COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

MÁSTER EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE MÁSTER

DESARROLLO DE UN SISTEMA DIGITAL PARA LA INTERCONEXIÓN DE DONACIONES DE SANGRE ENTRE HOSPITALES Y BANCOS DE SANGRE

Autor: Alejandro Ucelay Jiménez

Director: Celina Álvarez-Cedrón Benavente

Co-Director: Álvaro Morate Solís

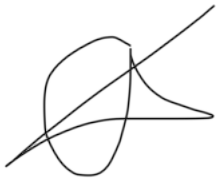
Madrid Agosto 2024

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título Desarrollo de un sistema digital para la interconexión de donaciones de sangre entre hospitales y bancos de sangre en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2023/2024 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.: Alejandro Ucelay Jiménez Fecha: 24 / 08 / 2024

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO



Fdo.: Celina Álvarez-Cedrón Benavente Fecha: 24 / 08 / 2024

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título Desarrollo de un sistema digital para la interconexión de donaciones de sangre entre hospitales y bancos de sangre en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2023/2024 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.: Alejandro Ucelay Jiménez Fecha: 24 / 08 / 2024

Autorizada la entrega del proyecto

EL CO-DIRECTOR DEL PROYECTO

Fdo.: Álvaro Morate Solís Fecha: 24 / 08 / 2024

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

MÁSTER EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE MÁSTER

DESARROLLO DE UN SISTEMA DIGITAL PARA LA INTERCONEXIÓN DE DONACIONES DE SANGRE ENTRE HOSPITALES Y BANCOS DE SANGRE

Autor: Alejandro Ucelay Jiménez

Director: Celina Álvarez-Cedrón Benavente

Co-Director: Álvaro Morate Solís

Madrid Agosto 2024

Dedicatoria

Este trabajo de fin de máster lo dedico con todo cariño a mis padres Fernando y Alejandra por su sacrificio y esfuerzo, por creer en mi capacidad, por darme la oportunidad de poder estudiar lo que realmente me apasiona y por su incondicional apoyo durante estos dos años de máster.

A mis hermanos Fernando y Gadea, por ser mi fuente de inspiración y motivación para poder superarme y dar la mejor versión de mí mismo cada día.

Agradecimientos

Quiero agradecer a mis tutores Celina y Alvaro por la ayuda durante la elaboración de este proyecto.

DESARROLLO DE UN SISTEMA DIGITAL PARA LA INTERCONEXIÓN DE DONACIONES DE SANGRE ENTRE HOSPITALES Y BANCOS DE SANGRE

Autor: Ucelay Jiménez, Alejandro.
Director: Celina Álvarez-Cedrón Benavente.
Co-Director: Álvaro Morate Solís.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Este proyecto se enfoca en desarrollar una aplicación web para optimizar el proceso de transfusiones sanguíneas, mejorando la trazabilidad y el monitoreo en tiempo real de las donaciones de sangre entre hospitales y bancos de sangre. El objetivo es incrementar la eficiencia y seguridad en las transfusiones, reducir errores y mejorar la gestión de los recursos sanguíneos.

Palabras clave: Desarrollo web, base de datos, servidor, interfaz de usuario, donaciones de sangre, hospitales, bancos de sangre, pacientes, donadores.

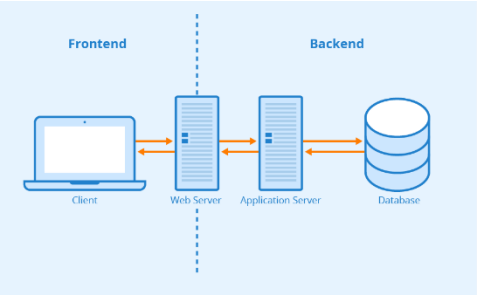
1. Introducción

El presente proyecto tiene como objetivo desarrollar una aplicación web que optimice la eficiencia del proceso integral de transfusiones sanguíneas al incorporar un componente de trazabilidad para rastrear de manera precisa y completa el ciclo de vida de una donación, así como monitorear en tiempo real el nivel de inventario de los diversos bancos de sangre. Para evaluar el correcto funcionamiento de la aplicación, se llevará a cabo una demostración en la que se simulará la solicitud de una dosis de un tipo de sangre específico para un paciente determinado, identificando los bancos de sangre con inventario disponible para satisfacer dicha petición.

---

2. Definición del proyecto

El desarrollo de la aplicación se basa en un servidor, una interfaz de usuario y un modelo de datos que interactúan mediante peticiones HTTP a través de servicios cliente-servidor, garantizando una comunicación efectiva y segura entre ellos. A continuación, se muestra un esquema que muestra la interconexión entre el servidor de la aplicación, la interfaz de usuario (front-end) y la base de datos.



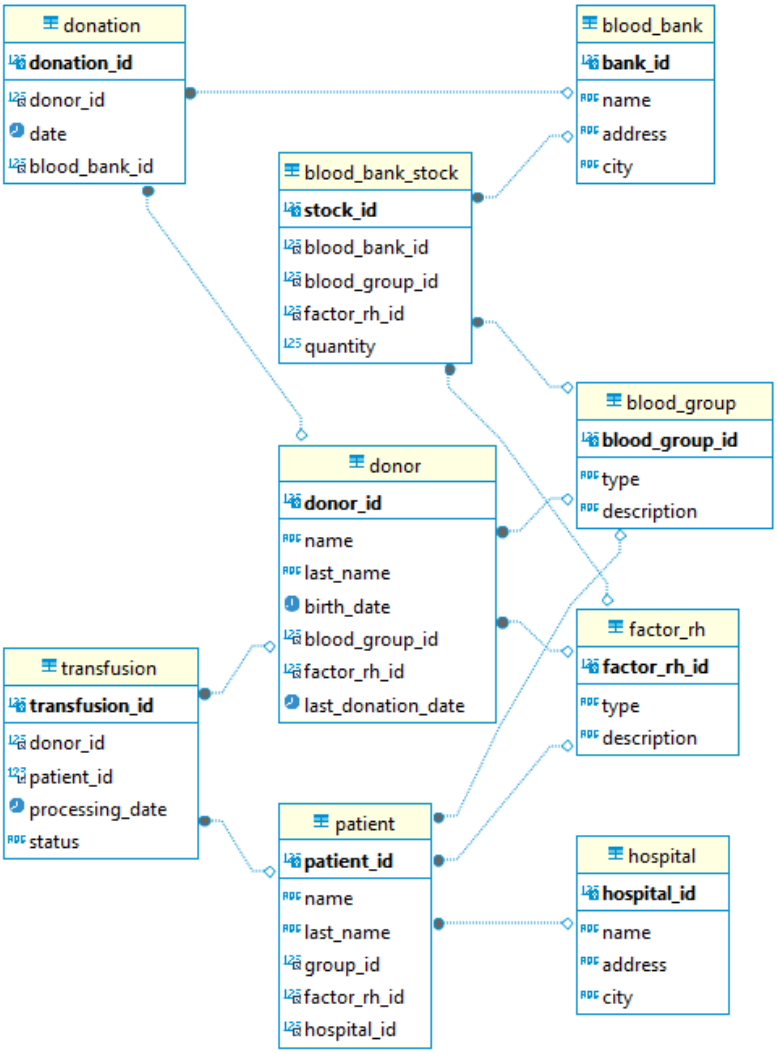
3. Solución

El desarrollo del servidor de la aplicación se ha realizado utilizando Python como lenguaje principal, aprovechando su robustez y versatilidad en la creación de aplicaciones web y la gestión de bases de datos. Para la conexión con la base de datos MySQL, se ha empleado la librería mysql-connector-python, la cual permite interactuar de manera eficiente y segura con la base de datos, ejecutando consultas SQL y gestionando transacciones de manera confiable. Además, se ha utilizado Flask, un microframework ligero pero poderoso que facilita la creación de aplicaciones web con una estructura sencilla y flexible. Flask ha sido esencial para manejar las rutas, gestionar las peticiones HTTP y asegurar una integración fluida entre el frontend y el backend. Para el modelado de la base de datos, se ha incorporado SQLAlchemy, un ORM (*Object-Relational Mapping*) que permite definir y manipular las tablas y relaciones de la base de datos en forma de clases y objetos en Python, simplificando considerablemente el proceso de interacción con la base de datos y asegurando un código más limpio y mantenible.

Por otro lado, el *frontend* de la aplicación se ha desarrollado utilizando JavaScript con la biblioteca React, una herramienta utilizada para construir interfaces de usuario dinámicas y altamente interactivas. React permite crear componentes reutilizables que gestionan de manera eficiente el estado y el flujo de datos dentro de la aplicación, mejorando así la experiencia del usuario. Además, se ha integrado la librería Ant Design, que proporciona una colección extensa de componentes predefinidos con un diseño moderno y profesional.

4. Resultados

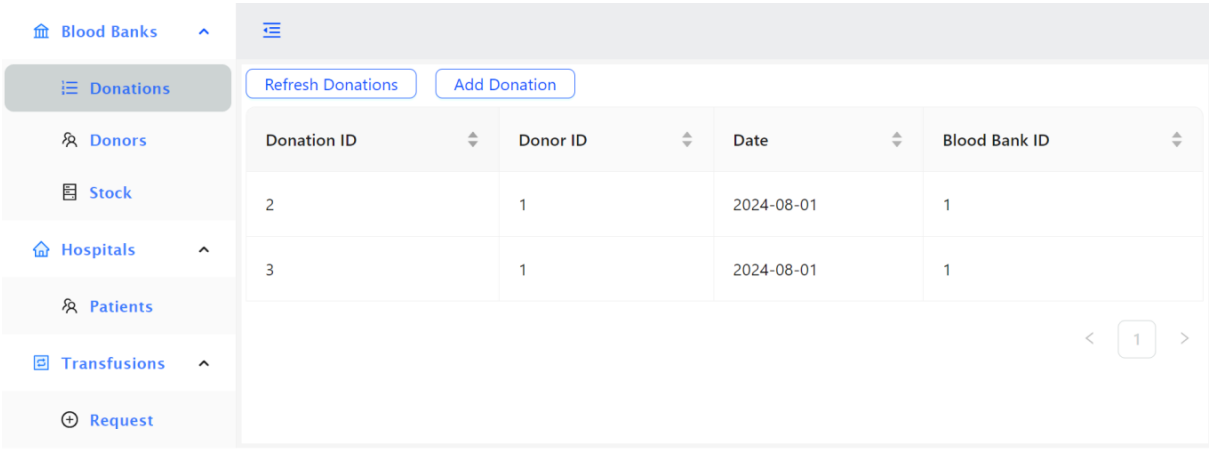
Se ha desarrollado la aplicación para la interconexión de donaciones de sangre entre hospitales y bancos de sangre. En primer lugar, se diseño el modelo lógico, definiendo las entidades necesarias para cubrir el objetivo del proyecto. Además de las entidades, se definieron las relaciones entre ellas resultando en el siguiente modelo lógico:



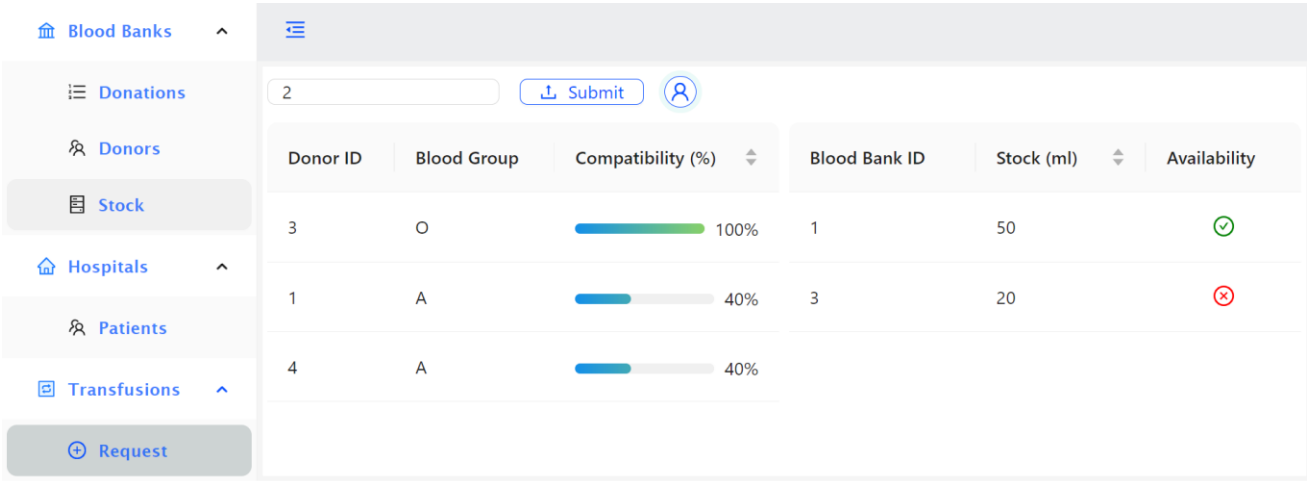
Una vez definido la base de datos del proyecto, se ha desarrollado el servidor de la aplicación utilizando el lenguaje de programación Python. Este servidor se compone de diferentes módulos que hacen referencia a las principales entidades del modelo lógico. En cada módulo, se llevan a cabo las diferentes peticiones HTTP (*GET*, *POST*, *UPDATE*, *DELETE*). De esta forma, el servidor establece una conexión con la base de datos y puede realizar peticiones para la obtención e inserta de datos. Para comprobar el correcto funcionamiento del servidor, se han llevado a cabo diferentes pruebas utilizando *Postman*, el cual permite la creación de un cliente virtual para poder realizar peticiones a los *endpoints* desarrollados en el servidor. Se ha comprobado que, para cada una de las direcciones, el servidor funciona correctamente y devuelve los datos solicitados.

Una vez el servidor funcionaba correctamente y devolvía los datos solicitados, se llevo a cabo el desarrollo de la interfaz de usuario utilizando el lenguaje de programación JavaScript y la librería React. Además, para el diseño de la misma se utilizo HTML y CSS. La interfaz de usuario dispone de un menú en la parte izquierda que permite al usuario final navegar por la aplicación. A continuación,

se puede observar el menú con los diferentes submenús y las tablas provenientes del servidor mostrando la información solicitada.



Una vez desarrollada la interfaz de usuario, se llevaron a cabo pruebas *end-to-end* para comprobar el correcto funcionamiento de la aplicación. Para ello, la prueba consistía en solicitar una transfusión ("*Request*") para un determinado paciente y la aplicación debería devolver un listado de los diferentes donadores con su respectiva compatibilidad (%) de sangre y un listado de los diferentes bancos de sangre definiendo si cuentan con inventarios para ese tipo de sangre o no. A continuación, se puede observar como para el paciente con número de ID (2), la aplicación devuelve los dos listados requeridos.



5. Conclusiones

Se puede concluir que el desarrollo del sistema digital para la interconexión de donaciones de sangre entre hospitales y bancos de sangre ha mejorado significativamente la eficiencia y seguridad en la gestión de transfusiones, al permitir el monitoreo en tiempo real de inventarios y los donantes disponibles compatibles para cada tipo de paciente. Este proyecto, construido con tecnologías

modernas como Flask y React, ha demostrado ser una herramienta efectiva para optimizar la disponibilidad de sangre y mejorar la toma de decisiones en situaciones de emergencia. Además, su diseño escalable y adaptable sienta las bases para futuras integraciones con sistemas hospitalarios más amplios, fortaleciendo así la infraestructura de salud y promoviendo prácticas más sostenibles y equitativas en la distribución de recursos médicos críticos.

6. Referencias

1. Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.
2. Sadalage, P. & Fowler, M. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley.
3. Sonmez, J. (2013). *Soft Skills: The software developer's life manual*. Manning Publications.
4. Khan, M. & Khan, S. S. (2022). "A Comparative Study of Web Application Frameworks: Django vs. Flask", *International Journal of Computer Science and Information Security*, 20(5), 123-130.
5. Lin, K., Li, Z., & Lee, Y. (2021). "Real-Time Blood Bank Management System Using Internet of Things", *International Journal of Healthcare Information Systems and Informatics*, 16(2), 56-72.
6. Pusztai, J., Menyhei, A., & Bessenyei, B. (2020). "A Cloud-Based Blood Donation and Transfusion Management System", *Journal of Cloud Computing and Data Science*, 8(1), 89-102.
7. Flask Documentation. (2024). *Flask*. https://flask.palletsprojects.com/
8. React Documentation. (2024). *React – A JavaScript library for building user interfaces*. https://reactjs.org/docs/getting-started.html
9. MySQL Documentation. (2024). *MySQL 8.0 Reference Manual*. https://dev.mysql.com/doc/
10. SQLAlchemy Documentation. (2024). *SQLAlchemy*. https://docs.sqlalchemy.org/
11. Ant Design Documentation. (2024). *Ant Design*. https://ant.design/docs/react/introduce
12. How to build a Rest API. https://restfulapi.net/
13. 1. Smith, J., & Johnson, A. (2022). Technological Advances in Blood Transfusions: A Comprehensive Review. Journal of Transfusion Medicine, 15(2), 87-104.
14. 2. García, L., & Pérez, M. (2023). Automation and Centralized Databases in Blood Transfusion Services: Enhancing Safety and Efficiency. International Journal of Hematology, 28(4), 321-335.
15. 3. Wang, S., et al. (2024). Robotics and Artificial Intelligence in Blood Transfusion: Current Applications and Future Perspectives. Journal of Medical Robotics, 10(1), 45-58.

16. 4. Hawashin D, Mahboobeh DAJ, Salah K, Jayaraman R, Yaqoob I, Debe M et al. Blockchain-based management of blood donation. IEEE Access. 2021;9:163016-32.
17. 5. Pradhan NR, Singh AP, Kumar V. Blockchain-enabled traceable, transparent transportation system for blood bank. In Advances in VLSI, Communication, and Signal Processing: Select Proceeding
18. 6. Castro M, Jara AJ, Skarmeta AF. Analysis of the future internet of things capabilities for continuous temperature monitoring of blood bags in terrestrial logistic systems. In Convergence and Hybrid Information Technology: 5th International Conference, ICHIT 2011, Daejeon, Korea. Proceedings, Springer Berlin Heidelberg. 2011;5:558- 66.
19. 7. Hammoudeh M, Ghafir I, Bounceur A, Rawlinson T. Continuous monitoring in mission-critical applications using the internet of things and blockchain. In Proceedings of the 3rd International Conference on Future Networks and Distributed Systems. 2019;1-5.
20. 8. Mohanta B, Das P, Patnaik S. Healthcare 5.0: A paradigm shift in digital healthcare system using artificial intelligence, IOT and 5G communication, In 2019 International Conference on Applied Machine Learning. 2019;191-6.

DESARROLLO DE UN SISTEMA DIGITAL PARA LA INTERCONEXIÓN DE DONACIONES DE SANGRE ENTRE HOSPITALES Y BANCOS DE SANGRE

Autor: Ucelay Jiménez, Alejandro.
Director: Celina Álvarez-Cedrón Benavente.
Co-Director: Álvaro Morate Solís.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

PROJECT SUMMARY

This project focuses on developing a web application to optimize the blood transfusion process by improving the traceability and real-time monitoring of blood donations between hospitals and blood banks. The goal is to increase the efficiency and safety of transfusions, reduce errors, and enhance the management of blood resources.

Key words: Web development, databases, server, user interface, hospitals, blood donations, blood banks, hospitals, patients, donors.

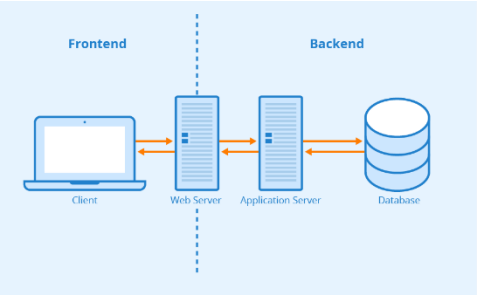
1. Introduction

The aim of this project is to develop a web application that optimizes the efficiency of the comprehensive blood transfusion process by incorporating a traceability component to accurately and comprehensively track the lifecycle of a donation, as well as monitor in real-time the inventory levels of various blood banks. To evaluate the proper functioning of the application, a demonstration will be conducted in which a request for a dose of a specific blood type for a particular patient will be simulated, identifying the blood banks with available inventory to fulfill the request.

---

2. Project definition

The development of the application is based on a server, a user interface, and a data model that interact through HTTP requests via client-server services, ensuring effective and secure communication between them. Below is a diagram that illustrates the interconnection between the application server, the user interface (front-end), and the database.



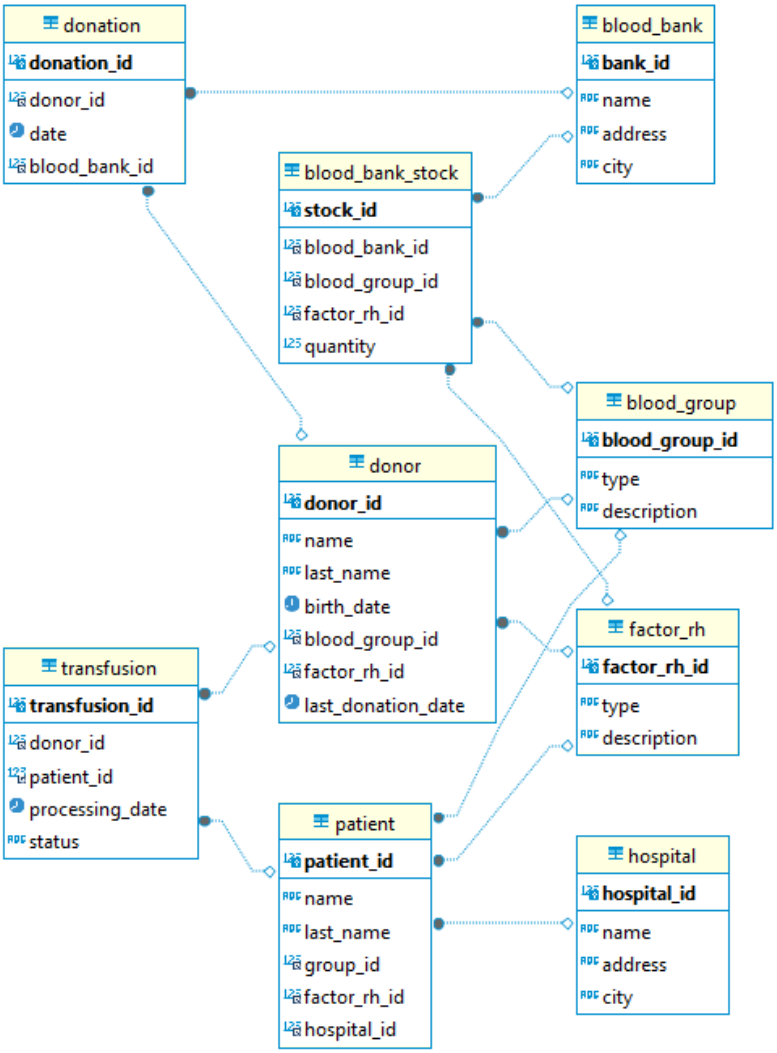
3. Solution

The development of the application server has been carried out using Python as the primary language, leveraging its robustness and versatility in creating web applications and managing databases. For connecting with the MySQL database, the mysql-connector-python library has been used, which allows for efficient and secure interaction with the database, executing SQL queries, and managing transactions reliably. Additionally, Flask, a lightweight yet powerful microframework, has been employed to facilitate the creation of web applications with a simple and flexible structure. Flask has been essential for handling routes, managing HTTP requests, and ensuring smooth integration between the frontend and backend. For database modeling, SQLAlchemy has been incorporated, an ORM (Object-Relational Mapping) that allows defining and manipulating database tables and relationships as classes and objects in Python, significantly simplifying the interaction with the database and ensuring cleaner, more maintainable code.

On the other hand, the frontend of the application has been developed using JavaScript with the React library, a tool used to build dynamic and highly interactive user interfaces. React enables the creation of reusable components that efficiently manage state and data flow within the application, thereby enhancing the user experience. Additionally, the Ant Design library has been integrated, providing an extensive collection of pre-defined components with a modern and professional design.

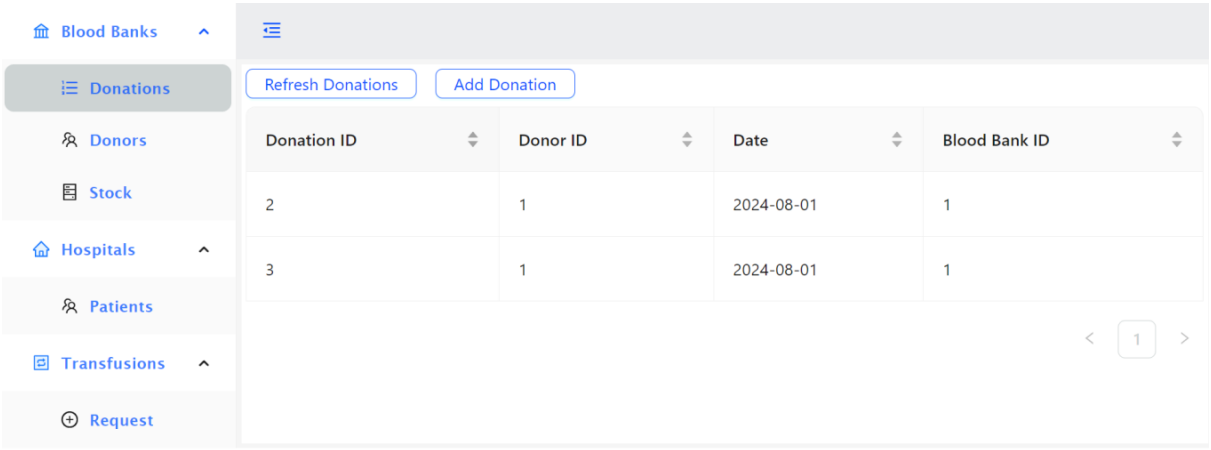
4. Results

The application for the interconnection of blood donations between hospitals and blood banks has been developed. First, the logical model was designed, defining the necessary entities to achieve the project's objective. In addition to the entities, the relationships between them were defined, resulting in the following logical model:

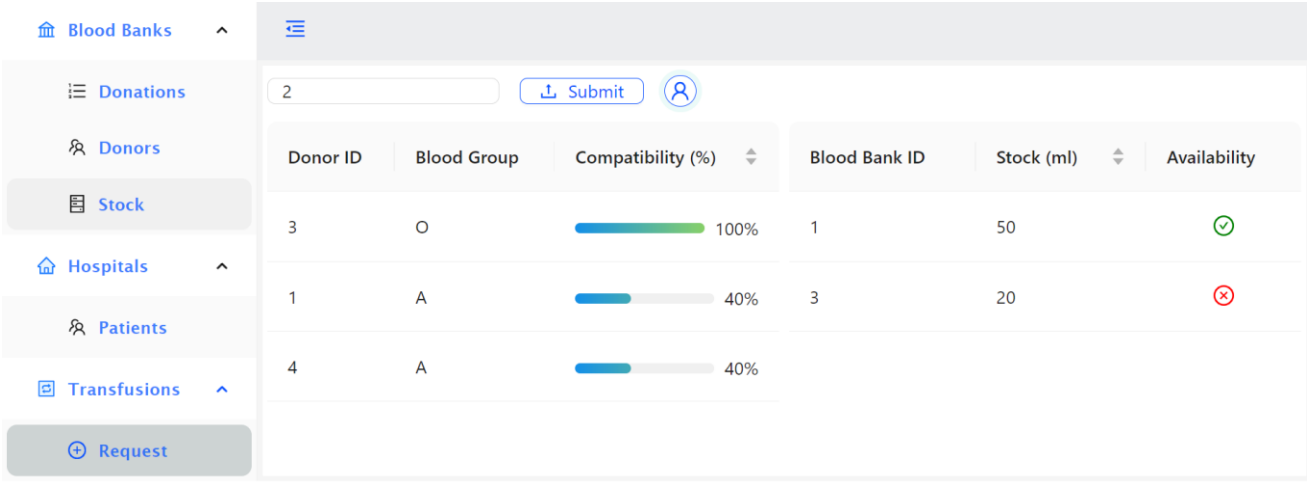


Once the project database was defined, the application server was developed using the Python programming language. This server is composed of different modules that refer to the main entities of the logical model. Each module handles various HTTP requests (GET, POST, UPDATE, DELETE). In this way, the server establishes a connection with the database and can perform requests for data retrieval and insertion. To verify the correct functioning of the server, various tests were conducted using Postman, which allows the creation of a virtual client to make requests to the endpoints developed on the server. It was confirmed that for each endpoint, the server works correctly and returns the requested data.

Once the server was functioning correctly and returning the requested data, the user interface was developed using the JavaScript programming language and the React library. Additionally, HTML and CSS were used for the design. The user interface features a menu on the left side that allows the end user to navigate through the application. Below, you can see the menu with the different submenus and the tables coming from the server displaying the requested information.



Once the user interface was developed, end-to-end tests were conducted to verify the correct functioning of the application. For this purpose, the test involved requesting a transfusion ("Request") for a specific patient, and the application should return a list of different donors with their respective blood compatibility percentage and a list of the different blood banks indicating whether or not they have inventory for that blood type. Below, you can see that for the patient with ID number (2), the application returns the two required lists.



5. Conclusions

It can be concluded that the development of the digital system for the interconnection of blood donations between hospitals and blood banks has significantly improved the efficiency and safety in transfusion management by enabling real-time monitoring of inventories and compatible donors for each type of patient. This project, built with modern technologies such as Flask and React, has proven to be an effective tool for optimizing blood availability and enhancing decision-making in emergency situations. Moreover, its scalable and adaptable design lays the foundation for future integrations with broader hospital systems, thereby strengthening healthcare infrastructure and promoting more

sustainable and equitable practices in the distribution of critical medical resources.

6. References

1. Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.
2. Sadalage, P. & Fowler, M. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley.
3. Sonmez, J. (2013). *Soft Skills: The software developer's life manual*. Manning Publications.
4. Khan, M. & Khan, S. S. (2022). "A Comparative Study of Web Application Frameworks: Django vs. Flask", *International Journal of Computer Science and Information Security*, 20(5), 123-130.
5. Lin, K., Li, Z., & Lee, Y. (2021). "Real-Time Blood Bank Management System Using Internet of Things", *International Journal of Healthcare Information Systems and Informatics*, 16(2), 56-72.
6. Pusztai, J., Menyhei, A., & Bessenyei, B. (2020). "A Cloud-Based Blood Donation and Transfusion Management System", *Journal of Cloud Computing and Data Science*, 8(1), 89-102.
7. Flask Documentation. (2024). *Flask*. https://flask.palletsprojects.com/
8. React Documentation. (2024). *React – A JavaScript library for building user interfaces*. https://reactjs.org/docs/getting-started.html
9. MySQL Documentation. (2024). *MySQL 8.0 Reference Manual*. https://dev.mysql.com/doc/
10. SQLAlchemy Documentation. (2024). *SQLAlchemy*. https://docs.sqlalchemy.org/
11. Ant Design Documentation. (2024). *Ant Design*. https://ant.design/docs/react/introduce
12. How to build a Rest API. https://restfulapi.net/
13. 1. Smith, J., & Johnson, A. (2022). Technological Advances in Blood Transfusions: A Comprehensive Review. Journal of Transfusion Medicine, 15(2), 87-104.
14. 2. García, L., & Pérez, M. (2023). Automation and Centralized Databases in Blood Transfusion Services: Enhancing Safety and Efficiency. International Journal of Hematology, 28(4), 321-335.
15. 3. Wang, S., et al. (2024). Robotics and Artificial Intelligence in Blood Transfusion: Current Applications and Future Perspectives. Journal of Medical Robotics, 10(1), 45-58.
16. 4. Hawashin D, Mahboobeh DAJ, Salah K, Jayaraman R, Yaqoob I, Debe M et al. Blockchain-based management of blood donation. IEEE Access. 2021;9:163016-32.
17. 5. Pradhan NR, Singh AP, Kumar V. Blockchain-enabled traceable, transparent transportation system for blood bank. In Advances in VLSI, Communication, and Signal Processing: Select Proceeding

18. 6. Castro M, Jara AJ, Skarmeta AF. Analysis of the future internet of things capabilities for continuous temperature monitoring of blood bags in terrestrial logistic systems. In Convergence and Hybrid Information Technology: 5th International Conference, ICHIT 2011, Daejeon, Korea. Proceedings, Springer Berlin Heidelberg. 2011;5:558- 66.
19. 7. Hammoudeh M, Ghafir I, Bounceur A, Rawlinson T. Continuous monitoring in mission-critical applications using the internet of things and blockchain. In Proceedings of the 3rd International Conference on Future Networks and Distributed Systems. 2019;1-5.
20. 8. Mohanta B, Das P, Patnaik S. Healthcare 5.0: A paradigm shift in digital healthcare system using artificial intelligence, IOT and 5G communication, In 2019 International Conference on Applied Machine Learning. 2019;191-6.

CAPÍTULO 1: INTRODUCCIÓN

En este primer capítulo se expone una visión general del proyecto. En la sección 1.3 *Estado del arte*, se explican diferentes estudios acerca de la tecnología a emplear en este proyecto. A su vez, se establecen los objetivos del proyecto, así como la motivación de este.

1.1 MOTIVACIÓN

La sociedad avanza hacia una transformación digital que está impactando de manera significativa los procesos de atención médica, incluyendo la gestión de donaciones de sangre y transfusiones entre hospitales y bancos de sangre. La demanda de soluciones inteligentes que permitan planificar, controlar y optimizar los servicios relacionados con las donaciones de sangre está en aumento, buscando crear mayor valor en todo el proceso de atención médica.

En este contexto, las aplicaciones que facilitan la interconexión entre hospitales y bancos de sangre desempeñan un papel crucial en la optimización de los procesos de transfusión. Sin embargo, garantizar la eficiencia y seguridad en la entrega y recepción de sangre supone un desafío significativo que requiere soluciones innovadoras para el seguimiento y localización de las donaciones en tiempo real.

Por lo tanto, el desarrollo de una aplicación para la interconexión de donaciones de sangre entre hospitales y bancos de sangre representa una oportunidad única para mejorar la calidad de la atención médica. Esta solución podría revolucionar la gestión de las donaciones de sangre mediante la implementación de tecnologías avanzadas que permitan la trazabilidad, transparencia y control eficiente de todo el proceso.

Personalmente, aunque no tenía experiencia previa en este campo, el potencial para mejorar la vida de las personas mediante la optimización de los procesos de donación y transfusión de sangre despertó mi admiración y curiosidad.

1.2 OBJETIVOS

El presente proyecto tiene como objetivo desarrollar una aplicación web que optimice la eficiencia del proceso integral de transfusiones sanguíneas al incorporar un componente de trazabilidad para rastrear de manera precisa y completa el ciclo de vida de una donación, así como monitorear en tiempo real el nivel de inventario de los diversos bancos de sangre. Para evaluar el correcto funcionamiento de la aplicación, se llevará a cabo una demostración en la que se simulará la solicitud de una dosis de un tipo de sangre específico para un paciente determinado, identificando los bancos de sangre con inventario disponible para satisfacer dicha petición. El desarrollo de la aplicación se basa en un servidor, una interfaz de usuario y un modelo de datos que interactúan mediante peticiones HTTP a través de servicios cliente-servidor, garantizando una comunicación efectiva y segura entre ellos. A continuación, en la *Ilustración 1* se muestra un esquema que muestra la interconexión entre el servidor de la aplicación, la interfaz de usuario (*front-end*) y la base de datos.

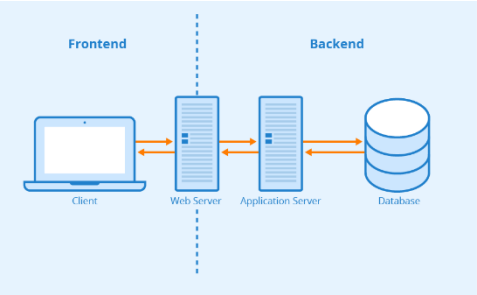


Ilustración 1: Esquema interconexión servidor-cliente-base de datos

Por lo tanto, se ha dividido el proyecto en cuatro fases:

1. Definición del modelo de datos relacional. Identificación de las principales entidades y las relaciones entre estas.
2. Desarrollo del servidor de la aplicación.
3. Desarrollo de la interfaz de usuario de la aplicación.
4. Realización de pruebas para comprobar el correcto funcionamiento.

1.3 ESTADO DEL ARTE

En el ámbito de las transfusiones sanguíneas, se ha evidenciado una evolución significativa a lo largo del tiempo, marcada por avances tecnológicos que han transformado la práctica clínica. La introducción

de nuevas tecnologías ha revolucionado el proceso de transfusión, brindando oportunidades para mejorar la eficiencia, seguridad y calidad de este procedimiento vital en la medicina moderna. La automatización ha desempeñado un papel crucial en la optimización de las transfusiones sanguíneas, agilizando los procesos de selección, preparación y administración de componentes sanguíneos.

La centralización de bases de datos ha facilitado la trazabilidad de las unidades de sangre, garantizando su adecuado almacenamiento y distribución. Asimismo, la implementación de sistemas informáticos especializados ha permitido una gestión más eficiente de los recursos sanguíneos, asegurando la disponibilidad oportuna de sangre segura y compatible para los pacientes que la requieran.

En el contexto actual, la integración de robots en los procesos de transfusión sanguínea ha representado un avance significativo, mejorando la precisión y reduciendo el riesgo de errores humanos.

Además, los algoritmos de inteligencia artificial han demostrado su utilidad en la predicción de necesidades de sangre, la optimización de la asignación de recursos y la identificación de patrones que pueden mejorar la eficacia de las transfusiones.

Estos avances tecnológicos han sentado las bases para una práctica más segura, eficiente y personalizada en el ámbito de las transfusiones sanguíneas.

1.3.1 Tecnologías actuales y tendencias futuras en los procesos de transfusión de sangre

El campo de la medicina transfusional se está viendo impactado por los avances tecnológicos en RFID, IoT, blockchain e inteligencia artificial, los cuales están transformando la forma en que se lleva a cabo y se gestiona esta práctica médica. A continuación, se presentan actualizaciones sobre estos desarrollos destinados a mejorar la calidad, la eficacia y la seguridad de los procedimientos transfusionales.

La identificación por radiofrecuencia (RFID) utiliza ondas electromagnéticas para reconocer y rastrear etiquetas digitalmente codificadas en elementos. Por otro lado, el Internet de las Cosas (IoT) conecta objetos físicos con sensores y tecnología para compartir datos a través de internet. Los desafíos clave incluyen una transmisión de datos sin problemas, costo-efectividad y conectividad máquina a máquina o dispositivo a dispositivo. El volumen de datos manejados por dispositivos IoT está aumentando rápidamente, planteando preocupaciones de seguridad y privacidad. La integración de tecnología IoT en las transfusiones de sangre permite un monitoreo en tiempo real de las bolsas de sangre para garantizar su calidad y seguridad durante el transporte y almacenamiento. Además, el uso de RFID y sensores mejora la identificación precisa de pacientes y productos, reduciendo errores y aumentando la eficacia de los servicios transfusionales.

Blockchain es un marco revolucionario que ofrece un nuevo diseño para almacenar y compartir datos entre los miembros de una red. Las actualizaciones actuales y futuras de la tecnología blockchain en los servicios de transfusión incluyen mejoras en la trazabilidad y la seguridad de los datos, y la posibilidad de realizar un seguimiento preciso de las donaciones de sangre. Un sistema de transfusiones basado en blockchain ayuda a acceder, rastrear, gestionar y compartir información de salud y relacionada con la sangre, y proporciona un entorno seguro que sirve como un centro de comunicación entre donantes, receptores, médicos y laboratorios de pruebas.

La inteligencia artificial, el aprendizaje automático y las redes neuronales profundas son avances modernos en transfusión de sangre. Estas tecnologías se utilizan como herramientas de apoyo para mejorar la calidad y eficacia, y reducir errores en los servicios transfusionales. Las actualizaciones recientes incluyen sistemas de apoyo a la toma de decisiones y la optimización de procesos transfusionales, como son las plataformas inteligentes de gestión de bancos de sangre basada en datos que pueden reducir la incertidumbre de la demanda de sangre al anticipar la recolección y demanda de sangre, así como disminuir el desperdicio y la escasez mediante el equilibrio entre donaciones y distribución, basándose en una gestión óptima del inventario de sangre. Además, utiliza algoritmos inspirados en la biología como algoritmos genéticos y estrategias evolutivas para optimizar la programación de sesiones de donación de sangre que sean convenientes tanto para el banco de sangre como para el donante.

CAPÍTULO 2: DESCRIPCIÓN DE LAS TECNOLOGÍAS

En este segundo capítulo se exponen las diferentes tecnologías utilizadas para el desarrollo de este proyecto. Se analizarán las ventajas de utilizar cada tecnología tanto para el modelado de la base de datos como para el desarrollo del servidor y la interfaz de usuario.

2.1 Herramientas para el diseño del modelo de datos

El diseño de un modelo de datos eficaz es esencial para desarrollar aplicaciones robustas y bien estructuradas. Para lograr esto, es necesario contar con herramientas que faciliten la creación, gestión y visualización de esquemas de bases de datos. En este capítulo, se presentan dos herramientas clave para este propósito: DBeaver y MySQL.

2.1.1 DBeaver

DBeaver es una herramienta de administración de bases de datos que ofrece una interfaz gráfica intuitiva y versátil para la gestión de diversos sistemas de bases de datos. Su flexibilidad permite trabajar con una amplia gama de DBMS, incluyendo MySQL, PostgreSQL, y Oracle, entre otros. Entre sus principales características destacan las siguientes:

- **Modelado de datos:** Ofrece potentes herramientas para crear diagramas de entidad-relación (ERD). Estos diagramas ayudan a visualizar y diseñar el esquema de la base de datos, mostrando tablas, relaciones y restricciones de manera gráfica.
- **Gestión de datos:** Permite realizar tareas de administración como insertar, actualizar y eliminar registros, además de ejecutar consultas SQL complejas para gestionar datos de forma eficiente.
- **Soporte para múltiples DBMS:** La capacidad de trabajar con diferentes sistemas de gestión de bases de datos hace que DBeaver sea una herramienta versátil y útil en entornos variados.
- **Importación y exportación de datos:** Facilita la transferencia de datos entre sistemas y la creación de respaldos a través de opciones de importación y exportación en diversos formatos.

2.1.2 MySQL

MySQL es uno de los sistemas de gestión de bases de datos relacionales más conocidos y utilizados en todo el mundo. Desarrollado por Oracle Corporation, MySQL es apreciado por su rendimiento, fiabilidad y facilidad de uso. Entre sus principales características destacan las siguientes:

- **Modelo de datos relacional:** Utiliza un modelo de datos basado en tablas que se relacionan entre sí mediante claves primarias y foráneas, lo que permite una gestión estructurada y eficiente de la información.
- **Lenguaje SQL:** Emplea el lenguaje SQL (Structured Query Language) para definir y manipular datos. SQL facilita la ejecución de consultas complejas y la gestión de la estructura de la base de datos.
- **Transacciones y control de concurrencia:** Soporta transacciones ACID, garantizando la integridad y consistencia de los datos durante operaciones concurrentes.
- **Escalabilidad y rendimiento:** Diseñado para manejar grandes volúmenes de datos y un alto número de transacciones, MySQL es adecuado tanto para aplicaciones pequeñas como para grandes sistemas empresariales.
- **Interoperabilidad y soporte:** Compatible con diversas plataformas y lenguajes de programación, MySQL cuenta con una amplia comunidad que proporciona soporte y contribuciones continuas.

En resumen, DBeaver y MySQL son herramientas complementarias que juegan un papel crucial en el diseño y gestión de bases de datos. DBeaver proporciona una interfaz gráfica avanzada y herramientas de modelado, mientras que MySQL ofrece una plataforma robusta para la administración de datos

2.2 Herramientas para el desarrollo de la aplicación

El desarrollo de aplicaciones web modernas implica la integración de diversas tecnologías que permiten crear aplicaciones funcionales, eficientes y de alta calidad. En este apartado, se abordan tres herramientas fundamentales para el desarrollo de aplicaciones: Python, JavaScript, y las tecnologías de diseño web HTML y CSS.

2.2.1 Python

Python es un lenguaje de programación de alto nivel conocido por su simplicidad y flexibilidad, lo que lo convierte en una opción destacada para el desarrollo de aplicaciones web. Su sintaxis clara y su enfoque en la legibilidad del código facilitan la creación y el mantenimiento de aplicaciones robustas

y eficientes. Entre sus principales características destacan las siguientes:

- **Simplicidad y legibilidad:** Python es valorado por su sintaxis intuitiva, que permite a los desarrolladores escribir código de manera rápida y fácil de entender. Esta claridad en la codificación es crucial para el mantenimiento y la evolución de las aplicaciones a lo largo del tiempo.
- **Bibliotecas y frameworks:** Python dispone de una amplia gama de bibliotecas y frameworks que simplifican el proceso de desarrollo. *Flask* y *Django* son ejemplos prominentes. Flask es un microframework ligero que ofrece flexibilidad y control sobre los componentes de la aplicación, mientras que Django proporciona una solución más estructurada y completa para aplicaciones más grandes.
- **Interacción con bases de datos:** La integración con bases de datos se facilita mediante bibliotecas como *SQLAlchemy*, que permite a los desarrolladores trabajar con bases de datos mediante un enfoque de mapeo objeto-relacional (ORM). Esto simplifica la gestión de datos y mejora la mantenibilidad del código.
- **Desarrollo del** ***backend*****:** Python es ampliamente utilizado en el desarrollo del lado del servidor (*backend*). Su capacidad para manejar lógica de negocio, procesamiento de datos y comunicación con bases de datos lo convierte en una herramienta clave para la creación de aplicaciones web dinámicas y escalables.

2.2.2 Javascript

JavaScript es un lenguaje de programación de alto nivel que juega un papel crucial en el desarrollo web, especialmente en el ámbito del *frontend*. Su capacidad para interactuar de manera directa con el navegador y manipular el contenido de las páginas web en tiempo real lo convierte en una herramienta indispensable para la creación de interfaces de usuario interactivas y dinámicas. En este apartado, se analizarán las características fundamentales de JavaScript que lo consolidan como una tecnología esencial en el desarrollo *frontend*. Entre sus principales características, se destacan las siguientes:

- **Interactividad y dinamismo:** Una de las principales ventajas de JavaScript es su habilidad para añadir interactividad y dinamismo a las páginas web mediante la manipulación del Document Object Model (DOM). Esta manipulación permite modificar el contenido y la estructura de la página de forma dinámica, en respuesta a las acciones del usuario, como clics de botones, desplazamientos y entradas de teclado. Gracias a esta capacidad, es posible actualizar el contenido de la página sin necesidad de recargarla completamente, mejorando así la fluidez y la experiencia del usuario. Además, JavaScript facilita la validación de formularios,

el manejo de eventos y la creación de animaciones y transiciones suaves, aspectos fundamentales para una experiencia de usuario atractiva y funcional.

- **Frameworks:** El ecosistema de JavaScript es vasto y diverso, lo cual se refleja en la amplia disponibilidad de bibliotecas y *frameworks* que simplifican y potencian el desarrollo de aplicaciones web. Entre estos, *React* se destaca como una de las bibliotecas más populares y ampliamente utilizadas para la construcción de interfaces de usuario. React permite a los desarrolladores crear componentes reutilizables, modularizando el código y facilitando su mantenimiento y escalabilidad. Además, React incluye un eficiente sistema de gestión del estado, lo que resulta fundamental para manejar la complejidad de las aplicaciones modernas.
- **Asincronía y rendimiento:** JavaScript soporta la programación asíncrona, una característica clave para mejorar la eficiencia y la capacidad de respuesta de las aplicaciones web. A través del uso de promesas y las funciones *async/await*, JavaScript permite la ejecución de tareas en segundo plano, como la obtención de datos de servidores externos, sin bloquear la interfaz de usuario. Esta capacidad es esencial para mantener una experiencia de usuario fluida y sin interrupciones, especialmente en aplicaciones que requieren realizar múltiples operaciones simultáneamente.

2.2.3 HTML y CSS aplicados al desarrollo web

HTML (*HyperText Markup Language*) y CSS (*Cascading Style Sheets*) son dos tecnologías fundamentales para la creación y diseño de páginas web. Mientras que HTML proporciona la estructura y el marco básico del contenido de una página, CSS se encarga de su estilo y presentación visual. Ambas tecnologías trabajan de manera conjunta para ofrecer una experiencia de usuario cohesiva y estética, adaptándose a las demandas de diseño y funcionalidad del desarrollo web moderno. Entre las principales características de HTML, se destacan las siguientes:

- **Estructuración del contenido**: HTML es el lenguaje de marcado estándar utilizado para definir la estructura del contenido de las páginas web. Utiliza una serie de etiquetas que permiten organizar la información de manera jerárquica y lógica. Estas etiquetas incluyen elementos para definir encabezados, párrafos, listas, enlaces e imágenes, entre otros, lo que facilita la organización y la claridad del contenido. La correcta estructuración con HTML es fundamental para asegurar que las páginas web sean comprensibles tanto para los usuarios como para los motores de búsqueda, mejorando así su accesibilidad y usabilidad.

- **Semántica y accesibilidad**: Con la introducción de HTML5, se han mejorado significativamente las capacidades semánticas del lenguaje. Se han añadido etiquetas semánticas específicas como <*header*>, <*article*>, <*nav*>, y <*footer*>, que proporcionan una descripción más clara y precisa de la función de cada sección de la página. Estas etiquetas no solo mejoran la estructura del código y la organización del contenido, sino que también incrementan la accesibilidad al permitir que los lectores de pantalla y otros dispositivos de asistencia comprendan mejor la disposición y función del contenido. Además, estas etiquetas ayudan a los motores de búsqueda a indexar y clasificar mejor las páginas, mejorando así la visibilidad y la optimización en buscadores (SEO).

Entre las principales características de CSS, se destacan las siguientes:

- **Estilo y presentación**: CSS es el lenguaje utilizado para describir la presentación visual de un documento escrito en HTML. Permite a los desarrolladores aplicar estilos a los elementos HTML, controlando aspectos como el color, la tipografía, el espaciado, los márgenes y la disposición de los elementos en la página. Esta capacidad de separar el contenido (HTML) de su presentación (CSS) proporciona una mayor flexibilidad en el diseño web y facilita la actualización y el mantenimiento de los estilos. Gracias a CSS, es posible mantener la consistencia visual a través de múltiples páginas de un sitio web, asegurando una experiencia de usuario uniforme y profesional.
- **Diseño responsivo**: Una de las características más importantes de CSS es su capacidad para soportar el diseño responsivo, esencial en la era de los dispositivos móviles. CSS proporciona herramientas como los media queries, que permiten definir estilos específicos para diferentes tamaños de pantalla y resoluciones. Esto asegura que las páginas web se adapten y se vean bien en una variedad de dispositivos, desde teléfonos móviles y tabletas hasta computadoras de escritorio. El diseño responsivo es crucial para ofrecer una experiencia de usuario óptima y accesible, independientemente del dispositivo que se utilice para acceder al sitio web.
- **Animaciones y transiciones**: CSS también incluye capacidades avanzadas para la creación de animaciones y transiciones, lo que añade un nivel adicional de interactividad y atractivo visual a las páginas web. Mediante el uso de propiedades de animación y transición, es posible crear efectos visuales que mejoran la experiencia del usuario al interactuar con la interfaz. Estas animaciones pueden variar desde simples cambios de color hasta transformaciones complejas de elementos, contribuyendo a una experiencia de navegación más dinámica y atractiva.

Finalmente, en la *Ilustración 2* se muestra la arquitectura software para el desarrollo de la aplicación con las diferentes tecnologías utilizadas:

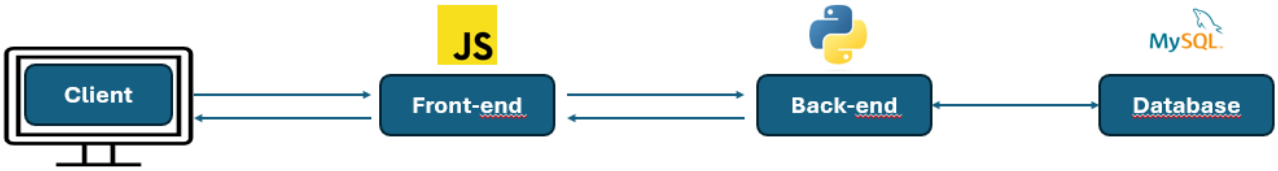


Ilustración 2: Arquitectura software

CAPÍTULO 3: DISEÑO DEL MODELO LÓGICO

En este tercer capítulo se explica cómo se ha llevado a cabo el modelo lógico de la aplicación. Se explicarán las principales entidades de la base de datos, así como su propósito en la aplicación y las relaciones entre las diferentes entidades.

3.1 Diseño del modelo de datos

El diseño del modelo de datos para el sistema de banco de sangre se ha desarrollado con el objetivo de gestionar de manera eficiente la información relacionada con donantes, donaciones, inventarios de sangre, pacientes, transfusiones y hospitales. Este modelo está compuesto por un total de nueve tablas, cada una de las cuales representa una entidad clave dentro del sistema. A continuación, se presenta una descripción detallada de cada una de estas tablas, junto con sus atributos y el propósito que cumplen en el modelo de datos.

3.1.1 Estructura del modelo: Entidades

A continuación, se presenta una descripción detallada de cada una de las entidades del modelo, así como sus principales atributos y el propósito que cumplen en el modelo de datos:

1. ***Donor* (Donante):**
   - **Atributos**: *donor_id*, *name* (nombre), *last_name* (apellido), *birth_date* (fecha de nacimiento), *blood_group_id* (ID del grupo sanguíneo), factor_rh_id (ID del factor Rh), *last_donation_date* (fecha de última donación).
   - **Descripción**: Esta tabla almacena información detallada sobre cada donante registrado en el sistema. Incluye datos personales, como el nombre y la fecha de nacimiento, así como información médica crítica, como el grupo sanguíneo y el factor Rh. El *last_donation_date* permite rastrear cuándo fue la última vez que el donante realizó una donación, lo cual es importante para cumplir con las normas de donación de sangre.
2. ***Donation* (Donación):**
   - **Atributos**: *donation_id*, *donor_id* (ID del donante), date (fecha de la donación), *blood_bank_id* (ID del banco de sangre).
   - **Descripción**: Esta tabla registra cada evento de donación de sangre realizado por los donantes. Almacena información sobre quién realizó la donación (*donor_id*), cuándo se realizó (*date*) y en qué banco de sangre tuvo lugar (*blood_bank_id*). Esta tabla es esencial

para mantener un historial completo de las donaciones y para gestionar el flujo de entrada de unidades de sangre en el sistema.

3. ***Blood Bank*** **(Banco de Sangre):**
   - **Atributos**: *bank_id*, *name* (nombre), *address* (dirección), *city* (ciudad).
   - **Descripción**: Representa las instalaciones donde se almacenan y procesan las donaciones de sangre. Esta tabla contiene información sobre cada banco de sangre, incluyendo su nombre, dirección y ciudad. Es fundamental para la gestión logística de las donaciones y transfusiones, así como para la coordinación de campañas de donación.
4. **Blood Bank Stock (Inventario del Banco de Sangre):**
   - **Atributos**: *stock_id*, *blood_bank_id* (ID del banco de sangre), *blood_group_id* (ID del grupo sanguíneo), factor_rh_id (ID del factor Rh), *quantity* (cantidad).
   - **Descripción**: Esta tabla se utiliza para mantener un registro actualizado del inventario de sangre disponible en cada banco de sangre. Clasifica las unidades de sangre por grupo sanguíneo y factor Rh, y registra la cantidad disponible de cada tipo. Esto permite a los bancos de sangre gestionar su stock de manera efectiva y responder rápidamente a las necesidades de transfusión.
5. ***Blood Group*** **(Grupo Sanguíneo):**
   - **Atributos**: blood_group_id, type (tipo de grupo sanguíneo), description (descripción).
   - **Descripción**: Esta tabla define los diferentes tipos de grupos sanguíneos disponibles, como A, B, AB y O. También puede incluir una descripción de cada tipo. Es fundamental para clasificar tanto a los donantes como a las unidades de sangre, asegurando que las transfusiones sean seguras y compatibles.
6. **Factor Rh:**
   - **Atributos**: *factor_rh_id*, *type* (tipo de factor Rh, positivo o negativo), *description* (descripción).
   - **Descripción**: Similar a la tabla de grupos sanguíneos, esta tabla se utiliza para clasificar el factor Rh de los donantes y las unidades de sangre (positivo o negativo). El factor Rh es un componente crítico en la compatibilidad de la sangre para transfusiones.
7. ***Patient*** **(Paciente):**

- **Atributos**: *patient_id*, *name* (nombre), *last_name* (apellido), *group_id* (grupo sanguíneo), *factor_rh_id* (factor Rh), *hospital_id* (ID del hospital).
- **Descripción**: Esta tabla contiene información sobre los pacientes que requieren transfusiones de sangre. Incluye datos personales, el grupo sanguíneo y el factor Rh del paciente, y el hospital donde se encuentra registrado. Este registro es esencial para garantizar que las transfusiones sean compatibles y para coordinar la atención médica con los hospitales.

8. ***Transfusion*** **(Transfusión):**
   - **Atributos**: *transfusion_id*, *donor_id* (ID del donante), *patient_id* (ID del paciente), *processing_date* (fecha de procesamiento), *status* (estado de la transfusión).
   - **Descripción**: Esta tabla registra cada evento de transfusión, conectando donantes con pacientes. Almacena detalles como el donante que proporcionó la sangre, el paciente que la recibió, la fecha de procesamiento de la transfusión y el estado actual de la transfusión (por ejemplo, completada, pendiente). Es crucial para el seguimiento de cada unidad de sangre desde su donación hasta su uso final.
9. **Hospital:**
   - **Atributos**: *hospital_id*, *name* (nombre), *address* (dirección), *city* (ciudad).
   - **Descripción**: Esta tabla representa los hospitales donde se llevan a cabo transfusiones de sangre y donde se atiende a los pacientes. Incluye información básica sobre el hospital, como su nombre, dirección y ciudad. Permite gestionar y coordinar las transfusiones entre los bancos de sangre y las instalaciones médicas.

3.1.1 Estructura del modelo: Relaciones

Una vez explicadas todas las entidades del modelo de datos, a continuación, se procede a la explicación de las relaciones existentes entre las diferentes entidades, definiéndose las diferentes *primary_keys* y *foreign_keys.*

1. **Relación entre *Donor* y *Donation***: Se establece una relación uno a muchos entre *donor* y *donation*, ya que un donante puede realizar múltiples donaciones a lo largo del tiempo. La clave foránea *donor_i*d en la tabla *donation* se utiliza para vincular cada donación con su donante correspondiente.

2. **Relación entre Donation y Blood Bank**: Cada donación se realiza en un banco de sangre

específico. Esta relación se define utilizando la clave foránea *blood_bank_id* en la tabla *donation*, que apunta a la tabla *blood_bank*. Esto permite rastrear en qué banco se realizó cada donación.

3. **Relación entre Blood Bank y Blood Bank Stock**: Se establece una relación uno a muchos entre *blood_bank* y *blood_bank_stock*, ya que cada banco de sangre puede tener múltiples tipos de sangre en su inventario. La clave foránea *blood_bank_id* en la tabla *blood_bank_stock* asegura que cada registro de inventario esté vinculado a un banco de sangre específico.
4. **Relación entre Donor y Blood Group**: Se utiliza una relación muchos a uno entre *donor* y *blood_group* para clasificar a los donantes según su tipo de sangre. La clave foránea *blood_group_id* en la tabla *donor* facilita esta clasificación, garantizando que cada donante esté asociado con un tipo de sangre específico.
5. **Relación entre Donor y Factor Rh**: Similar a la relación con el grupo sanguíneo, la relación entre *donor* y *factor_rh* clasifica a los donantes por su factor Rh. La clave foránea *factor_rh_id* en la tabla *donor* asegura que cada donante tenga un factor Rh asignado (positivo o negativo).
6. **Relación entre Blood Bank Stock, Blood Group y Factor Rh**: La tabla *blood_bank_stock* se conecta con *blood_group* y *factor_rh* mediante claves foráneas, permitiendo clasificar el inventario de sangre por tipo y factor Rh. Esto facilita la gestión del stock de sangre y asegura la compatibilidad en las transfusiones.
7. **Relación entre Transfusion, Donor y Patient**: La tabla *transfusion* establece una relación entre *donor* y *patient* para rastrear qué donante proporcionó sangre a qué paciente. Las claves foráneas *donor_id* y *patient_id* en la tabla transfusion permiten realizar un seguimiento detallado de las transfusiones.
8. **Relación entre Patient y Hospital**: Se define una relación muchos a uno entre *patient* y *hospital*, donde cada paciente está asociado a un hospital específico. La clave foránea *hospital_id* en la tabla *patient* asegura que cada paciente esté registrado en un hospital particular.
9. **Relación entre Patient y Blood Group, Factor Rh**: Al igual que los donantes, cada paciente tiene un grupo sanguíneo y un factor Rh. Estas relaciones se establecen mediante las claves foráneas *group_id* y *factor_rh_id* en la tabla *patient*, garantizando que se realicen transfusiones seguras y compatibles.

Una vez explicadas tanto las entidades como las relaciones entre ellas, a continuación, en la *Ilustración* 3 se muestra el diagrama lógico resultante:

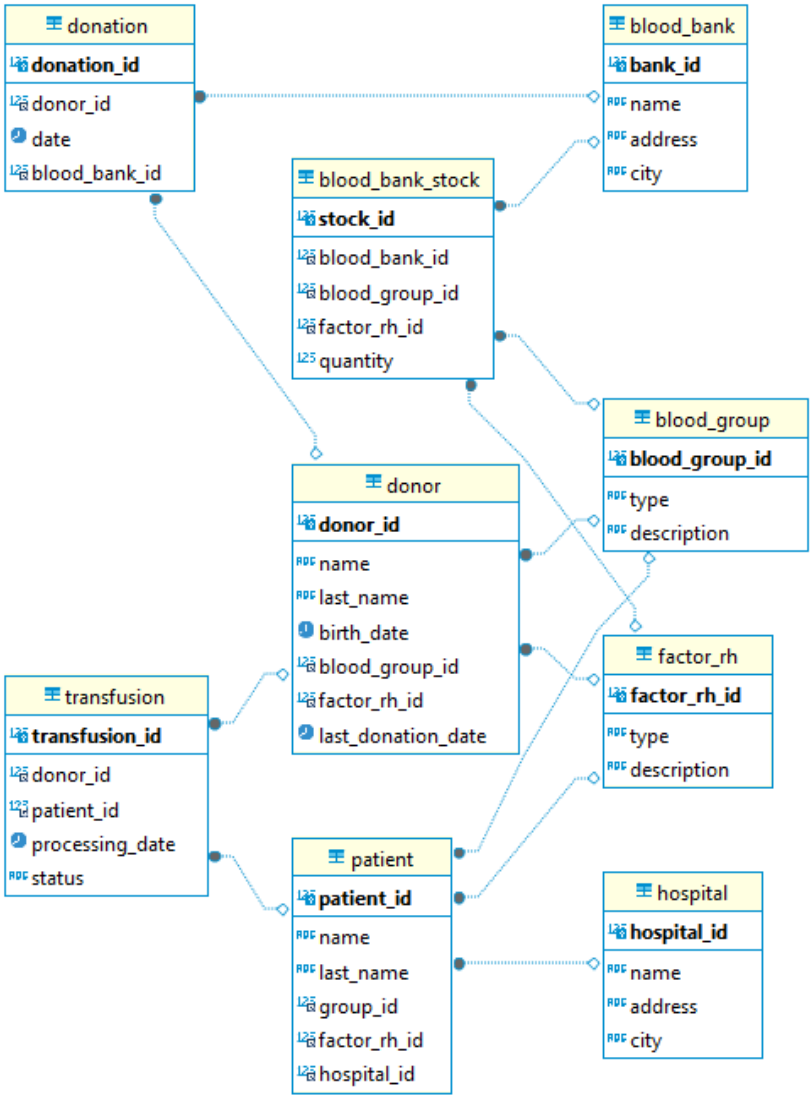


Ilustración 3: Modelo lógico resultante

CAPÍTULO 4: DESARROLLO DEL SERVIDOR

En este cuarto capítulo se explica el desarrollo del servidor de la aplicación, donde reside la lógica de esta. A su vez, se explicarán las diferentes librerías utilizadas y los principales *endpoints* desarrollados para cada una de las entidades.

4.1 Desarrollo del servidor

En este apartado se explica cómo se ha llevado a cabo el desarrollo del servidor donde reside la lógica de la aplicación. El desarrollo de este se ha dividido en tres partes:

1. Configuración: Se han definido las principales dependencias y las configuraciones para conectarse a la base de datos.
2. *Models* (Modelos): La carpeta Models hace referencia a los diferentes archivos para la creación del modelo lógico utilizando SQLAlchemy.
3. *Routes* (Rutas): En esta carpeta se almacenan las peticiones HTTP que se realizarán desde el servidor hacía la base de datos. Para cada una de las entidades del modelo, existen diferentes peticiones para la obtención, inserción, actualización y eliminación de datos.

A continuación, en la *Ilustración 4* se puede observar cómo se encuentra estructurado el servidor

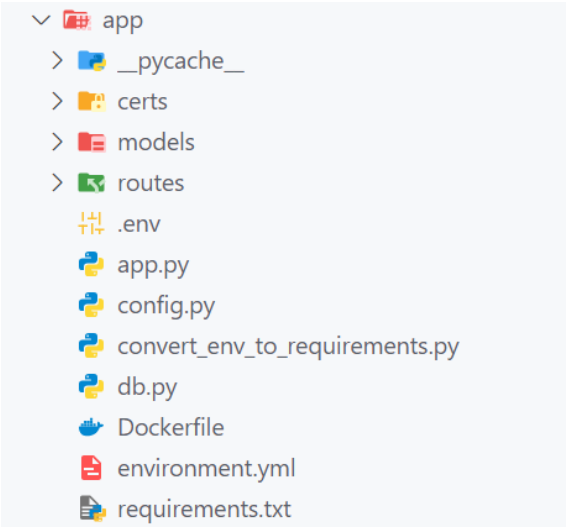


Ilustración 4: Carpetas del servidor

4.1.1 Desarrollo del servidor: Librerías utilizadas

Para el desarrollo del servidor, se han utilizado tres principales librerías: Flask, SQLAlchemy y MySQLconnection. A continuación, se explica en detalle cada una de ellas.

- **Flask**

Flask es un microframework para Python que facilita el desarrollo de aplicaciones web. Diseñado con la filosofía de "hacer una cosa y hacerla bien", Flask proporciona las herramientas esenciales para construir aplicaciones web de manera eficiente, sin imponer una estructura rígida. En su núcleo, Flask incluye un servidor de desarrollo integrado, soporte para enrutamiento de URL y una sencilla interfaz para gestionar las solicitudes HTTP. Esta simplicidad permite a los desarrolladores tener un control preciso sobre los componentes de su aplicación y, a su vez, favorece la extensión de sus funcionalidades mediante la integración de bibliotecas adicionales.

- **MySQLConnection**

MySQLConnection es un módulo que proporciona una interfaz para conectar y operar con bases de datos MySQL desde aplicaciones Python. Este módulo permite la conexión a una base de datos MySQL, así como la ejecución de consultas SQL y la manipulación de los datos retornados. A través de MySQLConnection, los desarrolladores pueden gestionar conexiones, realizar operaciones CRUD (crear, leer, actualizar y eliminar) y manejar transacciones dentro de la base de datos. El uso de MySQLConnection se destaca por su capacidad para interactuar directamente con MySQL sin la necesidad de un ORM (Object-Relational Mapping), lo que puede ser beneficioso para aplicaciones que requieren un control más granular sobre las operaciones SQL.

- **SQLAlchemy**

SQLAlchemy es una biblioteca de ORM para Python que proporciona una capa de abstracción para interactuar con bases de datos relacionales. A diferencia de MySQLConnection, SQLAlchemy permite a los desarrolladores trabajar con bases de datos a un nivel más alto de abstracción mediante el uso de objetos y clases, en lugar de escribir consultas SQL manualmente. SQLAlchemy se compone de dos componentes principales: el Core, que ofrece una interfaz de bajo nivel para ejecutar consultas SQL, y el ORM, que permite mapear clases Python a tablas de bases de datos y gestionar las relaciones entre ellas de manera automática. Esta capacidad para manejar la persistencia de objetos en bases de datos facilita el desarrollo de aplicaciones complejas y la implementación de patrones de diseño orientados a objetos.

Para la gestión de las diferentes librerías y asegurar una correcta gestión de versiones, se ha decidido crear un ecosistema de Conda con las diferentes librerías instaladas (paquetes). Se ha utilizado Conda ya que este permite la instalación de paquetes y bibliotecas con una amplia gama de versiones y dependencias específicas. A continuación, en la *Ilustración 5* se muestra cómo se gestionan las bibliotecas utilizando Conda.

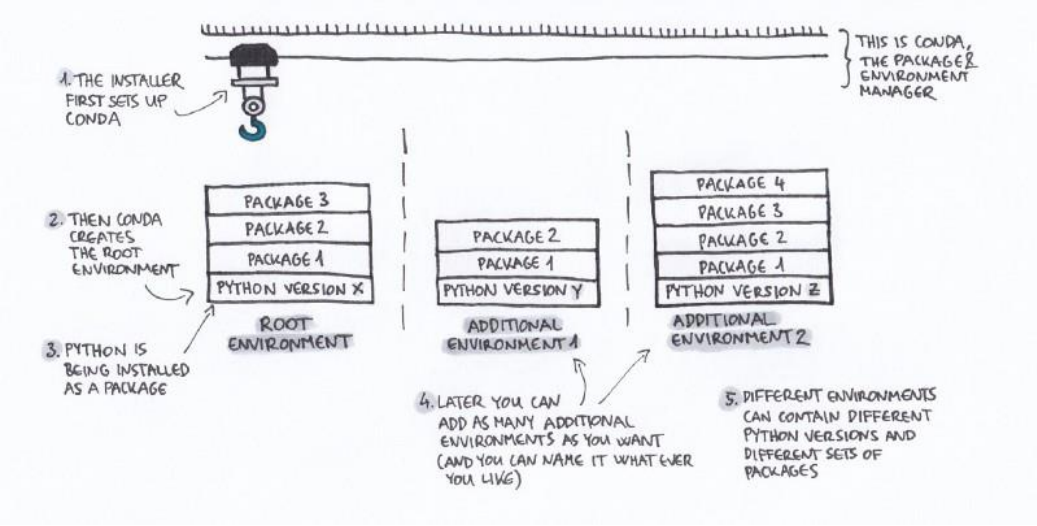


Ilustración 5: Conda para la gestión de librerías

4.1.2 Desarrollo del servidor: Endpoints

Para cada una de las entidades del modelo de datos, se han desarrollado diferentes rutas HTTP. De esta forma, se pueden introducir datos a la base de datos mediante peticiones *post*, obtener datos mediante peticiones *get*, actualizar datos mediante peticiones *put* y borrar datos mediante peticiones *delete*.

Al tratarse la aplicación de una *API REST*, las peticiones CRUD (Create, Read, Update, Delete) son fundamentales para la gestión de recursos. Estas peticiones se construyen utilizando el protocolo HTTP y se componen de varias partes esenciales que permiten la interacción adecuada entre el cliente y el servidor. A continuación, se detallan las partes principales de una petición CRUD:

- **Método HTTP**

  El método HTTP especifica la acción que se desea realizar sobre un recurso. Los métodos CRUD más comunes son:

  - ***POST***: Se utiliza para crear un nuevo recurso. Este método indica al servidor que debe agregar un nuevo elemento a la colección de recursos.
  - ***GET***: Este método es utilizado para recuperar información o leer un recurso existente.

Es una operación de solo lectura y no debe tener efectos secundarios.

- ***PUT***: Se emplea para actualizar un recurso existente de manera completa. Si el recurso no existe, en algunos casos, puede crear uno nuevo.
- ***PATCH***: Similar a PUT, pero se utiliza para realizar actualizaciones parciales en un recurso existente.
- ***DELETE***: Este método se utiliza para eliminar un recurso existente del servidor.

- **URL (*Uniform Resource Locator*)**

La URL define la dirección del recurso específico al que se está accediendo o manipulando. Es un identificador único que indica al servidor el recurso sobre el cual se desea operar.

- ***Headers* (Encabezados)**

Los encabezados de la petición contienen metadatos sobre la petición misma y sobre el contenido que se está enviando o esperando recibir. Algunos de los encabezados más comunes incluyen:

- ***Content-Type***: Especifica el formato de los datos enviados en el cuerpo de la petición. Comúnmente, este valor es application/json para datos en formato JSON.
- ***Authorization***: Proporciona las credenciales necesarias para autenticar la petición, como un token de acceso o una API key.
- ***Accept***: Indica al servidor el tipo de contenido que el cliente puede procesar, lo que facilita la negociación de contenido entre cliente y servidor.
- ***User-Agent***: Informa sobre el cliente que realiza la petición, incluyendo información sobre la versión del navegador o aplicación utilizada.
- ***Cache-Control***: Permite al cliente o servidor controlar la forma en que las respuestas pueden ser almacenadas en caché.

- ***Body* (Cuerpo de la Petición)**

El cuerpo de la petición contiene los datos que se envían al servidor en las operaciones que lo requieren, como en las peticiones POST, PUT o PATCH. Estos datos generalmente están estructurados en formato JSON o XML, y contienen la información necesaria para crear o actualizar un recurso.

- ***Query Parameters* (Parámetros de Consulta)**

Los parámetros de consulta son pares clave-valor que se añaden a la URL para proporcionar

filtros adicionales, ordenamiento, o paginación de los resultados. Estos parámetros se incluyen después de un signo de interrogación (?) en la URL y se separan por un signo igual (=) y ampersand (&).

- **Path Parameters (Parámetros de Ruta)**

Los parámetros de ruta son partes de la URL que varían para identificar un recurso específico. Estos parámetros se usan comúnmente en peticiones que apuntan a recursos individuales dentro de una colección.

- **Response Status Codes (Códigos de Estado de Respuesta)**

Aunque no forman parte de la petición en sí, los códigos de estado de respuesta son cruciales para entender el resultado de una operación CRUD. Estos códigos, enviados por el servidor, indican si la operación fue exitosa o si hubo algún error. Algunos de los códigos más comunes incluyen:

  - **200 OK**: La petición fue exitosa y se devolvió el recurso solicitado.
  - **201 Created**: Un nuevo recurso ha sido creado satisfactoriamente en respuesta a una petición POST.
  - **400 Bad Request**: La petición no pudo ser procesada debido a una sintaxis incorrecta o parámetros inválidos.
  - **401 Unauthorized**: La autenticación es necesaria y ha fallado o no ha sido proporcionada.
  - **404 Not Found**: El recurso solicitado no pudo ser encontrado.
  - **500 Internal Server Error**: Se ha producido un error en el servidor al procesar la petición.

A continuación, se muestra un ejemplo de las peticiones que se han desarrollado para una entidad del modelo de datos (*Patient*).

La siguiente petición tiene como función obtener y devolver una lista de todos los pacientes almacenados en la base de datos.

```
@patient_bp.route('/', methods=['GET'])
def get_patients():
    patients = session.query(Patient).all()
    return jsonify([patient.to_dict() for patient in patients])
```

A continuación, se muestra una petición que tiene como objetivo devolver el paciente seleccionado a través del *patient_id*. Luego, se obtienen detalles adicionales sobre el grupo sanguíneo y el factor Rh del paciente utilizando *group_id* y *factor_rh_id* de la tabla *Patient*, que se relacionan con las tablas

BloodGroup y *FactorRh*.

```
@patient_bp.route('/<int:patient_id>', methods=['GET'])
def get_patient(patient_id):
    patient = session.query(Patient).get(patient_id)
    patient_blood_group = session.query(BloodGroup).get(patient.group_id).type
    patient_factor_rh= session.query(FactorRh).get(patient.factor_rh_id).type
    patient_dict = {
        'name': patient.name,
        'last_name': patient.last_name,
        'patient_blood_group': patient_blood_group,
        'patient_factor_rh':  patient_factor_rh,
    }
    if not patient:
        return jsonify({'error': 'Patient not found'}), 404
    return jsonify(patient_dict)
```

Por último, se muestra una petición de tipo *post*, cuyo objetivo es introducir un nuevo paciente en la base de datos a partir de sus atributos, que conforman el *body* de la petición HTTP.

```
@patient_bp.route('/', methods=['POST'])
def create_patient():
    data = request.json
    name =  data.get('name')
    last_name = data.get('last_name')
    group_id = data.get('blood_group_id')
    factor_rh_id = data.get('factor_rh_id')
    hospital_id = data.get('hospital_id')

    # Validar los campos requeridos
    if not name or not last_name or not group_id or not factor_rh_id or not hospital_id:
        return jsonify({'error': 'Missing required fields'}), 400

    # Crear una nueva instancia de Donation
    new_patient = Patient(
        name = name,
        last_name = last_name,
        group_id=group_id,
        factor_rh_id=factor_rh_id,
        hospital_id = hospital_id
    )

    # Agregar la nueva donación a la sesión y confirmar
    session.add(new_patient)
    session.commit()

    return jsonify(new_patient.to_dict()), 201
```

4.2 Pruebas de la aplicación

En este apartado, se describen las pruebas realizadas en la aplicación, específicamente en el backend, utilizando la herramienta Postman para enviar peticiones HTTP. Estas pruebas son fundamentales para asegurar que la API REST funcione correctamente, cumpla con los requisitos establecidos y maneje adecuadamente los diferentes casos de uso y posibles errores.

Postman es una herramienta ampliamente utilizada para probar y desarrollar APIs. Permite enviar

peticiones HTTP de diferentes tipos (GET, POST, PUT, DELETE) y observar las respuestas del servidor. En este proyecto, se ha utilizado Postman para verificar la correcta funcionalidad de la API del backend, evaluando las operaciones CRUD (Create, Read, Update, Delete) implementadas en los diferentes endpoints.

4.2.1 Prueba de petición de todos los pacientes

El objetivo de esta prueba es verificar que el *endpoint* de recuperación de todos los pacientes (GET /patients) devuelve correctamente una lista de pacientes en formato JSON. A continuación, en la *Ilustración 6* se puede observar como se realiza la petición a la dirección HTTP: *http://127.0.0.1:5000/patients*

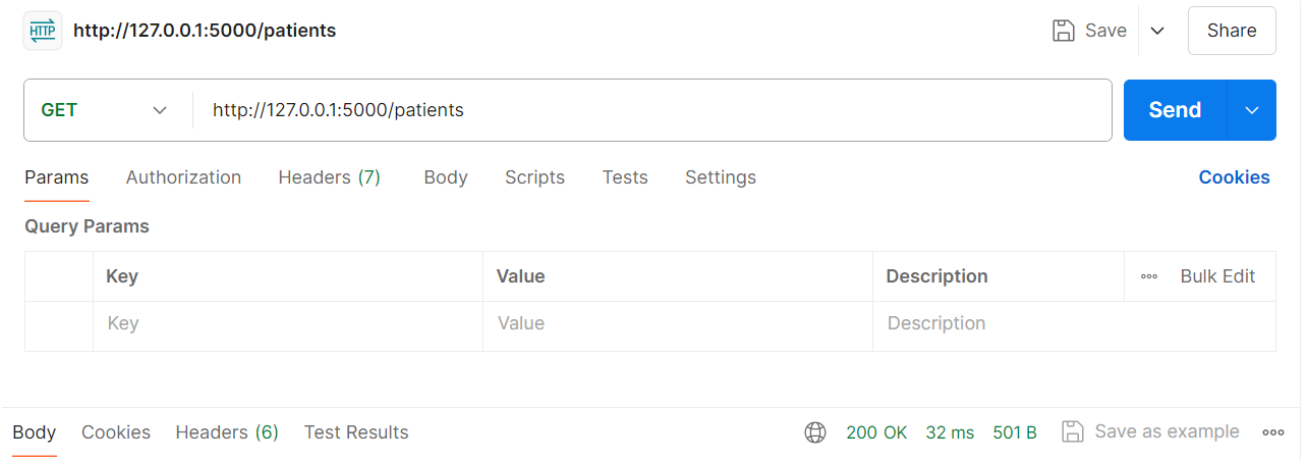


Ilustración 6: Petición de todos los pacientes

Una vez realizada la petición, se observa como el servidor se encuentra funcionando correctamente ya que se recibe un JSON con todos los pacientes disponibles en la base de datos.


```
[
    {
        "factor_rh_id": 1,
        "group_id": 7,
        "hospital_id": 2,
        "last_name": "Williams",
        "name": "Bob",
        "patient_id": 2
    },
    {
        "factor_rh_id": 1,
        "group_id": 1,
        "hospital_id": 1,
        "last_name": "Glez",
        "name": "Pablo",
        "patient_id": 4
    }
]
```

4.2.2 Prueba de petición de todos los donantes

El objetivo de esta prueba es verificar que el *endpoint* de recuperación de todos los donantes (GET /donors) devuelve correctamente una lista de pacientes en formato JSON. A continuación, en la *Ilustración 7* se puede observar cómo se realiza la petición a la dirección HTTP: *http://127.0.0.1:5000/donors*

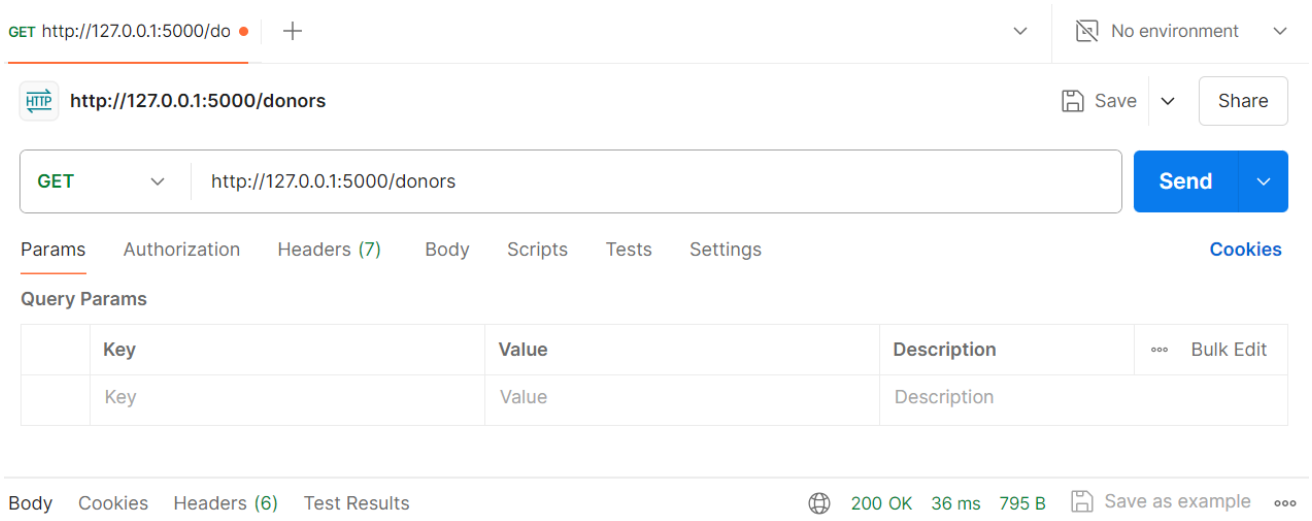


Ilustración 7: Petición de todos los donantes

Una vez realizada la petición, se observa como el servidor se encuentra funcionando correctamente ya que se recibe un JSON con todos los pacientes disponibles en la base de datos.


```
[
    {
        "birth_date": "1980-01-01",
        "blood_group_id": 1,
        "donor_id": 1,
        "factor_rh_id": 1,
        "last_donation_date": "2023-01-01",
        "last_name": "Doe",
        "name": "John"
    },
    {
        "birth_date": "1985-03-03",
        "blood_group_id": 7,
        "donor_id": 3,
        "factor_rh_id": 1,
        "last_donation_date": "2023-03-01",
        "last_name": "Brown",
        "name": "Jim"
    },
]
```

4.2.3 Prueba de petición de un paciente individual

El objetivo de esta prueba es verificar que el *endpoint* de recuperación de un paciente (GET /patient/patient_id) devuelve correctamente un paciente en formato JSON. A continuación, en la *Ilustración 8* se puede observar cómo se realiza la petición a la dirección HTTP: http://127.0.0.1:5000/patients/2 . En este caso, se quiere obtener información acerca del paciente con número de ID 2.

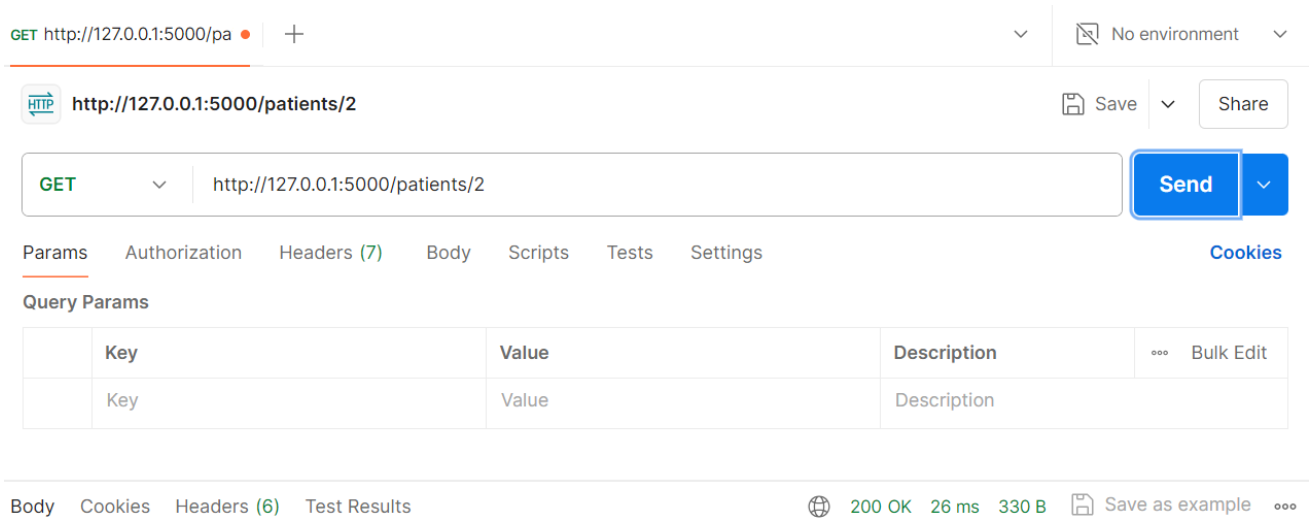


Ilustración 8: Petición de un paciente dado el id

Una vez realizada la petición, se puede observar como el estado es 200 OK, significando que la petición se ha llevado a cabo correctamente y se ha devuelto el paciente con número ID 2.


```
{
    "last_name": "Williams",
    "name": "Bob",
    "patient_blood_group": "O",
    "patient_factor_rh": "Positive"
}
```


4.2.4 Prueba de inserción de un paciente

Una vez completadas las pruebas de petición de datos, se procede a comprobar el correcto funcionamiento del servidor cuando se quieren insertar datos en la base de datos. Para ello, como en los anteriores casos, haremos uso de Postman para realizar una petición de tipo POST. Se introducirá un nuevo paciente a la ruta /patients. Para ello, es necesario incluir la información del paciente en el

cuerpo de la petición (body). Se puede observar en la *Ilustración 9* como una vez enviada la petición, el servidor devuelve el status 201 CREATED, completando la creación del nuevo paciente.

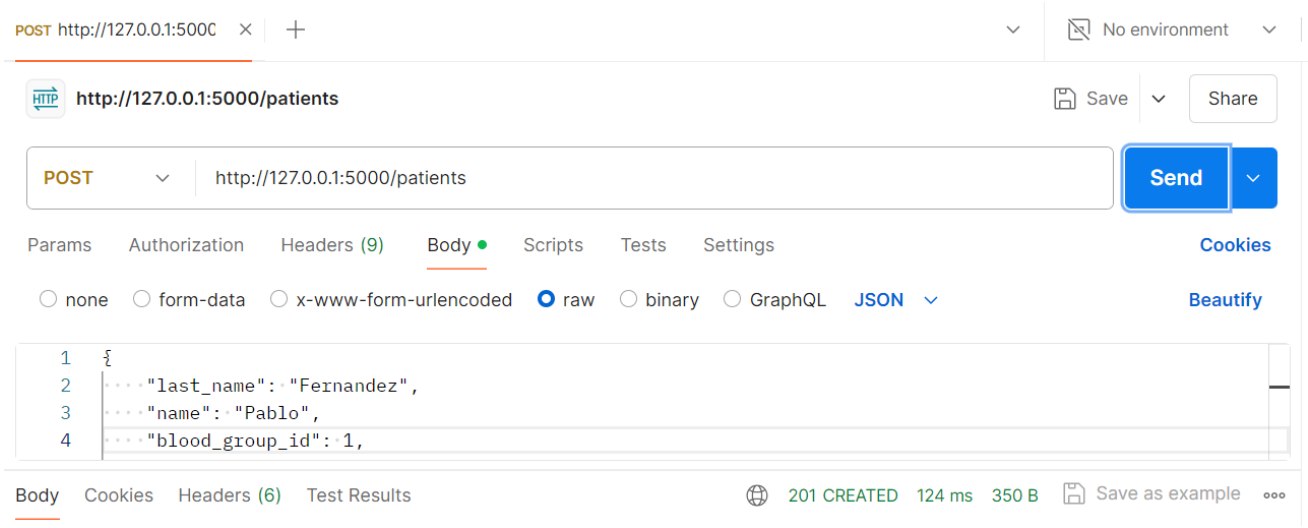


Ilustración 9: Petición de inserción de un paciente

CAPÍTULO 5: DESARROLLO DE LA INTERFAZ DE USUARIO

En este cuarto capítulo se explica el desarrollo del servidor de la aplicación, donde reside la lógica de esta. A su vez, se explicarán las diferentes librerías utilizadas y los principales *endpoints* desarrollados para cada una de las entidades.

5.1 Desarrollo de la interfaz de usuario (*front-end*)

En este apartado se explica cómo se ha llevado a cabo el desarrollo de la interfaz de usuario donde el cliente final podrá operar con la aplicación. Se ha tratado de desarrollar una interfaz que sea fácil de utilizar y funcional. El desarrollo de esta se ha dividido en tres partes:

1. Archivo app.js: Archivo donde se ha definido el diseño de la aplicación (layout) y desde donde se importan el resto de componentes (e.g. tablas).
2. Carpeta *blood_banks*: En esta carpeta se pueden encontrar diferentes archivos javascript en relación con las entidades relacionadas con los bancos de sangre (donadores, bancos de sangre, almacenamiento etc)
3. Carpeta *hospitals*: Por otro lado, en esta carpeta se pueden encontrar archivos

5.1.1 Desarrollo de la interfaz de usuario: Librerías utilizadas

Para el desarrollo de la interfaz de usuario, se han empleado dos principales librerías: React.js y AntDesign. A continuación, se detallan las características y el principal propósito de cada librería en el desarrollo de la aplicación.

React

React es una biblioteca de JavaScript de código abierto, desarrollada por Facebook, que se utiliza para construir interfaces de usuario, particularmente en aplicaciones web de una sola página. React adopta un enfoque basado en componentes, permitiendo a los desarrolladores descomponer la interfaz de usuario en elementos reutilizables y modulares. Cada componente en React gestiona su propio estado y lógica, lo que facilita el desarrollo y mantenimiento de interfaces complejas.

Una de las características distintivas de React es su uso del Virtual DOM, una representación en memoria del DOM real, que optimiza la actualización y el renderizado de componentes. Al realizar

cambios en el estado de un componente, React actualiza de manera eficiente solo aquellas partes del DOM que han cambiado, lo que mejora significativamente el rendimiento de la aplicación. Además, React utiliza un flujo de datos unidireccional, lo que significa que los datos se pasan de los componentes padres a los hijos a través de propiedades (props), asegurando una mayor previsibilidad y control en la gestión del estado de la aplicación.

React es altamente extensible y puede integrarse con otras bibliotecas y frameworks para añadir funcionalidades adicionales. Por ejemplo, se puede utilizar junto con Redux para la gestión global del estado o con React Router para manejar la navegación entre diferentes vistas de una aplicación. La popularidad de React ha dado lugar a una gran comunidad de desarrolladores y a un ecosistema robusto de herramientas y bibliotecas que facilitan y aceleran el desarrollo de aplicaciones web.

Ant Design

Ant Design es un marco de diseño de interfaz de usuario y una biblioteca de componentes desarrollada por Ant Financial, una filial de Alibaba. Diseñado específicamente para aplicaciones empresariales, Ant Design proporciona un conjunto completo de componentes UI preconstruidos que siguen un estilo visual coherente y moderno. Estos componentes, que incluyen formularios, botones, tablas, modales y más, están diseñados para ser altamente personalizables y responden a los principios de diseño de interfaz de usuario que priorizan la claridad y la consistencia.

Ant Design se integra de manera nativa con React, lo que permite a los desarrolladores utilizar componentes de Ant Design directamente en sus aplicaciones React. Esta integración proporciona una base sólida para el desarrollo de aplicaciones rápidas y con una experiencia de usuario rica. Los componentes de Ant Design vienen con soporte incorporado para la internacionalización, lo que facilita la creación de aplicaciones que pueden ser localizadas en múltiples idiomas y adaptarse a diferentes mercados.

Además de los componentes UI, Ant Design incluye un sistema de diseño que abarca tipografía, espaciado, paleta de colores, y patrones de interacción, lo que garantiza que las aplicaciones desarrolladas con Ant Design mantengan una estética visual coherente y una experiencia de usuario optimizada. Este enfoque sistemático no solo agiliza el proceso de diseño, sino que también mejora la cohesión y la calidad del producto final.

5.1.2 Desarrollo de la interfaz de usuario: Componentes desarrollados

A continuación, se procede a explicar las principales funcionalidades de la interfaz y como se encuentra estructurada. En primer lugar, la interfaz cuenta con un menú desplegable situado a la izquierda. En

este menú, el cliente puede consultar información respecto a hospitales, bancos de sangre o solicitar una transfusión para un paciente. En la *Ilustración 10* se puede observar el menú de la aplicación.

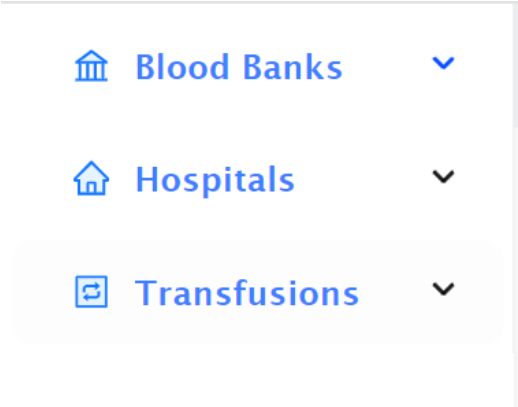


Ilustración 10: Menú colapsado de la aplicación

A su vez, el usuario puede navegar dentro de cada menú y consultar información como las últimas donaciones, el listado de pacientes y donadores o el inventario actual de sangre para cada uno de los bancos de sangre disponibles. En la *Ilustración 11* se puede observar el menú desplegado con los diferentes submenús.

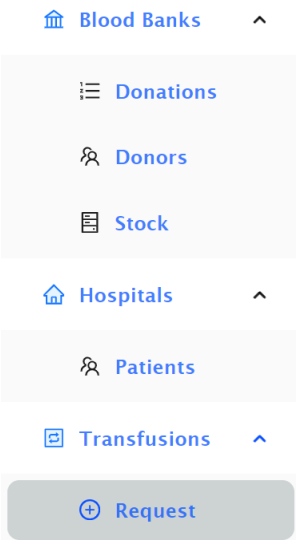


Ilustración 11: Menú desglosado de la aplicación

Para cada submenú, se ha desarrollado un componente en forma de tabla donde se almacena la información solicitada. Por ejemplo, si el usuario final selecciona “Donations”, la aplicación mostraría un listado con el histórico de donaciones, el ID del donante que ha llevado a cabo la donación, la fecha de la misma y por último, el banco de sangre donde se ha producido la donación.

A su vez, la aplicación permite incluir una nueva donación al sistema y un botón para refrescar el listado. A continuación, en la *Ilustración 12* se muestra el resultado de la aplicación en el caso que el cliente solicitase información sobre las donaciones.

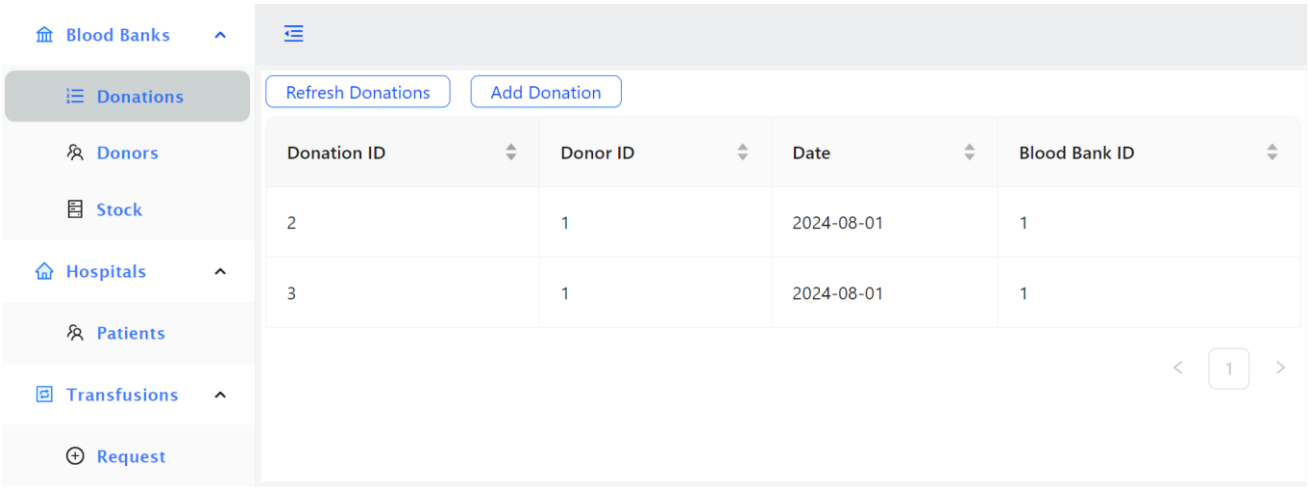


Ilustración 12: Diseño de la aplicación para las donaciones

De igual modo, si el usuario final decidiese obtener información sobre los donantes, la aplicación mostraría un listado con la información del histórico de los donantes. Se puede observar la última donación llevada a cabo por cada donante y el grupo de sangre y factor Rh. Además, en la *Ilustración 13*, se puede observar como la aplicación permite introducir nuevos donantes a la base de datos.

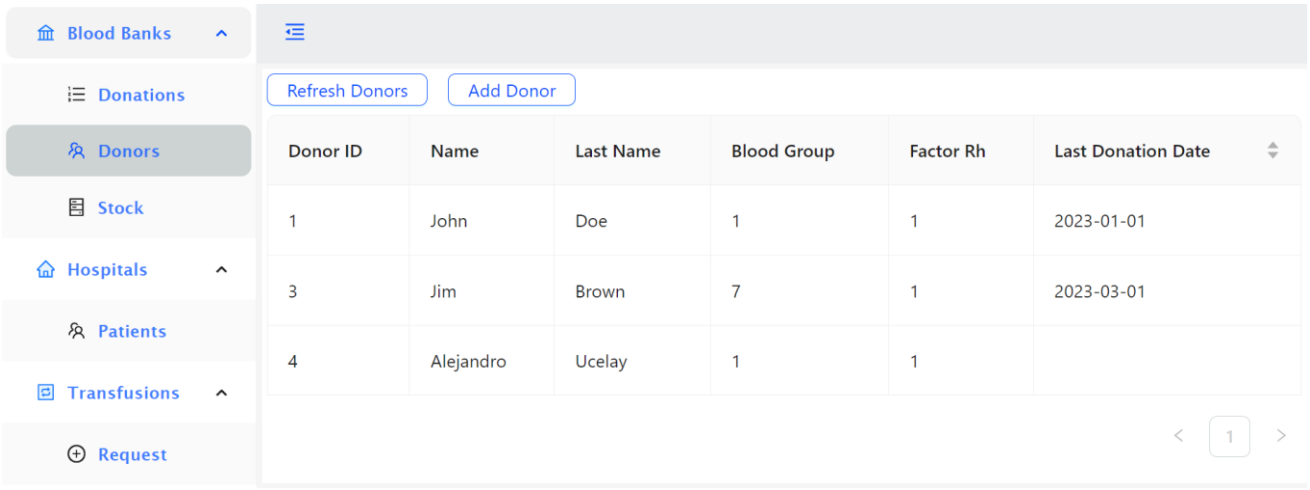


Ilustración 13: Diseño de la aplicación para donantes

En el caso que el usuario solicitase información sobre el inventario disponible, la aplicación mostraría un listado con los bancos de sangre disponibles y el inventario de cada uno de ellos para cada tipo de sangre y factor Rh (*Ilustración 14*). De esta forma, se puede conocer en tiempo real en que bancos de sangre existen existencias de un tipo de sangre en específico. A su vez, se añade un componente de trazabilidad ya que se es capaz de conocer el histórico de inventario de un banco de sangre.

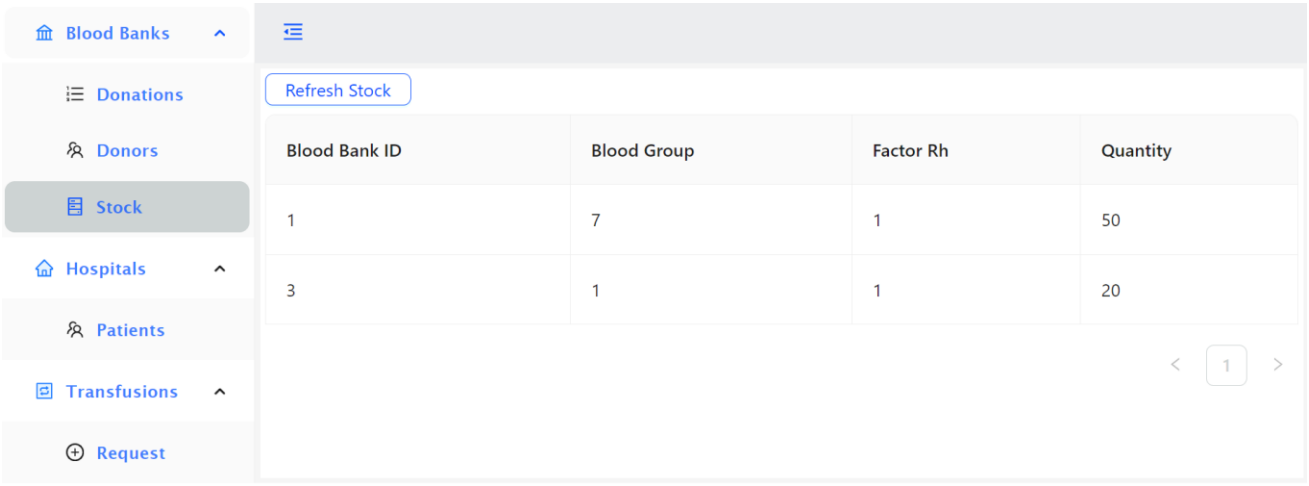


Ilustración 14: Diseño de la aplicación para los inventarios de los bancos de sangre

De manera análoga a la tabla de donantes, si el usuario quisiese obtener información sobre los pacientes, la aplicación mostraría una tabla con la información de todos los pacientes registrados en el sistema, así como su información. En la *Ilustración 15*, se puede observar como se muestra información relacionada con el tipo de sangre y factor Rh de cada paciente, así como el hospital al que pertenece. De esta forma, se puede conocer la distribución de pacientes para cada tipo de sangre en cada hospital y poder predecir en un futuro la necesidad de un tipo de sangre en específico.

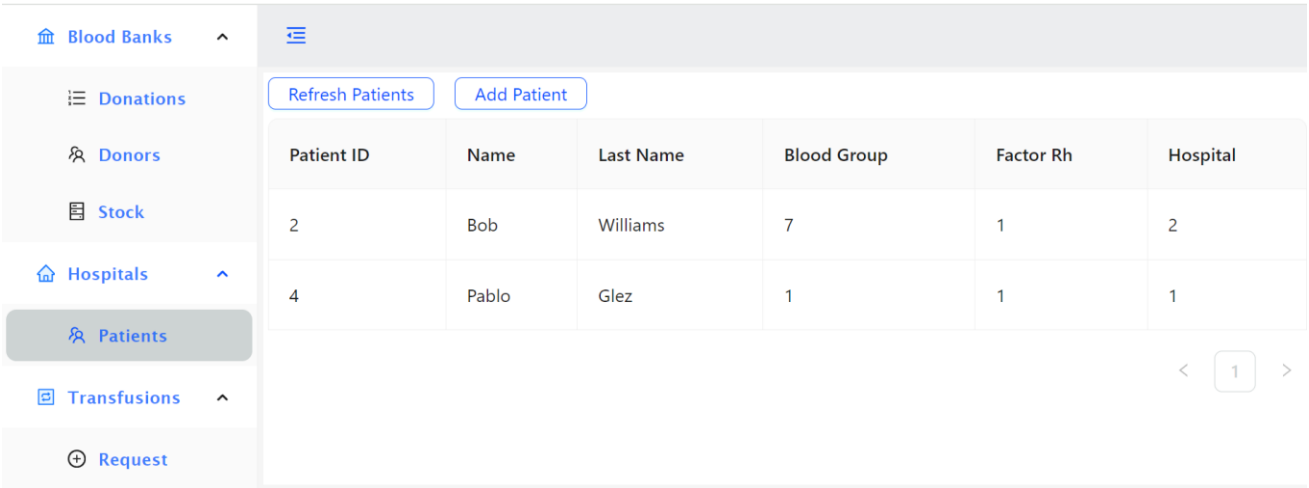


Ilustración 15: Diseño de la aplicación para mostrar los pacientes

Por último, si el cliente final decidiese solicitar una nueva transfusión para un paciente, debería seleccionar el submenú "*Request*" (*Ilustración 16*). Una vez seleccionado, se mostraría una página donde el cliente puede introducir el número de paciente en la plataforma y seleccionar el botón "*Submit*".

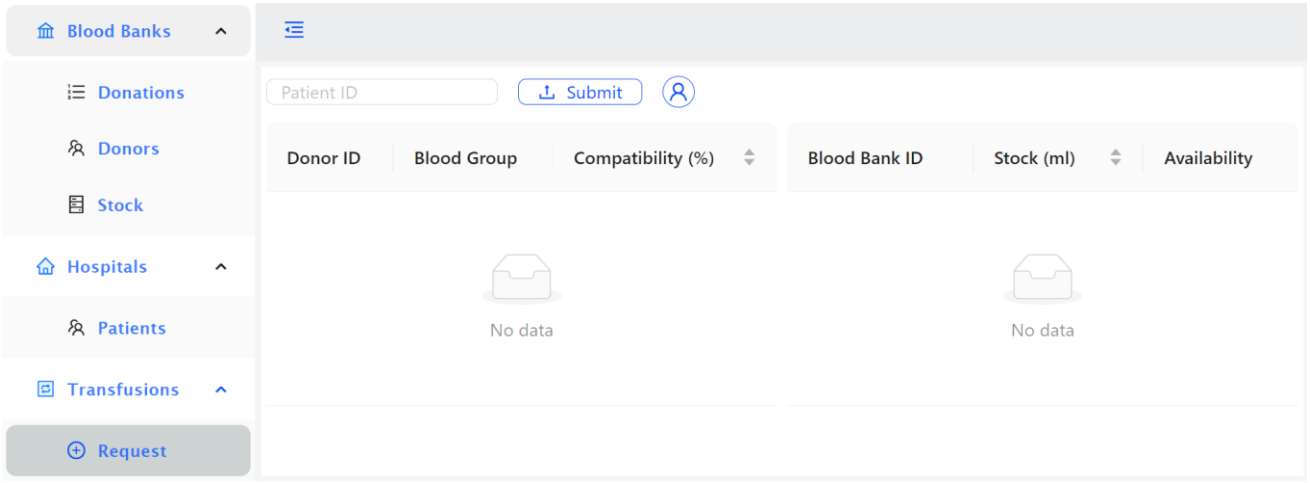


Ilustración 16: Diseño de la aplicación previo a realizar una petición de transfusión

Cuando el usuario introduce el número de paciente y este existe en la base de datos del sistema, la aplicación muestra dos tablas diferentes (*Ilustración 17*). La primera de ellas hace referencia a los donantes que existen para llevar a cabo esta transfusión. Se muestra información sobre su ID, su tipo de sangre y el porcentaje de compatibilidad entre el paciente introducido y cada donante. De esta forma, se puede conocer qué donante es el más compatible con el paciente introducido. Esto último es realmente útil, ya que, en caso de no disponer de inventario con el tipo de sangre requerido para el paciente, se puede contactar directamente con el donante más compatible.

Por otro lado, a la derecha se muestra una segunda tabla, en esta se muestra información sobre los bancos de sangre y si cuentan con inventario del tipo de sangre del paciente. El campo "*Availability*" muestra si el banco de sangre se encuentra disponible o no.

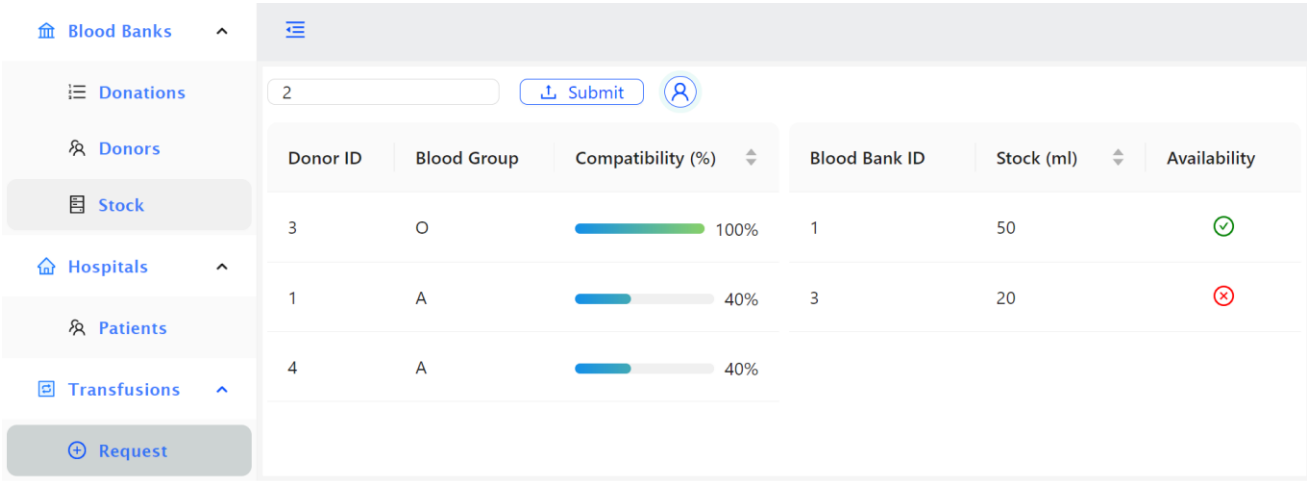


Ilustración 17: Diseño de la aplicación una vez realizada la petición de transfusión

A continuación, se muestra como ejemplo, el código desarrollado para desarrollar el componente JavaScript necesario para solicitar una transfusión para un paciente.

```
import React, { useState } from 'react';
import { Form, Input, Button, Space, Table, Progress, Tooltip } from 'antd';
import { UploadOutlined, UserOutlined, CheckCircleOutlined, CloseCircleOutlined} from '@ant-design/icons';

const columns1 = [
  {
    title: 'Donor ID',
    dataIndex: 'donor_id',
    key: 'donor_id',
  },
  {
    title: 'Blood Group',
    dataIndex: 'donor_blood_group',
    key: 'donor_blood_group',
  },
  {
    title: 'Compatibility (%)',
    dataIndex: 'compatibility',
    key: 'compatibility',
    sorter: (a, b) => b.compatibility - a.compatibility,
    render: (text) => (
      <Progress
        percent={text}
        status="active"
        strokeColor={{
          from: '#108ee9',
          to: '#87d068',
        }}
      />
    ),
  },
];

const columns2 = [
  {
    title: 'Blood Bank ID',
    dataIndex: 'bloodBankID',
    key: 'bloodBankID',
  },
  {
    title: 'Stock (ml)',
    dataIndex: 'quantity',
    key: 'quantity',
    sorter: (a, b) => b.quantity - a.quantity,
  },
  {
    title: 'Availability',
    dataIndex: 'availability',
    key: 'availability',
    render: (available) => (
      <div style={{ textAlign: 'center' }}>
        {available ? (
          <CheckCircleOutlined style={{ color: 'green', fontSize: '16px' }} />
        ) : (
            <CloseCircleOutlined style={{ color: 'red', fontSize: '16px' }} />
        )}
      </div>
    ),
  },
];
```

```
const MyComponent = () => {
  const [dataSource1, setDataSource1] = useState([]);
  const [dataSource2, setDataSource2] = useState([]);
  const [patientId, setPatientId] = useState('');
  const [patientInfo, setPatientInfo] = useState(null);

  const handlePatientIdChange = (e) => {
    setPatientId(e.target.value);
  };

  const handleSearch = async () => {
    try {
      // Fetch compatibility data
      const response = await fetch(`http://127.0.0.1:5000/transfusions/${patientId}`);
      if (!response.ok) {
        throw new Error(`Network response was not ok: ${response.statusText}`);
      }

      const data = await response.json();
      setDataSource1(data);

      // Fetch patient info
      const patientResponse = await fetch(`http://127.0.0.1:5000/patients/${patientId}`);
      if (patientResponse.ok) {
        const patientData = await patientResponse.json();
        setPatientInfo(patientData);
      } else {
        setPatientInfo(null);
      }

      // Fetch blood bank stock data
      const stockResponse = await fetch(`http://127.0.0.1:5000/blood_bank_stocks/${patientId}`);
      if (!stockResponse.ok) {
        throw new Error(`Network response was not ok: ${stockResponse.statusText}`);
      }

      const stockData = await stockResponse.json();
      setDataSource2(stockData);

    } catch (error) {
      console.error('Error fetching data:', error);
    }
  };

  return (
    <Space direction="vertical" size="small" style={{ display: 'flex' }}>
      <Form layout="inline">
        <Form.Item name="patientId" style={{ height: 10 }}>
          <Input
            placeholder="Patient ID"
            value={patientId}
            onChange={handlePatientIdChange}
            style={{ height: '20px' }}
          />
        </Form.Item>
        <Form.Item>
          <Button
            icon={<UploadOutlined />}
            htmlType="button"
            onClick={handleSearch}
            style={{ height: '20px', color: '#0B51FF', borderColor: '#0B51FF' }}
          >
            Submit
          </Button>
        </Form.Item>
```

```
        <Form.Item>
          <Tooltip
            title={patientInfo ? (
              <>
                <p><strong>Name:</strong> {patientInfo.name} {patientInfo.last_name}</p>
                <p><strong>Blood Group:</strong> {patientInfo.blood_group}</p>
                <p><strong>Rh Factor:</strong> {patientInfo.factor_rh}</p>
              </>
            ) : 'No patient information available'}
          >
            <Button
              icon={<UserOutlined />}
              htmlType="button"
              style={{
                height: '25px',
                width: '25px',
                borderRadius: '50%',
                color: '#0B51FF',
                borderColor: '#0B51FF',
              }}
            />
          </Tooltip>
        </Form.Item>
      </Form>
      <div style={{ display: 'flex', width: '100%', overflowX: 'auto' }}>
        <Table
          columns={columns1}
          dataSource={dataSource1}
          pagination={false}
          rowKey="donor_id" // Ensure unique key for each row
          style={{ flex: 1, marginRight: 8 }}
        />
        <Table
          columns={columns2}
          dataSource={dataSource2}
          pagination={false}
          rowKey="bloodBankID" // Ensure unique key for each row
          style={{ flex: 1 }}
        />
      </div>
    </Space>
  );
};

export default MyComponent;
```

CAPÍTULO 6: CONCLUSIONES Y TRABAJOS FUTUROS

En este último capítulo se explican las conclusiones del proyecto y se definen los trabajos futuros que se podrían implementar.

El desarrollo del sistema digital para la interconexión de donaciones de sangre entre hospitales y bancos de sangre ha permitido establecer una plataforma eficiente y confiable para la gestión integral del proceso de donación y transfusión sanguínea. La aplicación web diseñada no solo mejora la eficiencia operativa, sino que también introduce un nivel significativo de trazabilidad y monitoreo en tiempo real, elementos cruciales en la gestión de inventarios de sangre.

Uno de los principales logros del proyecto ha sido la implementación de un sistema que permite identificar rápidamente la disponibilidad de diferentes tipos de sangre en los bancos de sangre conectados, facilitando la toma de decisiones críticas en situaciones de emergencia. Esto no solo optimiza los tiempos de respuesta, sino que también mejora la capacidad de los hospitales para satisfacer las necesidades de sus pacientes de manera oportuna. Además, la aplicación facilita la trazabilidad completa del ciclo de vida de cada donación, lo que asegura un control riguroso sobre la calidad y seguridad de las transfusiones.

La utilización de tecnologías modernas y robustas, como Flask y React, en combinación con herramientas de modelado de bases de datos como SQLAlchemy, ha demostrado ser eficaz para el desarrollo de un sistema escalable y mantenible. La estructura cliente-servidor adoptada asegura una comunicación eficiente y segura entre la interfaz de usuario y el backend, proporcionando una experiencia de usuario fluida y confiable.

Finalmente, la simulación de casos de uso para la solicitud de dosis específicas de sangre y la identificación de bancos con inventario disponible ha confirmado la funcionalidad y fiabilidad del sistema. Esto valida que la aplicación no solo cumple con los requisitos establecidos, sino que también está preparada para ser implementada en un entorno real, donde pueda contribuir significativamente a mejorar la gestión y distribución de donaciones de sangre.

En cuanto a los trabajos futuros que se podrían implementar a raíz de este proyecto, a continuación, se destacan los siguientes:

- Integración con Sistemas Hospitalarios Existentes: Para maximizar el impacto del sistema, sería beneficioso integrar la aplicación con los sistemas de información hospitalaria (HIS) existentes. Esto permitiría un intercambio de datos más fluido y una mayor automatización en la gestión de solicitudes y distribución de sangre.
- Desarrollo de Funcionalidades de Predicción y Alerta: Implementar algoritmos de aprendizaje automático que analicen patrones históricos de donación y uso de sangre podría ayudar a predecir futuras demandas. Además, el sistema podría incorporar alertas automáticas para niveles bajos de inventario o fechas de vencimiento de los componentes sanguíneos, mejorando aún más la eficiencia operativa.
- Ampliación a Nivel Regional o Nacional: Actualmente, el sistema está diseñado para funcionar en una red local de hospitales y bancos de sangre. Un trabajo futuro podría enfocarse en ampliar la aplicación para soportar una interconexión a nivel regional o nacional, lo que aumentaría significativamente su utilidad y alcance.
- Seguridad y Cumplimiento de Normativas: La implementación de medidas de seguridad más avanzadas, como la encriptación de datos y la autenticación multifactor, sería crucial para garantizar la confidencialidad y protección de la información sensible. Además, asegurar el cumplimiento con normativas internacionales sobre privacidad y protección de datos, como la GDPR o HIPAA, sería esencial para la implementación en un entorno real.
- Desarrollo de Aplicaciones Móviles: Crear versiones móviles de la aplicación podría facilitar el acceso y la gestión de donaciones de sangre para profesionales de la salud y administradores de bancos de sangre, permitiéndoles realizar consultas y gestionar inventarios desde dispositivos móviles.

En conclusión, este proyecto sienta las bases para un sistema de gestión de donaciones de sangre interconectado y eficiente. Los trabajos futuros sugeridos proporcionan un camino claro para su evolución, con el potencial de hacer una contribución significativa al sector de la salud mediante la optimización de la disponibilidad y distribución de recursos sanguíneos críticos.

REFERENCIAS

1. Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.
2. Sadalage, P. & Fowler, M. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley.
3. Sonmez, J. (2013). *Soft Skills: The software developer's life manual*. Manning Publications.
4. Khan, M. & Khan, S. S. (2022). "A Comparative Study of Web Application Frameworks: Django vs. Flask", *International Journal of Computer Science and Information Security*, 20(5), 123-130.
5. Lin, K., Li, Z., & Lee, Y. (2021). "Real-Time Blood Bank Management System Using Internet of Things", *International Journal of Healthcare Information Systems and Informatics*, 16(2), 56-72.
6. Pusztai, J., Menyhei, A., & Bessenyei, B. (2020). "A Cloud-Based Blood Donation and Transfusion Management System", *Journal of Cloud Computing and Data Science*, 8(1), 89-102.
7. Flask Documentation. (2024). *Flask*. https://flask.palletsprojects.com/
8. React Documentation. (2024). *React – A JavaScript library for building user interfaces*. https://reactjs.org/docs/getting-started.html
9. MySQL Documentation. (2024). *MySQL 8.0 Reference Manual*. https://dev.mysql.com/doc/
10. SQLAlchemy Documentation. (2024). *SQLAlchemy*. https://docs.sqlalchemy.org/
11. Ant Design Documentation. (2024). *Ant Design*. https://ant.design/docs/react/introduce
12. How to build a Rest API. https://restfulapi.net/
13. 1. Smith, J., & Johnson, A. (2022). Technological Advances in Blood Transfusions: A Comprehensive Review. Journal of Transfusion Medicine, 15(2), 87-104.
14. 2. García, L., & Pérez, M. (2023). Automation and Centralized Databases in Blood Transfusion Services: Enhancing Safety and Efficiency. International Journal of Hematology, 28(4), 321-335.
15. 3. Wang, S., et al. (2024). Robotics and Artificial Intelligence in Blood Transfusion: Current Applications and Future Perspectives. Journal of Medical Robotics, 10(1), 45-58.
16. 4. Hawashin D, Mahboobeh DAJ, Salah K, Jayaraman R, Yaqoob I, Debe M et al. Blockchain-based management of blood donation. IEEE Access. 2021;9:163016-32.
17. 5. Pradhan NR, Singh AP, Kumar V. Blockchain-enabled traceable, transparent transportation system

for blood bank. In Advances in VLSI, Communication, and Signal Processing: Select Proceeding

18. 6. Castro M, Jara AJ, Skarmeta AF. Analysis of the future internet of things capabilities for continuous temperature monitoring of blood bags in terrestrial logistic systems. In Convergence and Hybrid Information Technology: 5th International Conference, ICHIT 2011, Daejeon, Korea. Proceedings, Springer Berlin Heidelberg. 2011;5:558- 66.
19. 7. Hammoudeh M, Ghafir I, Bounceur A, Rawlinson T. Continuous monitoring in mission-critical applications using the internet of things and blockchain. In Proceedings of the 3rd International Conference on Future Networks and Distributed Systems. 2019;1-5.
20. 8. Mohanta B, Das P, Patnaik S. Healthcare 5.0: A paradigm shift in digital healthcare system using artificial intelligence, IOT and 5G communication, In 2019 International Conference on Applied Machine Learning. 2019;191-6.

ANEXO A: CÓDIGO FUENTE SERVIDOR

En este apartado se adjunta el código fuente desarrollado para el servidor de la aplicación. Como se mencionó anteriormente, el servidor se ha desarrollado utilizando el lenguaje de programación Python, así como las librerías Flask, SQLAlchemy y MySQLconnection. Únicamente se adjunta el código del fichero principal (app.py) donde se importan el resto de módulos y el archivo de configuración (config.py)

App.py

```python
from flask import Flask, request, jsonify
from flask_cors import CORS
from routes.donor_routes import donor_bp
from routes.donation_routes import donation_bp
from routes.patient_routes import patient_bp
from routes.transfusion_routes import transfusion_bp
from routes.blood_group_routes import blood_group_bp
from routes.factor_rh_routes import factor_rh_bp
from routes.hospital_routes import hospital_bp
from routes.blood_bank_routes import blood_bank_bp
from routes.blood_bank_stock_routes import blood_bank_stock_bp

app = Flask(__name__)

# Configurar CORS para permitir solicitudes desde http://localhost:3000
CORS(app, resources={r"/*": {"origins": "http://localhost:3000", "methods": ["GET", "POST", "OPTIONS"],
"allow_headers": ["Content-Type"]}})

# Registrar los blueprints de rutas
app.register_blueprint(donor_bp, url_prefix='/donors')
app.register_blueprint(patient_bp, url_prefix='/patients')
app.register_blueprint(transfusion_bp, url_prefix='/transfusions')
app.register_blueprint(blood_group_bp, url_prefix='/blood_groups')
app.register_blueprint(factor_rh_bp, url_prefix='/factors_rh')
app.register_blueprint(hospital_bp, url_prefix='/hospitals')
app.register_blueprint(blood_bank_bp, url_prefix='/blood_banks')
app.register_blueprint(blood_bank_stock_bp, url_prefix='/blood_bank_stocks')
app.register_blueprint(donation_bp, url_prefix='/donations')

@app.before_request
def handle_preflight():
    if request.method == 'OPTIONS':
        response = jsonify({'message': 'CORS preflight response'})
        response.headers.add('Access-Control-Allow-Origin', 'http://localhost:3000')
        response.headers.add('Access-Control-Allow-Methods', 'GET, POST, OPTIONS')
        response.headers.add('Access-Control-Allow-Headers', 'Content-Type')
        return response

if __name__ == '__main__':
    app.run(debug=True)
```

Config.py

```
import os
from dotenv import load_dotenv

# Cargar las variables de entorno desde el archivo .env
load_dotenv()

class Config:
    DB_USER = os.getenv('DB_USER')
    DB_PASSWORD = os.getenv('DB_PASSWORD')
    DB_NAME = os.getenv('DB_NAME')
    DB_HOST = os.getenv('DB_HOST')
    DB_PORT = os.getenv('DB_PORT')
    DB_SSL_CA = os.getenv('DB_SSL_CA')  # Ruta al certificado CA, si es necesario

    SQLALCHEMY_DATABASE_URI = (
        f'mysql+mysqlconnector://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}'
    )
    SQLALCHEMY_TRACK_MODIFICATIONS = False
```

ANEXO B: CÓDIGO FUENTE INTERFAZ DE USUARIO

En este apartado se adjunta el código fuente desarrollado para la interfaz de usuario de la aplicación. Como se mencionó anteriormente, la interfaz se ha desarrollado utilizando el lenguaje de programación JavaScript, así como las librerías React y AntDesign. Únicamente se adjunta el código del fichero principal (app.js) donde se importan el resto de módulos y el archivo de dependencias (packages.json)

App.js

```
import React, { useState } from "react";
import { Route, Routes, Link } from "react-router-dom";
import {
  MenuFoldOutlined,
  MenuUnfoldOutlined,
  OrderedListOutlined,
  TeamOutlined,
  HddOutlined,
  HomeTwoTone,
  InteractionTwoTone,
  BankTwoTone,
  PlusCircleOutlined
} from "@ant-design/icons";
import { Button, Layout, Menu, theme, ConfigProvider } from "antd";
import "./App.css";
import BloodBanksTable from "./blood_banks/blood_banks";
import DonorsTable from "./blood_banks/donors";
import DonationsTable from "./blood_banks/donations";
import StockTable from "./blood_banks/stock";
import PatientsTable from "./hospitals/patients";
import TransfusionsTable from "./hospitals/transfusions";
import HospitalsTable from "./hospitals/hospitals";
import RequestElement from "./hospitals/request_transfusions";

const { Header, Sider, Content } = Layout;
const { SubMenu } = Menu;

const App: React.FC = () => {
  const [collapsed, setCollapsed] = useState(false);
  const {
    token: { colorBgContainer },
  } = theme.useToken();

  return (
    <ConfigProvider
      theme={{
        token: {
          colorPrimary: "#CDD3D3",
        },
        components: {
          Menu: {
            itemSelectedBg: "#CDD3D3",
            itemSelectedColor: "#0B51FF",
          },
        },
      }}
```

```
  >
    <Layout>
      <Sider trigger={null} collapsible collapsed={collapsed} theme="light">
        <div className="demo-logo-vertical" />
        <Menu
          theme="light"
          mode="inline"
          defaultSelectedKeys={["1"]}
          defaultOpenKeys={["sub1", "sub2"]}
        >
        <SubMenu
          key="sub1"
          icon={<BankTwoTone />}
          title={<Link to="/blood_banks"><span className="custom-menu-item">Blood Banks</span></Link>}
        >
            <Menu.Item key="2" icon={<OrderedListOutlined />}>
              <span className="custom-menu-item">Donations</span>
              <Link to="/donations" />
            </Menu.Item>
            <Menu.Item key="3" icon={<TeamOutlined />}>
              <span className="custom-menu-item">Donors</span>
              <Link to="/donors" />
            </Menu.Item>
            <Menu.Item key="4" icon={<HddOutlined />}>
              <span className="custom-menu-item">Stock</span>
              <Link to="/blood_stock" />
            </Menu.Item>
          </SubMenu>
          <SubMenu
            key="sub2"
            icon={<HomeTwoTone />}
            title={<Link to="/hospitals"><span className="custom-menu-item">Hospitals</span></Link>}
          >
            <Menu.Item key="5" icon={<TeamOutlined />}>
              <span className="custom-menu-item">Patients</span>
              <Link to="/patients" />
            </Menu.Item>
          </SubMenu>
          <SubMenu
          key="sub3"
          icon={<InteractionTwoTone />}
          title={<Link to="/transfusions"><span className="custom-menu-item">Transfusions</span></Link>}
          >
          <Menu.Item key="6" icon={<PlusCircleOutlined />}>
              <span className="custom-menu-item">Request</span>
              <Link to="/request_transfusion" />
            </Menu.Item>
        </SubMenu>
        </Menu>
      </Sider>
      <Layout>
        <Header className="custom-header">
          <Button
            type="text"
            icon={collapsed ? <MenuUnfoldOutlined /> : <MenuFoldOutlined />}
            onClick={() => setCollapsed(!collapsed)}
            style={{
              fontSize: "16px", // Ajuste del tamaño del ícono
              width: 40,
              height: 40,
              color: '#0B51FF',
            }}
          />
        </Header>
        <Content
```

```
            style={{
              margin: "4px", // Espaciado más consistente
              padding: 4,
              background: colorBgContainer,
              minHeight: 280,
            }}
          >
            <Routes>
              <Route path="/blood_banks" element={<BloodBanksTable />} />
              <Route path="/donors" element={<DonorsTable />} />
              <Route path="/patients" element={<PatientsTable />} />
              <Route path="/transfusions" element={<TransfusionsTable />} />
              <Route path="/hospitals" element={<HospitalsTable />} />
              <Route path="/donations" element={<DonationsTable />} />
              <Route path="/blood_stock" element={<StockTable />} />
              <Route path="/request_transfusion" element={<RequestElement />} />
              {/* Añade más rutas aquí */}
            </Routes>
          </Content>
        </Layout>
      </Layout>
    </ConfigProvider>
  );
};

export default App;
```

Packages.json

```
{
  "name": "transfusions_frontend",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "antd": "^5.19.3",
    "react": "^18.3.1",
    "react-dom": "^18.3.1",
    "react-router-dom": "^6.25.1",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
```

```
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

ANEXO C: CONTRIBUCIÓN A LOS OBJETIVOS DE LA ONU PARA EL DESARROLLO SOSTENIBLE

El desarrollo del sistema digital para la interconexión de donaciones de sangre entre hospitales y bancos de sangre se alinea con varios de los Objetivos de Desarrollo Sostenible (ODS) de las Naciones Unidas, proporcionando una solución tecnológica que promueve la salud y el bienestar, la innovación en infraestructura, y la reducción de desigualdades. A continuación, se detallan las contribuciones específicas de este proyecto a los ODS:

ODS 3: Salud y Bienestar (Garantizar una vida sana y promover el bienestar para todos en todas las edades)

El sistema desarrollado contribuye directamente al ODS 3 al mejorar la gestión de las donaciones de sangre y facilitar su distribución eficiente entre hospitales y bancos de sangre. Al permitir la trazabilidad completa del ciclo de vida de una donación de sangre, el sistema asegura que las transfusiones se realicen de manera segura y oportuna, minimizando los riesgos de errores y mejorando la calidad de la atención médica. Además, al optimizar el proceso de identificación de inventarios disponibles, se reduce el tiempo de respuesta en situaciones de emergencia, lo cual puede salvar vidas.

ODS 9: Industria, Innovación e Infraestructura (Construir infraestructuras resilientes, promover la industrialización inclusiva y sostenible y fomentar la innovación)

Este proyecto fomenta la innovación en la infraestructura de salud mediante la creación de un sistema digital interconectado que facilita la gestión de un recurso crítico como es la sangre. Al emplear tecnologías avanzadas como Flask, React, y SQLAlchemy, y al integrar componentes de trazabilidad y monitoreo en tiempo real, el sistema promueve una mayor eficiencia y sostenibilidad en la infraestructura hospitalaria. Además, la interoperabilidad entre diferentes hospitales y bancos de sangre mejora la resiliencia del sistema de salud, permitiendo una respuesta más coordinada y efectiva ante crisis sanitarias.

ODS 10: Reducción de las Desigualdades (Reducir la desigualdad en y entre los países)

El sistema de interconexión de donaciones de sangre tiene el potencial de reducir las desigualdades en el acceso a recursos de salud. En muchas regiones, la disponibilidad de sangre para transfusiones puede ser limitada y desigual, lo que afecta a los grupos más vulnerables. Al implementar una red de

donaciones interconectada, el sistema facilita una distribución más equitativa de los recursos sanguíneos, asegurando que los pacientes en diferentes hospitales, independientemente de su ubicación o recursos, tengan un acceso justo a las transfusiones de sangre necesarias.