicamente tenemos que introducir el siguiente código al inicio del componente, donde se declaran todas las variables necesarias, que es como se crea un 'array' de manera normal.

```
private insEC2: string []= [];
```

Después, lo único que tenemos que hacer es rellenarlo:

```
for (let i = 0; i < data.length; i++) {  
  let instance = data[i];  
  this.insEC2.push(instance);  
}
```

Es importante que en el desarrollo de la lógica, para saber que estamos haciendo o que valores toman ciertas variables, es muy útil el comando '*console.log()*'. Con él, en la consola de la ventana de inspección del navegador, podemos mostrar el valor de las variables que deseemos, y comprobar si todo esta funcionando como deseamos. Para acceder a ella, simplemente pulsamos F12 y Console. Ejemplo.

```
console.log('data');  
console.log(data);
```



Figura 67: Visión de las variables desde la consola del navegador.

Para visualizar el contenido de la variable 'insEC2' desde la vista, se ha decidido hacerlo en una tabla responsiva de bootstrap, cuyas columnas sean los parámetros que queremos monitorizar de los servicios. Para recorrer un 'array' desde la vista, Angular 2 provee de un comando denominado "**ngFor*", que funciona de manera similar al '*for-each*' de Java. Mostramos el código a continuación:

```
<tr *ngFor="let dato of insEC2">
  <td> {{dato.ec2InstanceName}}</td>
  <td> {{dato.ec2InstanceId}}</td>
  <td>{{dato.ec2InstanceLaunchDate}}</td>
  <td>{{dato.ec2InstanceRegion}}</td>
  <td>{{dato.ec2InstanceState}}</td>
  <td>{{dato.ec2InstanceType}}</td>
  <td>{{dato.ec2InstanceOwner}}</td>
  <td>{{dato.ec2InstanceDns}}</td>
  <td>{{dato.ec2InstancePlatform}}</td>
</tr>
```

Dentro de las columnas de la tabla, introducimos la lectura de la variable. A la variable dato le vamos a introducir el objeto JSON correspondiente a la posición en la que se vaya desarrollando el bucle, y, posteriormente, accedemos a los datos que queremos monitorizar, haciendo coincidir las columnas con los valores correspondientes. Para monitorizar el resto de servicios y los usuarios, el procedimiento es el mismo.

Para llamar a los métodos que contienen esta lógica, se han desarrollado tres botones, cada uno correspondiendo a cada tipo de servicio. Angular 2 posee otro comando muy útil denominado (*click*), que se añade a los botones y se indica a la función que se desea acudir. Evidentemente, las tablas al inicio estarán “escondidas”, es decir, sin mostrar, y al pulsar dichos botones, el booleano que hace que no estén visibles, cambiará su valor para hacer justo lo contrario, porque lo que queremos es que las tablas se muestren cada vez que se pulse el botón.

Para hacer que algo de la vista esté oculto, Angular provee de otra etiqueta denominada “hidden” que se iguala a un booleano. Al inicio dicho booleano valdrá 1 para que la tabla no esté visible, pero al inicio del método que se llame cuando se pulse el botón, el valor de dicho booleano cambiará para mostrar la tabla.

Esta manera de trabajar tiene un problema, y es el siguiente. Tal y cómo hemos explicado las cosas, cada vez que queramos monitorizar un servicio, debemos pulsar un botón, este llamará a un método que accederá a una URI que realizará la petición correspondiente a AWS y devolverá los datos.

Como comentamos en el apartado del servidor, al intentar sacar información de los servicios, se recorren todas las regiones en busca de alguna instancia localizada en una región incorrecta. Este proceso lleva tiempo y es lento, por lo que mostrando así los datos tardaremos unos tres minutos cada vez. Hoy en día una página web que tarde 3 minutos en realizar su función es una página web destinada al fracaso, ya que nadie está dispuesto a esperar esa cantidad de tiempo.

Llega el momento de explicar las tablas de la base de datos que no se explicaron anteriormente y si vamos a hacerlo ahora.

Para solucionar este problema, se pensó que en vez de estar pidiendo cada vez los datos a AWS, sería mucho más útil pedir los datos una vez y guardarlos en una base de datos. Varios son los motivos de esta decisión. El primero y más evidente, la velocidad de acceso. Coger los datos de una tabla con una *query* es muchísimo más rápido que esperar a todas las peticiones de AWS.

El segundo es que, no haciéndolo con base de datos, es un gasto de recursos innecesarios, ya que, aunque los servicios *cloud* estén muy de moda y están empezando a ser muy demandados, no se levantan instancias todos los días, por lo que es de esperar que los datos de los servicios no cambien rápidamente

Para ello, se han creado tablas en la base de datos para almacenar la información de cada tipo de servicio. Con los usuarios no se ha hecho ya que los usuarios no se comprueban en cada región. Por tanto, los cambios que hay que realizar en el código back con mínimos. Solo hay que crearse 3 métodos que lean de base de datos, uno para cada servicio. Y los métodos que mandaban la información almacenada en un objeto, guarden dicha información de las peticiones a AWS en la base de datos, por lo que ahora, cada vez que queramos monitorizar los servicios, accederemos a la tabla correspondiente y la respuesta será muy rápida.

Evidentemente, de esta manera, si se crea una instancia y la base de datos no está actualizada, dicha instancia no se va amostrar y tendríamos un problema, por lo que se van a realizar dos cosas. La primera es que la base de datos se va a actualizar por detrás cada cierto tiempo, para que esté prácticamente actualizada en cualquier momento.

Para ello, se hace uso de la clase Observable para TypeScript, que ejecutará el método “Interval(time)” que ejecuta una función cada cierto tiempo que nosotros especifiquemos. El código es el siguiente:


```
Observable.interval(2000 * 60).subscribe(x => {  
  this.refreshS3Table();  
});  
Observable.interval(1000 * 60).subscribe(x => {  
  this.refreshEC2Table();  
});  
Observable.interval(3000 * 60).subscribe(x => {  
  this.refreshCloudSearchTable();  
});
```

Pero, en cualquier caso, también se dará la opción de actualizar la base de datos manualmente desde la web a través de un botón.

Por otro lado, en cuanto a la creación de grupos desde la web, se hará a través de un formulario de Angular Material. Debemos cargar en el 'module.ts' el componente 'FormsModule'.

Para introducir dicho formulario, el código es el siguiente:

```
<table [hidden] = booleanInputGroup class="example-full-width"  
cellspacing="0"><tr>  
  <td><md-input-container class="example-full-width">  
    <input mdInput #groupName placeholder="Nombre del grupo">  
  </md-input-container></td>  
</tr></table>
```

Se le introduce también un booleano para que aparezca cuando se pulse el botón de crear un grupo. El valor del input se quedará guardado en la variable 'groupName' a través del código "#groupName". Posteriormente, para conectar en la API-REST con el método que crea los grupos, creamos un botón que llame a dicho método y acceda a la URI correspondiente.

```
<a href="#" class="btn btn-success">  
  <span class="glyphicon glyphicon-ok"></span> Añadir  
</a>
```

A continuación, pasaremos a explicar la creación de gráficos que se visualizarán en la aplicación. Serán tres. Dos de ellos, circulares, y el otro un gráfico de barras.

Representarán datos que lleven a un entendimiento rápido de lo que se tiene contratado.

El primero de los gráficos circulares representará la cantidad de instancias que tenemos de cada tipo. Como estamos evaluando 3 clases distintas, aparecerán como máximo tres colores diferentes. A día de hoy se tienen instancias de todos los tipos estudiados, por lo que se deduce que siempre aparecerán 3 partes distinguidas en el gráfico. A continuación, mostramos una imagen explicativa:

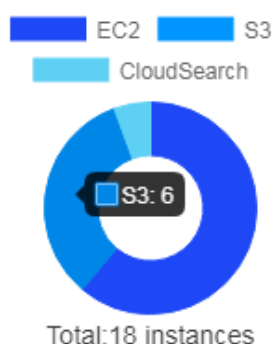


Figura 68: Gráfico de instancias por tipo

El otro gráfico circular representará cuántas instancias hay levantadas en la región adecuada (Irlanda) y cuántas no. De esta manera es fácil saber cuántas máquinas tiene la entidad en localizaciones no válidas, y, así, poder actuar rápidamente. Se rellenará mediante un formulario de entrada en el que se deberán indicar todos los parámetros correspondientes. Una vez completado, pulsando el botón el gráfico se actualizará solo.

Por último, el gráfico de barras representará cuantas instancias de cada tipo tiene cada usuario. Anteriormente, explicamos que, para la entidad, es útil saber que usuarios levantan las instancias. Ya sabemos que desde AWS sólo informan sobre la cuenta general de la entidad en Amazon, pero no informan sobre quién en concreto la levanta. Los datos se rellenarán obteniendo la información de la base de datos creada para obtener este tipo de información. Para introducir más datos cada vez que una instancia se levante, se deberá acceder al formulario que la aplicación proporciona en el botón “añadir instancia”. En el eje horizontal se mostrarán los usuarios actuales de AWS, y encima de ellos aparecerán 3 barras que indicarán la cantidad de instancias de cada tipo de ha levantado cada uno de ellos.

Es importante explicar cómo se han desarrollado cada uno de ellos por detrás. Al trabajar con componentes, tenemos una serie de ventajas que proporciona Angular 2 que, si trabajáramos de otra manera, no tendríamos. Cada uno de ellos, tiene la opción de implantar un código “*OnInit*”. Esta etiqueta hace que el código desarrollado dentro de esta etiqueta sea el que se ejecute primero cada vez que se arranca la aplicación. De esta manera, obtendremos todos los datos de las tablas que tenemos, haremos la lógica necesaria para tener los datos como deseamos, y se los introduciremos al gráfico, para que cada vez que se arranque la aplicación, los gráficos estén actualizados.

Finalmente, daremos paso a la explicación de cómo se ha llevado a cabo la gestión de la asignación de permisos de los usuarios dentro de AWS. Como comentamos anteriormente, dentro de la entidad hay un departamento de seguridad que se encarga, entre otras cosas, de asignar los permisos dentro de AWS. Para el desarrollo del proyecto, se intentó que concedieran los permisos para estas acciones al departamento de desarrollo, pero no pudieron hacerlo, asique esta parte del proyecto se vio comprometida.

Para solucionarlo, se decidió crear un archivo PDF que se envíe automáticamente al responsable del departamento de desarrollo para que lo compruebe, o, directamente, al departamento de seguridad encargado de asignar los permisos. Para ello, en la aplicación, en la pestaña de “*policies*”, se mostrarán dos tablas. Una de ellas contendrá todos los usuarios de la cuenta en AWS, y, la otra, una tabla con todos los permisos asignables. Ambas serán “*clickables*” para poder seleccionar a que usuarios deseamos asignarles permisos. Una vez que se tenga todo lo que se desea claro y pulsado, pulsando un botón llamado “*Generate PDF*”, se creará automáticamente el archivo pdf correspondiente en el que se indican claramente los permisos que se desea que sean asignados a los usuarios correspondientes.

Hasta este punto, hemos comentado todo el desarrollo funcional de nuestra parte *front-end*, pero aún queda un punto del desarrollo bastante importante por explicar. Se trata del tema de la seguridad web.

CORS (Cross Origin Resource Sharing) es el término que se emplea para explicar lo que sucede cuando accedemos a una página web, que intenta solicitar datos que no pertenecen a

su propio dominio. Es el caso de esta aplicación. Se ha desarrollado una web particular que accede a AWS y obtiene los datos de él. Al tratarse de dominios diferentes, surge un problema de seguridad en el navegador.

Para solucionar este problema, al estar la aplicación desarrollada con Spring, deberá hacerse con este *framework*, y para ello, se ha desarrollado una clase que solventa estos problemas para las URL, haciendo que se permita el acceso a otros dominios diferentes.

A continuación, mostramos un diagrama de actividad para mostrar lo que ocurre cuando se hace una petición a AWS.

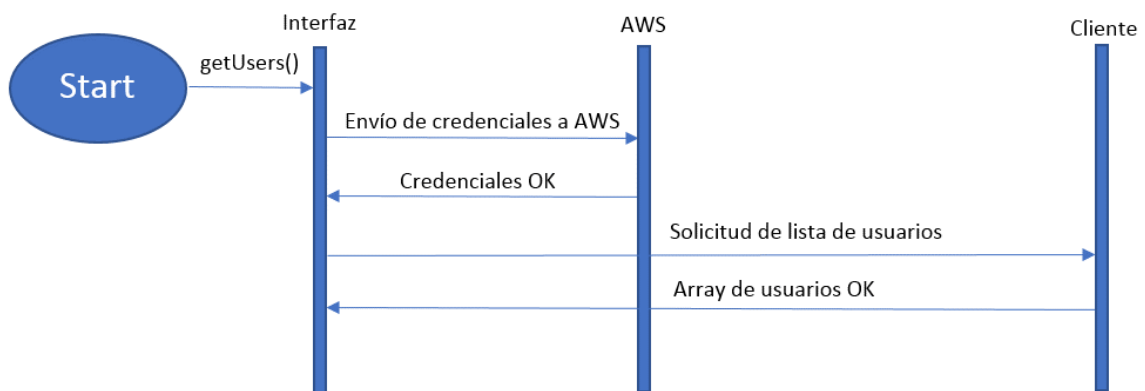


Figura 69: Diagrama de actividad con petición a AWS.

Capítulo 7. ANÁLISIS DE RESULTADOS

En este apartado se destacan los resultados más importantes y relevantes del proyecto, entre los que se encuentran el buen funcionamiento de la misma, los casos de uso para los que ha sido creada y alguna funcionalidad añadida.

También se visualizarán las alertas desarrolladas ante situaciones no deseadas, comprobando que efectivamente, se notifican de manera correcta.

Recordamos los objetivos principales de la aplicación. Dentro de todo lo relacionado con usuarios, se debe cumplir con la siguiente funcionalidad:

- Mostrar los usuarios dados de alta en AWS dentro de la cuenta de la entidad.
- Crear grupos de usuarios.
- Asignar los diferentes permisos disponibles a cualquier usuario.

En relación a los productos de AWS:

- Se deben mostrar todos los productos contratados en cualquier región disponible.
- Se deben mostrar también que instancias ha levantado cada usuario.

En cuanto a las alarmas:

- Se debe generar una alarma ante funcionalidad no deseada, como advertir de la localización incorrecta de una instancia, o de la contratación de una instancia EC2 con un sistema operativo más sofisticado de lo normal, como Red Hat.

Es importante comentar, que la aplicación no se implantará en ningún servidor hasta que esté terminada. En este proyecto se ha tratado la monitorización de los servicios de AWS, pero aún quedan por monitorizar otros servicios *cloud*, por lo que todas las pruebas se desarrollarán en local. En cualquier caso, es bastante probable que la aplicación acabe corriendo en una instancia propia de Amazon, como una EC2 o una S3 (las S3, dependiendo

del tipo que sean, a veces dan opción de almacenar desarrollos software para levantar aplicaciones). A continuación, incluiremos los resultados finales de la aplicación.

Antes de comentar objetivo a objetivo, es importante mencionar que la aplicación general tiene una interfaz muy amigable dentro de unos parámetros muy comunes en la modernidad de las aplicaciones web, gracias al uso de *frameworks* muy utilizados y que hacen que la aplicación sea propia de esta época.

En primer lugar, se ha conseguido mostrar información muy útil de manera rápida mediante los gráficos iniciales en la parte superior. En el primero de ellos, se pueden observar el número de instancias que hay de cada tipo, diferenciados por colores con una leyenda encima del mismo. El gráfico situado en la parte central nos informa de las instancias que ha levantado cada usuario, algo realmente útil también, ya que AWS no devuelve esta información, y por último, podemos observar un gráfico que indica cuántas instancias hay en Irlanda y fuera de ella. Este aspecto resulta de gran ayuda, ya que para la entidad es importante tener controladas las regiones donde tienen instancias levantadas. Por todo esto, los gráficos suponen una parte importante de la aplicación, mostrando información de forma rápida e intuitiva.

En segundo lugar, es necesario explicar la parte de servicios. Como se indica en los objetivos, es necesario mostrar todas las instancias levantadas por el banco en cualquier región disponible. Esta tarea se consigue satisfactoriamente ya que podemos mostrar distintas tablas separadas por tipo de instancia que muestran al usuario esta información. Como se ha comentado, es necesario mostrar instancias en cualquier región. Para ello, se necesita recorrer todas ellas haciendo peticiones a la API, lo que supone un tiempo de respuesta muy elevado, y, por tanto, no se podía gestionar de esta manera dicha tarea. Se optó por crear una base de datos donde se almacenará la información necesaria y leer de ella, ya que este tipo de acceso es muchísimo más rápido, y daría a la aplicación mayor velocidad.

Como es evidente, si hay algún cambio en las instancias, o se crean o borran algunas, debe notificarse de alguna manera. Para ello, es necesario actualizar la base de datos para que tenga los datos lo más actualizados posibles. Se ha dado la posibilidad de realizar esta tarea

de dos maneras diferentes, manual o automáticamente. A la derecha de cada título de las tablas, se encuentra un botón de actualizar la tabla de la base de datos perteneciente a dicho tipo de instancias. Al pulsarlo, se realizan las peticiones correspondientes a AWS para que nos devuelva la información actualizada y poder mostrarla en la aplicación. Esta es la manera manual. La manera automática elimina la anterior opción, haciendo que los botones estén desactivados. Esta forma de trabajar se puede activar pulsando un botón situado en la parte superior derecha, que activa la actualización automática cada dos minutos. Para saber si la base de datos se está actualizando o no, hay un icono de carga al lado de cada tabla que aparece cuando se está recargando. Cuando hay algún cambio, los gráficos circulares se actualizan automáticamente si necesidad de recargar la página. El gráfico central muestra información que no proporciona AWS, por lo que se debe gestionar de manera manual. Para hacerlo, se debe rellenar un formulario de entrada indicando los parámetros correspondientes para añadir una instancia a la tabla de la base de datos que da vida al gráfico y que lo actualiza correctamente. Por todas estas razones, podemos concluir que la muestra de instancias se realiza de manera más que satisfactoria.

Por otro lado, hay que decir que la gestión de situaciones no deseadas se realiza en cada actualización de la base de datos. Actualmente son dos, aunque en un futuro puede que se aumente el número de ellas. Que una instancia esté fuera de Irlanda, o que se levante con un sistema operativo como Red Hat son las tratadas en este proyecto. Cada vez que se pide la información a AWS, se comprueban estos hechos, y se evalúan. Si realmente se producen, se le notifica al responsable mediante correo el aspecto que haya sucedido y, por tanto, está al tanto de que hay funcionalidad no deseada que está ocurriendo. Una vez notificado, es problema suyo tomar medidas o no. Por ello, podemos concluir que la gestión de alarmas también se realiza de manera satisfactoria.

En cuanto al *Billing*, hay que decir que no ocurre lo mismo. Desde AWS, proveen un *dashboard* genérico que informa sobre los costes generales que tiene la cuenta de la entidad. Dichos costes están separados por tipo de servicio (EC2, S3, etc...), pero no informan sobre los costes concretos de cada instancia en particular, algo que es realmente útil para llevar un mejor control de los costes de la cuenta y que era lo que se quería realizar en este apartado.

Una de las formas de hacerlo, es analizar reportes de facturación que Amazon te proporciona en un *bucket* S3 si se lo indicas. El problema que se tuvo es que esta tarea no corresponde al departamento de desarrollo, y no nos pudieron ceder los permisos para pedir dichos reportes por lo que no se pudieron analizar. La API de AWS tampoco concede métodos para saber dichos costes por lo que las opciones se agotaron. Debido a estos problemas, no se pudo realizar de como deseábamos este aspecto. Por ello, se optó por poner un enlace directo al *dashboard* oficial para tener una idea general de los costes, pero sin llegar a la funcionalidad querida. Con esta explicación la parte de servicios queda cerrada.

Por otro lado, se deben comentar los resultados respecto a los usuarios y los permisos, que van bastante relacionados. El primero de los objetivos referentes a esta parte del proyecto era mostrar los usuarios actuales en AWS. De igual manera, somos capaces de mostrar una tabla con todos ellos y los atributos más relevantes. Hay que decir que esta información no se lee de base de datos ya que la respuesta sobre estos datos es mucho más rápida que la respuesta sobre las instancias. También damos la posibilidad de recargar la tabla mediante un botón de actualizar, de igual modo que con las instancias.

Dentro de AWS, se pueden crear grupos para asignar posteriormente usuarios. Esta tarea es simplemente para separar a los usuarios por algún tipo de funcionalidad o por jerarquía o por lo que se desee. No podemos asignar usuarios a grupos por falta de permisos, pero si podemos crear los grupos, que era el objetivo inicial. Para ello, simplemente se rellena un formulario de entrada indicando el nombre del mismo. Hasta aquí, todo se realiza de manera correcta. A continuación, explicamos la gestión de los permisos.

Desde la aplicación, se deben poder asignar permisos a los usuarios. Hay muchos tipos de permisos dentro de AWS. De lectura, de escritura, *full access*, son algunos de ellos. El responsable de los equipos es el que decide que permisos asignar y a quien, por lo que esta tarea formaba parte de los objetivos fundamentales del proyecto.

De nuevo, nos encontramos ante la misma problemática que anteriormente. Dentro del banco, el departamento de desarrollo no dispone de los permisos suficientes para asignar permisos a usuarios, ya que esta tarea corresponde a otro departamento. Ante la negativa

sobre poder obtener los permisos, se optó por una solución intermedia e ingeniosa, la creación de un PDF dinámico que indique a que usuarios hay que asignarles permisos de AWS. Obviamente, estos usuarios deben pertenecer a la cuenta de AWS. Desde la aplicación se muestran dos tablas, que indican los usuarios y las *policies* disponibles respectivamente. Simplemente seleccionando los usuarios y los permisos que se desean, se genera un PDF con esta información que se envía directamente al departamento correspondiente, evitando tener que estar generando correos y contactos con el departamento cada vez que se quiera realizar esta tarea. De esta manera, es bastante más sencillo y automático, por lo que podemos concluir que, aunque el objetivo no se realice de manera directa, se ha desarrollado una solución más que suficiente para satisfacer el objetivo inicial.

Para terminar, a modo de conclusión, podemos decir que ha quedado una aplicación bastante completa y con margen de mejora al mismo tiempo. En este proyecto se ha tratado la monitorización de AWS, y se han cumplido todos los objetivos que se han podido realizar. En cuanto a los objetivos no desarrollados, habrá que esperar a tener los permisos dentro del banco para poder acceder a la información necesaria y, en ese momento, gestionarlos. Por otro lado, respecto al margen de mejora, nos referimos a la monitorización del resto de servicios *cloud* que tenga el banco contratados, para poder juntarlos a todos ellos y tener un control más adecuado, que, como ya hemos comentado, dicha tarea no forma parte de este proyecto. La aplicación queda de base para el futuro análisis de los mismos, y que la entidad pueda tener gestionarlos mejor y más fácilmente.

Capítulo 8. CONCLUSIONES Y TRABAJOS FUTUROS

Tras finalizar el período de prácticas en la entidad, después de 6 meses, se ha obtenido una aplicación web que monitoriza los servicios contratados por el banco a AWS, y estructurada para monitorizar el resto de servicios *cloud* que el banco tenga.

La aplicación es capaz de pedir la información correspondiente a AWS a través de su API y obtener los datos para ser posteriormente mostrados en un interfaz gráfico. En relación a la planificación, es importante decir que se ha seguido con bastante precisión los plazos decididos al inicio del mismo.

La implantación de la web se ha realizado en un solo equipo de la manera planificada y la monitorización de la información correspondiente, explicada en los objetivos, ha funcionado correctamente como se puede ver en el apartado de los resultados, cubriendo así el objetivo principal del proyecto.

Por otro lado, es importante comentar que el sistema de alertas, también explicado en el capítulo anterior, funciona correctamente. Este aspecto es muy positivo, ya que de un sistema de monitorización se espera que notifique las situaciones no deseadas de manera correcta y eficaz.

La aplicación ha cumplido con los objetivos, pudiendo mostrar los usuarios dados de alta en AWS. Evidentemente los usuarios que se vayan dando de alta también se mostrarán. Se podrán crear grupos, para la futura agregación de usuarios. Dicha agregación podrá realizarse en función de los permisos, de la jerarquía dentro de la empresa, o según cualquier otro aspecto significativo para el responsable de la cuenta. Un ejemplo de acción futura al visualizar los usuarios de AWS, es eliminar aquellos que no tengan ninguna función dentro de los servicios *cloud*. La aplicación es capaz de mostrar la última fecha de conexión del perfil. Si a un usuario le aparece como “Nunca conectado”, teniendo en cuenta que la entidad lleva con estos servicios un año, es bastante probable que dicho usuario no tenga ninguna relación con AWS, aunque como siempre, este aspecto debe ser comprobado por el responsable y como tal, la decisión debe ser suya.

La aplicación también es capaz de mostrar todas las instancias levantadas, junto con todas sus características correspondientes. Gracias a los gráficos, se puede obtener rápidamente una conclusión sobre los datos de la cuenta en AWS en el momento de ejecutar la aplicación.

En definitiva, se ha desarrollado una aplicación bastante completa y que, de alguna manera, inicia el camino de desarrollo para la monitorización de otros servicios de computación en la nube contratados en el banco. De esta manera, la entidad podrá disponer de una aplicación donde pueda visualizar todos los datos que requiera de todos sus servicios juntos, y, en función de lo que se muestre, tomar las decisiones correspondientes. Además, la gestión de dichos servicios será mucho más sencilla.

8.1 TRABAJOS FUTUROS

En este apartado se describen brevemente algunas futuras modificaciones que podrían mejorar la implementación actual de la web añadiendo nuevas funcionalidades o características o para solucionar ciertas debilidades, o para mejorar el propio funcionamiento de la aplicación.

8.1.1 BILLING

En primer lugar, hay que decir que, en un sistema de monitorización, siempre es útil mostrar los costes que tiene la entidad al contratar los servicios de AWS. Para ello, dentro del banco, es necesario que nos asignen los permisos correspondientes para acceder a esa clase de información. Actualmente, AWS no contiene métodos en su API para poder hacer peticiones sobre lo que cuesta tener contratada una instancia en su empresa. Sin embargo, tiene una opción de poder almacenar en un *bucket* S3 todos los reportes de costes de las instancias por tipo de acción. Si se pudiera tener acceso a dichos archivos (asignando los permisos al perfil del desarrollador), se podrían analizar y obtener la información que se desea para monitorizarla con algunos gráficos que representen los costes globales de tener contratados estos servicios.

Por esta razón, no se puede llevar a cabo un estudio detallado de los costes que está teniendo la cuenta, analizándolo por tipo de instancia, región, sistema operativo, etc...

Por ello, simplemente se ha puesto un hipervínculo al *dashboard* general que proporciona AWS, que no muestra la información que deseamos, pero sí da una idea general de lo que se está consumiendo

8.1.2 MAYOR NÚMERO DE COMPONENTES

En cuanto a la estructura del cliente Angular 2, explicado en el capítulo de desarrollo, es posible que una división en mayor número de componentes hiciera que el código estuviera más limpio y que, por tanto, la estructura del cliente fuera más clara.

Actualmente, se tienen dos componentes, que son el “*header*” y el “*body*”, que pueden representar la estructura de la web a alto nivel. Como el *header* solo contiene dos fotos, no puede estar dividido en un mayor número de componentes porque sería una medida bastante absurda, sin embargo, el *body*, sí que podría. Este último, está dividido en 3 pestañas principales. Ciertamente es que dos de ellas (Google y Microsoft) no están desarrolladas aún, pero cada tipo de servicio, deberá ser un componente distinto. En este caso, el desarrollo de la lógica sería muy parecido al desarrollado con AWS, pero cambiando las peticiones y la forma de trabajar con la API. Con AWS se deben importar las librerías y utilizar sus clases propias, pero con Google, por ejemplo, también se da la posibilidad de acceder a la información mediante peticiones Http directamente, en lugar de tener que desarrollar el código para obtener la misma respuesta. En cualquier caso, el desarrollo sería un trabajo parecido. Así mismo, dentro de AWS, se deben juntar las sub-pestañas creadas en clientes diferentes, es decir, que la vista de usuarios, instancias y permisos, sean componentes aislados e independientes. De esta manera, conseguiremos que la estructura este más definida y clara.

Capítulo 9. BIBLIOGRAFÍA

- [1] Wikipedia, Maven. <https://es.wikipedia.org/wiki/Git>
- [2] Alberto Básalo. Evolución de la plataforma. <http://academia-binaria.com/Angular-2-primeras-impressiones/>
- [3] Wikipedia, Git. <https://es.wikipedia.org/wiki/Git>
- [4] Carey WodeHouse. “GitLab vs GitHub : How Are They different?”
<https://www.upwork.com/hiring/development/gitlab-vs-github-how-are-they-different/>
- [5] Wikipedia, Opex. <https://es.wikipedia.org/wiki/Opex>
- [6] Página oficial de AWS, CloudSearch. <https://aws.Amazon.com/es/cloudsearch/>
- [7] Wikipedia, Scrum. [https://es.wikipedia.org/wiki/Scrum_\(desarrollo_de_software\)](https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software))
- [8] Wikipedia, identificador de recursos uniformes (URI)
https://es.wikipedia.org/wiki/Identificador_de_recursos_uniforme
- [9] Página oficial de BBVA informativa sobre recursos.
<https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos>
- [10] Página oficial de AWS, S3. <https://aws.amazon.com/es/s3/>
- [11] Página oficial de AWS, EC2. <https://aws.amazon.com/es/ec2/>
- [12] Página oficial de AWS, Instance Types. <https://aws.amazon.com/es/ec2/instance-types/>
- [13] Wikipedia, SMTP. https://es.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol
- [14] Página oficial de AWS, IAM. <https://aws.amazon.com/es/iam/>
- [15] Jeisson Andrés García Rodríguez, Noviembre, 2014. “¿Qué es JPA?”,
<http://oraclejuniors.blogspot.com.es/2014/11/que-es-jpa-java-persistence-api.html>
- [16] Eduard Tomàs, Abril, 2014. Qué es REST. <https://desarrolloweb.com/articulos/que-es-rest-caracteristicas-sistemas.html>
- [17] Darío Blasco, Abril, 2016. “Ventajas y desventajas de programar en *Angular JS*”,
https://www.atraura.com/Angular_JS/
- [18] IBM Developers Works. “¿Simplemente qué es Node.js?”
<https://www.ibm.com/developerworks/ssa/opensource/library/os-Node.js/index.html>

ANEXO A: GUÍA DE INSTALACIÓN

En este Anexo se tratará de explicar todo lo referente a la instalación de las tecnologías necesarias para un correcto uso del proyecto desarrollado.

En primer lugar, es necesario tener instalado Java, ya que toda la lógica que se ha desarrollado está hecha con este lenguaje de programación.

1.1 INSTALACIÓN DE JAVA

El primer paso para la instalación de Java es descargarse el *Java Development Kit* (JDK) del siguiente enlace. <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>.

Java SE Development Kit 8u131
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement Decline License Agreement

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.87 MB	jdk-8u131-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.81 MB	jdk-8u131-linux-arm64-vfp-hflt.tar.gz
Linux x86	164.66 MB	jdk-8u131-linux-i586.rpm
Linux x86	179.39 MB	jdk-8u131-linux-i586.tar.gz
Linux x64	162.11 MB	jdk-8u131-linux-x64.rpm
Linux x64	176.95 MB	jdk-8u131-linux-x64.tar.gz
Mac OS X	226.57 MB	jdk-8u131-macosx-x64.dmg
Solaris SPARC 64-bit	139.79 MB	jdk-8u131-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.13 MB	jdk-8u131-solaris-sparcv9.tar.gz
Solaris x64	140.51 MB	jdk-8u131-solaris-x64.tar.Z
Solaris x64	96.96 MB	jdk-8u131-solaris-x64.tar.gz
Windows x86	191.22 MB	jdk-8u131-windows-i586.exe
Windows x64	198.03 MB	jdk-8u131-windows-x64.exe

Java SE Development Kit 8u131 Demos and Samples Downloads
You must accept the Oracle BSD License. to download this software.

Accept License Agreement Decline License Agreement

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	9.94 MB	jdk-8u131-linux-arm32-vfp-hflt-demos.tar.gz
Linux ARM 64 Hard Float ABI	9.97 MB	jdk-8u131-linux-arm64-vfp-hflt-demos.tar.gz
Linux x86	52.66 MB	jdk-8u131-linux-i586-demos.rpm
Linux x86	52.51 MB	jdk-8u131-linux-i586-demos.tar.gz
Linux x64	52.72 MB	jdk-8u131-linux-x64-demos.rpm
Linux x64	52.58 MB	jdk-8u131-linux-x64-demos.tar.gz
Mac OS X	53.09 MB	jdk-8u131-macosx-x86_64-demos.zip
Solaris SPARC 64-bit	13.54 MB	jdk-8u131-solaris-sparcv9-demos.tar.Z
Solaris SPARC 64-bit	9.33 MB	jdk-8u131-solaris-sparcv9-demos.tar.gz
Solaris x64	13.59 MB	jdk-8u131-solaris-x64-demos.tar.Z
Solaris x64	9.29 MB	jdk-8u131-solaris-x64-demos.tar.gz
Windows x86	53.8 MB	jdk-8u131-windows-i586-demos.zip
Windows x64	53.82 MB	jdk-8u131-windows-x64-demos.zip

Figura 1: Versiones del JDK

Elegimos el correspondiente JDK en función de nuestro sistema operativo. Al instalar el JDK, Java se instalará de forma automática y no hará falta hacerlo manualmente.

Una vez instalado el JDK, lo más probable es que si se ha seguido la configuración por defecto, el JDK se haya instalado en el disco donde esté almacenado el sistema operativo, en la carpeta “Program Files/Java”. Lo que se debe hacer ahora es modificar la variable de entorno Path para que el sistema sepa que tiene instalado Java.

Para ello, lo primero que debemos hacer es crearnos una variable de entorno llamada “JAVA_HOME” que va a tener la ruta donde está almacenado el JDK que nos acabamos de descargar.

Para ello, vamos a: Inicio – Panel de control – Sistema – configuración avanzada del sistema – Variables de entorno. Aparecerá la siguiente ventana.

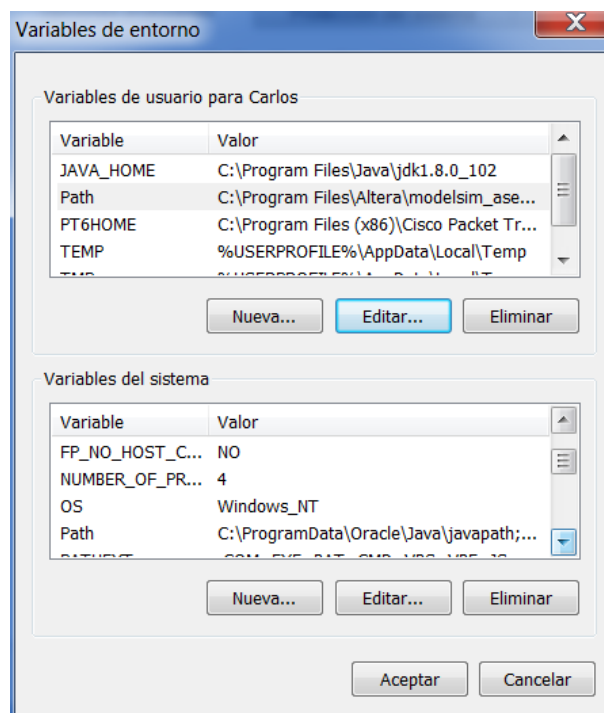


Figura 2: Variables de entorno

Aquí, podemos ver dos tipos de variables de entorno, las del usuario y las del sistema.

Dentro de las del usuario, damos a Nueva, y escribimos lo siguiente:

Nombre de la variable: JAVA_HOME.

valor de la variable: Debemos indicar la ruta del JDK, que en nuestro caso es: C:\Program Files\Java\jdk1.8.0_102.

Una vez hecho esto, accedemos a la variable de entorno Path, y añadimos a su contenido actual la siguiente cadena: “;%JAVA_HOME%\bin”, de manera que el resultado debe ser el siguiente:

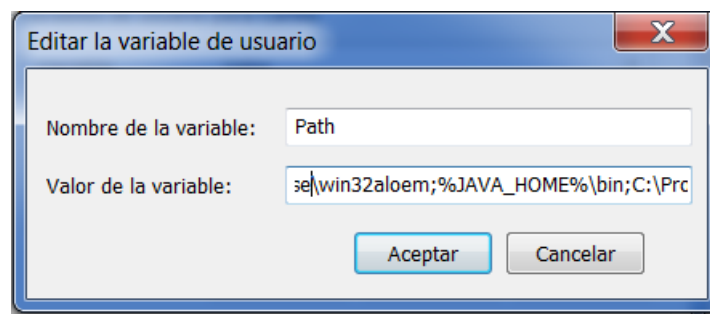


Figura 3: Variable de entorno “Path”

De esta manera queda instalado Java.

Como el proyecto no está monitorizando todos los servicios *cloud*, y, por tanto, no está al completo de su funcionalidad, no se va a almacenar en ninguna instancia de AWS ni en ningún servidor, por lo que, hasta el momento, se va a seguir desarrollando todo en local. Para ello, hace falta un entorno de desarrollo. Como ya hemos comentado anteriormente que hemos usado el IntelliJ, propiedad de JetBrains, será su proceso de instalación el que explicaremos a continuación.

1.2 INSTALACIÓN DEL ENTORNO DE DESARROLLO INTELLIJ

En primer lugar, deberemos acceder a la página del fabricante, que es la siguiente: <https://www.jetbrains.com/>.

Después, deberemos acceder al apartado de IDEs y seleccionar el IntelliJ Idea, ideado para desarrollar en código Java.

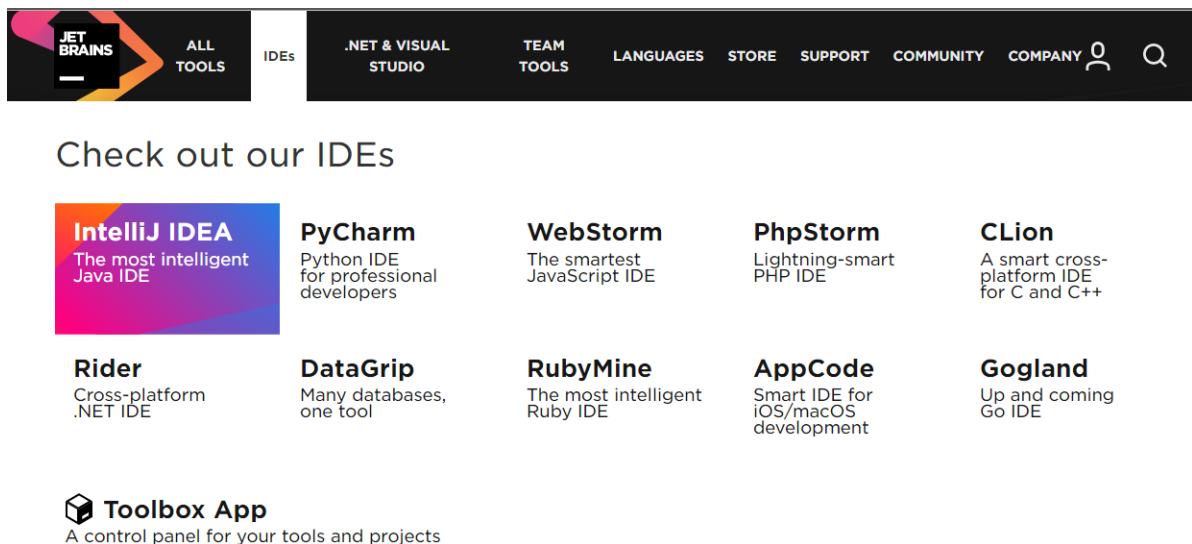


Figura 4: Página principal de JetBrains.

Este entorno de desarrollo posee dos versiones, la versión estándar, y la versión *ultimate*.

Esta versión mejorada está disponible para estudiante, por lo que nosotros podremos disfrutar de este entorno en su completo esplendor. La diferencia con la versión estándar es que permite desarrollos en lenguaje TypeScript, lo que ha resultado fundamental para el desarrollo del proyecto. Por tanto, nos descargaremos la versión premium.

Una vez en la ventana de IntelliJ, pulsaremos el botón “Download”, que nos llevará a la siguiente ventana.



Figura 5: Versiones del IntelliJ

Pulsamos el botón de descargar de la versión ultimate. Una vez descargado el archivo ejecutable, lo pulsamos, y comenzará la instalación.

En la primera ventana, pulsamos Next...



Figura 6: Setup del IntelliJ

Aquí tenemos la opción de directorio, pero dejaremos el que está por defecto.

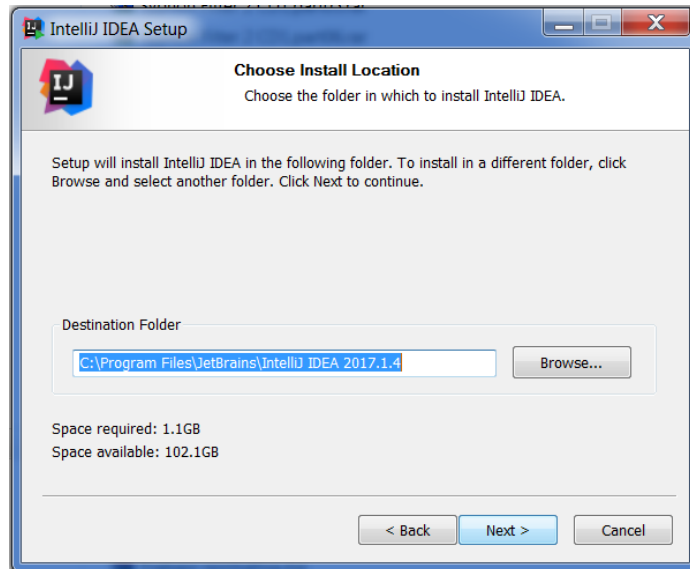


Figura 7: Localización de la carpeta de instalación.

En nuestro caso, únicamente crearemos un icono en el escritorio para abrir el programa. Como nuestro sistema operativo es de 64 bit, lo creamos de 64. No lo referenciamos a ningún tipo de archivo.

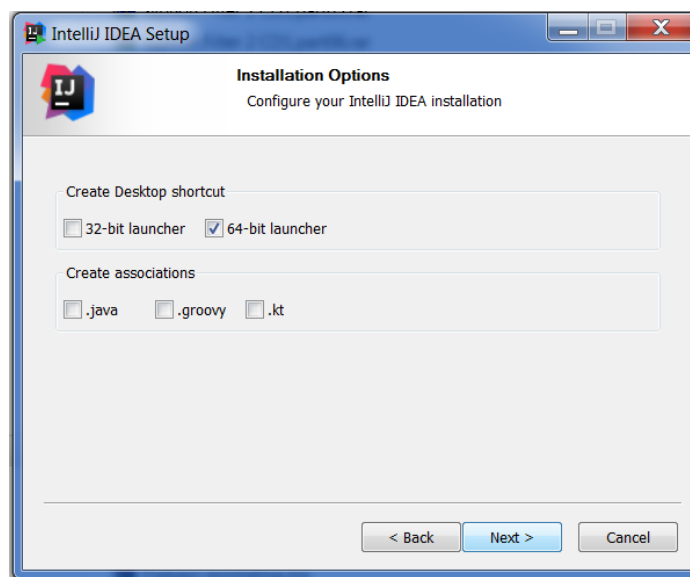


Figura 8: Opciones de la instalación

Introduciremos los datos en la carpeta JetBrains. Aceptamos la instalación en dicha carpeta y dará comienzo la instalación.

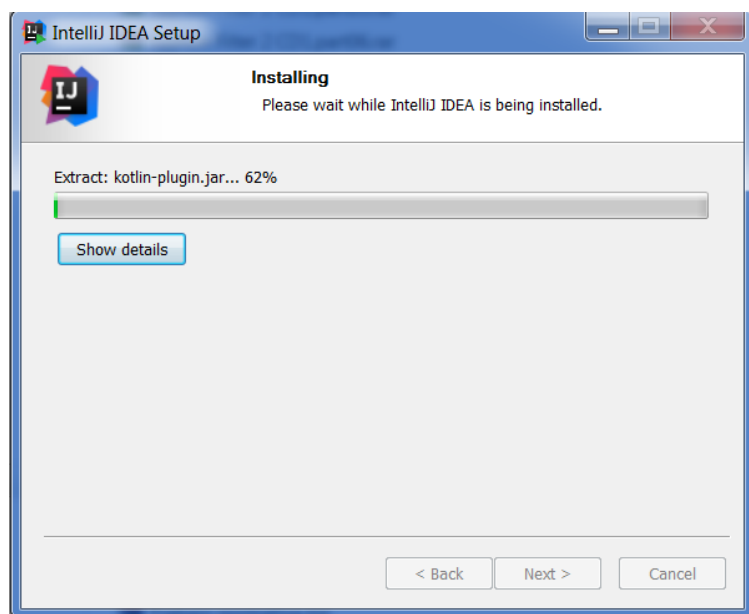


Figura 9: Instalación de IntelliJ

Una vez instalado el programa, la primera vez que lo abramos, aparecerá la siguiente ventana. En nuestro caso, como no hemos tenido nunca otra versión instalada, no necesitaremos importar nada.

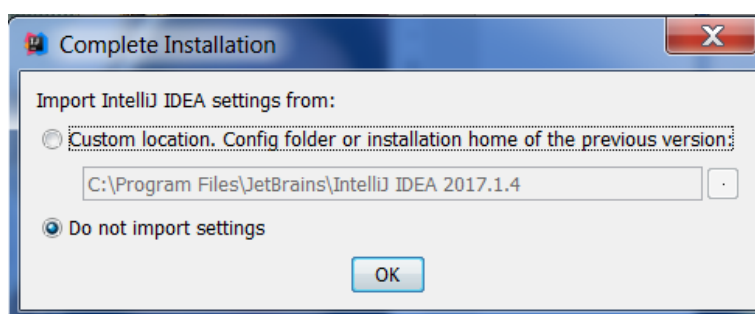


Figura 10: Asistente de instalación

Posteriormente, aparecerá una ventana que nos preguntará si tenemos licencia o código de activación o algún otro recurso para poder acceder a la versión *ultimate*. Como somos estudiante, es nuestro caso.

Para descargarnos una licencia, debemos ir al apartado de nuestro perfil en la página oficial de JetBrains, en el botón que se muestra a continuación.

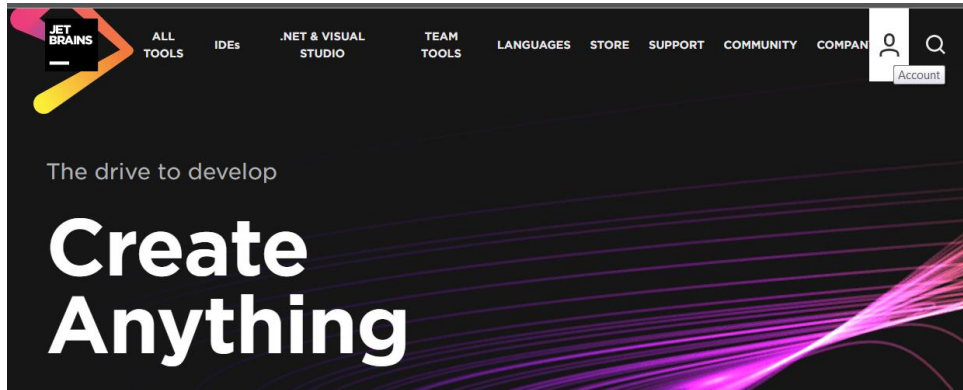


Figura 11: Acceso al perfil personal en JetBrains

Una vez aquí, deberemos ingresar con nuestro nombre de cuenta o nuestro correo de la universidad, y la clave que hayamos puesto. Evidentemente, anteriormente se debe crear una cuenta en JetBrains asociada a nuestro correo de la universidad. Entramos.

Welcome to JetBrains Account



Access your purchases
and view your order history



Identify expired and outdated licenses,
order new licenses and upgrades



Manage your company licenses
and distribute them to end users

Sign in with existing account

Sign In

[Forgot password?](#)

Not registered yet?

Create JetBrains Account

Sign Up

Figura 12: Introduciendo datos del perfil.

Una vez en nuestro perfil, aparecerá la imagen siguiente:

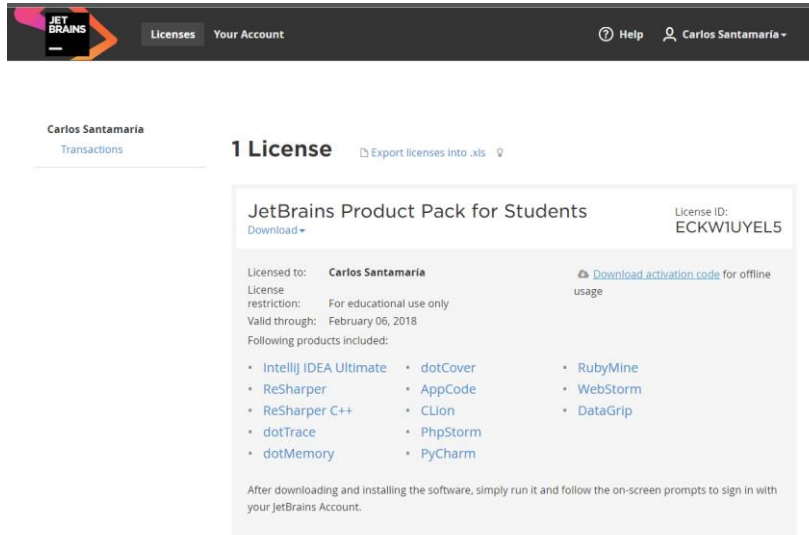


Figura 13: Opciones del perfil de estudiante en JetBrains.

Deberemos pulsar en “Download activation code” para descargar la licencia. Será un archivo .txt con la información correspondiente. Una vez tengamos este archivo descargado y listo para usar, deberemos arrastrarlo hasta la ventana de activación del IntelliJ y lo detectará automáticamente. Previamente debemos pulsar la opción de Activation Code.

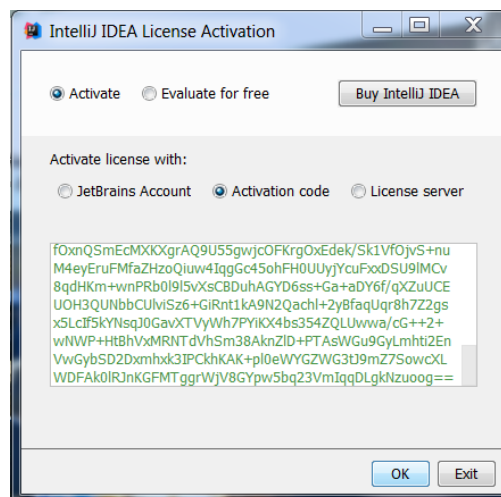


Figura 14: Activación del código para la versión ultimate.

Una vez realizado este paso, ya hemos indicado que somos estudiantes y que podemos optar a la versión premium, y por tanto, ya estamos listos para usar IntelliJ,



Figura 15: Abriendo IntelliJ

Como ver, al iniciarlo aparece a quién esta concedida la licencia de este programa. Evidentemente, en este caso es al desarrollador de este proyecto. Hay ciertas tecnologías que se deberían instalar aparte, como por ejemplo Maven, pero al instalar IntelliJ, quedan instaladas automáticamente.

1.3 NODE.JS Y NPM

Node.js y npm son las tecnologías que se deben instalar para poder realizar proyectos Angular 2. Node es el entorno de ejecución para JavaScript. Los proyectos Angular 2 son desarrollados en TypeScript, que es igual que JavaScript pero tipado, por tanto, todo el código que desarrollemos en este lenguaje, pasará por el compilador Node.js.

Npm por otro lado, es el ecosistema de paquetes que almacena todas las librerías necesarias para poder hacer un buen desarrollo en Angular 2. Por tanto, deberemos instalar ambas, aunque con instalar Node.js, npm se instala automáticamente.

El primer paso, es acceder a la página oficial de Node.js. Es el siguiente enlace:

<https://node.js.org/es/>.



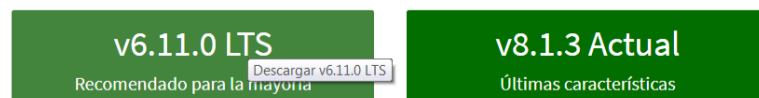
Node.js® es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Node.js usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. El ecosistema de paquetes de Node.js, npm, es el ecosistema mas grande de librerías de código abierto en el mundo.

Figura 16: Página principal de Node.js

Una vez en la web, descargaremos el paquete de Node para la mayoría, es decir, la versión latest (LTS).

Node.js® es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Node.js usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. El ecosistema de paquetes de Node.js, npm, es el ecosistema mas grande de librerías de código abierto en el mundo.

Descargar para Windows (x64)



Otras Descargas | Cambios | Documentación del API Otras Descargas | Cambios | Documentación del API

Ó revise la [Agenda de LTS](#).

Figura 17: Última versión de Node para descargar.

Se descargará un archivo de tipo msi, que deberemos ejecutar para comenzar la instalación. Lo primero que aparecerá será el asistente de configuración, que deberemos aceptar.

Aparecerá la siguiente ventana. Pulsaremos el botón “Next” para ir al siguiente paso de la instalación.



Figura 18: Comienzo de la instalación de Node.js

Deberemos indicar la carpeta donde queremos que se instale. En este caso, se dejará la carpeta por defecto.

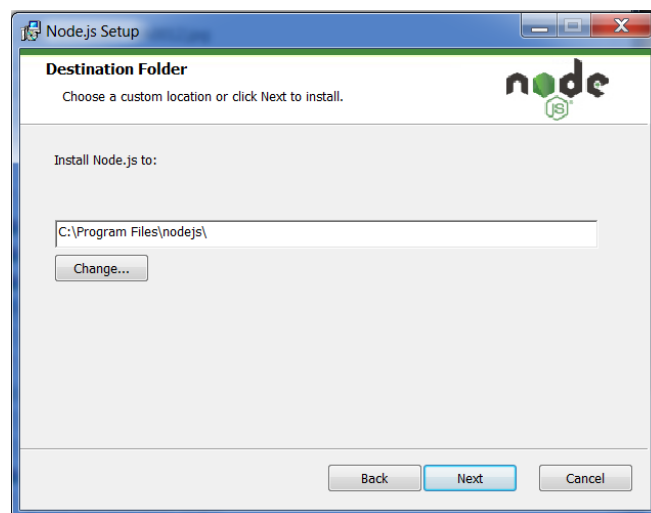


Figura 19: Carpeta de instalación

Aceptamos la siguiente configuración, y pasamos al siguiente punto de la instalación, que será la instalación el paquete propiamente dicha.

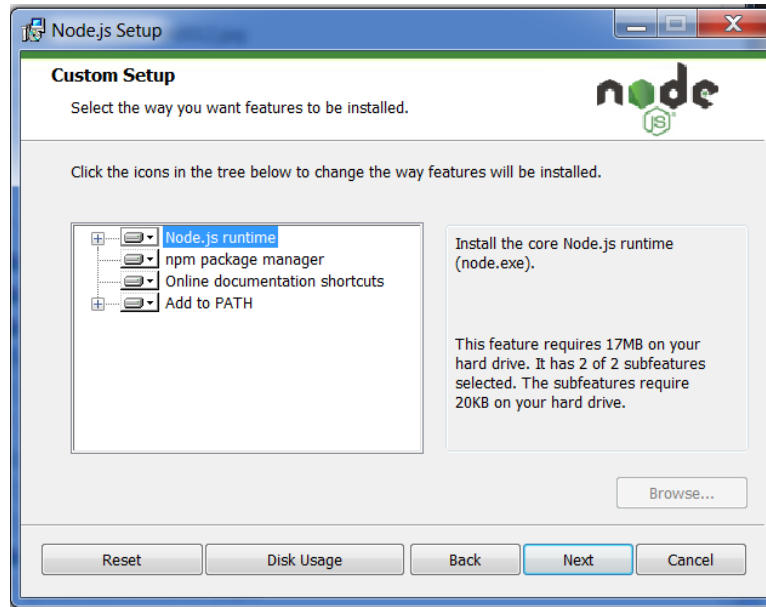


Figura 20: Opciones de instalación de Node.js

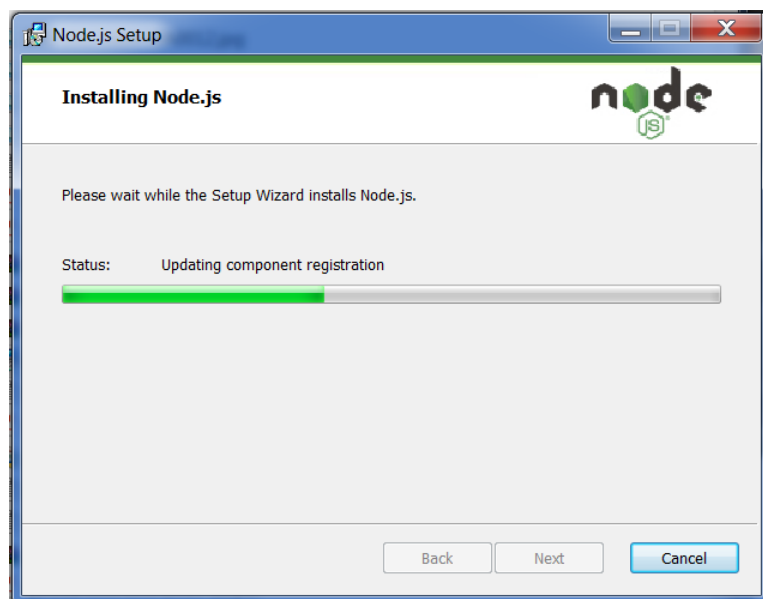


Figura 21: Instalando Node.js

Una vez instalado Node.js y npm, ya disponemos de todas las herramientas necesarias para poder ejecutar nuestro proyecto de manera satisfactoria.

1.4 INSTALACIÓN DE LAS CREDENCIALES DE AWS

Para poder hacer uso de este proyecto, es necesario de alguna manera indicar la identidad respecto a AWS. Para ello, se deben seguir los siguientes pasos. Debemos acceder a nuestra cuenta en AWS.

La imagen, es la siguiente, dónde podemos observar el nombre de la cuenta y el formulario para introducir el usuario y la contraseña.

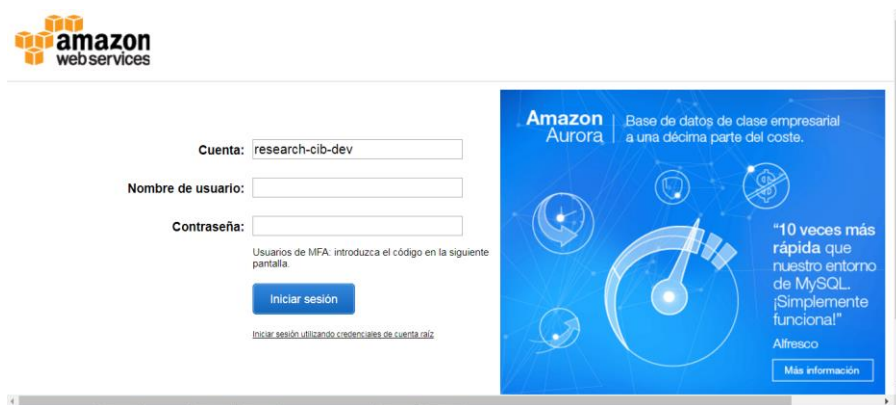


Figura 22: Accediendo a AWS

Al acceder a nuestro perfil, podremos observar todos los servicios *cloud* que AWS ofrece, pero en este caso, nosotros nos centraremos en uno, el IAM, que es el servicio que permite gestionar las cuentas.

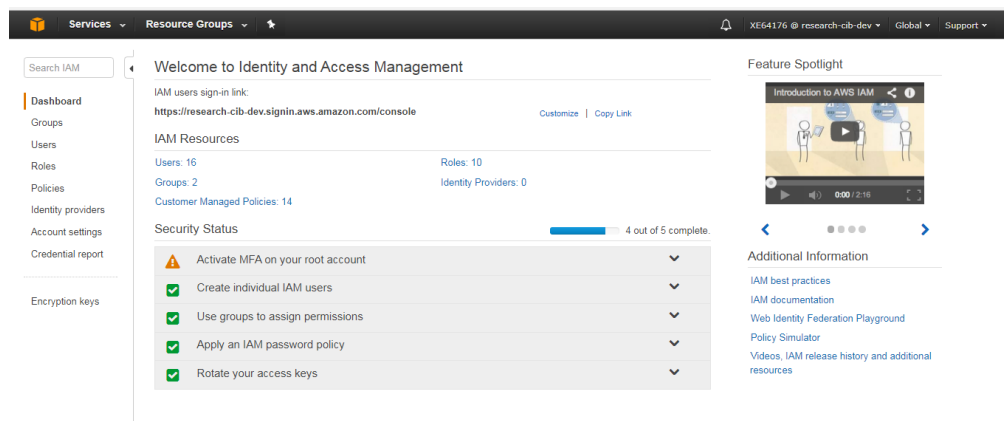


Figura 23: Servicio IAM dentro de AWS

Dentro del menú situado a la izquierda, debemos pulsar en usuarios para mostrar todos los usuarios que hay, y, posteriormente, seleccionar el nuestro. Aparecerá la siguiente ventana:

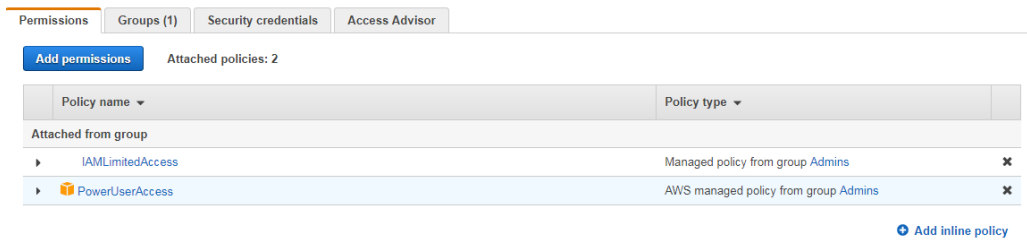


Figura 24: Menú particular de un usuario en AWS

Dentro de la ventana de “Security Credentials”, es dónde se almacenan las credenciales de cada usuario. Cada uno puede tener como máximo dos, y la contraseña únicamente se puede ver en el momento de crear una credencial. Por tanto, si se tienen dos, se debe borrar una e incorporar otra, pulsando la “x” en la que deseamos eliminar, y pulsando “Create Access Key” para crear una nueva. Como hemos dicho antes, es muy importante ver y apuntar la clave de la credencial de seguridad, ya que no podremos volver a acceder a este dato. Por temas de confidencialidad, no se mostrarán capturas de este proceso.

Una vez tengamos el ID y la *password* de la correspondiente credencial, debemos introducirla en nuestro sistema de la siguiente manera. Accedemos a las variables de entorno y creamos dos. El nombre de cada una de ellas debe ser:

- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY

El valor de la primera debe ser el id de la credencial, y el valor de la segunda, la contraseña. Una vez establecidos estos datos, ya estamos indicando al sistema esta credencial y, por tanto, a AWS quienes somos cada vez que le enviemos peticiones. El proyecto detectará automáticamente esta credencial y por tanto, obtendrá la información correspondiente a la cuenta a la que pertenezca esta credencial, que, en nuestro caso, es el BBVA.

ANEXO B: MANUAL DE USUARIO

En este manual de usuario se explica, a través de un ejemplo de ejecución, los pasos que el usuario debe realizar y las acciones que debe llevar a cabo para realizar diferentes funcionalidades. Se hace uso de capturas de pantalla que expliquen la acción concreta que es está explicando.

Antes de nada, es muy importante tener un buen conocimiento sobre las credenciales que proporciona AWS a sus usuarios para poder hacer peticiones a su API. Para el proyecto, se necesitan, y sin ellas, nada de lo que se explica a continuación funcionaría. En primer lugar, este proyecto solo podrá ejecutarlo alguien con los permisos correspondientes, o en su defecto, podrá ejecutarlo cualquiera que posea un ordenador con estas credenciales, aunque esta situación no es la normal.

Para saber cómo obtenerlas, acceder al apartado “Instalación de las credenciales de AWS”. Al tenerla en las variables de entorno del sistema, el proyecto debería detectarlas automáticamente para obtener la identidad de quién está haciendo uso del proyecto aquí desarrollado.

Es importante comentar que, al tratarse de una aplicación web, tendremos que levantar dos tipos de proyectos diferentes, el servidor y el cliente. El servidor es donde se encuentra toda la lógica desarrollada en Java. Lo hemos dividido en un pequeño menú para facilitar la comprobación de ciertas tareas, pero cuando se complete la aplicación monitorizando todos los servicios *cloud*, se dejará una sola opción (una sola clase) para ejecutarla como clase principal. Para ello, de momento, deberemos crearnos una nueva configuración de ejecución en el menú superior, en la parte derecha, para ejecutar nuestra aplicación.

La nueva configuración deberá crearse de tipo “*Application*” y contener la clase principal de nuestro proyecto, que en nuestro caso se denomina “Main”. Así mismo debe indicarse el JRE correspondiente al SDK de Java del que dispongamos (Es bastante posible que IntelliJ

lo detecte automáticamente). Una vez creada una configuración para la ejecución de un proyecto ya podemos dar paso a la ejecución del mismo.

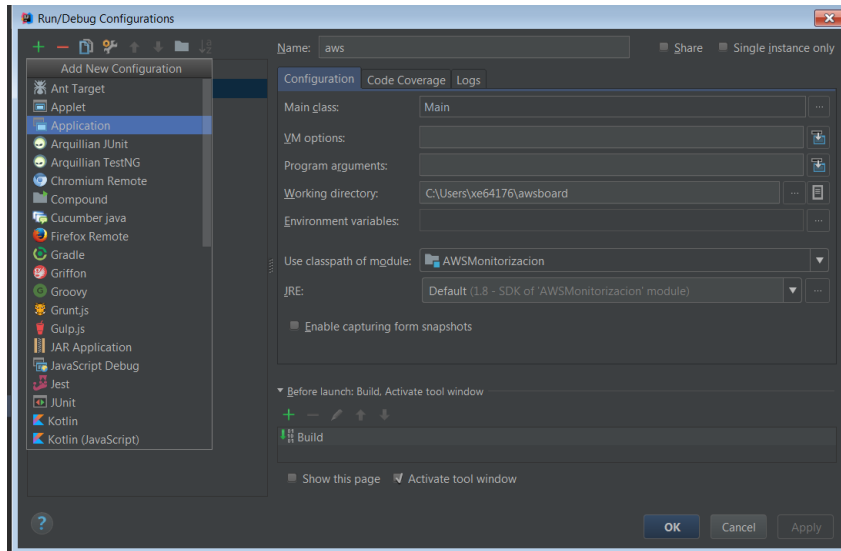


Figura 1: Configuración de la ejecución para IntelliJ

En nuestro caso, deberemos elegir la opción 6, que levantará el servidor basado en Spring por el acceso a la base de datos y a toda la API-REST para poder hacer peticiones desde el cliente Angular 2.

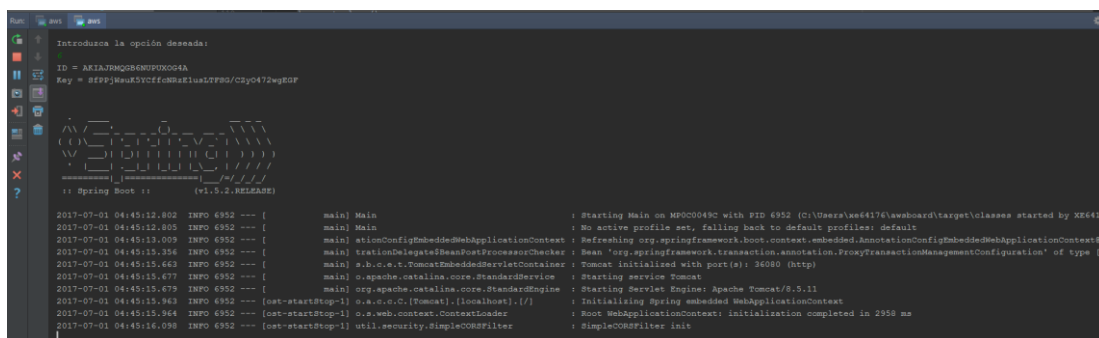


Figura 2: Levantamiento del servidor con Spring

Por otro lado, debemos seguir un camino similar para levantar nuestro cliente Angular 2, que será el que interactúe con el usuario para realizar toda la funcionalidad que tenga la aplicación desarrollada.

ANEXO B: MANUAL DE USUARIO

Para ejecutar un cliente Angular 2, debemos crearnos una configuración npm, que dispondrá de varios métodos que ejecutar. Se debe elegir el método “start” que levanta un servidor virtual en un puerto de la máquina, y simula que el cliente está almacenado ahí.

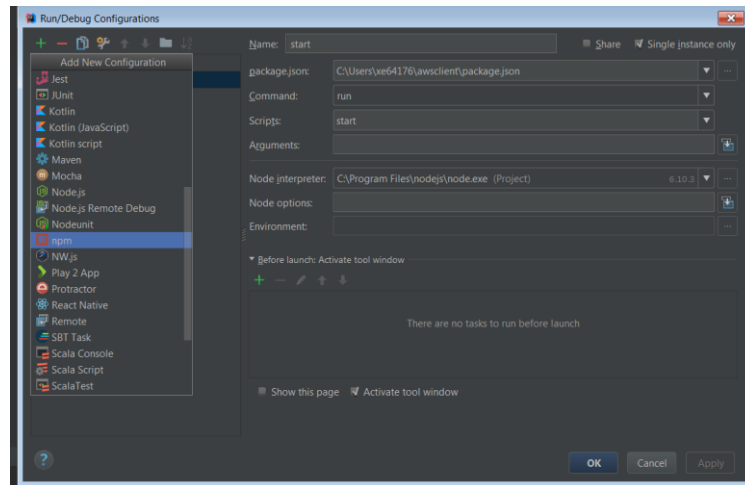


Figura 3: Configuración de npm en IntelliJ

Al ejecutarse, en el terminal del IDE, se nos indicará la dirección y el puerto donde se ha levantado nuestro cliente Angular 2, y por tanto, sólo tendremos que copiar dicha dirección en el navegador, y visualizaremos la aplicación web desarrollada. En este caso <http://localhost:4200>.

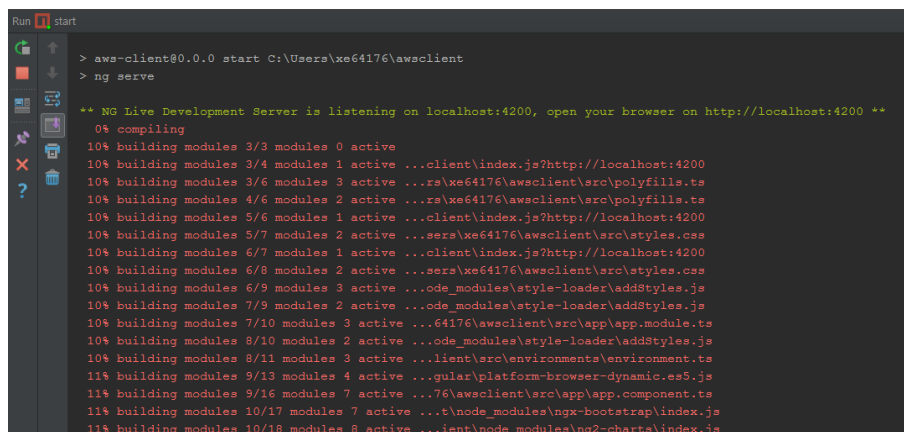


Figura 4: Ejecución del cliente Angular 2

Una vez levantados ambos en nuestra máquina local, se puede empezar a realizar toda la funcionalidad disponible. Es muy importante comentar, que la base de datos es persistente, por lo que, si no se ha importado el archivo donde está almacenada, la primera vez que ejecutemos el proyecto no aparezca ningún dato en los gráficos ni en las tablas. Lo único que se debe hacer es esperar a que la base de datos se actualice sola, o se actualiza manualmente desde los botones correspondientes.

Una vez levantados ambas partes del proyecto, y teniendo la base de datos en orden, el aspecto inicial de la aplicación es el siguiente:

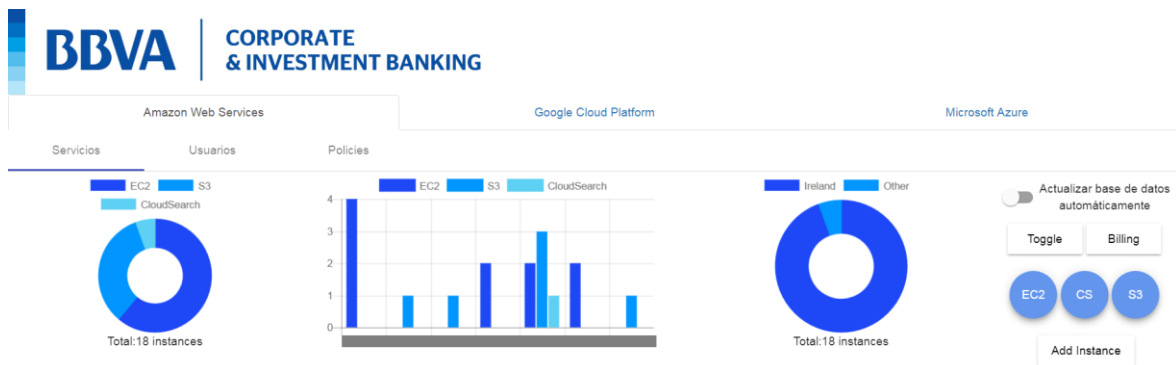


Figura 5: Inicio de la aplicación

En la primera pestaña, se pueden realizar ciertas funcionalidades. Si se desea cambiar el tipo de gráfico, se deberá pulsar el botón de “Toggle”.

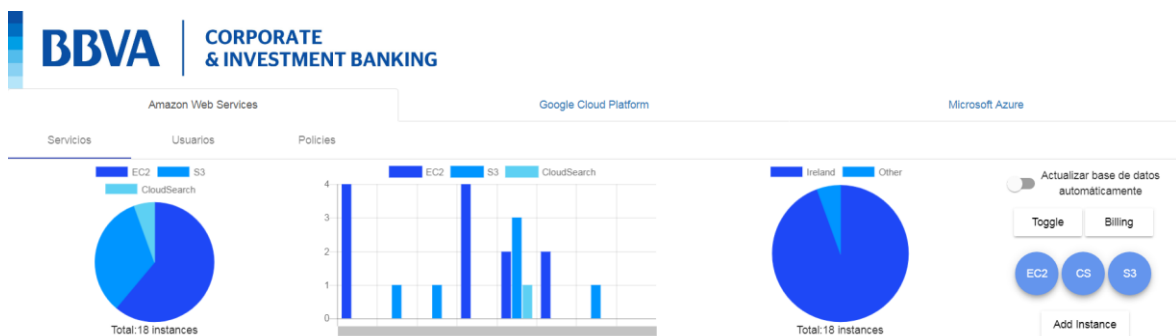


Figura 6: Inicio de la aplicación con los gráficos cambiados.

Si se desea acceder al *billing* oficial, se deberá pulsar el botón llamado “*Billing*”. Aparecerá la siguiente ventana. Evidentemente, el coste variará en función de la fecha de acceso. El coste se renueva cada mes. Esta imagen está tomada a principios de mes.

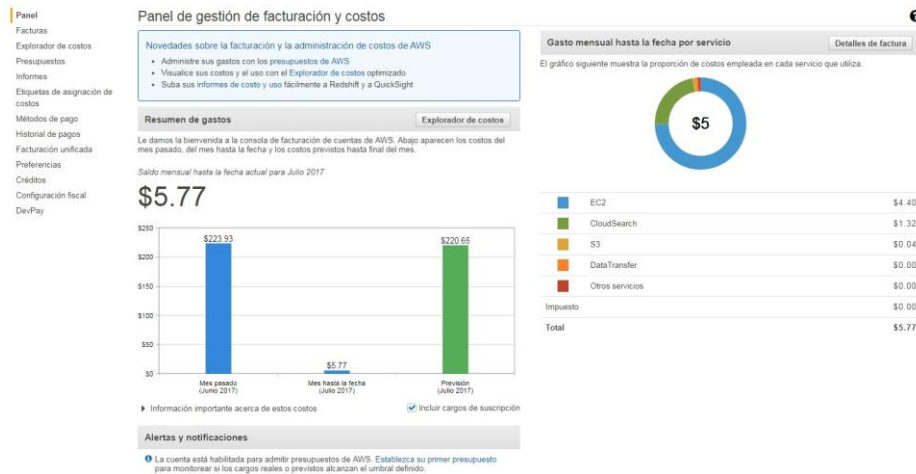


Figura 7: Dashboard oficial de billing en AWS

Para mostrar cualquier tipo de instancias, lo único que debemos hacer es pulsar el botón correspondiente a cada tipo de instancia. Se hará un *scroll* automático para que no tenga que realizarlo el usuario.

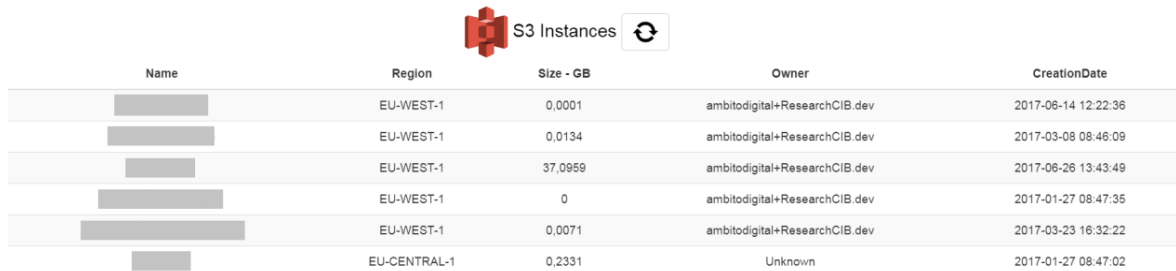



Figura 8: Botones para las instancias

EC2 Instances

Name	Id	LaunchDate	Region	State	Type	Owner	Dns	Platform
[Redacted]	i-0704da293804c948c	2017-04-09 09:35:18	eu-west-1b	running	t2.micro		ip-172-31-39-191	Other Linux
[Redacted]	i-0d473d8afd56f1ad9	2017-06-29 16:02:08	eu-west-1b	stopped	t2.micro		ip-172-31-44-178	Red Hat
[Redacted]	i-0625a23ef5686a230	2017-01-11 14:27:29	eu-west-1b	stopped	t2.micro		ip-172-31-43-50	Amazon Linux
[Redacted]	i-01b55ea35904b050e	2017-06-02 10:09:54	eu-west-1b	stopped	t2.micro		ip-172-31-39-10	Other Linux
[Redacted]	i-0f24174c0f846c66	2017-01-25 15:33:26	eu-west-1a	stopped	m4.xlarge		ip-172-31-19-149	Other Linux
[Redacted]	i-0ff243d2e4f2feb5	2017-05-24 07:21:40	eu-west-1a	running	t2.medium		ip-172-31-20-12	Other Linux
[Redacted]	i-0992f8dee88d246c9	2017-04-26 13:30:07	eu-west-1b	running	t2.micro		ip-172-31-41-35	Amazon Linux
[Redacted]	i-0755d2dc499e39806	2017-06-26 15:15:28	eu-west-1b	stopped	m4.2xlarge		ip-172-31-36-19	Other Linux
[Redacted]	i-02513240c1b045127	2017-03-02 16:09:04	eu-west-1b	running	t2.micro		ip-172-31-35-111	Other Linux
[Redacted]	i-07210b9e9f48145c2	2017-04-11 12:45:04	eu-west-1b	running	t2.nano		ip-172-31-32-162	Amazon Linux
[Redacted]	i-01a2c100e99c3c383	2016-10-20 13:48:56	eu-west-1b	running	m4.2xlarge	CIB_TA	ip-172-31-44-166	Other Linux

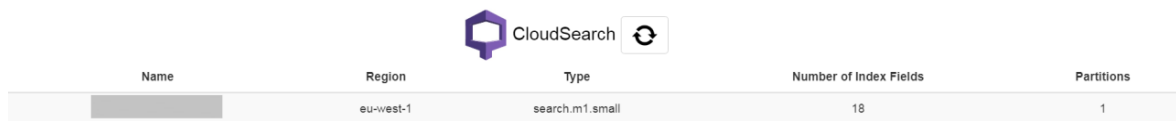
Figura 9: Tabla de instancias EC2




S3 Instances 

Name	Region	Size - GB	Owner	CreationDate
[redacted]	EU-WEST-1	0,0001	ambitodigital+ResearchCIB.dev	2017-08-14 12:22:36
[redacted]	EU-WEST-1	0,0134	ambitodigital+ResearchCIB.dev	2017-03-08 08:46:09
[redacted]	EU-WEST-1	37,0959	ambitodigital+ResearchCIB.dev	2017-08-26 13:43:49
[redacted]	EU-WEST-1	0	ambitodigital+ResearchCIB.dev	2017-01-27 08:47:35
[redacted]	EU-WEST-1	0,0071	ambitodigital+ResearchCIB.dev	2017-03-23 16:32:22
[redacted]	EU-CENTRAL-1	0,2331	Unknown	2017-01-27 08:47:02

Figura 10: Tabla de instancias S3



CloudSearch 

Name	Region	Type	Number of Index Fields	Partitions
[redacted]	eu-west-1	search.m1.small	18	1

Figura 11: Tabla de instancias CloudSearch

Si se quiere actualizar manualmente la base de datos para tener actualizados los datos, únicamente se debe pulsar el botón de actualizar situado al lado de cada tipo de servicio.

Si el botón situado en la parte superior izquierda (el de actualizar la base de datos automáticamente) no está pulsado, podremos actualizar la base de datos manualmente, pero, si por el contrario, si no está, se realizará esta acción automáticamente



Figura 12: Botón activado.



Por otro lado, cuando se levanta una instancia de manera oficial, se debe comunicar a la aplicación a través del formulario que aparece cuando se pulsa el botón “Add Instance”. Se deberán introducir todos los datos necesarios para tener la aplicación actualizada y que, por tanto, se muestren las instancias levantadas por cada usuario en el gráfico de barras.



Figura 13: Formulario para crear una instancia

Por otro lado, si se desea suprimir algún tipo de dato en cualquiera de los gráficos, basta con pulsar sobre el dato que se desea quitar en la leyenda del gráfico.

En cuanto a los usuarios, se pueden realizar dos tipos de acciones. La primera y básica, es mostrarlos. Para ello, únicamente se deberá pulsar el botón llamado “Mostrar usuarios”, y a continuación, aparecerá una tabla con todos los usuarios actuales de AWS en la cuenta del banco.

 Usuarios 

IdBvva	Id	Última conexión	Fecha de alta
██████████	██████████	2017-06-06 06:54:34	2016-10-19 16:54:28
██████████	██████████	2017-04-20 09:38:43	2016-08-26 10:31:04
██████████	██████████	2017-04-28 09:06:37	2016-08-26 10:31:04
██████████	██████████	Nunca conectado	2016-09-06 08:08:29
██████████	██████████	Nunca conectado	2016-10-19 16:54:28
██████████	██████████	2017-02-17 07:10:12	2017-01-09 17:09:20
██████████	██████████	2017-06-01 20:16:27	2017-01-09 17:09:20
██████████	██████████	2017-01-16 09:17:20	2017-01-11 07:23:51
██████████	██████████	2016-10-20 09:34:39	2016-10-19 16:54:28
██████████	██████████	Nunca conectado	2016-10-19 16:55:06
██████████	██████████	2017-06-30 07:31:05	2016-10-19 16:54:28
██████████	██████████	Nunca conectado	2016-10-19 16:55:06
██████████	██████████	2017-07-04 07:12:41	2016-10-19 16:54:28
██████████	██████████	2017-07-03 14:34:26	2016-10-19 16:55:06
██████████	██████████	2017-07-05 07:57:37	2017-02-14 16:14:30
██████████	██████████	2017-04-26 11:26:52	2017-01-30 15:08:12

Figura 14: Usuarios actuales en AWS

Por otro lado, para crear grupos en AWS, se deberá pinchar en el botón correspondiente y completar el formulario que aparecerá en el momento de hacerlo.

Mostrar Usuarios

Crear grupo

Nombre del grupo

✓ Añadir

Figura 15: Formulario de entrada para crear grupos.

Respecto a los permisos, en la web aparecerán dos tablas diferentes. El uso es muy intuitivo. El usuario que haga uso de la aplicación simplemente deberá seleccionar que *policias* desea asignar a cualquier usuario de la cuenta. Una vez hecho esto, deberá pulsar el botón llamado “Generate PDF” para obtener el PDF donde se muestre la información que se desea enviar al departamento correspondiente a la seguridad de la cuenta.

A continuación, mostramos una imagen de las tablas y del PDF correspondiente:

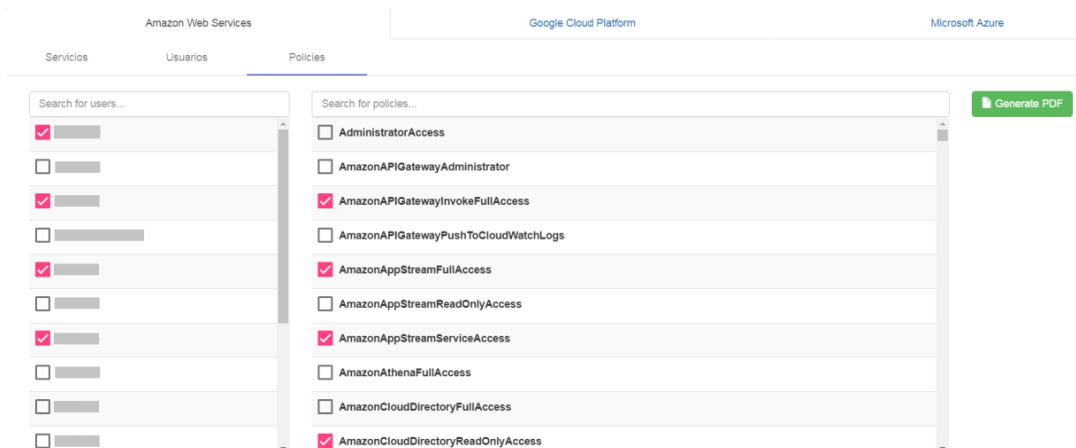


Figura 16 : Tablas de usuarios y policias



Figura 17: PDF generado

Con esta última explicación, se ha terminado el manual de usuario. Como se ha podido comprobar, es bastante preciso y conciso, indicando claramente que se puede hacer y cómo.