Diseño del Componente Backend del Software para el Censo de Camas del Hospital Universitario de Santander (HUS).

Valentina Goyeneche Calderón

Trabajo de Grado para Optar al Título de Ingeniera de Sistemas

Director

Gabriel Rodrigo Pedraza Ferreira

PhD en Ciencias de la Computación

Tutor:

Nelly Méndez Mesa

Ingeniera de Sistemas

Universidad Industrial de Santander

Facultad de Ingenierías Físico-Mecánicas

Escuela de Ingeniería de Sistemas e Informática

Bucaramanga

2022

Dedicatoria

A mi hermana Carolina, porque siempre ha sido mi mayor apoyo, por ser mi mejor amiga y mi mejor compañía.

A mis padres, por estar a mi lado, por darme tantas oportunidades, por educarme de la mejor manera.

A mi pareja y amigos, por hacer de los últimos cinco años una experiencia inolvidable.

Agradecimientos

A la Universidad Industrial de Santander, por abrirme las puertas y ayudarme a formar como profesional y como persona.

Al profesor Gabriel Rodrigo Pedraza, por ser un excelente ejemplo y un mejor director de proyecto.

Al Hospital Universitario de Santander, por permitirme desarrollar este proyecto y aportar un granito a esa gran institución.

Tabla de Contenido

Pág.

Introducción 13

1. Planteamiento y Justificación del Problema 14

2. Objetivos 16

2.1 Objetivo General 16

2.2 Objetivos Específicos 16

3. Marco de Referencia 17

3.1 Marco Contextual 17

3.1.1 Gestión Hospitalaria 17

3.1.2 Hospitalización 17

3.1.3 Camas Hospitalarias 17

3.1.4 Traslado de Camas 18

3.1.5 Liberación de Camas 18

3.1.6 Censo de Camas 18

3.2 Marco Tecnológico 19

3.2.1 Aplicación Web 19

3.2.2 Single Page Application (SPA) 19

3.2.3 API REST 20

3.2.4 Framework 20

3.2.5 Java 21

3.2.6 Spring Boot 21

3.2.7 Spring Data JPA 21
3.2.8 IntelliJ IDEA 21
3.2.9 SQL Server 2012 22
3.2.10 Postman 22
4. Metodología 22
4.1 Análisis 23
4.2 Identificación de Requerimientos 23
4.3 Diseño 23
4.4 Prototipo 24
4.5 Validación y Retroalimentación 24
4.6 Entrega 24
5. Requerimientos 24
5.1 Listado de Requerimientos 24
5.1.1 Requerimientos Funcionales 25
5.1.2 Requerimientos No Funcionales 28
5.2 Análisis de Requerimientos 28
5.2.1 Diagramas de Casos de Uso 28
5.2.2 Diagramas de Actividades 31
6. Diseño de la Solución 35
6.1 Arquitectura 35
6.2 Diseño de Interfaces 37
6.3 Diseño de Datos 41
6.3.1 Modelo de base de datos 41

6.3.2 Descripción del modelo 41

7. Implementación del Prototipo 42

7.1 Primer Prototipo 43

7.1.1 Modelo 44

7.1.2 Repositorio 46

7.1.3 Servicio 47

7.1.4 Controlador REST 49

7.2 Segundo Prototipo 51

7.2.1 Autenticación de Usuarios 51

7.2.2 Listado de Peticiones HTTP 52

8. Validación del Prototipo 55

8.1 Pruebas de Software 56

8.2 Pruebas de Usuario 59

9. Conclusiones 62

Referencias Bibliográficas 64

Lista de Tablas

Pág.

Tabla 1. *Requerimientos Funcionales* 25

Tabla 2. *Requerimientos no funcionales* 28

Tabla 3. *Principales peticiones de la API* 37

Tabla 4. *Servicios de Ingreso* 52

Tabla 5. *Servicios de Cama* 53

Tabla 6. *Servicios de Grupo* 53

Tabla 7. *Servicios de Subgrupo* 53

Tabla 8. *Servicios de Tipo Cama* 54

Tabla 9. *Servicios de Usuario* 54

Tabla 10. *Servicios de Paciente* 55

Tabla 11. *Prueba iniciar sesión usuario* 56

Tabla 12. *Prueba registrar usuario* 56

Tabla 13. *Prueba cambiar contraseña* 56

Tabla 14. *Prueba filtro de búsqueda de ingresos* 57

Tabla 15. *Prueba de traslado de camas con ingreso registrado* 58

Tabla 16. *Prueba de liberación de camas con ingreso registrado* 58

Tabla 17. *Prueba filtro de búsqueda de camas* 58

Tabla 18. *Prueba búsqueda de cama por código* 59

Lista de Figuras

Pág.

Figura 1. *Imagen explicativa de una SPA.* 20

Figura 2. *Metodología del proyecto (fuente: Autora)* 23

Figura 3. *Diagrama de Casos de Uso de Usuario Administrador* 29

Figura 4. *Diagrama de Casos de Uso de Usuario Jefe de Enfermería* 30

Figura 5. *Diagrama de Casos de Uso de Usuario Enfermera* 31

Figura 6. *Diagrama de actividades para el traslado de pacientes.* 32

Figura 7. *Diagrama de actividades para la liberación de pacientes.* 33

Figura 8. *Diagrama de actividades para el censo de camas.* 34

Figura 9. *Arquitectura del software* 36

Figura 10. *Ejemplo de petición GET en Postman* 38

Figura 11. *Ejemplo de petición GET antes de un cambio* 39

Figura 12. *Ejemplo de petición PUT que realiza cambios en la información* 39

Figura 13. *Ejemplo de petición GET después de un cambio* 40

Figura 14. *Diagrama Entidad-Relación* 41

Figura 15. *Diagrama de base de datos del primer prototipo* 44

Figura 16. *Fragmento de código de una clase tipo Entidad* 45

Figura 17. *Fragmento de código de una clase tipo Repositorio* 46

Figura 18. *Ejemplo de métodos generalizados en un repositorio* 47

Figura 19. *Ejemplo de métodos que utilizan parámetros* 47

Figura 20. *Ejemplo de query utilizando lenguaje nativo SQL* 47

Figura 21. *Fragmento de código de una clase tipo Servicio* 48

Figura 22. *Ejemplo de varios repositorios en una misma clase Servicio* 49

Figura 23. *Ejemplo de implementación de DTO* 49

Figura 24. *Fragmento de código de clase tipo Controlador REST* 50

Figura 25. *Ejemplo de petición de tipo GET en un controlador* 50

Figura 26. *Implementación para restringir autorización en métodos* 52

Figura 27. *Resultados de la primera pregunta* 60

Figura 28. *Resultados de la segunda pregunta* 60

Figura 29. *Resultados de la tercera pregunta* 61

Figura 30. *Resultados de la cuarta pregunta* 61

Lista de Apéndices

(Los apéndices están adjuntos y puede visualizarlos en la base de datos de la biblioteca UIS)

Apéndice A. Javadoc.

Apéndice B. Validación de los servicios de la API.

Apéndice C. Manual Técnico del proyecto.

Apéndice D. Manual de Usuario del proyecto.

Apéndice E. Código fuente.

Resumen

Título: Diseño del componente backend del software para el censo de camas del Hospital Universitario de Santander (HUS) [*]

Autor: Valentina Goyeneche Calderón [**]

Palabras Clave: componente backend, censo de camas, Hospital Universitario de Santander, aplicación web.

Descripción: El Hospital Universitario de Santander (HUS) es un referente de las instituciones públicas prestadoras de salud en la región nororiental del país. Con más de 380 camas en servicio, el hospital recibe diariamente un alto flujo de pacientes. A pesar de contar con un sistema de censo de camas, este resulta impreciso, dificultando la agilidad de procesos y provocando que el personal opte por una acomodación y análisis de disponibilidad manual. Por tanto, el desarrollo de un software de gestión eficiente permitiría agilizar procesos y mejorar la atención hospitalaria.

En este trabajo se presenta el prototipo del componente backend propuesto como solución para una plataforma de aplicación web, este software fue desarrollado con una arquitectura basada en capas elaborada en el lenguaje de programación Java utilizando el framework Spring Boot. Esta configuración permite integrar el componente frontend diseñado en el framework Angular a través de una API REST para procesar y consultar grandes volúmenes de datos almacenados en una base de datos SQLServer 2012.

Para validar el enfoque propuesto, se puso a prueba la funcionalidad de este prototipo de software con ayuda del personal en las instalaciones de la institución. Con un conjunto de datos real suministrado por el hospital, se demostró la funcionalidad de esta aplicación para la asignación, control y consulta del servicio de censo de camas.

[1*] Trabajo de Grado

[2**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Gabriel Rodrigo Pedraza Ferreira. PhD en Ciencias de la Computación.

Abstract

Title: Design of the backend component of a software for bed census for the Hospital Universitario de Santander (HUS) *

Author(s): Valentina Goyeneche Calderón **

Key Words: backend, bed census, Hospital Universitario de Santander, web application

Description: The Santander University Hospital (HUS) is a reference for public health care institutions in the northeastern region of the country. With more than 380 beds, the hospital receives a high flow of patients daily. Despite having a bed census system, it is imprecise, obstructing the agility of processes and causing staff to opt for manual accommodation and availability analysis. Therefore, the development of an efficient management software would make it possible to have more efficient processes and improve hospital care.

In this work, the prototype of the backend component is proposed as a solution for a web application platform. This software was developed with an architecture based on layers elaborated in Java using the Spring Boot framework. This configuration allows integrating the frontend component designed in Angular framework using a REST API to process and query large volumes of data stored in a SQLServer 2012 database.

To validate the proposed approach, the functionality of this software prototype was tested with the help of the hospital staff at the institution's facilities. With a set of real data provided by the hospital, the functionality of this application for the assignment, control and management of the bed census service was demonstrated.

[3] *Degree Work

[4]**Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Gabriel Rodrigo Pedraza Ferreira. PhD en Ciencias de la Computación.

Introducción

El buen funcionamiento de un hospital depende de muchos factores, uno de los más importantes es la vigencia y confiabilidad de la información que se maneja en el proceso de atención de los pacientes. Este factor es relevante para la seguridad del paciente, y además facilita que se preste un servicio de calidad, creando así una experiencia más satisfactoria.

Actualmente el personal de enfermería del Hospital Universitario de Santander tiene dificultades con la gestión del sistema de censo de camas, ya que se obtiene por medio de un proceso manual, que además afecta la disponibilidad, confiabilidad, coherencia y el acceso de la información con respecto a la ocupación del hospital. La organización ha solicitado que se cree una herramienta tecnológica que les ayude con esta labor, con el propósito de agilizar otras operaciones que dependen de la ocupación de las camas, como asignación y traslado de cama a los pacientes, y liberación de camas, además de mejorar el tiempo de espera del paciente.

Este proyecto tiene como propósito central diseñar el componente backend de un prototipo de aplicación web que tenga como función principal apoyar el proceso institucional de censo de camas del hospital, haciendo uso de conocimientos de arquitectura de software para garantizar que el resultado final sea una prototipo funcional de alta calidad, además de crear una API que permita la comunicación con el componente frontend de manera que las funcionalidades creadas para el procesamiento de la información puedan ser desplegadas en una interfaz amigable al usuario.

1. Planteamiento y Justificación del Problema

El Hospital Universitario de Santander (HUS) cuenta con una amplia infraestructura, presta siete servicios de salud de mediana y alta complejidad, y su estructura la constituye un edificio de 11 pisos con una capacidad de 384 camas. Ya que el HUS es un punto referente del nororiente colombiano recibe una gran cantidad de pacientes a diario, además dentro del área metropolitana es un hospital público de tercer nivel, es decir, que consta con instalaciones adecuadas para el servicio médico y personal altamente capacitado(Recavar, 2020).

Debido al alto flujo de pacientes que tiene el hospital es necesario tener información actualizada, idealmente en tiempo real, del censo de camas de hospitalización. Sin embargo, en este momento el sistema que usan para determinar la disponibilidad de una cama es manual, donde un empleado pasa periódicamente por las diferentes áreas del hospital actualizando el estado de las camas, por lo que no solo es ineficiente sino que en el momento de consultar en el sistema la información muchas veces no es precisa, ocasionando que el proceso de ingreso de hospitalización de un paciente sea una operación lenta, comprometiendo la atención que el hospital presta y dificultando el trabajo de sus empleados.

Teniendo en cuenta la poca funcionalidad del sistema actual, el hospital ha optado por buscar una solución que pueda automatizar y optimizar el trabajo de los empleados encargados del censo, y facilite la visualización del estado de las camas. Por esto, se busca el desarrollo de un software que pueda ayudar en la organización y actualización de información de la ocupación hospitalaria, para que sea un sistema eficiente y confiable, que permita agilizar procesos como asignación de cama, traslado de pacientes, alta hospitalaria y liberación de cama, lo que mejora la

atención a los pacientes. En este sentido, se ha decidido diseñar una aplicación web con arquitectura SPA (Single Page Application), que estará dividida en dos componentes, un componente frontend que se enfoca en la interacción del usuario con la aplicación, y el componente de backend, que es el que le compete a este proyecto, que se encarga de la lógica del negocio, la persistencia de los datos y la integración con otros sistemas ya existentes dentro del hospital.

2. Objetivos

2.1 Objetivo General

Diseñar el componente backend de una aplicación web que permita gestionar el censo de camas soportando las actividades de asignación, control y consulta de disponibilidad para el servicio de hospitalización del Hospital Universitario de Santander.

2.2 Objetivos Específicos

- Especificar los requerimientos de software, centrados en el componente backend del sistema.
- Diseñar el componente backend de una aplicación web con base en el análisis de requerimientos de software.
- Construir un prototipo del diseño del software propuesto.
- Diseñar un plan de pruebas para comprobar la funcionalidad del prototipo.

3. Marco de Referencia

3.1 Marco Contextual

3.1.1 Gestión Hospitalaria

Se define como "*el proceso en el que las instituciones prestadoras de salud, o IPS, administran efectiva y eficientemente cualquier aspecto dentro de la institución, como el registro de pacientes, programación de citas, […], seguridad de los medicamentos, generación de reportes, gestión del personal, gestión de proceso ambulatorios, entre otros*" (Brownley, 2020).

Para el proyecto se tuvieron en cuenta tres procesos principales dentro de la gestión hospitalaria: traslado de camas, liberación de camas, y censo de camas.

3.1.2 Hospitalización

El glosario del Hospital Universitario de Santander define hospitalización como: "*Proceso en el cual por las condiciones del paciente se requiere de una atención dentro de la unidad hospitalaria para su evaluación y manejo, definida por el médico del servicio de urgencias*" (HUS, 2021).

3.1.3 Camas Hospitalarias

Una cama hospitalaria se define como una cama que presta un centro médico dentro de sus instalaciones para el uso de los pacientes durante su estadía de hospitalización (MINSA,2013). El Hospital Universitario de Santander maneja un sistema de clasificación de camas hospitalarias, el cual consiste en grupo, subgrupo, y tipo de cama. Este sistema se basa en los servicios que presta el hospital en los que es posible que un paciente pueda ser hospitalizado, estos son: hospitalización, quirúrgico, UCI, y urgencias.

3.1.4 Traslado de Camas

Este proceso consiste en asignar una nueva cama a un paciente que haya ingresado a la institución, puede ser un nuevo ingreso por urgencias o consulta externa, o por que deba ser movido de un servicio a otro. Es importante tener presente que este proceso depende del estado vigente de las camas, que pueden ser los siguientes:

- Desocupada
- Ocupada
- Bloqueada

3.1.5 Liberación de Camas

Ocurre cuando un paciente es dado de alta de la institución, entonces se debe registrar que la cama que ha estado ocupando estará nuevamente libre para ser asignada. En el caso del proyecto se plantea un proceso rápido para darle prioridad a registrar información en tiempo real, por lo que solo es necesario conocer el nombre o número de documento del paciente, o el número de ingreso que esté vinculado con la cama ocupada para poder liberarla.

3.1.6 Censo de Camas

Proceso en el cual se comprueba la ocupación hospitalaria, es decir, se hace un recuento de las camas existentes y de su respectivo estado, que puede ser: ocupada, desocupada o bloqueada. Se puede hacer una búsqueda por servicios con los siguientes filtros: grupo, subgrupo, tipo de cama y estado. Tipo de cama y estado son campos opcionales al momento de generar el censo.

3.2 Marco Tecnológico

3.2.1 Aplicación Web

Se define como aplicación web a los programas con arquitectura cliente-servidor, donde el navegador, o dispositivo desde el que se está interactuando con la aplicación, tiene el rol de cliente. Cuando el usuario ingresa entradas al programa, el cliente las envía al servidor web, que las recibe y las procesa para poder enviar una respuesta de vuelta (Shklar y Rosen, 2003).

Esto quiere decir que cuando se ejecuta una aplicación web como usuario se busca una ruta URL a través de un navegador, y esa acción provoca una búsqueda cuya respuesta trae información enviada desde el servidor hacia el navegador, y que concluye con el navegador exhibiendo en la aplicación el resultado de la búsqueda.

Las aplicaciones web tienen dos componentes diferentes que se ejecutan por separado, pero trabajan de manera sincronizada (Pisuwala, 2021). Esos componentes son el código que programa el cliente (frontend) y el código que programa el servidor (backend), los cuales se desarrollan individualmente. Sin embargo, necesitan de funciones que establezcan la conexión entre ambos (Sagara, 2019).

3.2.2 Single Page Application (SPA)

Este tipo de aplicaciones web están conformadas por una sola “página” como su nombre lo indica en inglés. Las SPA solo tienen una página de HTML desde la cual se expone toda la información disponible, así cuando se carga por primera vez, se descargan todos los recursos necesarios (HTML, CSS, JavaScript, datos, etc). En el momento en el que se interactúa con la aplicación, la vista contenida se reemplaza en el HTML inicial usando mecanismos de frontend de rutas y plantillas, pero sigue siendo la misma página (Puskaric et. al, 2019).

Figura 1. *Imagen explicativa de una SPA.*

Tomada de Lawson (2021).

3.2.3 API REST

Una Application Programming Interface (API), o interfaz de programación de aplicaciones en español, es un conjunto de reglas que definen cómo se pueden conectar y comunicar unas aplicaciones (o dispositivos) con otras. A su vez, una API REST es un tipo de arquitectura que usa el protocolo HTTP para hacer peticiones y hacer uso de la información, por medio de métodos como GET, PUT, POST, y DELETE, que se conocen como procesos de lectura, actualización, creación y eliminación de los recursos que se están consumiendo (Gillis, 2020).

3.2.4 Framework

En programación, un framework se puede definir como una estructura base que ofrece un conjunto de funcionalidades que aceleran el desarrollo de software. En otras palabras, es una plantilla compuesta de código que ya está implementado y estructurado, que simplifica y agiliza la construcción de aplicaciones (Rodríguez, 2020).

3.2.5 Java

Es un popular lenguaje de programación orientado a objetos, basado en clases diseñado para reducir dependencias al momento de programar. También se define como una plataforma de desarrollo. Reduce costos, acorta los plazos de desarrollo, impulsa la innovación y mejora los servicios de las aplicaciones (HARTMAN,2021).

3.2.6 Spring Boot

Es un framework de código abierto para Java, enfocado en microservicios web, es una infraestructura ligera que facilita la creación de aplicaciones basadas en Spring de una manera rápida, autónoma y con características deseables para producción (SPRING IO, 2021).

3.2.7 Spring Data JPA

Es una de las muchas herramientas que proporciona Spring Boot para facilitar el desarrollo de aplicaciones. Esta herramienta trabaja en conjunto con JPA, o Java Persistence API, y se puede definir como una capa agregada que simplifica el acceso a información que está almacenada en bases de datos relacionales (PHALTANKAR, 2021).

3.2.8 IntelliJ IDEA

Es un IDE inteligente y sensible al contexto para trabajar con Java y otros lenguajes JVM como Kotlin, Scala y Groovy en todo tipo de aplicaciones. Proporciona asistencia a la codificación inteligente para todos los lenguajes que soporta. La indexación inicial del código fuente permite al IDE crear un mapa virtual de su proyecto. Gracias a la información del mapa virtual, puede detectar errores sobre la marcha, sugerir variantes de finalización de código con un conocimiento del contexto preciso, realizar refactorizaciones y más (JETBRAINS, 2021).

3.2.9 SQL Server 2012

Microsoft SQL Server se considera uno de los sistemas de administración de bases de datos relacionales más populares, con décadas de uso en el mundo corporativo. SQL proviene del término en inglés Structured Query Language, es decir, lenguaje de consulta estructurada. Es un lenguaje que permite leer, insertar, actualizar y eliminar datos, principalmente en las bases de datos (Sanchez, 2021).

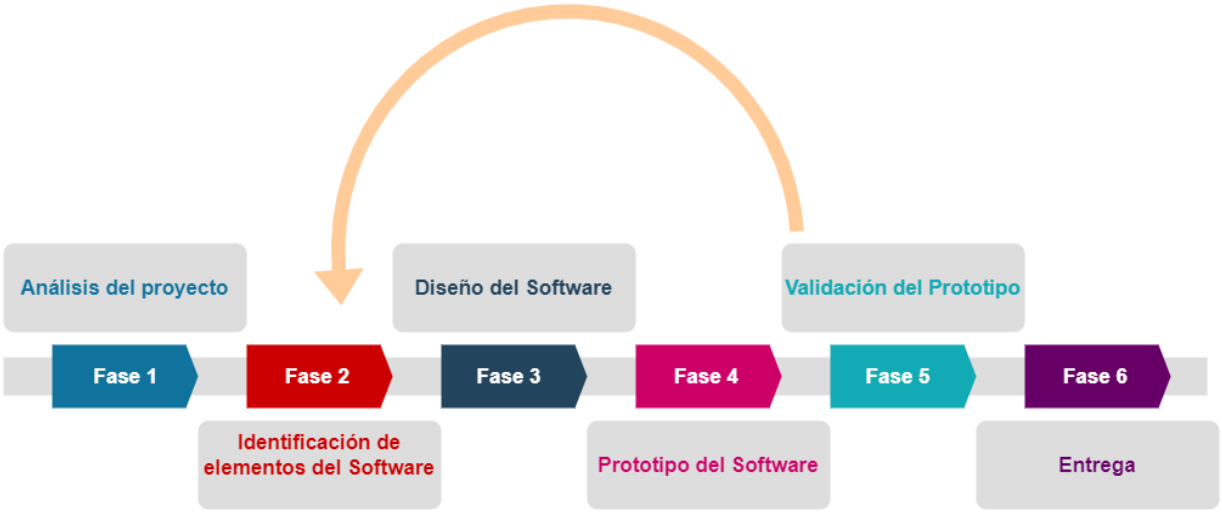
3.2.10 Postman

Postman es una herramienta multiplataforma muy popular entre los desarrolladores que permite publicar, compartir, probar y documentar APIs de manera sencilla y rápida. A través de la aplicación es posible crear peticiones HTTP/HTTPs de la complejidad deseada y a su vez obtener la respuesta que las peticiones desencadenan (Sharir,2020).

4. Metodología

En este apartado se describe la metodología que se implementó durante el desarrollo del proyecto. Como se planteó en la etapa de planeación del proyecto, se tuvieron en cuenta las siguientes fases: análisis, identificación de requerimientos, diseño, prototipo, validación y retroalimentación, y entrega del prototipo final.

Figura 2. *Metodología del proyecto*



4.1 Análisis

Esta fase se enfoca en el estudio del sistema de hospitalización y gestión de camas, se determinan los procesos que entran dentro del alcance del proyecto, se investiga más rigurosamente el marco teórico, y se contextualiza con las herramientas de software con las que se va a desarrollar el proyecto.

4.2 Identificación de Requerimientos

Para poder iniciar con el proceso de desarrollo se debe realizar la recolección de información, con las que se va a definir las funcionalidades que se quieren implementar y las herramientas con las que se va a trabajar.

4.3 Diseño

Se aplican los conocimientos de arquitectura de software, se comienza a trabajar con los requerimientos establecidos y a producir los diferentes diagramas que ayudan a estructurar el sistema. Durante el proceso de diseño del software se debe tener en cuenta la persistencia de los

datos, la lógica del negocio y la integración con los sistemas que ya existan dentro de la institución, además se tiene la responsabilidad de exponer una API que permita que el componente de frontend pueda acceder al componente backend.

4.4 Prototipo

Una vez se haya terminado con la primera versión del diseño del software se procede a crear el prototipo, dependiendo del avance del proyecto, se implementarán parcial o totalmente los requerimientos aceptados.

4.5 Validación y Retroalimentación

Se debe comprobar que el prototipo realizado cumple con los requerimientos del sistema propuestos en la segunda fase, en caso de que haya errores o que se pueda mejorar el prototipo, se deben revisar los resultados de las fases anteriores y realizar los cambios necesarios.

4.6 Entrega

Se debe comprobar que el prototipo realizado cumple con los requerimientos del sistema propuestos en la segunda fase, en caso de que haya errores o que se pueda mejorar el prototipo, se deben revisar los resultados de las fases anteriores y realizar los cambios necesarios.

5. Requerimientos

5.1 Listado de Requerimientos

Para el desarrollo de la aplicación web se definieron un conjunto de requerimientos funcionales y requerimientos no funcionales. Los requerimientos funcionales como su nombre lo indica describen las funcionalidades mínimas que un software debe implementar. Mientras que

los requerimientos no funcionales son aquellos que describen el comportamiento que debería tener la aplicación.

5.1.1 Requerimientos Funcionales

Tabla 1. *Requerimientos Funcionales*

Requerimiento: RF001	**Nombre:** Registrar usuario	**Prioridad:** Alta
Descripción: El usuario con permisos de tipo *Administrador* podrá registrar nuevos usuarios desde el módulo de usuario con la siguiente información: usuario, nombre, clave genérica, roles.		
Requerimiento: RF002	**Nombre:** Iniciar sesión	**Prioridad:** Alta
Descripción: El usuario podrá iniciar sesión en la aplicación con los siguientes datos: usuario y contraseña, generando un token que lo autentica y autoriza.		
Requerimiento: RF003	**Nombre:** Cerrar sesión	**Prioridad:** Alta
Descripción: El usuario podrá cerrar sesión en la aplicación, eliminando el token que lo autentica.		
Requerimiento: RF004	**Nombre:** Cambiar contraseña	**Prioridad:** Media
Descripción: El usuario podrá cambiar su contraseña con los siguientes datos: contraseña actual, y contraseña nueva.		
Requerimiento: RF005	**Nombre:** Restablecer contraseña de usuario	**Prioridad:** Media
Descripción: El usuario de tipo *Administrador* podrá restablecer la contraseña de cualquier usuario en caso de que se le haya olvidado.		

Requerimiento: RF006	**Nombre:** Consultar paciente	**Prioridad:** Alta
Descripción: El usuario podrá consultar los pacientes registrados en la institución cuyo estado de ingreso sea *Registrado*.		
Requerimiento: RF007	**Nombre:** Filtrar búsqueda pacientes	**Prioridad:** Alta
Descripción: El usuario podrá filtrar la búsqueda de pacientes por: nombre del paciente, documento del paciente, o número de ingreso.		
Requerimiento: RF008	**Nombre:** Seleccionar paciente	**Prioridad:** Alta
Descripción: El usuario podrá seleccionar el paciente con el que desee hacer el proceso de traslado o liberación.		
Requerimiento: RF009	**Nombre:** Consultar camas	**Prioridad:** Alta
Descripción: El usuario podrá consultar las camas de los servicios hospitalarios.		
Requerimiento: RF010	**Nombre:** Filtrar búsqueda camas	**Prioridad:** Alta
Descripción: El usuario podrá filtrar la búsqueda de camas por: grupo, subgrupo, tipo de cama (opcional) y estado (opcional).		
Requerimiento: RF011	**Nombre:** Seleccionar cama	**Prioridad:** Alta
Descripción: El usuario podrá seleccionar la cama a la que se desea trasladar el paciente, debe estar en estado Desocupada.		

Requerimiento: RF012	**Nombre:** Trasladar paciente	**Prioridad:** Alta
Descripción: El usuario podrá realizar el traslado de cama de un paciente, seleccionando el paciente que se desea trasladar y una cama en estado *Desocupada.*		
Requerimiento: RF013	**Nombre:** Liberar cama	**Prioridad:** Alta
Descripción: El usuario podrá liberar una cama, seleccionando el paciente vinculado a dicha cama.		
Requerimiento: RF014	**Nombre:** Generar reporte de censo	**Prioridad:** Alta
Descripción: El usuario podrá generar un reporte del censo de camas y su ocupación, con ayuda del filtro de búsqueda.		
Requerimiento: RF015	**Nombre:** Generar reporte de censo en vista gráfica.	**Prioridad:** Alta
Descripción: El usuario podrá generar un reporte del censo de camas de forma gráfica dentro de la aplicación donde se visible la información más relevante para el proceso: código de la cama, estado, y paciente si el estado es Ocupada.		
Requerimiento: RF016	**Nombre:** Generar reporte de censo formato PDF.	**Prioridad:** Media
Descripción: El usuario podrá generar un archivo PDF con el resultado de la búsqueda del censo de camas que se descargará automáticamente.		
Requerimiento: RF017	**Nombre:** Generar reporte de censo en vista tabla.	**Prioridad:** Media
Descripción: El usuario podrá visualizar el resultado de la búsqueda del censo de camas en forma de tabla.		

5.1.2 Requerimientos No Funcionales

Tabla 2. *Requerimientos no funcionales*

Requerimiento: RNF001	**Nombre:** Persistencia de datos	**Prioridad:** Alta
Descripción: Se debe garantizar la persistencia de datos durante todas las operaciones realizadas dentro de la aplicación.		
Requerimiento: RNF002	**Nombre:** Accesibilidad desde cualquier navegador	**Prioridad:** Alta
Descripción: Se podrá acceder a la aplicación desde cualquier navegador disponible en los equipos de la institución.		
Requerimiento: RNF003	**Nombre:** Interfaz gráfica como SPA	**Prioridad:** Alta
Descripción: La interfaz gráfica de usuario seguirá una arquitectura de una aplicación de una sola página, o SPA.		
Requerimiento: RNF004	**Nombre:** Confiabilidad de la información	**Prioridad:** Alta
Descripción: La aplicación permitirá hacer operaciones CRUD en tiempo real, garantizando la validez y confiabilidad de la información.		

5.2 Análisis de Requerimientos

5.2.1 Diagramas de Casos de Uso

En este apartado se muestran los diagramas de casos de uso de la aplicación web de cada tipo de usuario, basados en los requerimientos funcionales ya definidos.

Figura 3. *Diagrama de Casos de Uso de Usuario Administrador*

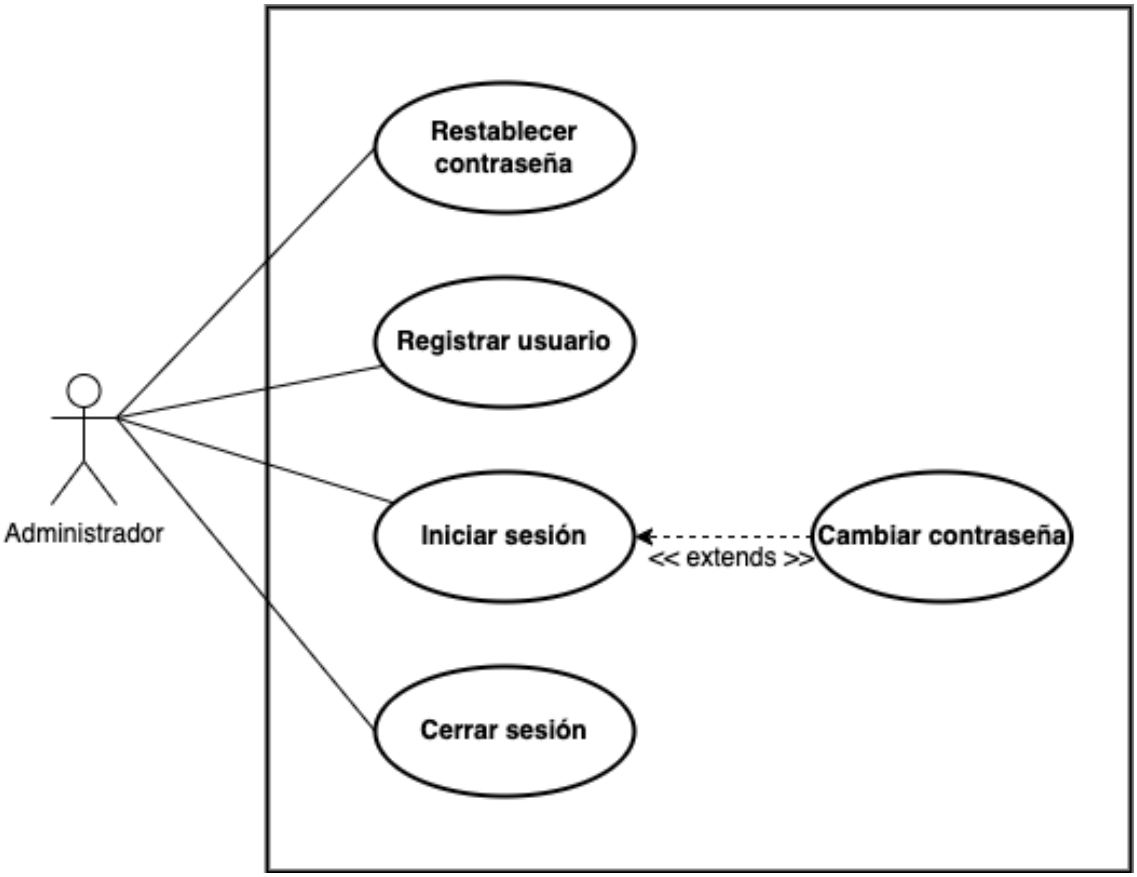


Figura 4. *Diagrama de Casos de Uso de Usuario Jefe de Enfermería*

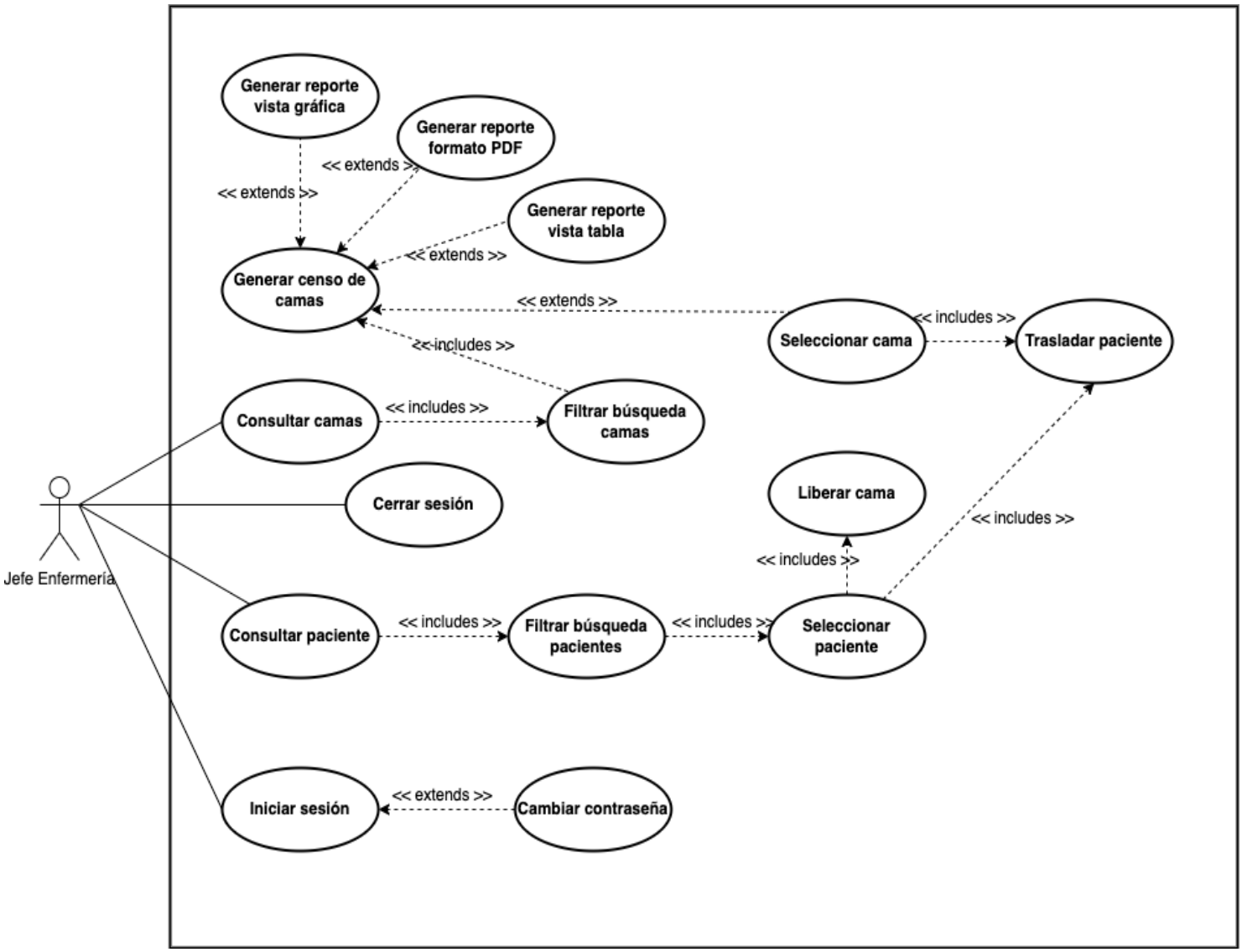
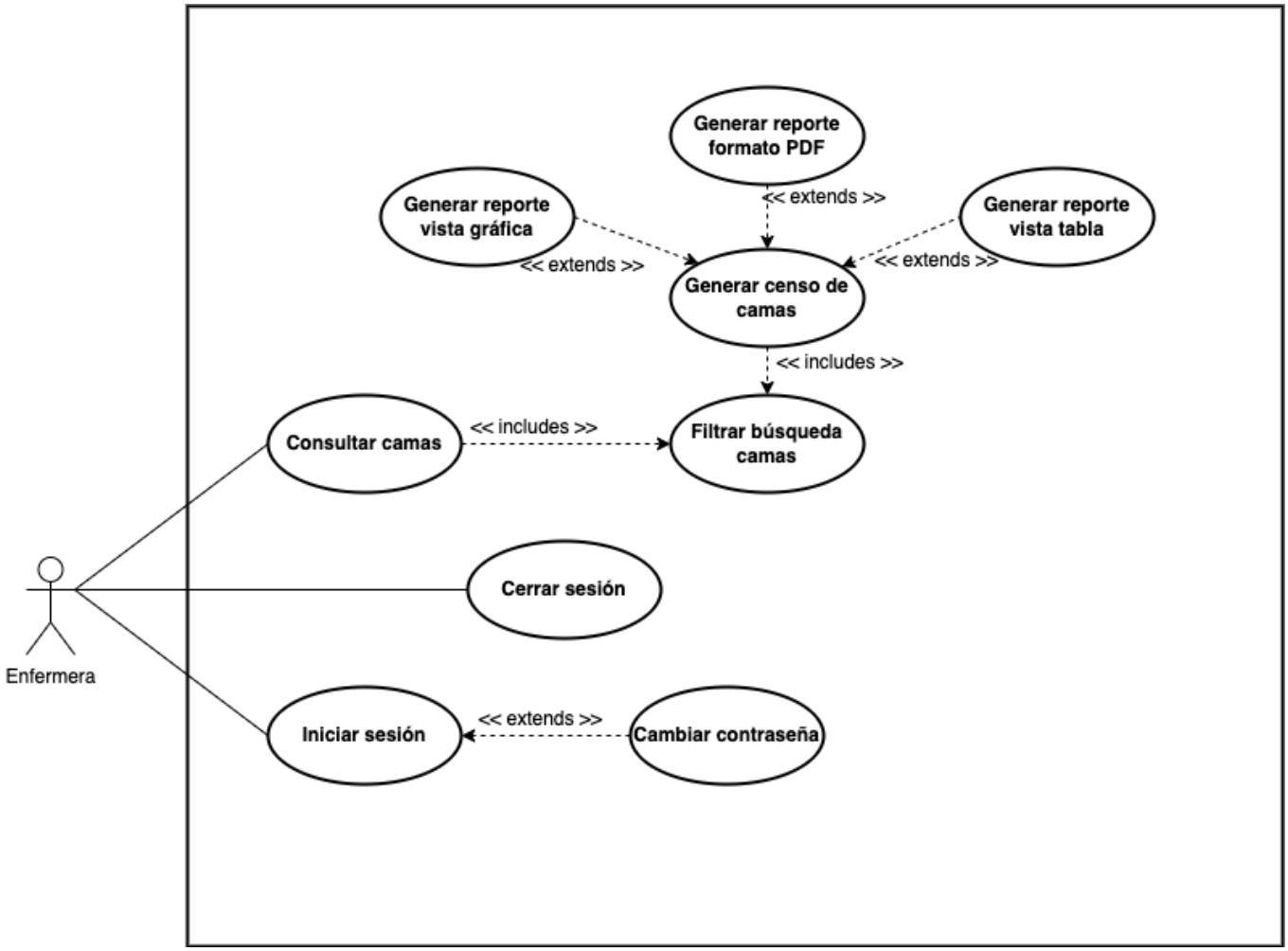


Figura 5. *Diagrama de Casos de Uso de Usuario Enfermera*



5.2.2 Diagramas de Actividades

En este apartado se presentan los diagramas de actividades de las principales funcionalidades de la aplicación web de censo de camas: Traslado, Liberación y Censo de Camas.

Figura 6. *Diagrama de actividades para el traslado de pacientes.*

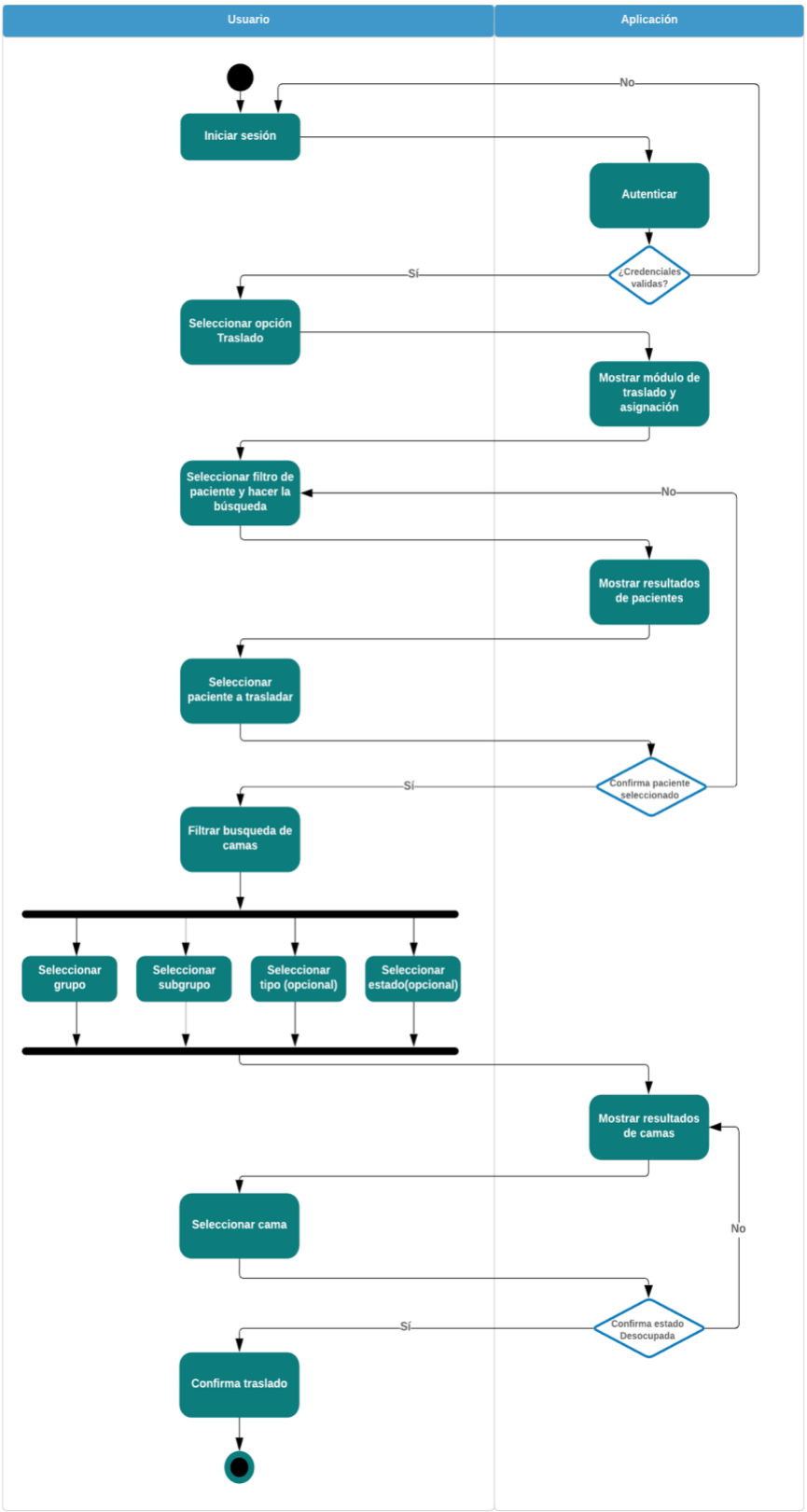


Figura 7. *Diagrama de actividades para la liberación de pacientes.*

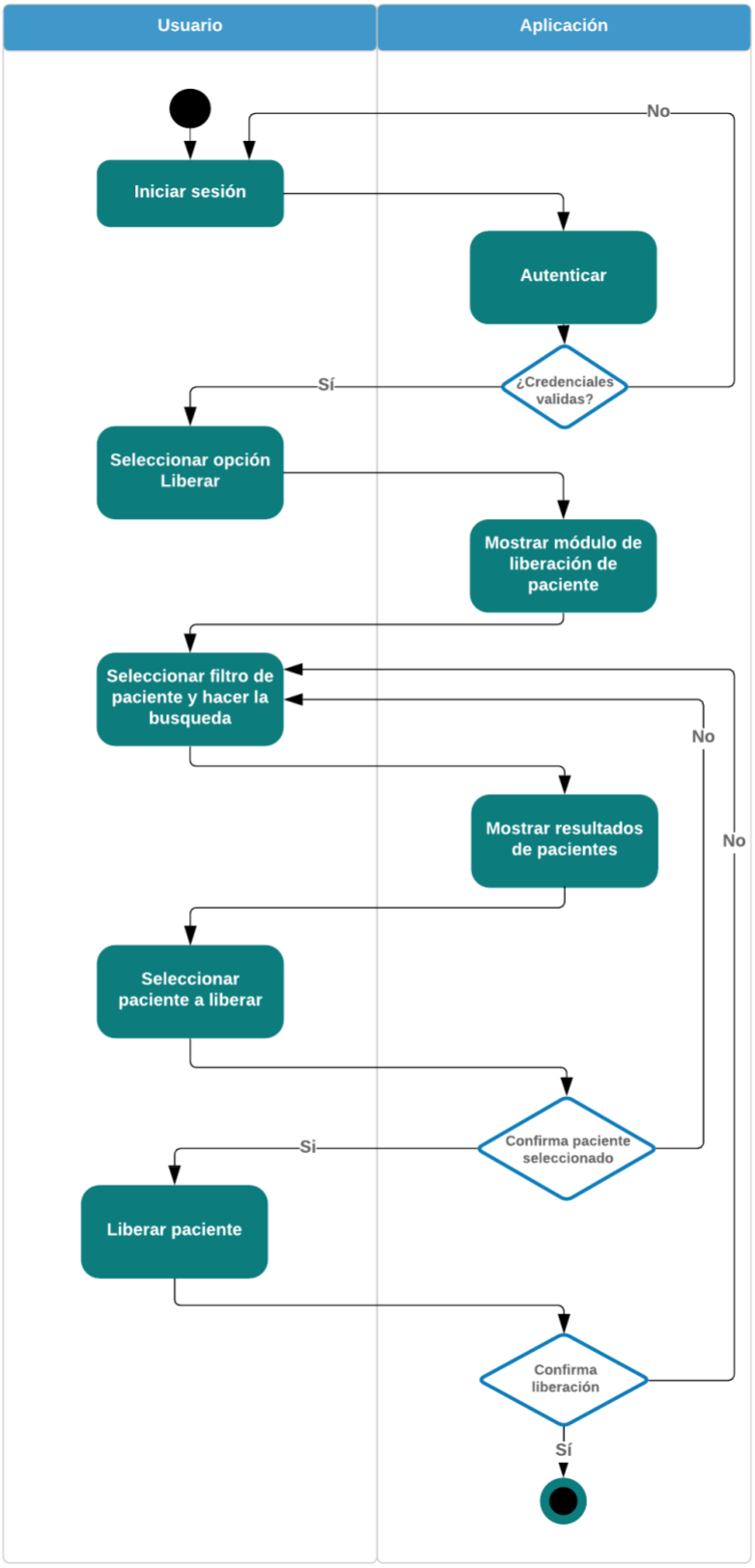
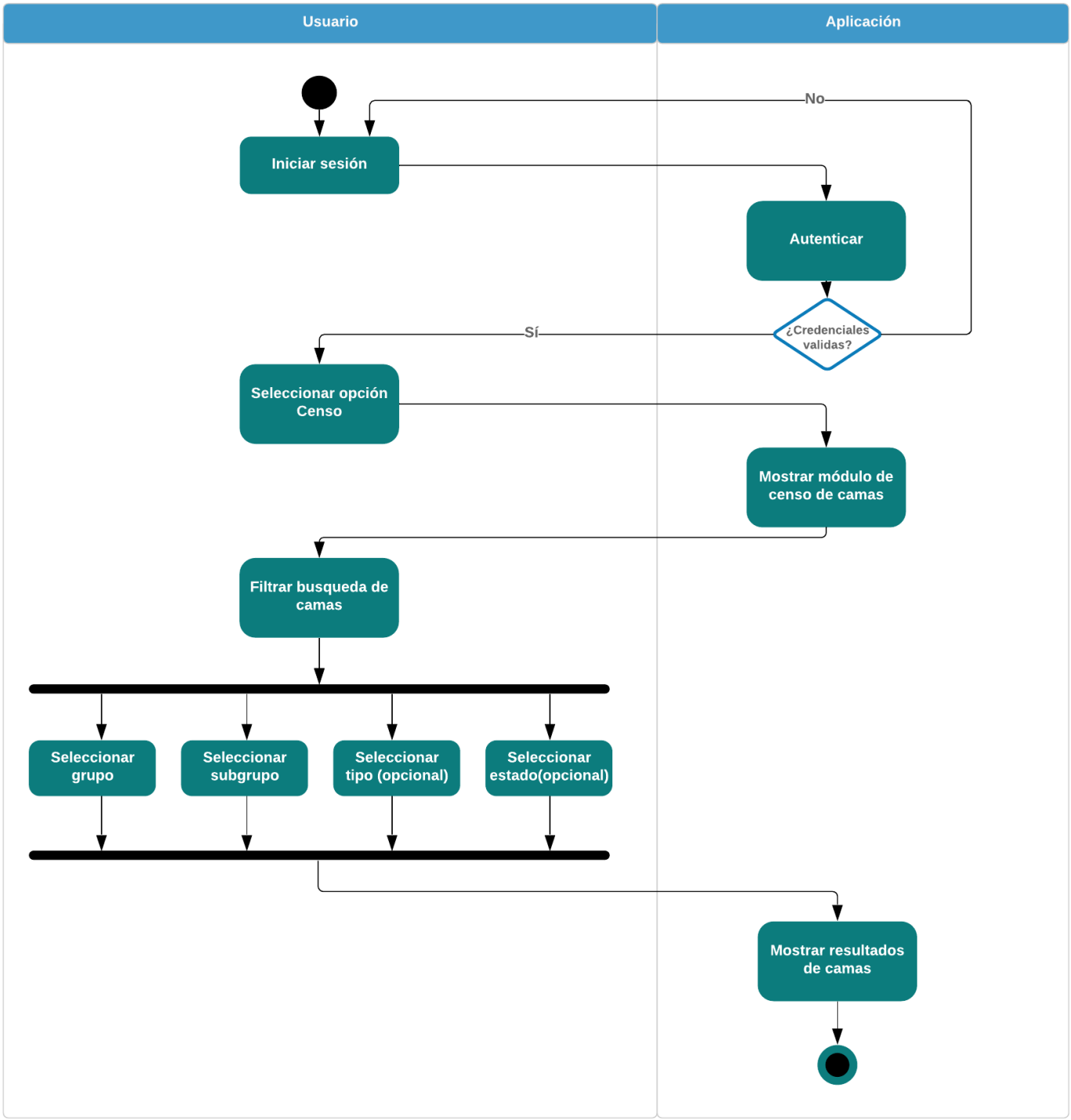


Figura 8. *Diagrama de actividades para el censo de camas.*



6. Diseño de la Solución

6.1 Arquitectura

Teniendo en cuenta que el presente proyecto hace parte de un macro proyecto, debido a su amplio alcance, en esta sección se describe la arquitectura general del aplicativo haciendo énfasis en los elementos que le competen al componente backend.

De acuerdo con las herramientas de desarrollo utilizadas, en la figura 8 se observa la arquitectura en la que se basó el diseño del proyecto. Para el componente frontend se trabajó con el framework Angular, en el cual se desarrolló una aplicación cuyo propósito es desplegar la interfaz gráfica del software de censo de camas, donde el usuario podrá llevar a cabo los respectivos procesos. El frontend utiliza los protocolos HTTP y HTTPs para comunicarse con el servidor que aloja el componente backend.

Por su parte, el componente backend se puede dividir en cuatro capas: modelo, repositorios JPA, servicios, y controladores REST. El modelo se refiere a las clases creadas que representan de manera abstracta los diferentes elementos que participan en los procesos de gestión hospitalaria.

Los repositorios JPA son una de las funcionalidades que ofrece Spring Data JPA, son interfaces que definen métodos query de manera intuitiva sin necesidad de escribir más código. Por medio de ellos es posible comunicarse con la base de datos para realizar las peticiones que el usuario solicite.

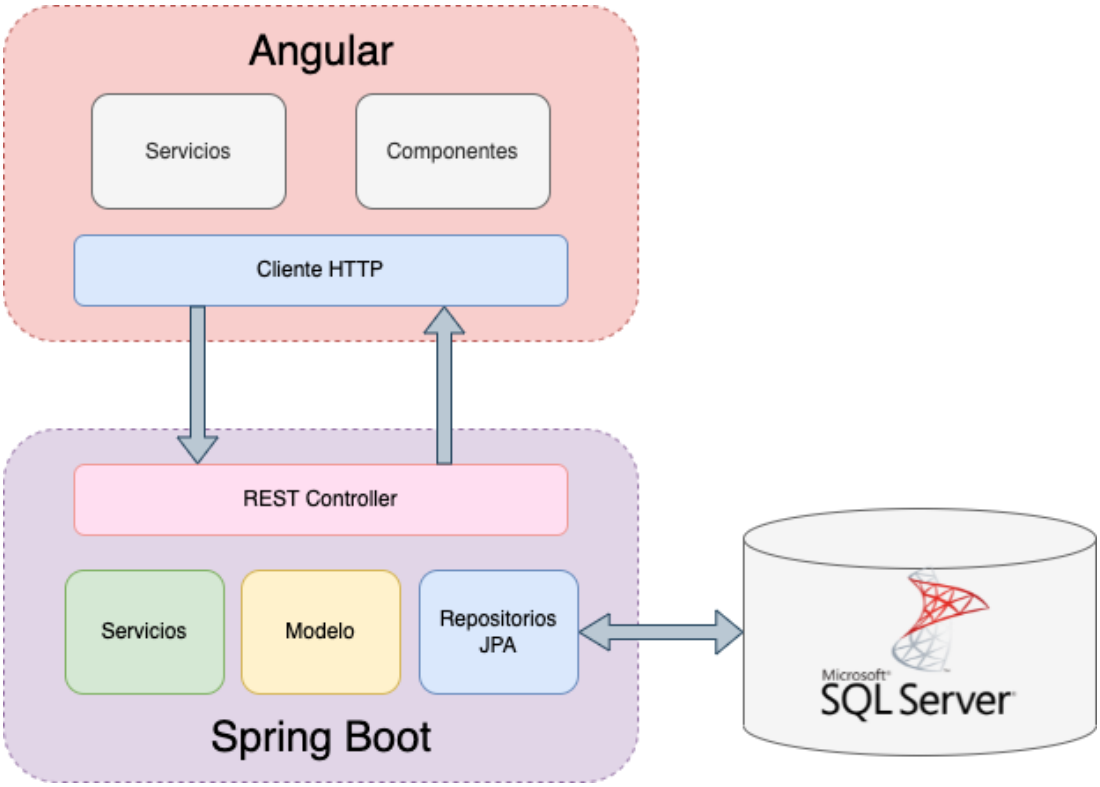
Por otro lado, los servicios son la parte del código que genera las funciones que son llamadas para hacer el debido procesamiento de la información desde el backend. Estos servicios, por medio de los métodos definidos en los repositorios, se encargan de generar búsquedas, realizar cambios, agregar o eliminar información, y validar errores que puedan suceder desde el lado del cliente, es decir, errores que puedan cometer los usuarios en el momento de hacer alguna solicitud.

Por último, el controlador REST, o API, es la sección del código que se encarga de la comunicación entre el cliente y el servidor. En el caso de este proyecto, está compuesto por instrucciones con métodos GET, PUT, POST, y DELETE, que se implementan desde el frontend para realizar peticiones al backend por medio de los protocolos HTTP y HTTPs, como se ha dicho

anteriormente, y de esta manera es posible ver la información y el procesamiento de los datos del lado del cliente (IBM Cloud Education, 2021).

Finalmente, está la base de datos SQLServer 2012, que fue provista por la institución para la que se desarrolló el software. Esta base de datos es la misma que usan actualmente los servicios de hospitalización, por lo que asegura la integridad y persistencia de la información a la que accede nuestro aplicativo.

Figura 9. *Arquitectura del software*



Adaptado de Frontbackend, (2021). https://frontbackend.com/spring-boot/angular-11-spring-boot-2-postgresql

6.2 Diseño de Interfaces

En esta sección se describirá la API diseñada para la comunicación de los dos componentes que forman el macro proyecto.

Como se mencionó anteriormente, la API o REST controller es la sección del código que contiene instrucciones que por medio de operaciones CRUD permite al servidor responder peticiones que se realicen del lado del cliente.

De acuerdo a las necesidades del proyecto, se crea un script para cada uno de los elementos que interactúan en los procesos de gestión hospitalaria que le competen al software.

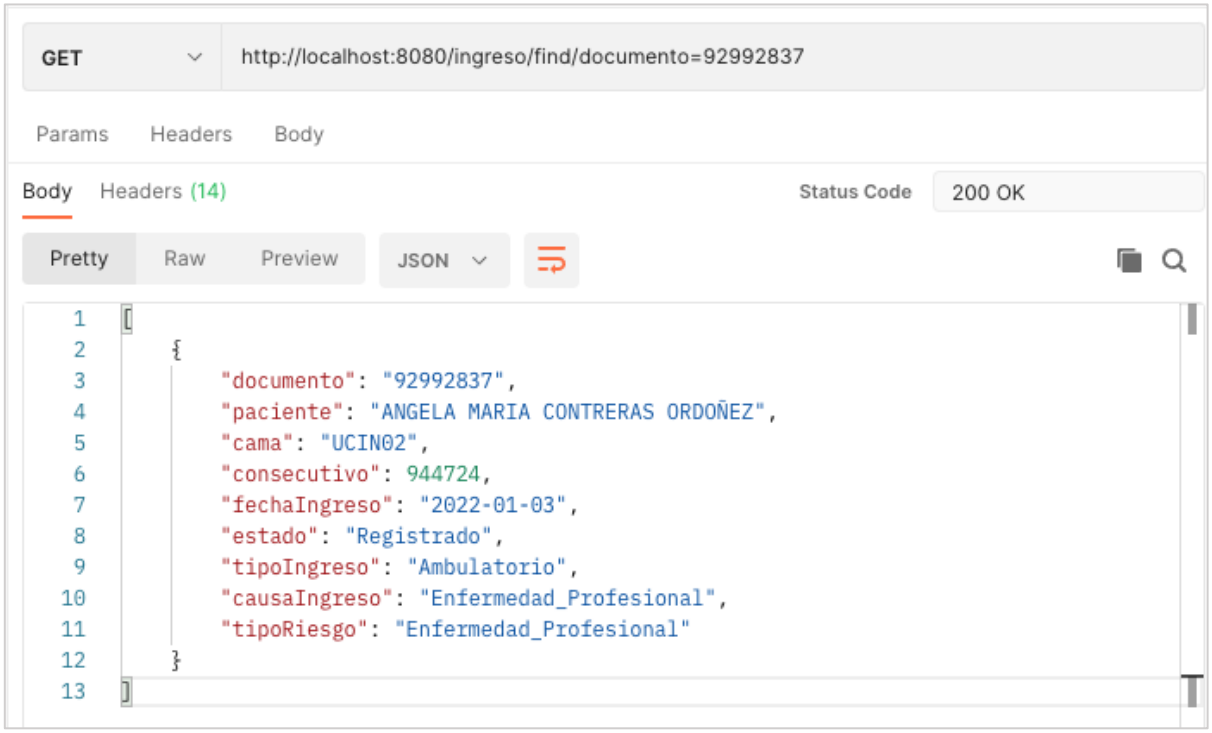
En la siguiente tabla se pueden ver algunas de las peticiones de la API construida:

Tabla 3. *Principales peticiones de la API*

Módulo	Método	URI	Descripción
Ingreso	GET	/ingreso/find/nombre={nombre}	Obtiene los pacientes y su respectivo ingreso, cuyo nombre coincida
Ingreso	PUT	/ingreso/traslado/ingreso={ing}/cama={cama}	Registra un traslado de cama
Ingreso	PUT	/ingreso/liberar/ingreso={ing}	Libera un cama con el ingreso vinculado
Cama	GET	/camas/find/grupo={def}/subgrupo={sub} /tipo={tipo}/estado={est}	Obtiene las camas por grupo, subgrupo, tipo y estado
Usuario	POST	/auth/signup	Registra un usuario en la aplicación
Usuario	DELETE	/auth/delete/usuario	Elimina un usuario de la aplicación

Para un mayor entendimiento de los servicios de la API, a continuación se muestran un par de ejemplos de peticiones realizadas por medio de la herramienta POSTMAN con el fin de imitar el comportamiento de un cliente desde el aplicativo.

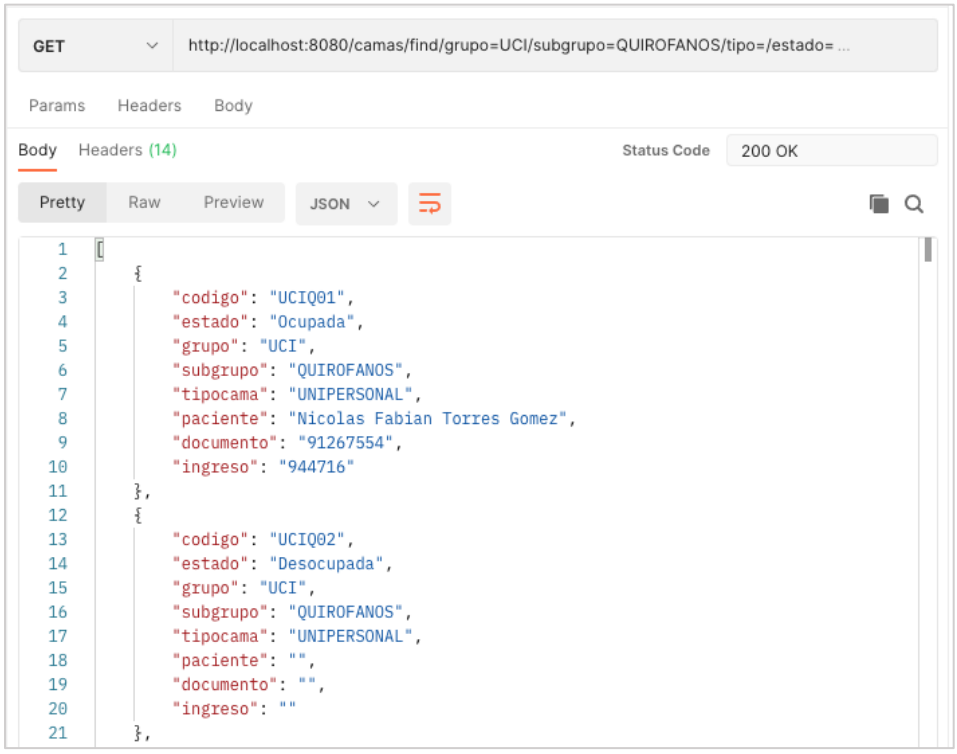
Figura 10. *Ejemplo de petición GET en Postman*



En el ejemplo se está probando el servicio de búsqueda por documento del módulo Ingreso. El cual, en caso de encontrar una coincidencia en el número de documento de un paciente vinculado a un ingreso activo o en estado “registrado”, nos retorna la información más relevante del ingreso encontrado. Es posible confirmar que el paciente se encuentra ocupando una cama, y que el estado del ingreso es “registrado”.

Otro ejemplo relevante es cuando se realizan cambios en la base de datos a través de las peticiones HTTP. Es posible comprobar estos cambios implementando una serie de peticiones, como se puede observar en las siguientes figuras.

Figura 11. *Ejemplo de petición GET antes de un cambio*

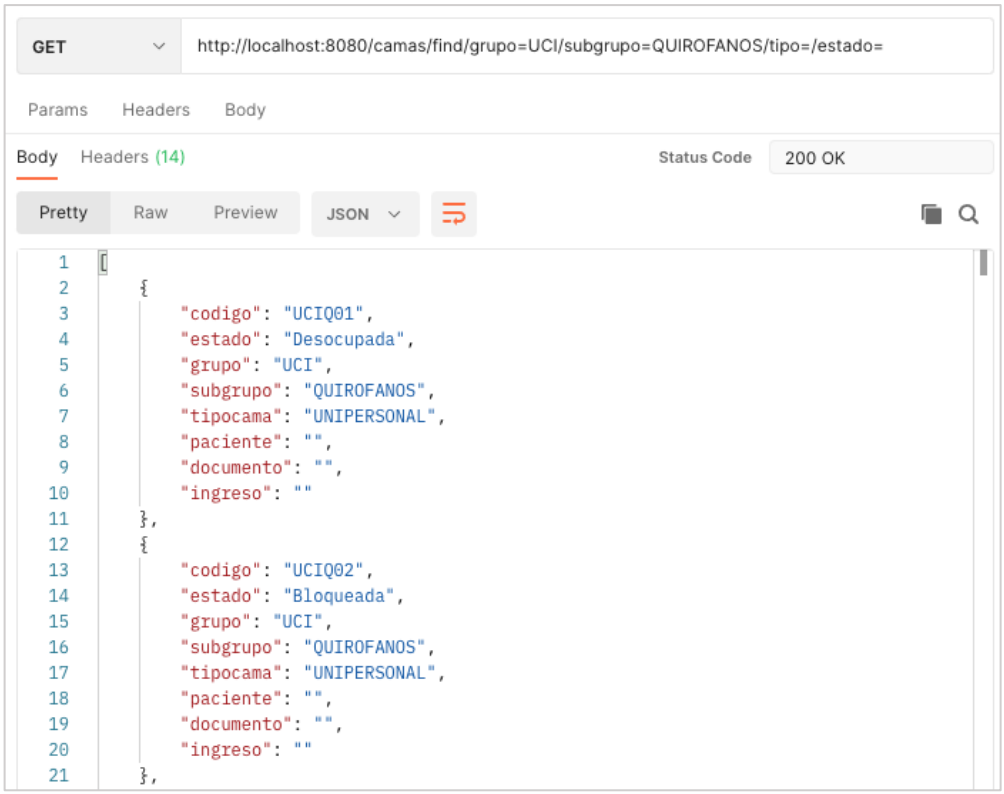


En este ejemplo, se está probando la petición de consulta de camas usando el filtro de búsqueda por grupo, subgrupo, específicamente. Se comprueba que se obtiene un listado de camas con información relevante para los usuarios, y se utilizará la información de la cama *UCIQ01* y el ingreso *944716* para verificar que los cambios realizados se registran en la base de datos.

Figura 12. *Ejemplo de petición PUT que realiza cambios en la información*

En la figura 12 se observa que la cama del ejemplo ha sido liberada por medio de una petición PUT.

Figura 13. *Ejemplo de petición GET después de un cambio*



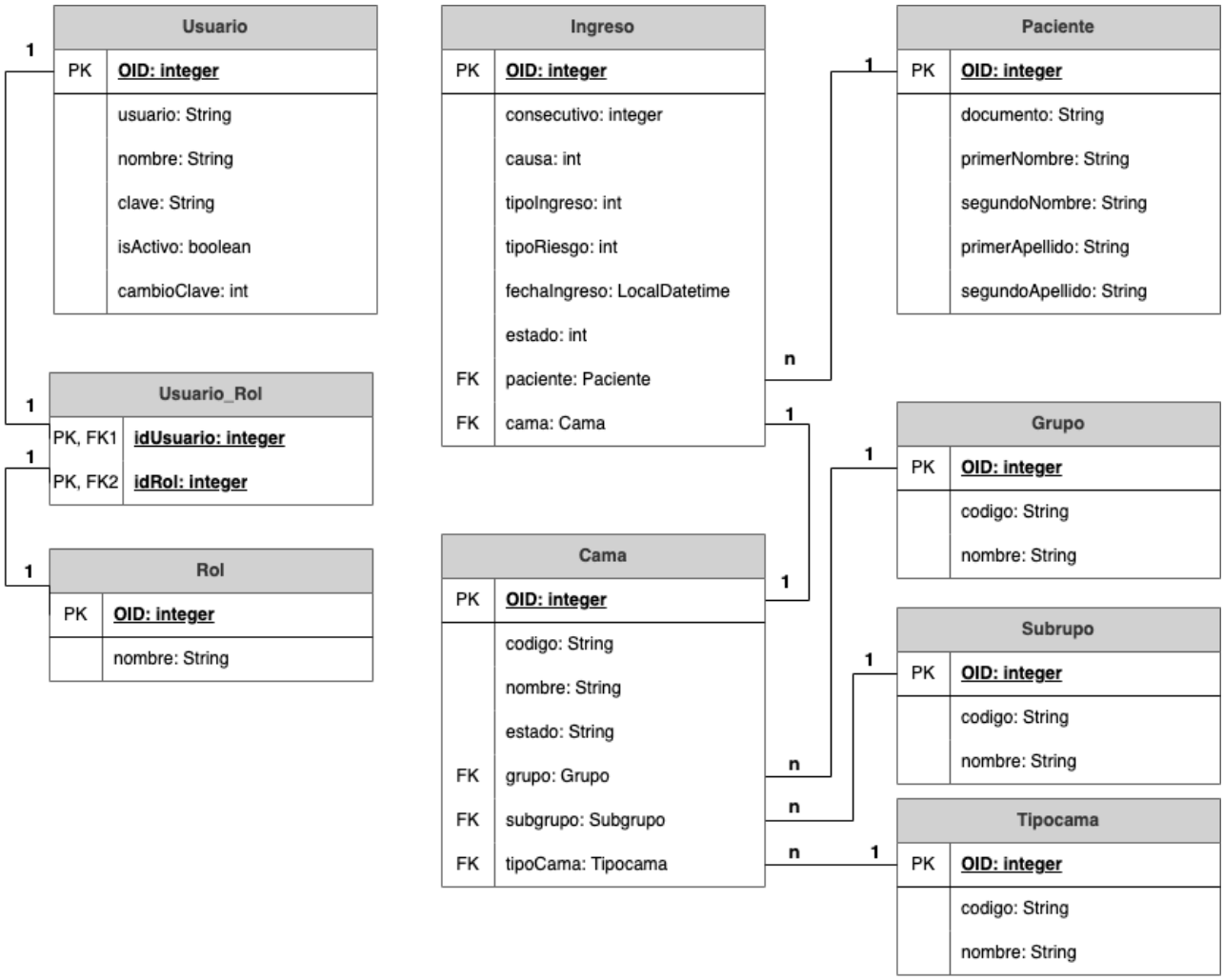
Finalmente, la figura 13 muestra la misma petición de consulta de camas con los mismos filtros de búsqueda, sin embargo se observa un cambio en la cama *UCIQ01*, ya que la petición anterior ha registrado correctamente los cambios.

En el *apéndice B* se encuentran los registros de todos los servicios creados como en los ejemplos anteriores, con el fin de corroborar su funcionalidad.

6.3 Diseño de Datos

6.3.1 Modelo de base de datos

Figura 14. *Diagrama Entidad-Relación*



6.3.2 Descripción del modelo

A continuación se hará una breve descripción del modelo de datos con el que se construyó la aplicación. Es importante aclarar que el proyecto se desarrolló sobre la base de datos que proporcionó la institución, y en su mayoría se utilizaron tablas que ya estaban creadas.

6.3.2.1 Ingreso. Tabla que registra la información sobre los ingresos clínicos de los pacientes en la institución. Esta tabla facilita los resultados de búsqueda de pacientes de las funcionalidades de Traslado y Censo de Camas.

6.3.2.2 Paciente. Tabla que registra la información del paciente, importante para poder mostrar los datos necesarios para escoger un paciente en las funcionalidades de Traslado y Liberación.

6.3.2.3 Cama. Tabla que registra la información de las camas existentes en el hospital. Los atributos más relevantes de la tabla son el código de la cama, el estado, el grupo, subgrupo y tipo de cama.

6.3.2.4 Grupo. Tabla que registra los grupos en los que se clasifican las camas en la institución.

6.3.2.5 Subgrupo. Tabla que registra los subgrupos en los que se clasifican las camas en la institución.

6.3.2.6 Tipo cama. Tabla que registra los tipos en los que se clasifican las camas en la institución.

6.3.2.7 Usuario. Tabla creada para el proyecto, necesaria para la creación y autenticación de los usuarios de la aplicación.

6.3.2.8 Rol. Tabla creada para el proyecto, necesaria para la asignación de los permisos a los usuarios creados. Se crearon tres tipos de rol que permiten delimitar funciones y permisos dentro de la aplicación: *superadmin* para los Administradores, *admin* para las Jefes de enfermería, y *user* para las enfermeras.

6.3.2.9 Usuario_Rol. Tabla creada para el proyecto, registra la relación entre el usuario y sus roles asignados ya que es una relación muchos a muchos.

7. Implementación del Prototipo

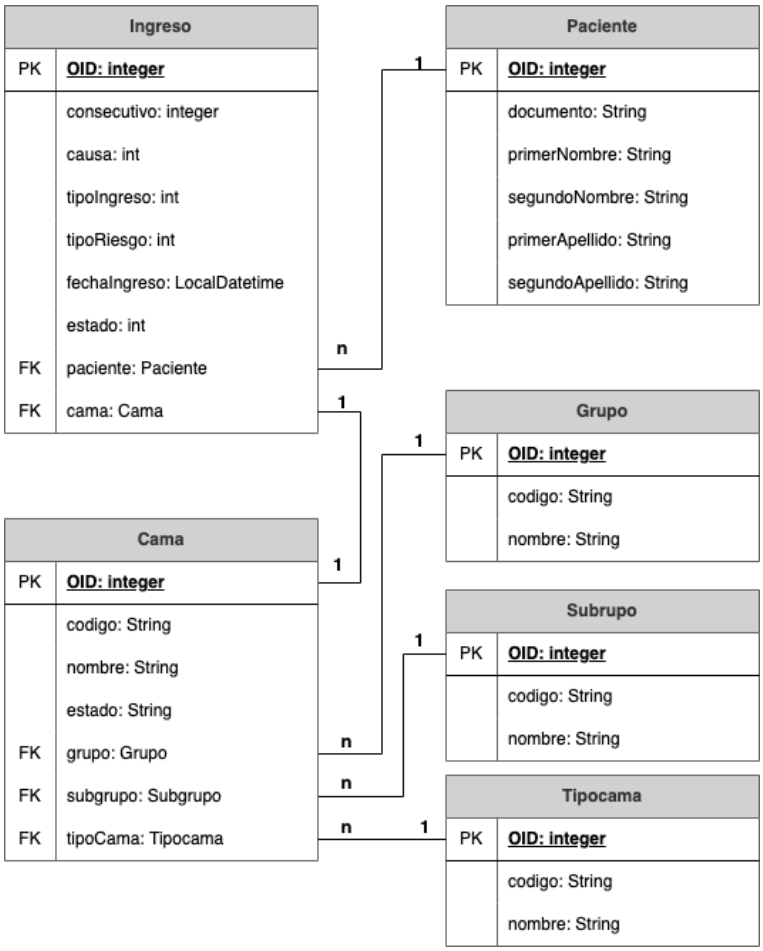
De acuerdo con la metodología planteada, se establece una etapa de validación del prototipo y realimentación. Una vez recibida la retroalimentación por parte de los potenciales usuarios, se realizaron las respectivas correcciones y recomendaciones, por lo que el presente proyecto cuenta con dos versiones del prototipo.

7.1 Primer Prototipo

En primera instancia la aplicación se desarrolló enfocándose en los requerimientos de mayor prioridad según el criterio de la institución. Es decir, los requerimientos que estuvieran directamente vinculados con los procesos de traslado de paciente a una cama, liberación de una cama, filtros de búsqueda y reportes de censo.

Desde el comienzo del proyecto se tenía clara la necesidad de autenticación y autorización de los usuarios del aplicativo, adicional a esto se pretendía reutilizar las credenciales que usa el personal del hospital actualmente para ingresar a su sistema de gestión hospitalaria Dinámica Gerencial. Sin embargo, esto no fue posible por lo que este prototipo no tiene en cuenta la implementación de autenticación de usuario.

Figura 15. *Diagrama de base de datos del primer prototipo*



Como describe la arquitectura del componente backend, este prototipo cuenta con cuatro capas: modelo, repositorio, servicio y controlador.

Para tener un mayor entendimiento del código fuente, se describirán pequeños fragmentos junto con las anotaciones utilizadas.

7.1.1 Modelo

Cuando se habla de modelo, se hace referencia a las entidades involucradas en los procesos de gestión de camas, a su vez, una entidad también representa una tabla en la base de datos. Las

clases entidad pretenden abstraer la información necesaria de cada componente que modelará las tablas en la base de datos.

Figura 16. *Fragmento de código de una clase tipo Entidad*

```
@Entity
@Table(name = "ADNINGRESO")
public class Ingreso implements Serializable {
    @Requerimiento
    @Column(name = "OID", updatable = false,nullable = false)
    private Integer id;
    @ManyToOne
    @JoinColumn(name="GENPACIEN", referencedColumnName = "OID")
    private Paciente paciente;
}
```

La anotación @Entity declara la clase como una entidad JPA, y a su vez indica que corresponde a una tabla dentro de la base de datos. Por defecto, si no se usa la anotación @Table la clase entidad interpreta que el nombre de la tabla es el mismo que el de la clase definida. En nuestro caso, es necesario utilizar esta anotación dado que las tablas utilizadas estaban previamente definidas y tenían un nombre específico que debía ser reconocido por la aplicación.

Dentro de las clases entidad, se definen los atributos que representan los campos de interés dentro de las tablas de la base de datos. En el proyecto se utilizaron las siguientes anotaciones:

- **@Id**: indica que el atributo representa la llave primaria en la tabla de la base de datos.
- **@Column:** permite personalizar el nombre del campo que representa el atributo, además validar si puede ser nulo o si puede ser actualizado.
- **@GeneratedValue:** indica que el valor del campo será generado automáticamente en caso de que un nuevo objeto de esa clase sea creado. Además, es posible declarar atributos que representen una relación entre una entidad y

otra. Al igual que en las relaciones entre tablas en una base de datos, hay diferentes tipos de relaciones y cada una tiene su respectiva anotación:

- **@OneToOne:** indica una relación de uno a uno, es decir que solo puede existir una relación entre las dos entidades.
- **@OneToMany:** indica una relación de uno a muchos.
- **@ManyToOne:** indica una relación muchos a uno.
- **@ManyToMany:** indica una relación muchos a muchos, en este caso esto representa una tabla diferente en la base de datos, sin necesidad de tener una entidad diferente.
- **@JoinColumn:** define la columna que representa la relación con otra tabla, se indica el nombre de la columna, y el campo de la otra tabla que guarda la relación, es decir la clave foránea.

7.1.2 Repositorio

El repositorio permite la comunicación entre el aplicativo y la base de datos, de esta forma es posible hacer peticiones y registrar operaciones.

Figura 17. *Fragmento de código de una clase tipo Repositorio*

```
public interface GrupoRepo extends JpaRepository<Grupo, Integer>
```

Si se quiere definir una clase como repositorio, se implementa como interfaz y se extiende JPA repository. Cada repositorio pertenece a una única clase entidad, por lo que se especifica la clase a la que pertenece, en este caso sería Grupo, y el tipo de atributo que define su clave primaria o @Id.

Figura 18. *Ejemplo de métodos generalizados en un repositorio*

```
List<Grupo> findAll();

Grupo findById(Integer id);
```

JPA Repository provee métodos generalizados como findAll, findById, add, delete, y save.

Figura 19. *Ejemplo de métodos que utilizan parámetros*

```
Grupo findByNombre(String nombre);

Paciente findByNombreAndApellido(String nombre, String apellido);
```

También permite definir métodos CRUD sin necesidad de escribir más líneas de código, simplemente se debe definir de manera intuitiva la operación que se desea realizar.

Figura 20. *Ejemplo de query utilizando lenguaje nativo SQL*

```
@Query(value = "Select DISTINCT g.HGRNOMBRE from HPNGRUPOS as g", nativeQuery = true)
List<String>findAllNames();
```

Además, también se pueden crear querys en lenguaje SQL utilizando la anotación @Query y el parámetro nativeQuery con valor *true*, para poder hacer peticiones más especializadas como el ejemplo anterior.

7.1.3 Servicio

La capa de servicio se encarga de la lógica de negocio cuando su complejidad aumenta y no es posible incluirla en el modelo. En el caso del proyecto esto ocurre gracias a la estrecha

relación entre algunas de las entidades modeladas, como es el caso de la entidad Ingreso, Paciente y Cama.

Figura 21. *Fragmento de código de una clase tipo Servicio*

```
@Service
@Transactional
public class IngresoService {
    private final IngresoRepo ingresoRepo;
    private final CamaRepo camaRepo;
    private final PacienteRepo pacienteRepo;
...
}
```

Para definir una clase de este tipo se utiliza la anotación @Service, esto permite que Spring Boot la reconozca como servicio.

Aunque la anotación @Transactional es opcional, se implementó en el proyecto ya que uno de los requerimientos era la persistencia de datos. Esta anotación provoca que cuando un método sea llamado, se implemente como si fuera una transacción, por lo que en caso de que un error arroje una excepción y haya ocurrido algún cambio dentro de la base de datos previamente, Spring Boot pueda detectarlo y que sea posible retroceder al estado inicial de la transacción. Gracias a este comportamiento es posible garantizar hasta cierto punto la persistencia de la información.

Dentro de la clase servicio se definen los repositorios necesarios para implementar la lógica de negocio. En este caso, se importan tres repositorios diferentes por la complejidad de las relaciones como se mencionó anteriormente.

Figura 22. *Ejemplo de varios repositorios en una misma clase Servicio*

```
public IngresoDTO findByConsecutivo(int consecutivo){
    Ingreso ingreso = ingresoRepo.findIngresoRegistradoByConsecutivo(consecutivo)
    .orElseThrow(()->
    new ObjectNotFoundException("\nNo hay ingreso con estado Registrado con ese id "+consecutivo+"\n"));
    Paciente paciente  = pacienteRepo.findPacienteById(ingreso.getIdPaciente())
            .orElseThrow(() -> new ObjectNotFoundException("No se encuentra paciente"));
    return mapIngresoDTO(paciente, ingreso);
}
```

Por ejemplo, se puede observar que uno de los métodos del servicio de Ingreso es findByConsecutivo. Teniendo en cuenta el fragmento de la *sección 7.1.1*, la entidad Ingreso no contiene información sobre el nombre del paciente, simplemente su identificador asociado, por lo que con el objetivo de que la información sea clara y amigable con el usuario, era necesario obtener el nombre del paciente vinculado. Debido a esto se hace uso del repositorio de Paciente.

Figura 23. *Ejemplo de implementación de DTO*

```
private IngresoDTO mapIngresoDTO(Paciente paciente, Ingreso ingreso){
    IngresoDTO nuevo = new IngresoDTO();
    nuevo.setDocumento(paciente.getDocumento());
    nuevo.setPaciente(paciente.getNombreCompleto().toUpperCase(Locale.ROOT));
    nuevo.setCama(setCamaIngreso(ingreso));
    return nuevo;
}
```

En el caso de algunos servicios, se crearon DTOs, o Data Transfer Object, que permiten transformar la información y facilitan la transferencia de datos del backend hacia el frontend. En este caso, se utilizó para cambiar el formato del nombre del paciente.

7.1.4 Controlador REST

El controlador REST tiene como propósito comunicar a los usuarios con los servicios creados en el componente backend por medio de peticiones HTTP.

Figura 24. *Fragmento de código de clase tipo Controlador REST*

```
@RestController
@RequestMapping("/ingreso")
public class IngresoResource {…}
```

La anotación @RestController permite crear servicios web REST. Esta anotación permite que Spring Boot reconozca la clase como controlador REST y además, incluye por defecto la anotación @RequestBody en los métodos de la clase, por lo que infiere que todos van a retornar una respuesta HTTP.

La anotación @RequestMapping asigna una petición HTPP a un método o clase. En este caso agrupa un conjunto de métodos bajo un prefijo "/ingreso" en las peticiones definidas dentro de la clase.

Figura 25. *Ejemplo de petición de tipo GET en un controlador*

```
@GetMapping("/find/id={id}")
public ResponseEntity<IngresoDTO> getIngresoById(@PathVariable("id") int id) {
    return ResponseEntity.ok().body(ingresoService.findByConsecutivo(id));
}
```

En el fragmento anterior, tendríamos como petición final lo siguiente: "/ingreso/find/id={id}".

Además, se usan diferentes etiquetas para determinar que tipo de petición espera el servidor por parte del usuario, y estas etiquetas deben definir la ruta de la petición:

- **@GetMapping**
- **@PutMapping**

- **@PostMapping**
- **@DeleteMapping:** Dentro de los métodos del controlador, se utilizan las siguientes anotaciones para indicar que tipo de atributo espera la petición:
- **@PathVariable:** como en el fragmento de código, indica que el atributo se encuentra dentro de la petición HTTP. Un ejemplo de petición sería "/ingreso/find/id=123".
- **@RequestBody:** indica que el atributo se obtiene desde el cuerpo de la petición, en forma de objeto o formato JSON.
- **@RequestParam:** indica que el atributo viene de los parámetros de la petición. Un ejemplo sería "/ingreso?id=222".

7.2 Segundo Prototipo

En el segundo prototipo, después de la etapa de retroalimentación por parte del personal del hospital, se implementó la autenticación de los usuarios, se agregó el filtro de nombre de paciente en la búsqueda de ingresos, y se agregó el filtro de estado en la sección de búsqueda de camas.

7.2.1 Autenticación de Usuarios

Se agregó un modelo, repositorio, servicio y controlador para representar la entidad de Usuario dentro de la aplicación. Adicional a esto, se crearon tres nuevas tablas dentro de la base de datos: usuarios, roles, y usuario_rol. Se puede encontrar una explicación más detallada de las tablas creadas durante el desarrollo del proyecto en la *sección 6.3.2*.

Gracias a la implementación de la autenticación y la definición de los roles, fue posible delimitar las acciones dentro de la aplicación de cada usuario dependiendo de su rol.

Figura 26. *Implementación para restringir autorización en métodos*

```
@PreAuthorize("hasRole('SUPERADMIN')")
@PostMapping("/signup")
public ResponseEntity<?> nuevo(@Validated @RequestBody NuevoUsuario nuevoUsuario){...}
```

Por ejemplo, con el fin de restringir el acceso a la aplicación únicamente a personal autorizado dentro del hospital, se utilizó la anotación @PreAuthorize que verifica el rol del usuario antes realizar una petición. En este caso, solo los usuarios con rol Superadmin pueden registrar usuarios en la aplicación.

7.2.2 Listado de Peticiones HTTP

A continuación, se presentan los servicios creados por entidad en la aplicación, con una breve descripción.

Tabla 4. *Servicios de Ingreso*

URI	Método	Descripción
/ingreso/find/id={id}	GET	Obtiene ingresos por Id Consecutivo
/ingreso/find/documento={doc}	GET	Obtiene ingresos por documento del paciente vinculado
/ingreso/find/nombre={nombre}	GET	Obtiene ingresos por nombre del paciente vinculado

/ingreso/traslado/ingreso={ing}/ cama={cama}	PUT	Realiza traslado de cama de un paciente
/ingreso/liberar/ingreso={ing}	PUT	Realiza liberación de una cama dependiendo del ingreso vinculado que ocupa la cama

Tabla 5. *Servicios de Cama*

URI	Método	Descripción
/camas/find/grupo={def}/subgrupo={sub}/ tipo={tipo}/estado={est}	GET	Obtiene las camas por grupo, subgrupo, tipo* y estado* *opcional
/camas/find/codigo={codigo}	GET	Obtiene cama por código
/camas/bloquear/codigo={codigo}	PUT	Bloquea cama por código
/camas/desbloquear/codigo={codigo}	PUT	Desbloquea cama por código

Tabla 6. *Servicios de Grupo*

URI	Método	Descripción
/grupo/all	GET	Obtiene todos los grupos registrados
/grupo/find/nombre={nombre}	GET	Obtiene grupo por el nombre

Tabla 7. *Servicios de Subgrupo*

URI	Método	Descripción

/subgrupo/all	GET	Obtiene todos los subgrupos registrados
/subgrupo/find/nombre={nombre}	GET	Obtiene subgrupo por el nombre
/subgrupo/find/grupo={gr}	GET	Obtiene los subgrupos por grupo al que pertenecen

Tabla 8. *Servicios de Tipo Cama*

URI	Método	Descripción
/tipocama/all	GET	Obtiene todos los tipos de camas registrados
/tipocama/find/nombre={nombre}	GET	Obtiene tipo de cama por el nombre
/tipocama/find/grupo={g}&subgrupo={s}	GET	Obtiene los tipos de camas por grupo y subgrupo al que pertenecen

Tabla 9. *Servicios de Usuario*

URI	Método	Descripción
/auth/signup	POST	Registra un usuario
/auth/login	POST	Inicia sesión
/auth/cambiar-clave	POST	Permite cambia la clave

/auth/reset-clave	POST	Permite al administrador restaurar la clave de un usuario
/auth/find/username={username}	GET	Obtiene usuario por el username
/auth/habilitar/{username}	PUT	Habilita un usuario
/auth/deshabilitar/{username}	PUT	Deshabilita un usuario
/auth/delete/{usuario}	DELETE	Elimina un usuario

Tabla 10. *Servicios de Paciente*

URI	Método	Descripción
/pacientes/find/{id}	GET	Obtiene pacientes por el ID de la tabla
/pacientes/encontrar/documento /{documento}	GET	Obtiene pacientes por el número de documento
/pacientes/encontrar/nombre /{nombrePaciente}	GET	Bloquea cama por código

8. Validación del Prototipo

Una vez se consideró que el aplicativo cumplía con todos los requerimientos establecidos en la fase inicial del proyecto, se diseñó un plan de pruebas como se propuso en la metodología, con el fin de validar y verificar el correcto comportamiento de todas las funcionalidades implementadas. Estas validaciones consistieron en pruebas de software y pruebas de usuario.

8.1 Pruebas de Software

Las siguientes tablas muestran el proceso y los resultados de las pruebas de software de los procesos más relevantes dentro del componente backend. Es importante recordar los actores que se definieron en la *sección 5.2.1*: superadmin, jefe de enfermería que tiene un rol de administrador (admin) y enfermera que tiene un rol de usuario (user).

Tabla 11. *Prueba iniciar sesión usuario*

HTTP request	/auth/login	/auth/login	/auth/login
Usuario	superadmin		superadmin
Contraseña	superadmin	superadmin	
API	Status: 200 OK Devuelve token de autorización	Status: 400 Bad Request “Usuario o contraseña incorrecta”	Status: 400 Bad Request “Usuario o contraseña incorrecta”
Resultado	Correcto	Correcto	Correcto

Tabla 12. *Prueba registrar usuario*

HTTP request	/auth/signup	/auth/signup	/auth/signup
Rol del usuario que registra	superadmin	admin	user
Usuario	GARA	EAP	MOPS
Contraseña	sadmin1	admin1	user1
API	Status: 200 OK Devuelve token de autorización	Status: 401 Unauthorized “No autorizado”	Status: 401 Unauthorized “No autorizado”
Resultado	Correcto	Correcto	Correcto

Tabla 13. *Prueba cambiar contraseña*

HTTP request	/auth/cambiar-clave/old=superadmin/new=sadmin1	/auth/cambiar-clave/old=admin1/new=superadmin1
Rol del usuario	superadmin	admin

Usuario	GARA	GARA
Contraseña actual	superadmin	admin1
Contraseña nueva	sadmin1	superadmin1
API	Status: 200 OK "Cambio de contraseña exitoso"	Status: 400 Bad Request "Contraseña actual incorrecta"
Resultado	Correcto	Correcto

Tabla 14. *Prueba filtro de búsqueda de ingresos*

HTTP request	/ingreso/find/nombre	/ingreso/find/documento	/ingreso/find/id	/ingreso/find/nombre
Rol del usuario	user	user	user	user
Filtro seleccionado	Nombre	Nº de documento	Nº de ingreso	Nº de ingreso
Información ingresada	María	10991929	111211	123
Estado Ingreso	Registrado	Registrado	Registrado	Bloqueado
API	Status: 200 OK Listado de ingresos que coinciden con el nombre	Status: 200 OK Listado de ingresos que coinciden con el documento	Status: 200 OK Ingreso que coincide con el id consecutivo	Status: 200 OK Ingreso que coincide con el id consecutivo
Resultado	Correcto	Correcto	Correcto	Incorrecto
Error	-	-	-	El número de ingreso indica que es un ingreso muy antiguo, y que tiene estado diferente a Registrado, por lo que no debería aparecer en la búsqueda.
Corrección				Se valida en la lógica del repositorio que deben filtrarse la búsqueda solo por ingresos con estado registrado

Como ha sido probado y corregido el filtro de búsqueda de ingresos, se hace la prueba de traslado y liberación de camas con ingresos que han sido validados por la versión final del filtro.

Tabla 15. *Prueba de traslado de camas con ingreso registrado*

HTTP request	/ingreso/traslado/ingreso={ing}/cama={cama}			
Rol del usuario	admin	admin	admin	user
Ingreso	991921	991933	991984	991928
Cama	I06	UCIQ02	UCIQ01	UCIQ01
Estado Cama	Ocupada	Bloqueada	Desocupada	Desocupada
API	Status: 400 Bad Request "La cama está ocupada"	Status: 400 Bad Request "La cama está bloqueada"	Status: 200 OK "El paciente ha sido trasladado"	Status: 401 Unauthorized "No autorizado"
Resultado	Correcto	Correcto	Correcto	Correcto

Tabla 16. *Prueba de liberación de camas con ingreso registrado*

HTTP request	/ingreso/liberar/ingreso={ing}			
Rol del usuario	admin	admin	admin	user
Ingreso	991921	991933	991984	991928
Cama	N06	UCIQ02	UCIQ01	UCIQ01
Estado Cama	Ocupada	Bloqueada	Desocupada	Desocupada
API	Status: 400 Bad Request "La cama está ocupada"	Status: 400 Bad Request "La cama está bloqueada"	Status: 200 OK "El paciente ha sido trasladado"	Status: 401 Unauthorized "No autorizado"
Resultado	Correcto	Correcto	Correcto	Correcto

Tabla 17. *Prueba filtro de búsqueda de camas*

HTTP request	/camas/find/grupo={def}/subgrupo={sub}/tipo={tipo}/estado={est}

Rol del usuario	admin	admin	user	-
Grupo	UCI	Urgencias	-	UCI
Subgrupo	UCI adultos	Urgencias-pediátricas	-	-
Tipo de cama	Unipersonal	-	-	-
Estado	-	Desocupada	Desocupada	Ocupada
API	Status: 200 OK Listado de camas	Status: 200 OK Listado de camas	Status: 400 Bad Request “Campos grupo y subgrupo son obligatorios”	Status: 401 Unauthorized “No autorizado”
Resultado	Correcto	Correcto	Correcto	Correcto

Tabla 18. *Prueba búsqueda de cama por código*

HTTP request	/camas/find/codigo={codigo}			
Rol del usuario	superadmin	admin	user	-
Código	N06	UCIQ02	UCIQ	UCIQ01
API	Status: 401 Unauthorized “No autorizado”	Status: 200 OK Cama que coincide con el código	Status: 400 Bad Request “No encuentra cama con ese codigo”	Status: 401 Unauthorized “No autorizado”
Resultado	Correcto	Correcto	Correcto	Correcto

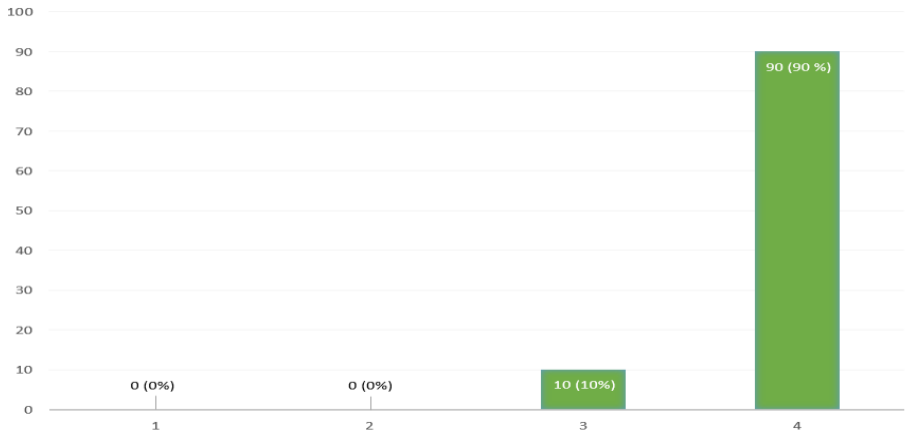
8.2 Pruebas de Usuario

Durante el periodo de entrega y validación del prototipo final, después de realizar una prueba piloto para corroborar que el aplicativo funcionaba de manera correcta en las instalaciones del hospital, se entregó una encuesta a 100 miembros del personal de la institución que tuvieron acceso al prototipo del software y que probaron su funcionalidad.

A continuación, se muestran los resultados de la encuesta.

1. ¿Qué confiabilidad le proporciona los servicios ofrecidos dentro del aplicativo? Los valores proporcionados en la encuesta están dados como 1: ‘poco confiable’ y 4: ‘muy confiable’.

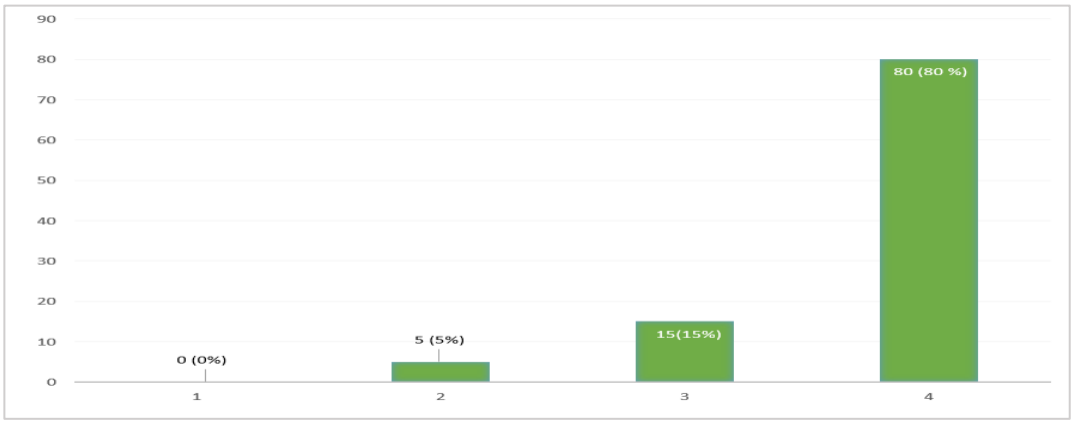
Figura 27. *Resultados de la primera pregunta*



2. ¿Qué tan satisfecho está con la facilidad de uso al realizar sus servicios en el software? Los valores proporcionados en la encuesta están dados como 1: ‘insatisfecho’ y 4: ‘muy satisfecho’.

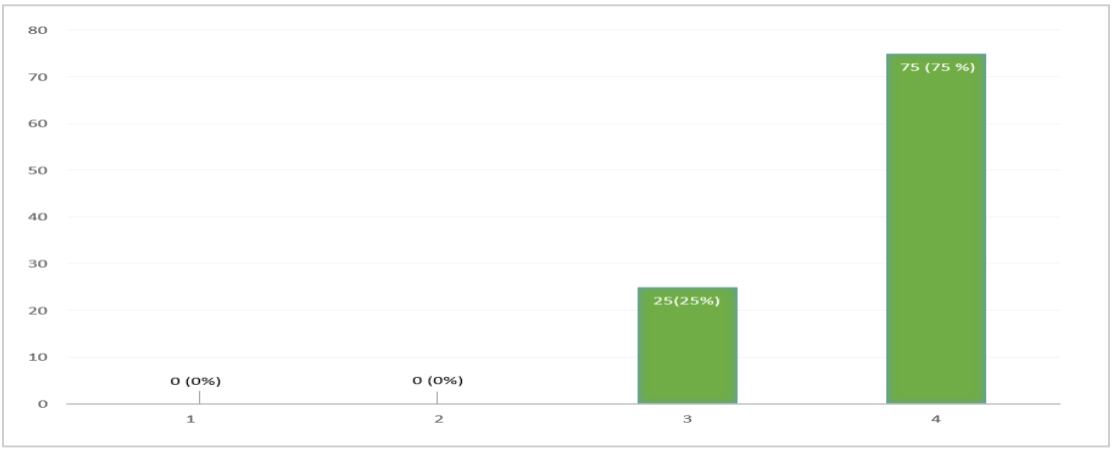
Figura 28. *Resultados de la segunda pregunta*

3. ¿Qué tan satisfecho está con la apariencia del software?



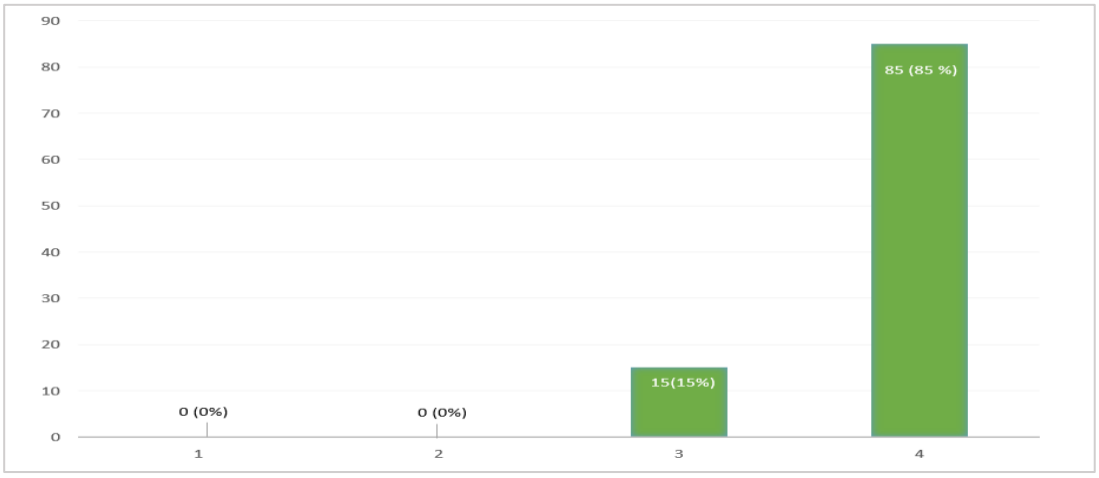
Los valores proporcionados en la encuesta están dados como 1: ‘insatisfecho’ y 4: ‘muy satisfecho’.

Figura 29. *Resultados de la tercera pregunta*



4. ¿Qué tan rápido funciona el software para realizar sus procesos? Los valores proporcionados en la encuesta están dados como 1: ‘muy lenta’;2: lenta, 3: rápida y 4: ‘muy rápida’.

Figura 30. *Resultados de la cuarta pregunta*



9. Conclusiones

Se puede concluir que los objetivos propuestos fueron alcanzados satisfactoriamente, debido a que en este trabajo se logró diseñar el componente backend de una aplicación web con arquitectura Single Page Application, que permite al personal de Enfermería administrar los procesos de traslado, asignación, liberación y censo de camas, de una manera mucho más eficiente, comparando esta solución tecnológica con la alternativa manual que implementan hoy en día.

Este proyecto inició por una necesidad que el Hospital Universitario de Santander expuso a los estudiantes de la Universidad Industrial de Santander, y gracias a esta oportunidad se pudo realizar un aporte considerable a la institución que abre las puertas para que se pueda seguir mejorando el servicio hospitalario de uno de los hospitales más importantes del departamento de Santander.

Quedó demostrado el cumplimiento de los requerimientos que la institución estableció en el periodo de planeación del proyecto, a través de las pruebas de validación y las encuestas en las que participaron los potenciales usuarios del software de censo de camas, donde se valida la satisfacción de los usuarios y además se corrobora la eficacia del sistema diseñado.

Debido al gran alcance del proyecto, y a la culminación del periodo de las prácticas empresariales, se propone por parte de los estudiantes complementar el proyecto con las siguientes recomendaciones finales por parte de los directivos de enfermería involucrados en el proceso:

1. Crear un módulo que permita al usuario, seleccionar una cama que esté ocupada para realizar una petición de traslado, de forma que cuando esta se desocupe en el sistema sea posible asignar al paciente de forma automática y asíncrona.

2. Crear un módulo de egreso que registre la información competente y que integre el proceso del área de facturación.

Referencias Bibliográficas

Brownley, K. (2020). What is Hospital Management - Definition and Look into Responsibilities - Gov Health IT. Recuperado a partir de: https://www.govhealthit.com/what-is-hospital-management/

Gillis, A. (2020). REST API (RESTful API). Recuperado a partir de: https://searchapparchitecture.techtarget.com/definition/RESTful-API

Shklar, L., & Rosen, R. (2009). Web application architecture: Principles, protocols, and practices.

Puskaric, H., Djordjevic, A., Stefanovic, M., & Zahar Đorđević, M. (2019). DEVELOPMENT OF WEB BASED APPLICATION USING SPA ARCHITECTURE. Proceedings on Engineering Sciences, 1, 457–464.

Pisuwala, U. (2021). Fundamentals of web application architecture. Recuperado a partir de: https://www.peerbits.com/blog/web-application-architecture.html

Rodriguez, E. (2020). Qué es un framework y por qué es conveniente usarlo. Recuperado a partir de: https://www.seoestudios.es/blog/que-es-un-framework

Sanchez, E. (2021). What is SQL Server? (Definition, Versions, Editions). Recuperado a partir de: https://codingsight.com/what-is-sql-server-definition-versions-editions/

Phaltankar, A. (2021). What is JPA, Spring Data and Spring Data JPA. Recuperado a partir de: https://www.amitph.com/jpa-and-spring-data-jpa/

Lawson, K. (2021). What is a Single Page Application? Recuperado a partir de: https://www.bloomreach.com/en/blog/2018/07/what-is-a-single-page-application.html

HUS. (2021). Glosario Hospital Universitario de Santander. Recuperado a partir de: http://www.hus.gov.co/glosario/

IBM Cloud Education. (2021). What is a REST API? Recuperado a partir de: https://www.ibm.com/cloud/learn/rest-apis

Sharir, J. (2020). How to use Postman for API Testing Automation. Recuperado a partir de: https://www.blazemeter.com/blog/how-use-postman-manage-and-execute-your-apis

Mekhiya, K. (2016). Software PROTOTYPING — Worth or Not? Recuperado a partir de: https://medium.com/content-writing/software-prototyping-worth-or-not-369a19ba6758

Sagara. (2019). The Fundamentals of Front End and Back End Development. (2021). Recuperado a partir de: https://sagaratechnology.medium.com/the-fundamentals-of-front-end-and-back-end-development-5973ac0910cf

RecaVar. (2020). Hospitales de tercer nivel: descubre los servicios que ofrecen. Recuperado a partir de: https://www.recavar.org/hospitales-de-tercer-nivel

Ministerio de Salud de Perú - MINSA. (2013). Indicadores de gestión y evaluación hospitalaria, para hospitales, institutos y DIRESA. Lima. Recuperado a partir de: http://bvs.minsa.gob.pe/local/MINSA/2739.pdf