PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

Unidad Académica de Formación Técnica y Tecnológica – PUCE TEC

SISTEMA DE GESTIÓN PARA LA ASIGNACIÓN DE AULAS EN LA DIRECCIÓN DE INFORMÁTICA DE LA PUCE

Proyecto de titulación previo a la obtención del título de: Tecnología Superior Universitaria en Desarrollo De Software

Autor: Estévez Toapanta Armando Xavier

y

Guijarro Vallejo Nelson Ricardo

Tutor: Galarraga Cañizares José Luis

Quito, Ecuador

2026

Dedicatoria

Nelson Guijarro:

Dedico este trabajo a mis padres, por su apoyo incondicional, esfuerzo constante y confianza a lo largo de toda mi formación académica. A mi hermana, por su compañía, ánimo y comprensión en cada etapa de este proceso. Y a mi tutor, por su guía, paciencia y valiosos conocimientos que hicieron posible la culminación de esta investigación.

Sin el apoyo y esfuerzo de cada uno de los nombrados, este trabajo no se hubiera podido realizar de manera adecuada y satisfactoria.

Armando Estévez:

Dedico el presente trabajo a las personas que han estado apoyándome desde el primer momento en todos mis sueños y aspiraciones, mis padres y mi hermana. Su confianza y ánimo hicieron que este proyecto sea posible

A mi tutor por la paciencia y los consejos para que podamos superar cada obstáculo.

Y a todos los que nos dieron un apoyo para poder cumplir con nuestro objetivo de ser buenos profesionales.

Tabla de contenidos

DECLARACIÓN y AUTORIZACIÓN

Yo, **Armando Xavier Estévez Toapanta** con C.I. 1718528662 autor(a) del trabajo de titulación intitulado: **"SISTEMA DE GESTIÓN PARA LA ASIGNACIÓN DE AULAS EN LA DIRECCIÓN DE INFORMÁTICA DE LA PUCE"**, previa a la obtención del título de **Tecnólogo Superior en Desarrollo de Software** en la Unidad Académica de Formación Técnica y Tecnológica PUCE TEC:

1.- Declaro tener pleno conocimiento de la obligación que tiene la Pontificia Universidad Católica del Ecuador, de conformidad con el artículo 144 de la Ley Orgánica de Educación Superior, de entregar a la SENESCYT en formato digital una copia del referido trabajo de graduación para que sea integrado al Sistema Nacional de Información de la Educación Superior del Ecuador para su difusión pública respetando los derechos de autor.

2.- Autorizo a la Pontificia Universidad Católica del Ecuador a difundir a través de sitio web de la Biblioteca de la PUCE el referido trabajo de titulación, respetando las políticas de propiedad intelectual de Universidad.

Quito, 06 de febrero de 2026

Armando Xavier Estévez Toapanta

C.I. 1718528662

DECLARACIÓN y AUTORIZACIÓN

Yo, **Nelson Ricardo Guijarro Vallejo** con C.I. 1725279242 autor(a) del trabajo de titulación intitulado: **"SISTEMA DE GESTIÓN PARA LA ASIGNACIÓN DE AULAS EN LA DIRECCIÓN DE INFORMÁTICA DE LA PUCE"**, previa a la obtención del título de **Tecnólogo Superior en Desarrollo de Software** en la Unidad Académica de Formación Técnica y Tecnológica PUCE TEC:

1.- Declaro tener pleno conocimiento de la obligación que tiene la Pontificia Universidad Católica del Ecuador, de conformidad con el artículo 144 de la Ley Orgánica de Educación Superior, de entregar a la SENESCYT en formato digital una copia del referido trabajo de graduación para que sea integrado al Sistema Nacional de Información de la Educación Superior del Ecuador para su difusión pública respetando los derechos de autor.

2.- Autorizo a la Pontificia Universidad Católica del Ecuador a difundir a través de sitio web de la Biblioteca de la PUCE el referido trabajo de titulación, respetando las políticas de propiedad intelectual de Universidad.

Quito, 06 de febrero de 2026

Nelson Ricardo Guijarro Vallejo

C.I. 1725279242

Agradecimientos

Expresamos nuestro sincero agradecimiento a la Pontificia Universidad Católica del Ecuador que nos brindó la oportunidad de formarnos académicamente, proporcionándonos los conocimientos, recursos y valores necesarios para nuestro desarrollo profesional a lo largo de toda la carrera. La formación recibida ha sido fundamental para la realización del presente trabajo de titulación.

De manera especial, agradecemos a nuestro tutor, por su orientación constante, paciencia y valiosas observaciones, las cuales fueron determinantes para el correcto desarrollo de esta investigación. Su acompañamiento académico y disposición para guiarnos durante cada etapa del proceso contribuyeron significativamente a la culminación exitosa de este trabajo.

Asimismo, extendemos nuestro agradecimiento a todos los docentes que formaron parte de nuestra formación a lo largo de los diferentes semestres, quienes con su profesionalismo, compromiso y conocimientos impartidos aportaron de manera directa e indirecta a la construcción de las bases académicas que hoy se reflejan en este proyecto.

Agradecemos profundamente a nuestras familias por su apoyo incondicional, comprensión, paciencia y motivación constante durante todo este proceso académico. Su respaldo emocional fue fundamental para afrontar los desafíos presentados a lo largo de la carrera y nos impulsó a perseverar hasta alcanzar este objetivo.

Finalmente, agradecemos a todas las personas que, de una u otra manera, aportaron con su apoyo y colaboración para la realización del presente trabajo, haciendo posible la culminación de esta etapa importante de nuestra formación académica.

Introducción

En este proyecto se plantea el desarrollo de un sistema que automatice el proceso de asignación de aulas en la Dirección de Informática (la cual nombraremos DI a partir de este momento) de la Pontificia Universidad Católica del Ecuador (la cual nombraremos PUCE a partir de este momento). Actualmente, ese proceso se realiza de forma manual mediante el uso de herramientas como Excel, correos electrónicos y el sistema SARS, lo que representa un aumento de trabajo para los técnicos y aumenta el riesgo de errores humanos.

Asimismo, el diseño y estructuración de este sistema automatizado permitirá la asignación inmediata de reservas en función de parámetros antes establecidos como son: la disponibilidad horaria, el número de estudiantes, los recursos tecnológicos requeridos, el nombre de docente, y la modalidad del uso del aula (ocasional o semestral).

El proyecto busca generar una interfaz amigable con pasos muy fáciles de seguir para los gestores de cada facultad responsables, técnicos de la DI y administradores para gestionar diferentes permisos de cada rol, lo que facilitará el proceso de obtención de datos y su validación en tiempo real. Asimismo, el aplicativo se acoplará a la validación de las facultades autorizadas por parte de la DI que puedan acceder a los servicios de asignación de aulas, verificando los parámetros y requisitos establecidos para cada reserva.

Mediante la automatización del proceso de asignación, se reducirán considerablemente los tiempos de respuesta a las solicitudes, se minimizarán los errores derivados de la gestión manual y se proporcionará a los usuarios finales (gestores de facultad, técnicos y administradores) una experiencia más eficiente y transparente.

Antecedentes

La asignación de aulas es un proceso administrativo que permite gestionar el uso de aulas para la DI de la PUCE. De acuerdo con este concepto, el proceso de asignación involucra que los gestores de facultad realicen reservas especificando la modalidad requerida (ocasional para usos puntuales o semestral para usos continuos), y que los técnicos de la Dirección de Informática puedan gestionar las reservas (esto incluye crear, editar, cancelar y ver reservas), la fecha y hora de disponibilidad necesaria, el número de estudiantes, el docente responsable, la asignatura y los recursos tecnológicos específicos; el sistema valida automáticamente que el aula cumpla con la capacidad, disponibilidad horaria y recursos solicitados, y procede a la asignación inmediata si todos los parámetros son válidos. Estudiantes o personal administrativo para el desarrollo de clases, reuniones, talleres, exámenes u otras actividades. Este proceso es útil para optimizar el uso de aulas disponibles que existen en este espacio de trabajo que es fundamental para el seguimiento de las clases dentro del periodo académico correspondiente.

El técnico actualmente; ha realizado la asignación de aulas de forma manual, mediante el uso de Excel y correos electrónicos. Este método suele generar dificultades como demoras en la aprobación de solicitudes, especialmente cuando varias facultades o departamentos comparten espacios comunes debido a que se debe mantener constante comunicación innecesaria con recursos manuales con los gestores de cada facultad y con el personal administrativo.

Ante esta situación, surge la necesidad de desarrollar un Sistema de Gestión para la Asignación de Aulas, que permita automatizar y centralizar este proceso. Dicho sistema busca mejorar la eficiencia en la administración de las aulas, reducir los errores

por parte de gestores y técnicos y brindar una herramienta accesible y segura para todos los usuarios involucrados.

Planteamiento del problema

Actualmente en la PUCE, el proceso de asignación de aulas dentro de la DI contiene varios problemas en lo que se refiere a lo técnico y lo operativo que dificulta una gestión estructurada de las aulas. Existen procesos manuales que requieren mucho tiempo y elaboración por parte de los técnicos, elevando la carga laboral para los mismos, la probabilidad de equivocaciones humanas y el plazo de respuesta hacia los gestores y los mismos hacia los docentes.

Desde un punto de vista técnico, el sistema actual usado en la DI no muestra un seguimiento fiable de las reservas de forma clara. También carece de una representación visual amigable y actual de la disponibilidad de aulas, lo que restringe la capacidad para planificar y responder a imprevistos. El hecho de tener que realizar tantos pasos para ver una reserva, docente, fecha, materias y demás constituye un riesgo para la DI en el tema de las reservas de aulas, ya que obstaculizan el uso eficaz de los recursos y complican el monitoreo del uso de las aulas.

Doble ingreso de datos: Los encargados deben introducir la misma información en dos programas distintos (Excel por correo, Excel local para visualización, SARS), esto hace que existan redundancias y falta de eficiencia. Restricciones técnicas del sistema actual: No se dispone de seguimiento de reservas ni visualización en tiempo real de la disponibilidad de aulas.

Consecuencias para la DI: Estas limitaciones complican una gestión controlada y eficaz en el uso de las aulas.

Justificación

El Sistema de Gestión de Aulas en la DI en la PUCE aparece con la intención de permitir a los gestores de facultad y a los técnicos, realizar las asignaciones de forma sencilla, ordenada y sistemática, esto con el fin de garantizar un flujo de trabajo eficiente tanto para los docentes como para los estudiantes.

Con la creación de este proyecto la situación de asignación de aulas cambiara por completo, dando la capacidad al técnico de poder realizar otras tareas evitando hacer este proceso de forma manual, y a los gestores, garantizar de forma oportuna un espacio para el docente que ha hecho la petición.

Con este sistema el gestor de facultad podrá visualizar el estado del aula, es decir: disponible o reservada, además de poder elegir el tipo de aula que se requiere como especializada, normal o Mac, y con la modalidad de uso, elegir entre ocasional o semestral, también podrá escoger el aula de acuerdo con la cantidad de estudiantes, la materia que se va a impartir y colocar el horario que requiere.

Al tener estos datos, los técnicos podrán tomar decisiones de mejor manera en cuanto aparezca alguna petición urgente como es en los casos de exámenes, pruebas de periodo o ubicación, que son los llamados “reserva ocasional”, el sistema facilitara el estado o la situación del aula mostrando los horarios y especialmente indicando cual se puede ocupar.

Objetivos

Objetivo General:

Desarrollar un sistema de escritorio automatizado para la asignación eficiente de aulas ocasionales y semestrales a los docentes de las distintas facultades que realicen sus peticiones de reserva a sus gestores de facultad para la Dirección de Informática.

Objetivos específicos:

- Implementar sistema de gestión de asignación de aulas que permita la creación, edición, actualización, búsqueda y cancelación de reservas de acuerdo al rol.

- Implementar un módulo de validación y asignación que permita verificar de forma automática la disponibilidad de las aulas en función de parámetros como fecha, horario, capacidad y requerimientos tecnológicos. Este proceso se realizará mediante algoritmos de búsqueda y coincidencia de datos, con el fin de descartar solicitudes no válidas y garantizar un uso eficiente y adecuado de las aulas disponibles.

- Generar funcionalidades adicionales como reportes automáticos e historial de reserva, así como la visualización de disponibilidad, implementando módulos de consulta, para apoyar la toma de decisiones de los técnicos y planificar el uso de aulas.

- Diseñar control de permisos por roles para que los usuarios de acuerdo con su rol accedan a sus módulos de gestiones y no tener que realizar tantos pasos para realizar su tarea.

- Diseñar una interfaz clara y comprensible para los gestores de cada facultad, técnicos y administradores, empleando herramientas de Frontend con una interacción intuitiva y buena experiencia de usuario, para mejorar la comunicación entre las facultades y la Dirección de Informática.

Capítulo I

Levantamiento de Requisitos y Diseño del Sistema

Alcance

Para el desarrollo propuesto a continuación se definirán los módulos con respecto a sus funcionalidades.

Módulo Gestión de Roles

Gestor de facultad: Este rol podrá visualizar las aulas ya reservadas, y llenar el formulario correspondiente a la petición de docente para la asignación del aula.

Técnico: Este rol podrá gestionar reservas, que aula está disponible, asignar aulas ocasionales, cancelar si es necesario o establecer como pendiente.

Administrador: Este rol podrá controlar el sistema y sus gestiones, es decir: realizar las actividades de eliminar, agregar, editar y actualizar campos. Además de proporcionar los permisos correspondientes a los roles.

Invitado: Este rol solo tendrá acceso al sistema para visualizar las reservas. Este rol se asigna en un registro público el cual tendrá que comunicarse con el administrador para obtener el permiso y rol adecuado.

Módulo Gestión de Reserva

Para la gestión de reserva se realizará la recepción, registro, organización.

Cada rol podrá realizar reservas de diferente tipo

- Gestor: reservas semestrales y ocasionales.
- Técnico: reservas ocasionales o semestrales.
- Administrador: Todo.

Módulo Gestión de Control de acceso

Cada usuario tendrá un correo y contraseña para el inicio de sesión. En caso de no tenerlo, se registrará con rol invitado y el administrador procederá a otorgar los permisos adecuados.

Además, en el menú principal que aparece al acceder, mostrará las gestiones dependiendo el permiso de cada rol.

Funcionalidades fuera del alcance

- Alcance para facultades no aprobadas por la Dirección de Informática.
- Control de ingreso y salida automática de docentes del aula asignada.
- Gestión de docentes, facultades y materias.
- Implementación de aplicaciones externas como SARS y Gooverlan.
- Autenticación doble factor en Microsoft para el registro o creación de usuario.
- Reinicio de contraseña automático.
- Aplicativo móvil y web.

Análisis de Requisitos

Los requisitos de un sistema son descripciones de lo que el sistema debe hacer, tomando en cuenta las restricciones y opciones que requieran el sistema para funcionar y optimizar el mismo, reflejando las necesidades de los clientes. (Sommerville, 2011)

"El análisis de requisitos es un proceso de estudio y comprensión de los requisitos establecidos por las partes interesadas" (Jain, 2025)

Requisitos funcionales

El sistema cuenta con una base de datos relacional con modelado en E-R (Entidad - Relación) para el funcionamiento de todo el sistema.

Módulo de gestión de roles

El sistema contará con tres roles importantes (gestor de facultad, técnico, administrador) y un rol de invitado para solo lectura de reserva, cada uno con sus permisos otorgados por el administrador.

Módulo de gestión de reserva

El sistema contará con un formulario de reserva con opciones para la solicitud de asignación de aula.

Los datos del sistema se editarán y actualizarán si existen cambios en la reserva.

Los datos de las reservas se guardarán en una tabla dentro de la interfaz para visualizar.

Módulo Gestión de Control de acceso

El sistema contará con servicio de inicio de sesión y registro para invitados de solo lectura de reservas.

El sistema cuenta con permisos por rol para garantizar el acceso restringido.

Requisitos no funcionales

El sistema tendrá rutas protegidas de acuerdo con los permisos otorgados y su rol.

El sistema tendrá acceso a una interfaz con su respectiva opción única de acuerdo con los permisos otorgados a ese rol.

El sistema contará con asignaciones de aulas de acuerdo con cada facultad.

El sistema contará con una actualización automática de las reservas por detrás para que todos los usuarios de la aplicación puedan visualizarla.

El sistema contará con validaciones internas para cada opción en el formulario.

El sistema contará con botones adicionales como una barra de búsqueda para facilitar la navegación de cada usuario que use esta aplicación.

Las tablas de reservas mostrarán la disponibilidad de uso.

Modelado de negocio

Diagrama de flujo

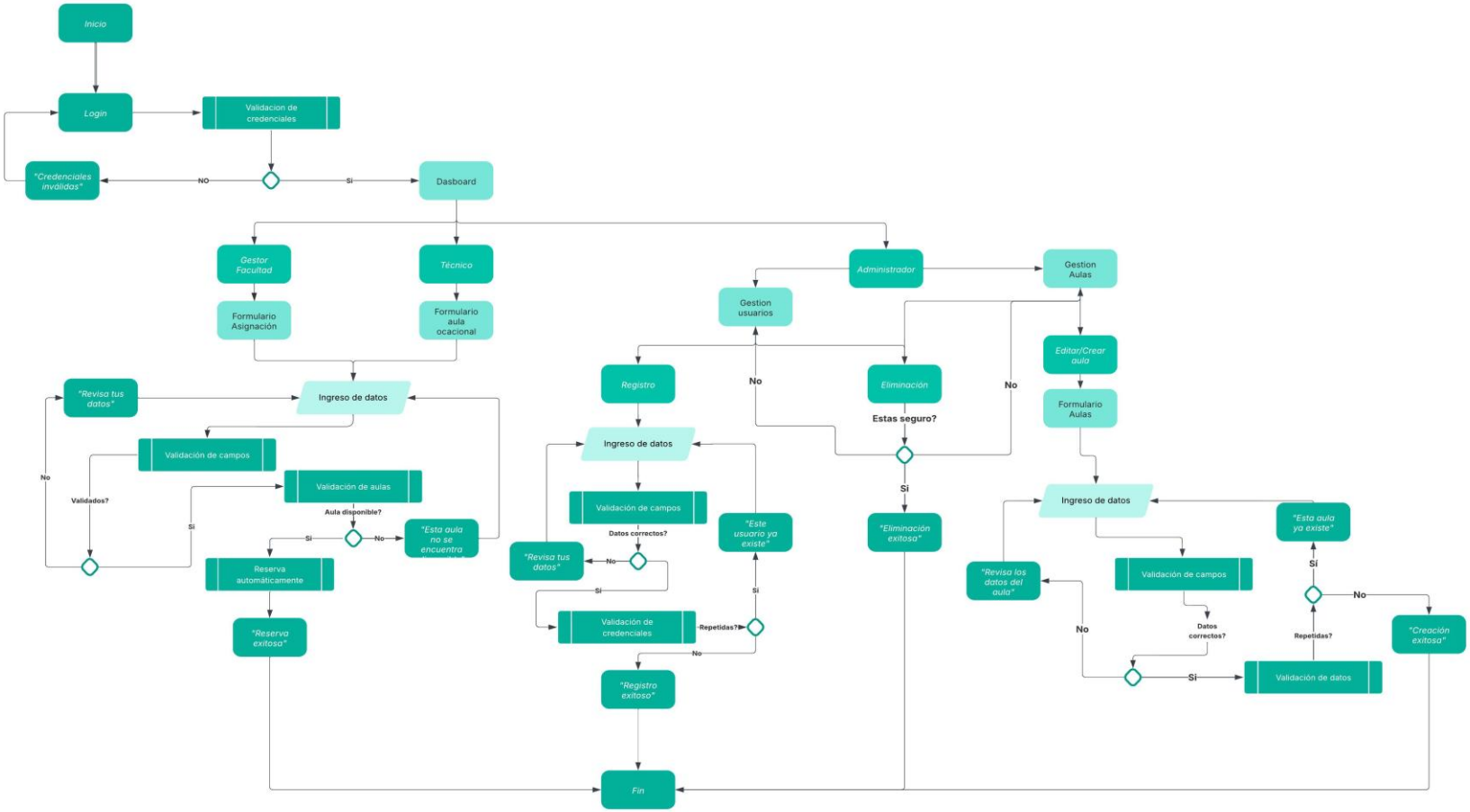
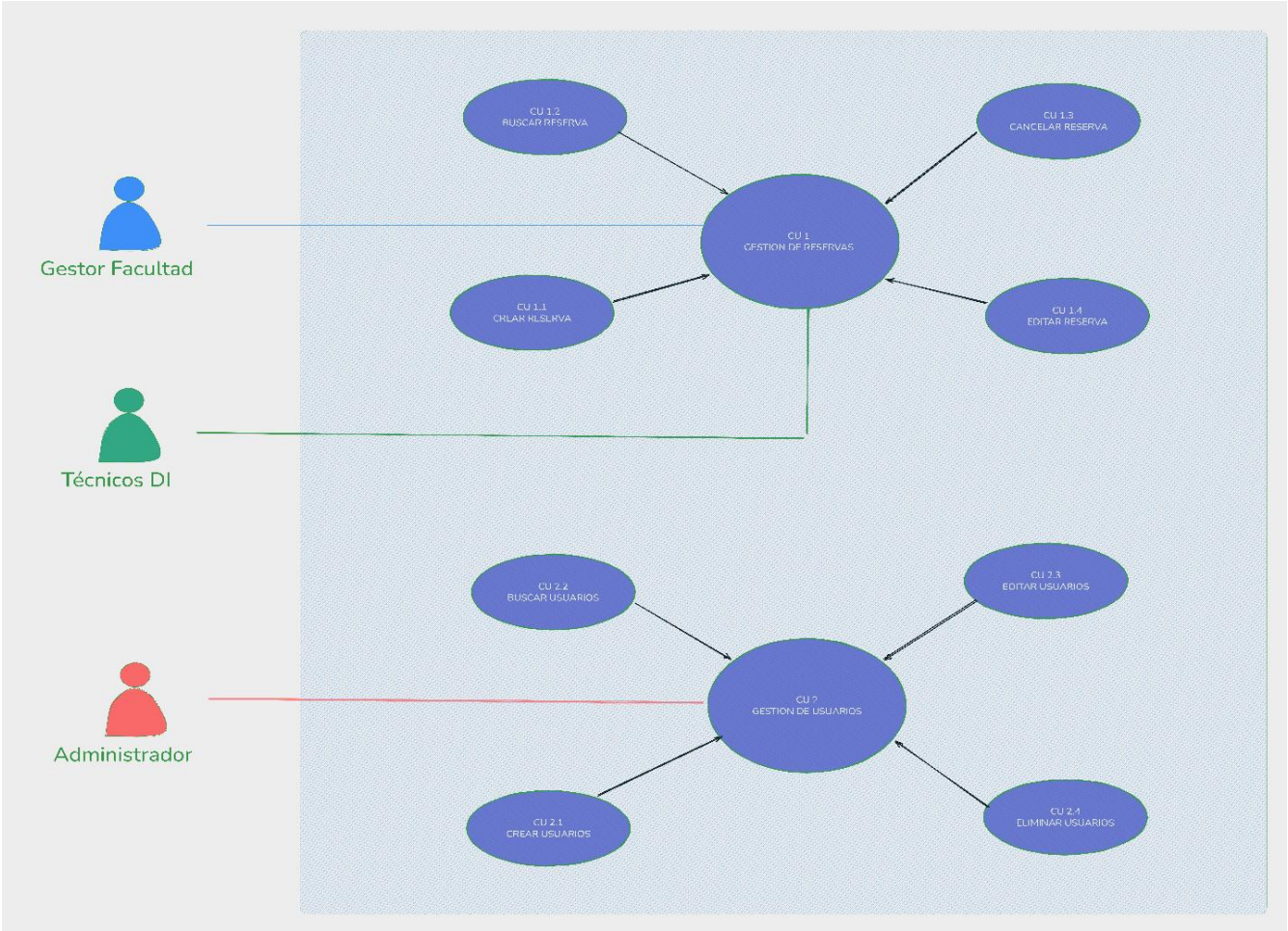


Figura 1: Diagrama de flujo Asignación de Aulas

Identificación Actores

UML caso de uso



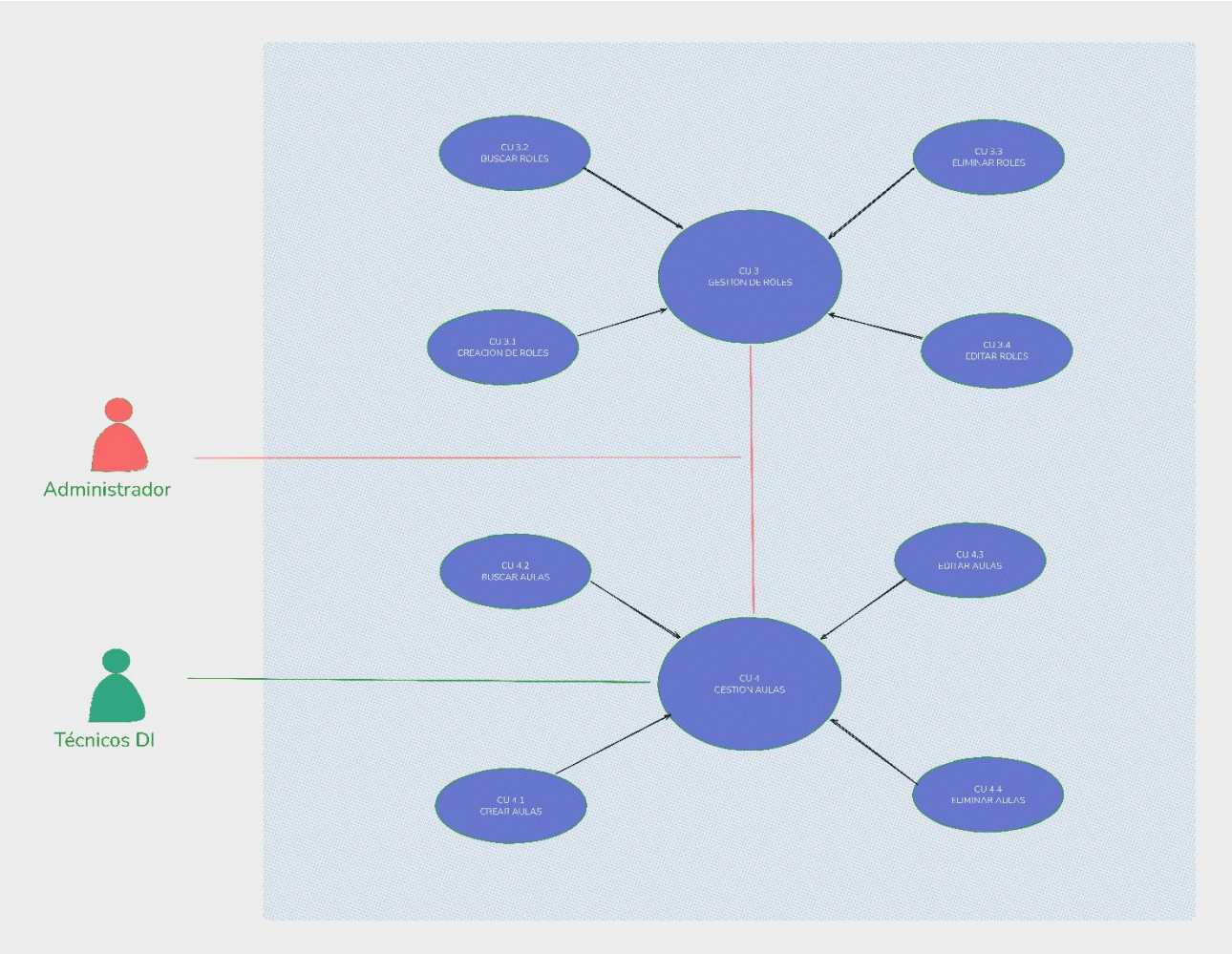


Figura 2: Diagrama de Caso de uso Sistema Asignación de Aulas

Caso de uso extendido

Nombre	CU 1.1 Crear reserva
Descripción: Permite al usuario crear una reserva de acuerdo con los parámetros establecidos en el formulario. Este caso de uso permite la creación de reservas que está directamente relacionado con el caso de uso gestión de reservas.	
Actores:	Gestor Facultad, Técnico, Administrador
Precondiciones:	Los gestores facultad, técnicos y administradores estén autenticados.
Requisitos no funcionales: - Los usuarios pueden consultar su historial de creación de reservas. - Los usuarios recibirán notificaciones de errores de coincidencia de aulas u horarios.	
Flujo de Eventos: 1. Se abre el apartado correspondiente al rol que inició sesión. 2. El usuario presiona el botón “Crear reserva” 3. El sistema muestra el formulario de crear reserva. 4. El usuario llena el formulario para la creación de la reserva. 5. El sistema valida los datos ingresados. 6. El usuario envía el formulario. 7. El sistema procede a reservar el aula con los parámetros establecidos en el formulario. 8. El sistema envía automáticamente los datos a una tabla de reservas.	
Flujo Alterno: 5.1 Si el sistema detecta algún error, muestra un mensaje de error al usuario y regresa al paso 3. 6.1 Si el sistema no realiza la petición HTTP correctamente, envía un mensaje de error y regresa al paso 3.	

Nombre	CU 1.2 Buscar reservas
Descripción: Permite al usuario buscar una reserva ya creada. Este caso de uso permite la búsqueda de reservas que está directamente relacionado con el caso de uso gestión de reservas.	
Actores:	Gestor Facultad, Técnico, Administrador.
Precondiciones:	- Los gestores facultad, técnicos y administradores estén autenticados. - Exista reservas.
Requisitos no funcionales: - La muestra de las reservas aparece de forma secuencial instantáneamente.	
Flujo de Eventos: 1. Se abre el apartado correspondiente al rol que inició sesión. 2. El usuario entra al apartado “Buscar reservas”. 3. El sistema muestra las reservas creadas por el usuario. 4. El usuario selecciona visualizar, cancelar o editar su reserva creada anteriormente.	
Flujo Alterno: 3.1 Si el sistema no detecta ninguna reserva, muestra un mensaje de aviso que no existen reservas. 4.1 Si el usuario decide visualizar su reserva, esta muestra los datos detalladamente.	

4.2 Si el usuario decide cancelar su reserva, se envía al CU1.3 Cancelar reserva. 4.3 Si el usuario decide editar su reserva, se envía al CU1.4 Editar reserva.

Nombre	CU 1.3 Cancelar reserva
Descripción: Permite al usuario cancelar su reserva creada. Este caso de uso permite la cancelación de reservas que está directamente relacionado con el caso de uso gestión de reservas.	
Actores:	Gestor Facultad, Técnico, Administrador.
Precondiciones:	- Los gestores facultad, técnicos y administradores estén autenticados. - Existe reservas
Requisitos no funcionales: - Mensajes de alerta para la confirmación de cancelación. - Manejo de errores de opción de cancelación.	
Flujo de Eventos: 1. Se abre el apartado correspondiente al rol que inició sesión. 2. El usuario entra al apartado “Buscar reservas”. 3. El sistema muestra las reservas creadas por el usuario. 4. El usuario elige una reserva y le da click al botón cancelar. 5. El sistema valida y muestra mensaje de confirmación “¿Está seguro de que desea cancelar la reserva?”. 6. El sistema muestra un cuadro para ingreso de la razón de la cancelación. 7. El usuario ingresa el detalle y acepta. 8. El sistema realiza el proceso de vaciar los datos de la reserva cancelada por el usuario. 9. Se recarga la página e información.	
Flujo Alterno: 3.1 Si el sistema no detecta ninguna reserva, muestra un mensaje de aviso que no existen reservas. 5.1 Si el sistema detecta que el usuario no tiene permisos para hacer dicha acción, envía un mensaje de alerta y envía al usuario al paso 3. 5.2 Si el usuario cancela esta acción, lo envía al paso 3.	

Nombre	CU 1.4 Editar reservas
Descripción: Permite al usuario editar su reserva creada. Este caso de uso permite la edición de reservas que está directamente relacionado con el caso de uso gestión de reservas.	
Actores:	Gestor Facultad, Técnico, Administrador.
Precondiciones:	- Los gestores facultad, técnicos y administradores estén autenticados. - Existe reservas
Requisitos no funcionales: - Mensajes de alerta para la confirmación de edición. - Manejo de errores de opción de edición.	
Flujo de Eventos: 1. Se abre el apartado correspondiente al rol que inició sesión. 2. El usuario entra al apartado “Buscar reservas”. 3. El sistema muestra las reservas creadas por el usuario.	

4. El usuario elige una reserva y le da click al botón editar.
5. El sistema muestra un formulario de edición similar al de creación
6. El usuario ingresa la información en los campos que requiere cambiar.
7. El usuario le da click al botón guardar.
8. El sistema valida y muestra mensaje de confirmación “Está seguro de que desea editar la reserva”.
9. El usuario acepta.
10. El sistema edita la reserva automáticamente.
11. Se recarga la página e información.

Flujo Alterno:
3.1 Si el sistema no detecta ninguna reserva, muestra un mensaje de aviso que no existen reservas.
8.1 Si el sistema detecta algún error dentro del formulario, envía un mensaje de alerta y regresa al paso 5.
9.1 Si el usuario cancela esta acción, lo envía al paso 3.

Nombre	CU 2.1 Crear usuario
Descripción: Permite al administrador crear un usuario. Este caso de uso permite la creación de usuarios que está directamente relacionado con el caso de uso gestión de usuarios.	
Actores:	Administrador
Precondiciones:	- El administrador esté autenticado.

Requisitos no funcionales:
- Cada usuario puede tener un alias.
- Cada usuario puede tener descripción.
- Manejo de errores de datos del usuario.

Flujo de Eventos:
1. Se abre el apartado correspondiente al rol que inició sesión.
2. El administrador presiona el botón “Crear usuario”.
3. El sistema muestra el formulario para crear el usuario.
4. El administrador llena el formulario para la creación del usuario.
5. El sistema valida los datos ingresados.
6. El administrador crea el usuario.
7. El sistema procede a crear el usuario con los parámetros establecidos en el formulario.
8. El sistema envía automáticamente los datos a una tabla de usuarios.

Flujo Alterno:
5.1 Si el sistema detecta algún error, muestra un mensaje de error al administrador y regresa al paso 3.
6.1 Si el administrador presiona en cancelar creación regresa al paso 3.
7.1 Si el sistema no realiza la petición HTTP correctamente, envía un mensaje de error y regresa al paso 3.

Nombre	CU 2.2 Buscar usuario
Descripción: Permite al administrador buscar un usuario. Este caso de uso permite la búsqueda de usuarios que está directamente relacionado con el caso de uso gestión de usuarios.	

Actores:	Administrador
Precondiciones:	- El administrador esté autenticado. - Existe usuarios.
Requisitos no funcionales: - Manejo de errores - Avisos de alerta	
Flujo de Eventos: 1. Se abre el apartado correspondiente al rol que inició sesión. 2. El administrador entra al apartado “Buscar usuario”. 3. El sistema muestra usuarios creados. 4. El administrador selecciona visualizar, cancelar o editar un usuario.	
Flujo Alterno: 3.1 Si el sistema no detecta ningún usuario, muestra un mensaje de aviso que no existen usuarios. 4.1 Si el administrador decide visualizar un usuario, esta muestra los datos detalladamente. 4.2 Si el administrador decide eliminar un usuario, se envía al CU 2.3 Editar usuario. 4.3 Si el administrador decide editar un usuario, se envía al CU 2.4 Eliminar usuario.	

Nombre	CU 2.3 Editar usuario
Descripción: Permite al administrador Editar un usuario. Este caso de uso permite la edición de usuarios que está directamente relacionado con el caso de uso gestión de usuarios.	
Actores:	Administrador
Precondiciones:	- El administrador esté autenticado.
Requisitos no funcionales: - Manejo de errores - Avisos de alerta	
Flujo de Eventos: 1. Se abre el apartado correspondiente al rol que inició sesión. 2. El administrador entra al apartado “Buscar usuarios”. 3. El sistema muestra los usuarios creados. 4. El administrador elige un usuario y le da click al botón editar. 5. El sistema muestra un formulario de edición similar al de creación 6. El administrador ingresa la información en los campos que requiere cambiar. 7. El administrador le da click al botón guardar. 8. El sistema valida y muestra mensaje de confirmación “¿Está seguro de que desea actualizar los datos?”. 9. El administrador acepta. 10. El sistema actualiza los datos del usuario automáticamente. 11. Se recarga la página e información.	
Flujo Alterno: 3.1 Si el sistema no detecta ningún usuario, muestra un mensaje de aviso que no existen usuarios. 8.1 Si el sistema detecta algún error dentro del formulario, envía un mensaje de alerta y regresa al paso 5. 9.1 Si el usuario cancela esta acción, lo envía al paso 3.	

Nombre	CU 2.4 Eliminar usuario
Descripción: Permite al administrador eliminar un usuario. Este caso de uso permite la eliminación de usuarios que está directamente relacionado con el caso de uso gestión de usuarios.	
Actores:	Administrador
Precondiciones:	- El administrador esté autenticado. - Exista un usuario
Requisitos no funcionales: - Manejo de errores - Avisos de alerta	
Flujo de Eventos: 1. Se abre el apartado correspondiente al rol que inició sesión. 2. El usuario entra al apartado "Buscar usuario". 3. El sistema muestra los usuarios creados por el administrador. 4. El administrador elige un usuario y le da click al botón eliminar. 5. El sistema valida y muestra mensaje de confirmación "¿Está seguro de que desea eliminar el usuario?". 6. El administrador acepta. 7. El sistema realiza el proceso de eliminar los datos del usuario cancelada por el administrador. 8. Se recarga la página e información.	
Flujo Alterno: 3.1 Si el sistema no detecta ningún usuario, muestra un mensaje de aviso que no existen usuarios. 4.1 Si el sistema detecta que el usuario no tiene permisos para hacer dicha acción, envía un mensaje de alerta y envía al usuario al paso 3. 7.2 Si el administrador cancela esta acción, lo envía al paso 3.	

Nombre	CU 3.1 Crear Roles
Descripción: Permite al administrador crear un rol. Este caso de uso permite la creación de un rol con sus respectivos permisos.	
Actores:	Administrador
Precondiciones:	- El administrador este autenticado.
Requisitos no funcionales: - Las configuraciones están en un apartado lateral superpuesta en la página. - Mantiene muestra de permisos en orden de CRUD. - Manejo de errores	
Flujo de Eventos: 1. Se abre el apartado correspondiente al rol que inició sesión. 2. El administrador ingresa los permisos de ese rol y los campos a llenar. 2.1 El sistema valida si los campos son válidos y estén llenados. 3. El administrador da clic en crear rol. 4. El sistema muestra en pantalla la creación del rol. 5. Se recarga la página e información.	
Flujo Alterno: 2.1 Si el administrador presiona en cerrar el apartado de configuraciones regresa al paso 1.	

Nombre	CU 3.2 Buscar Roles
Descripción: Permite al administrador visualizar los roles creados. Este caso de uso permite la visualización de roles.	
Actores:	Administrador
Precondiciones:	- El administrador esté autenticado.
Requisitos no funcionales: - Se muestra en detalle la información del rol. - Muestra de roles de forma instantánea. - Manejo de errores.	
Flujo de Eventos: 1. Se abre el apartado correspondiente al rol que inició sesión. 2. El sistema muestra los detalles de los roles. 3. El usuario puede visualizar toda la información del rol.	
Flujo Alterno: 3.1 Si el administrador presiona en cerrar el apartado de visualización se regresa al paso 1.	

Nombre	CU 3.3 Eliminar roles
Descripción: Permite al administrador eliminar un rol. Este caso de uso permite la eliminación de roles que está directamente relacionado con el caso de uso gestión de roles.	
Actores:	Administrador
Precondiciones:	- El administrador esté autenticado. - Exista un rol
Requisitos no funcionales: - Manejo de errores. - Avisos de alerta.	
Flujo de Eventos: 1. Se abre el apartado correspondiente al rol que inició sesión. 2. El usuario entra al apartado “Buscar rol”. 3. El sistema muestra los roles creados por el administrador. 4. El administrador elige un rol y le da click al botón eliminar. 5. El sistema valida y muestra mensaje de confirmación “¿Está seguro de que desea eliminar el rol?”. 6. El administrador acepta. 7. El sistema realiza el proceso de eliminar los datos del rol cancelada por el administrador. 8. Se recarga la página e información.	
Flujo Alterno: 3.1 Si el sistema no detecta ningún rol, muestra un mensaje de aviso que no existen roles. 4.1 Si el sistema detecta que el usuario no tiene permisos para hacer dicha acción, envía un mensaje de alerta y envía al usuario al paso 3. 7.2 Si el administrador cancela esta acción, lo envía al paso 3.	

Nombre	CU 3.4 Editar roles
Descripción: Permite al administrador editar un rol. Este caso de uso permite la edición de roles que está directamente relacionado con el caso de uso gestión de roles.	
Actores:	Administrador
Precondiciones:	- El administrador esté autenticado.
Requisitos no funcionales: - Manejo de errores - Avisos de alerta	
Flujo de Eventos: 12. Se abre el apartado correspondiente al rol que inició sesión. 13. El administrador entra al apartado "Buscar roles". 14. El sistema muestra los roles creados. 15. El administrador elige un rol y le da clic al botón editar. 16. El sistema muestra un formulario de edición similar al de creación 17. El administrador ingresa la información en los campos que requiere cambiar. 18. El administrador le da click al botón guardar. 19. El sistema valida y muestra mensaje de confirmación "¿Está seguro de que desea actualizar los datos?". 20. El administrador acepta. 21. El sistema actualiza los datos del usuario automáticamente. 22. Se recarga la página e información.	
Flujo Alterno: 3.1 Si el sistema no detecta ningún rol, muestra un mensaje de aviso que no existen roles. 8.1 Si el sistema detecta algún error dentro del formulario, envía un mensaje de alerta y regresa al paso 5. 9.1 Si el usuario cancela esta acción, lo envía al paso 3.	

Nombre	CU 4.1 Crear aula
Descripción: Permite al usuario crear un aula. Este caso de uso permite la creación de aulas que está directamente relacionado con el caso de uso gestión de aulas.	
Actores:	Técnico, administrador
Precondiciones:	- Los usuarios estén autenticados.
Requisitos no funcionales: - Manejo de errores	
Flujo de Eventos: 1. Se abre el apartado correspondiente al rol que inició sesión. 2. El usuario presiona el botón "Crear aula" 3. El sistema muestra el formulario de crear aula. 4. El usuario llena el formulario para la creación de aula. 5. El sistema valida los datos ingresados. 6. El usuario envía el formulario. 7. El sistema envía automáticamente los datos a una tabla de aulas existentes.	
Flujo Alterno: 5.1 Si el sistema detecta algún error, muestra un mensaje de error al usuario y regresa al paso 4. 7.1 Si el sistema no realiza la petición HTTP correctamente, envía un mensaje de error y regresa al paso 4.	

Nombre	CU 4.2 Buscar aula
Descripción: Permite al usuario buscar un aula. Este caso de uso permite la búsqueda de aulas que está directamente relacionado con el caso de uso gestión de aulas.	
Actores:	Gestor facultad, Técnico, Administrador
Precondiciones:	- Los usuarios estén autenticados. - Existe aulas.
Requisitos no funcionales: - Utiliza filtro de búsqueda para mantener facilidad de búsqueda de aulas. - Manejo de errores	
Flujo de Eventos: 1. Se abre el apartado correspondiente al rol que inició sesión. 2. El usuario entra al apartado "Buscar aula". 3. El sistema muestra todas las aulas creadas. 4. El usuario selecciona visualizar, cancelar o editar el aula a elegir.	
Flujo Alterno: 3.1 Si el sistema no detecta ninguna aula, muestra un mensaje de aviso que no existen aulas. 4.1 Si el usuario decide visualizar su aula, esta muestra los datos detalladamente. 4.2 Si el usuario decide cancelar su aula, se envía al CU4.3 Eliminar aula. 4.3 Si el usuario decide editar su aula, se envía al CU4.4 Editar aula.	

Nombre	CU 4.3 Eliminar aula
Descripción: Permite al usuario eliminar un aula. Este caso de uso permite la eliminación de aulas que está directamente relacionado con el caso de uso gestión de aulas.	
Actores:	Técnico, Administrador
Precondiciones:	- Los usuarios estén autenticados. - Existe aulas.
Requisitos no funcionales: - Mensajes de alerta para la confirmación de cancelación. - Manejo de errores.	
Flujo de Eventos: 1. Se abre el apartado correspondiente al rol que inició sesión. 2. El usuario entra al apartado "Buscar aulas". 3. El sistema muestra todas las aulas creadas. 4. El usuario elige un aula y le da click al botón eliminar. 5. El sistema valida y muestra mensaje de confirmación "¿Está seguro de que desea eliminar el aula?". 6. El usuario acepta. 7. El sistema realiza el proceso requerido. 8. Se recarga la página e información.	
Flujo Alterno: 3.1 Si el sistema no detecta ninguna aula, muestra un mensaje de aviso que no existen aulas. 5.1 Si el sistema detecta que el usuario no tiene permisos para hacer dicha acción, envía un mensaje de alerta y envía al usuario al paso 3. 6.2 Si el usuario cancela esta acción, lo envía al paso 3.	

Nombre	CU 4.4 Editar aula
Descripción: Permite al usuario editar un aula. Este caso de uso permite la edición de aulas que está directamente relacionado con el caso de uso gestión de aulas.	
Actores:	Técnico, Administrador
Precondiciones:	- Los usuarios estén autenticados. - Existe aulas.
Requisitos no funcionales: - Mensajes de alerta para la confirmación de edición.	
Flujo de Eventos: 1. Se abre el apartado correspondiente al rol que inició sesión. 2. El usuario entra al apartado “Buscar aulas”. 3. El sistema muestra todas las aulas. 4. El usuario elige un aula y le da click al botón editar. 5. El sistema muestra un formulario de edición similar al de creación. 6. El usuario ingresa la información en los campos que requiere cambiar. 7. El usuario le da click al botón guardar. 8. El sistema valida y muestra mensaje de confirmación “Está seguro de que desea editar el aula”. 9. El usuario acepta. 10. El sistema edita el aula automáticamente. 11. Se recarga la página e información.	
Flujo Alterno: 3.1 Si el sistema no detecta ningún aula, muestra un mensaje de aviso que no existen aulas. 8.1 Si el sistema detecta algún error dentro del formulario, envía un mensaje de alerta y regresa al paso 5. 9.1 Si el usuario cancela esta acción, lo envía al paso 3.	

Nombre	CU Adicional 2 Iniciar sesión
Descripción: Permite al usuario iniciar sesión. Este caso de uso permite el inicio de sesión de usuarios que está directamente relacionado con el caso de uso gestión de control de acceso.	
Actores:	Gestor facultad, Técnico, Administrador
Precondiciones:	- Tener un usuario - Tener contraseña
Requisitos no funcionales: - Puede ingresar usando su correo institucional. - Puede ingresar usando su nombre.	
Flujo de Eventos: 1. El usuario abre el aplicativo de escritorio 2. El sistema muestra un formulario de inicio de sesión 3. El usuario llena los datos requeridos 4. El sistema valida 5. El usuario presiona en iniciar sesión. 6. El sistema abre el apartado correspondiente al rol que inició sesión.	
Flujo Alterno: 4.1 Si el sistema detecta que los datos ingresados no son correctos, muestra un mensaje de error y envía al paso 2	

Herramientas Tecnológicas

Las herramientas tecnológicas mencionadas se eligieron de una lista provista por la propia DI.

Herramientas Backend:

Base de datos: Se eligió PostgreSQL versión 16.3 como base de datos principal del servidor porque tiene buen soporte para consultas complejas y manejo de concurrencia. Almacena toda la información del sistema: aulas, reservas, usuarios, roles, facultades y reportes. Para la administración de la base se usó PGAdmin 4 versión 8.12, que facilitó la creación del modelo entidad-relación y las consultas de prueba durante el desarrollo.

Lenguaje: PHP versión 8.3, se escogió este lenguaje porque Laravel lo requiere.

Framework de PHP: Laravel versión 11, se eligió porque tiene una comunidad grande, hay bastante documentación y ejemplos, y viene con funcionalidades que se necesitaban para el proyecto como autenticación, manejo de rutas y un ORM para la base de datos. (LaRavel, n.d.)

Herramientas Frontend:

Framework de JS: Vue.js versión 3, se usó porque se integra de forma directa con Laravel a través de Inertia.js, lo que evitó tener que crear una API REST separada para el frontend. Además, su sistema de componentes reactivos permitió manejar las interfaces de manera más ordenada. (Vue, n.d.)

Lenguaje: JavaScript para la lógica de interacción y eventos del lado del cliente, y HTML para definir la estructura de las vistas. Ambos son estándar y no requerían herramientas adicionales.

Editor de código fuente: Visual Studio Code versión 1.94+. Se utilizó durante todo el desarrollo por su compatibilidad con PHP, JavaScript y Vue, y por las extensiones que facilitan el trabajo con Laravel como Laravel Blade Snippets y PHP Intelephense.

Arquitectura General del sistema

En esta aplicación de escritorio se utilizará parte de la arquitectura Modelo-Vista-Controlador (MVC) que proporciona Laravel por defecto unido con la Arquitectura en capas (Layered Architecture).

El modelo MVC nos ayuda a separar la lógica de negocio de la lógica de eventos y comunicaciones. En esta arquitectura está presente tres componentes, los cuales se encargarán de realizar diferentes aspectos en la aplicación y son: Modelo-Vista-Controlador. (contributors, s.f.)

La Arquitectura en capas se centra en la organización de capas por niveles del proyecto, dando la responsabilidad única a cada capa manteniendo la comunicación de forma clara con sus capas contiguas. (LatamTech, n.d.)

A continuación, se explica de forma general el proceso de esta arquitectura en la aplicación de escritorio.

Está el Modelo, este se encargará de comunicarse con la base de datos que en este caso es SQLite que tiene por defecto NativePHP y se sincroniza con PostgreSQL usando el ORM que proporciona Larave que es Eloquent Models, siendo un puente de acceso a la información, mediante la API Resouce para transformar los datos a JSON y los Traits para evitar la duplicación de código y la reutilización de código en varias clases, en este caso es la comunicación correcta a SQLite sincronizado con PostgreSQL, así este modelo se encargará de asegurar la entrega de información de las consultas

sobre aulas, reservas hechas, códigos de aula, cantidad alumnos, horarios, herramientas a utilizar, roles, usuarios y reportaría, también del acceso al aplicativo especificadas en los requerimientos.

Está la capa de negocio, aquí entramos a la arquitectura por capas, en donde se usa la capa Action: que en este caso servirá para la creación de usuario y actualizar el usuario, esto se comunica con el kit de herramienta usado para la "sesión del usuario" que son Fortify y Jetstream que proporciona Laravel.

Está la capa de Servicio: en esta capa esta toda la lógica de negocio del sistema.

No solo contiene las operaciones CRUD básicas (crear, leer, actualizar, eliminar) para cada entidad, sino que también maneja la lógica más compleja. Por ejemplo, el servicio de reservas (ReservationService) se encarga de validar conflictos de horario, verificar que un docente no esté asignado en dos aulas al mismo tiempo y comprobar la capacidad del aula antes de confirmar una reserva. El servicio de reservas semestrales (SemestralReservationService) gestiona la creación de reservas recurrentes, es decir, cuando un docente necesita un aula todos los lunes del semestre, por ejemplo, este servicio genera automáticamente todas las reservas individuales verificando que no haya conflictos en ninguna de las fechas. Otros servicios como ReportService generar los datos para los reportes en PDF. Es decir, la capa de servicios está compuesta por varias clases, cada una con una responsabilidad definida, lo que facilita el mantenimiento y las pruebas del código.

Está la capa de Sync: esta capa ayuda a resolver un problema que se presento en el proeycto. Como la aplicación de escritorio usa NativePHP con SQLite como base de datos local, se necesita que los datos estén siempre actualizados tanto en la máquina del usuario como en el servidor PostgreSQL. Para esto se implementó el patrón local-first,

que funciona de la siguiente manera: cuando el usuario realiza una acción (por ejemplo, crear una reserva), el cambio se guarda primero en SQLite para que la aplicación responda sin depender de la conexión a internet. Luego, en segundo plano, el sistema intenta sincronizar ese cambio con PostgreSQL. Si la sincronización falla por un problema de red, el cambio se encola en una tabla llamada SyncQueue. Esta cola tiene un mecanismo de reintentos con espera progresiva, es decir que el primer reintento se hace a los 5 minutos, el segundo a los 15, el tercero a los 45 y así sucesivamente para no llenar el servidor. Si después de varios intentos sigue fallando, el registro se marca como fallido para que se revise de forma manual. La sincronización se organiza por entidad, cada tabla del sistema (usuarios, aulas, reservas, roles, facultades, docentes, materias, reportes) tiene su propio servicio de sincronización que hereda de una clase base abstracta, lo que permitió reutilizar la lógica común y solo definir las particularidades de cada entidad.

Está el Controlador, este se encargará de responder a los eventos o acciones que requiera el usuario a través las peticiones HTTP, se comunicará con la capa de Service y usará la capa Action, para retornar los JSON. En esta capa del controlador, se encontrará el enrutamiento y los middlewares lo cuales mapearan las rutas disponibles y filtraran las autenticaciones, respectivamente.

Está la Vista, ya que se usará Vue 3 conjunto con Inertia.js, se requiere usar la arquitectura SPA (Single-Page-Application) esto ayuda a que se comunique el backend con el frontend de forma no convencional es decir sin usar una API Rest directa.

Diagrama de arquitectura

Arquitectura de tecnología

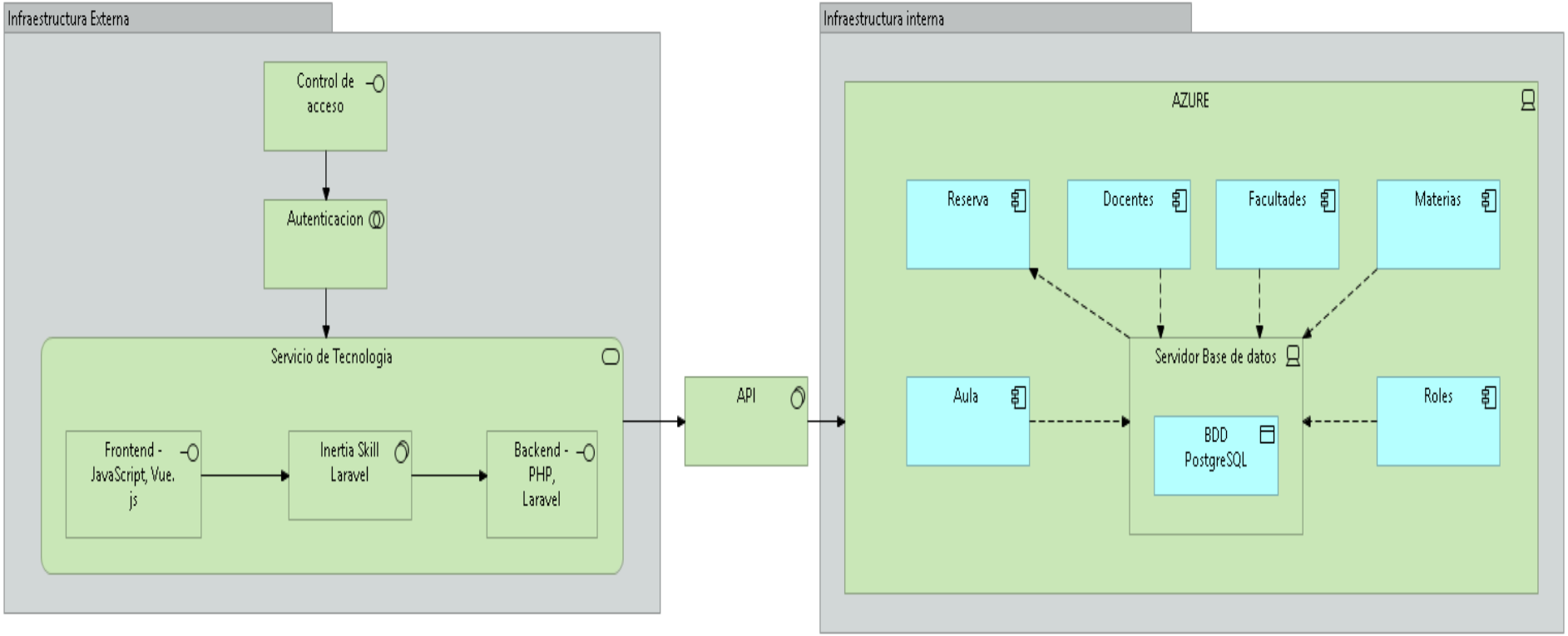


Figura 3: Diagrama de Arquitectura de tecnología del Sistema Asignación de Aulas

Arquitectura de aplicación

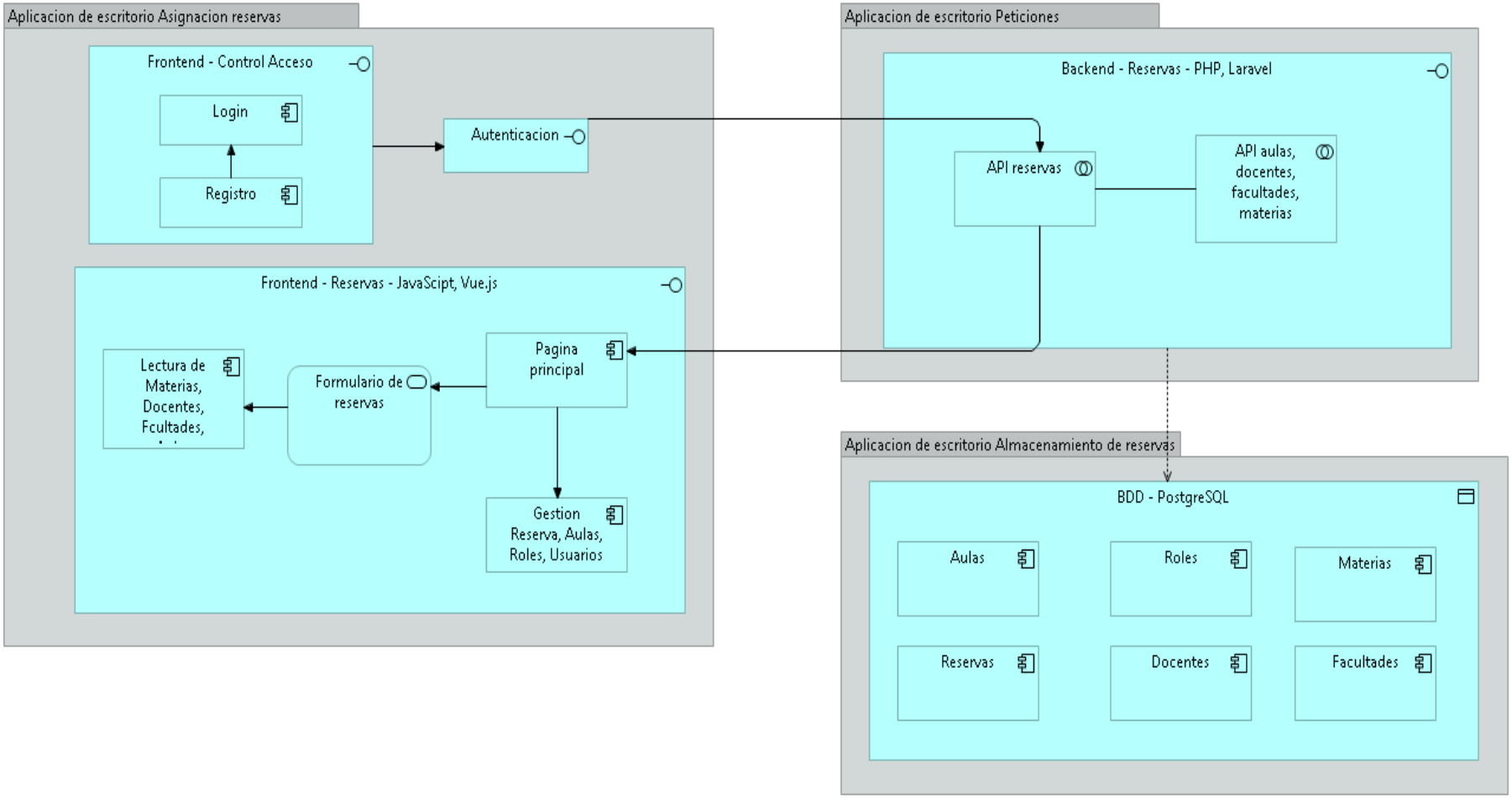


Figura 4: Diagrama de Arquitectura de aplicación del Sistema Asignación de Aulas

Diseño modular

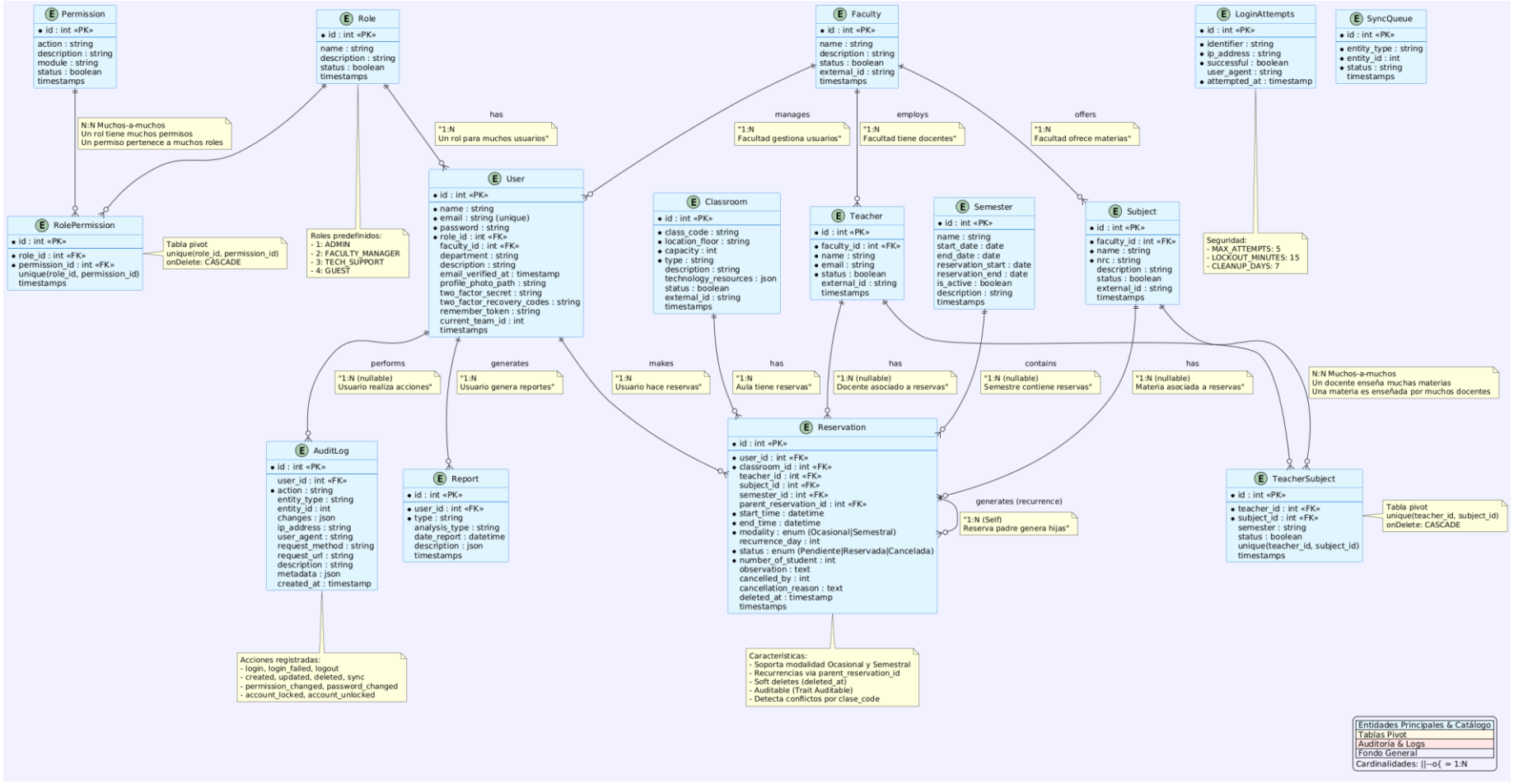


Figura 5 y 6: Diagrama del diseño modular y de clases del Sistema Asignación de

Diseño de interfaces de usuario (Mockup)

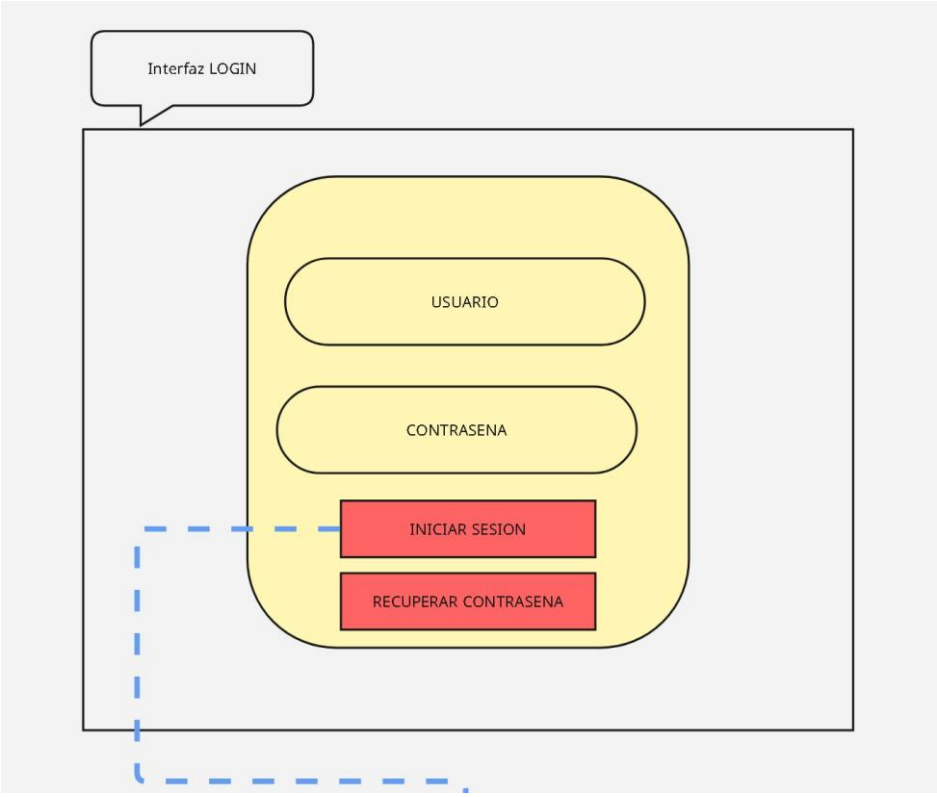


Figura 7: Mockup de LOGIN.

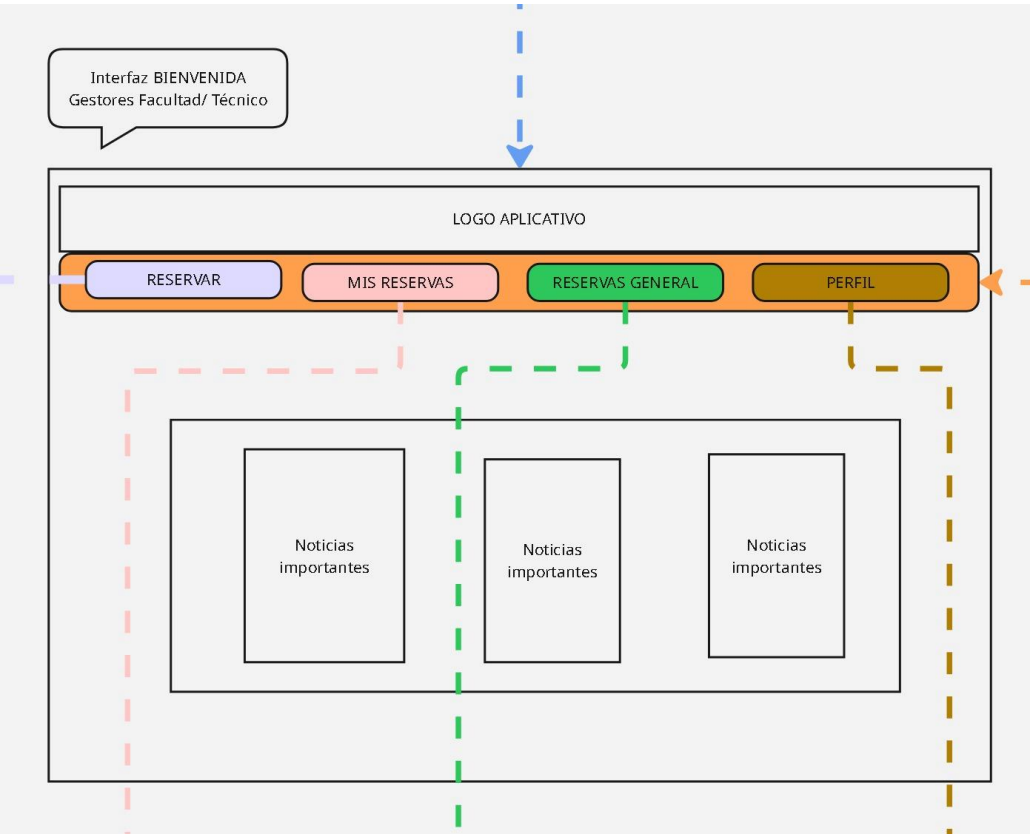


Figura 7: Mockup de Bienvenida.

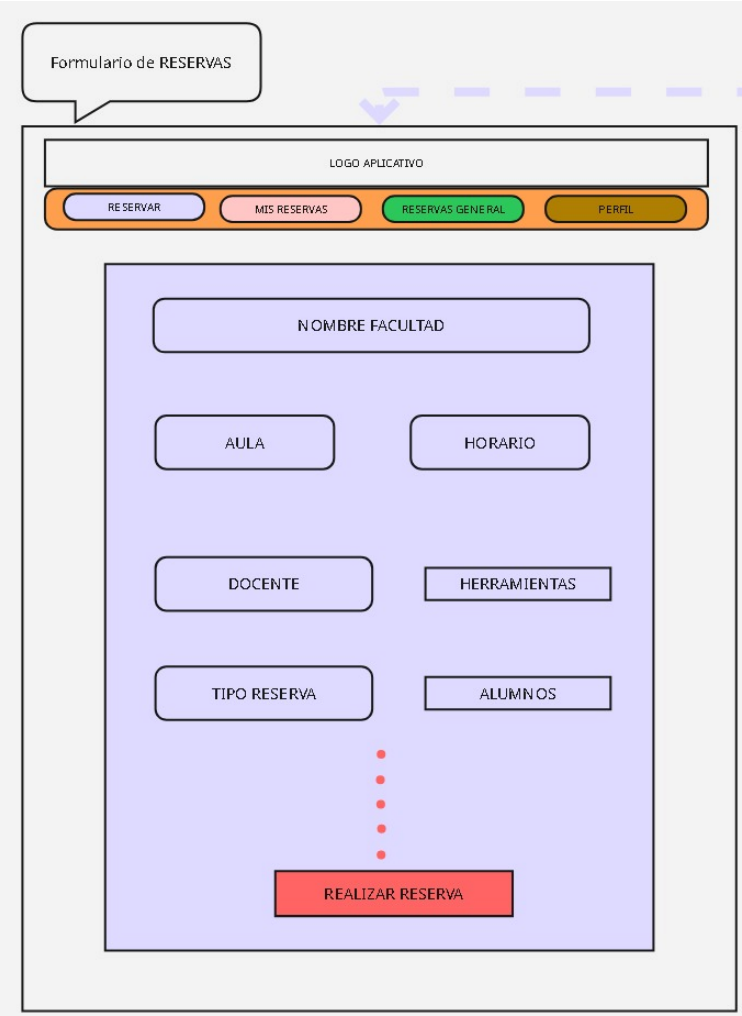


Figura 8: Formulario de RESERVAS.

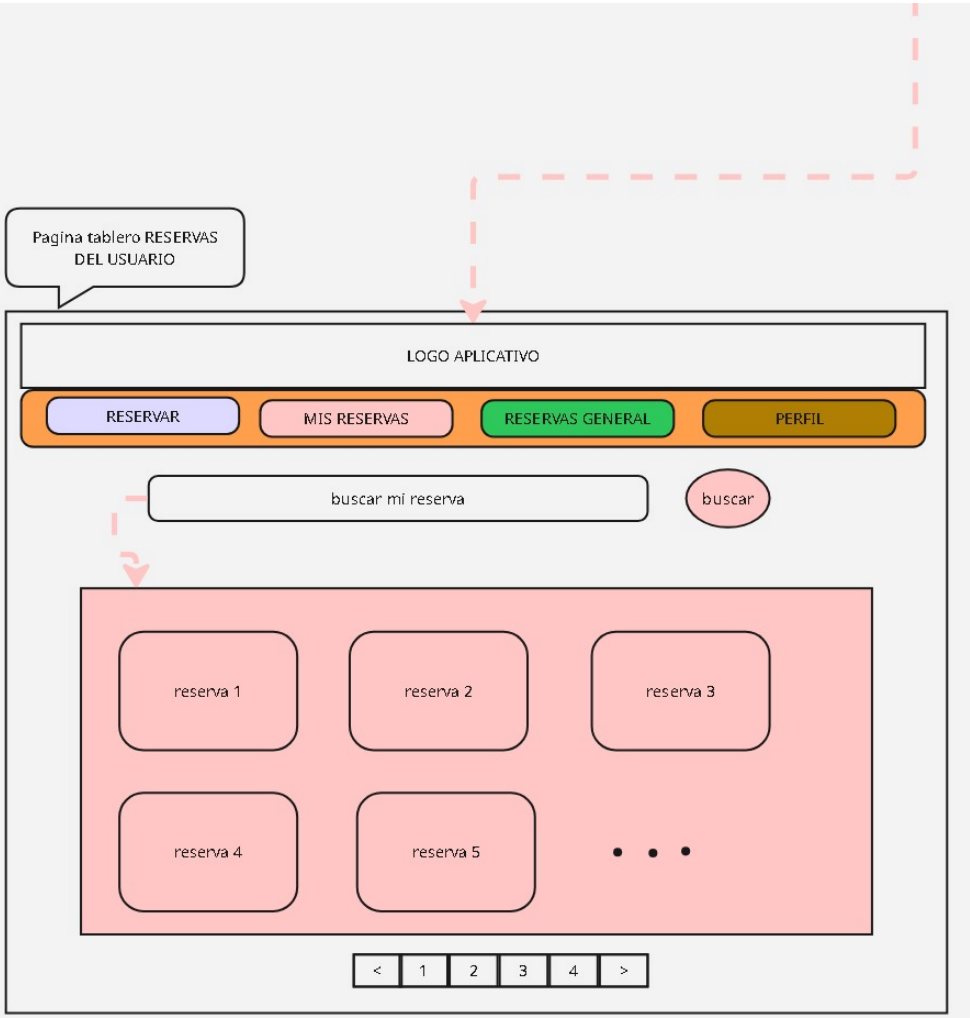


Figura 9: Tablero reservas de USUARIO.

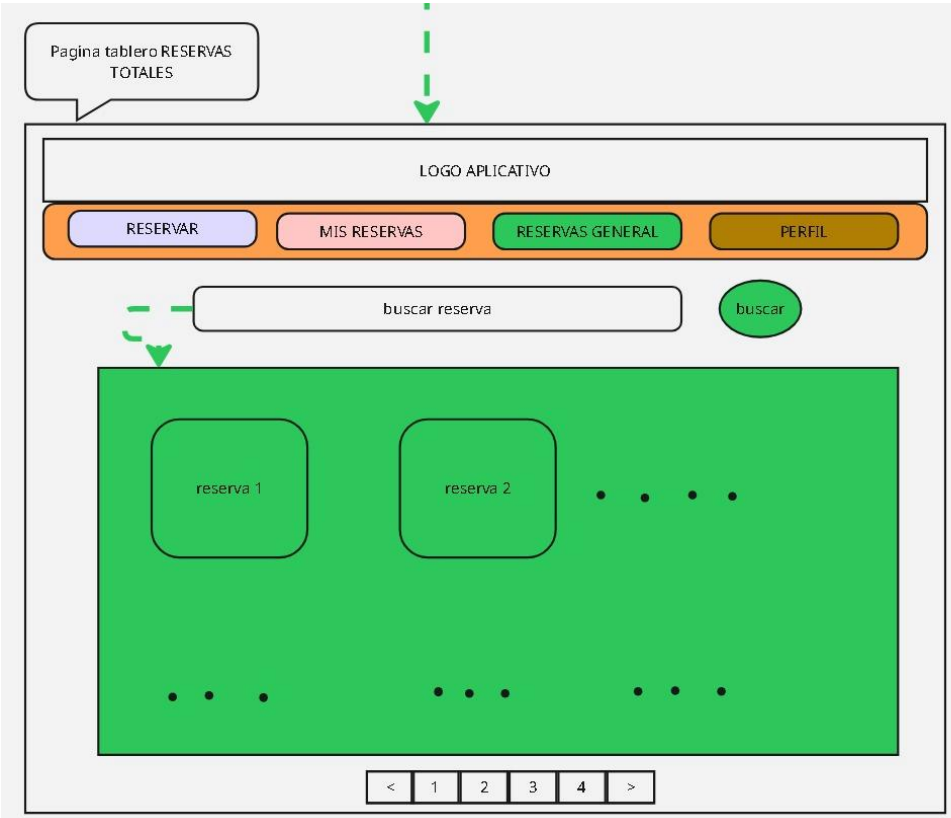


Figura 10: Tablero reservas generales.

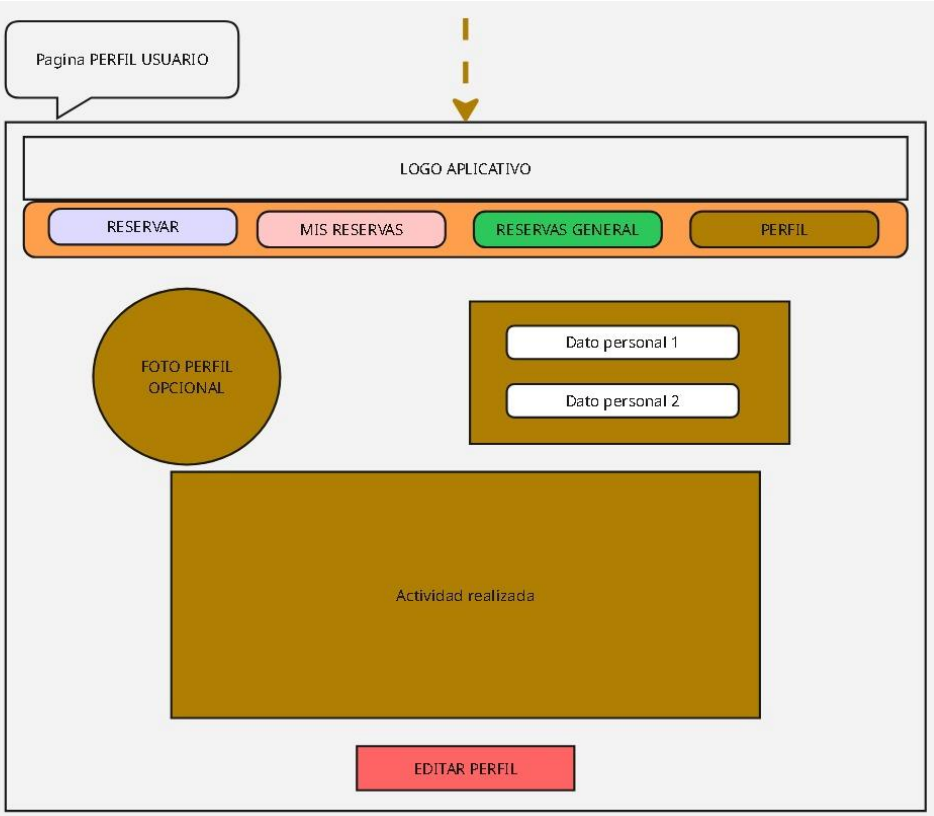


Figura 11: Perfil de USUARIO.

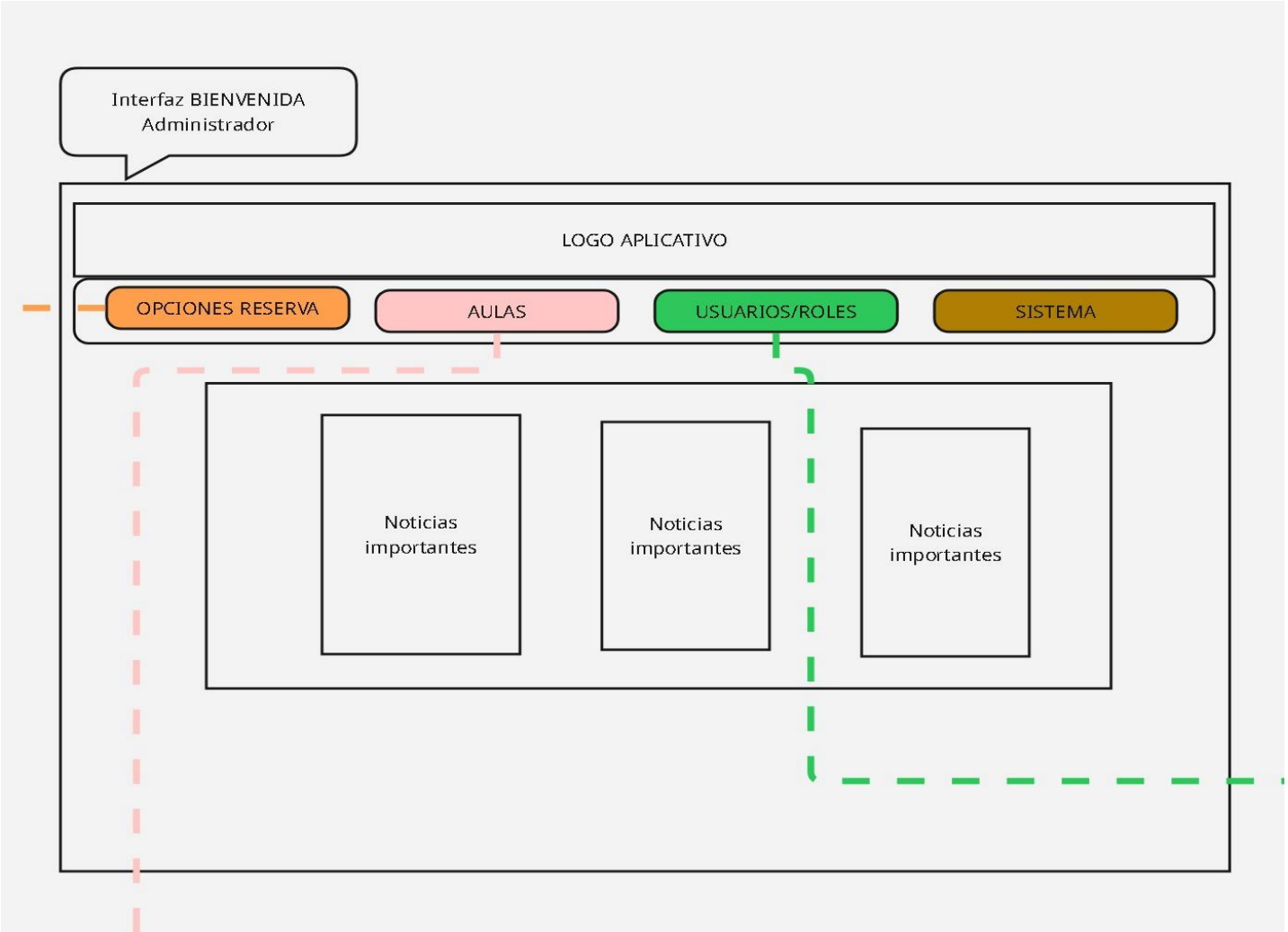


Figura 12: Mockup de Bienvenida Administrador.

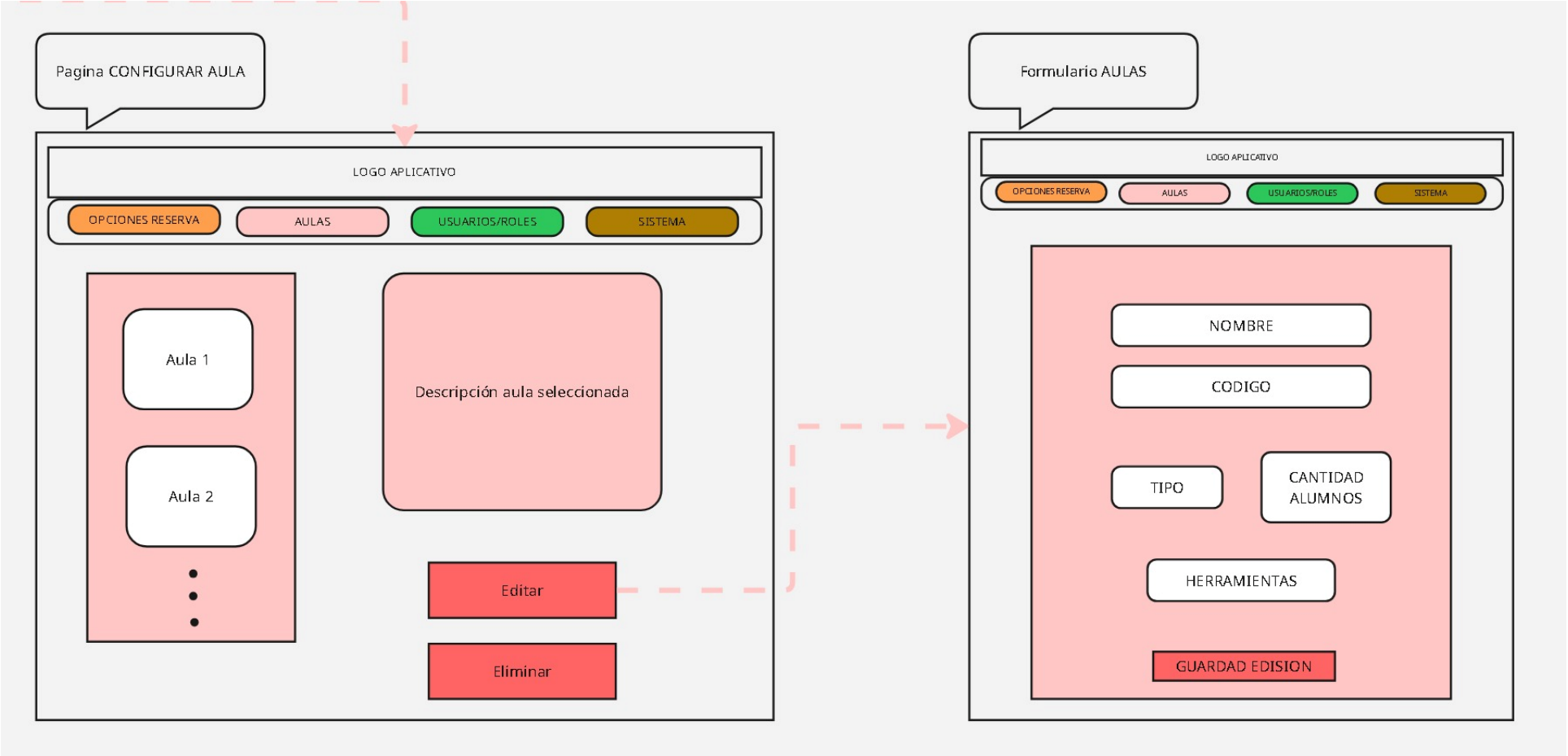


Figura 12: Tablero administrador configuración AULAS.

Figura 13: Tablero edición de AULAS.

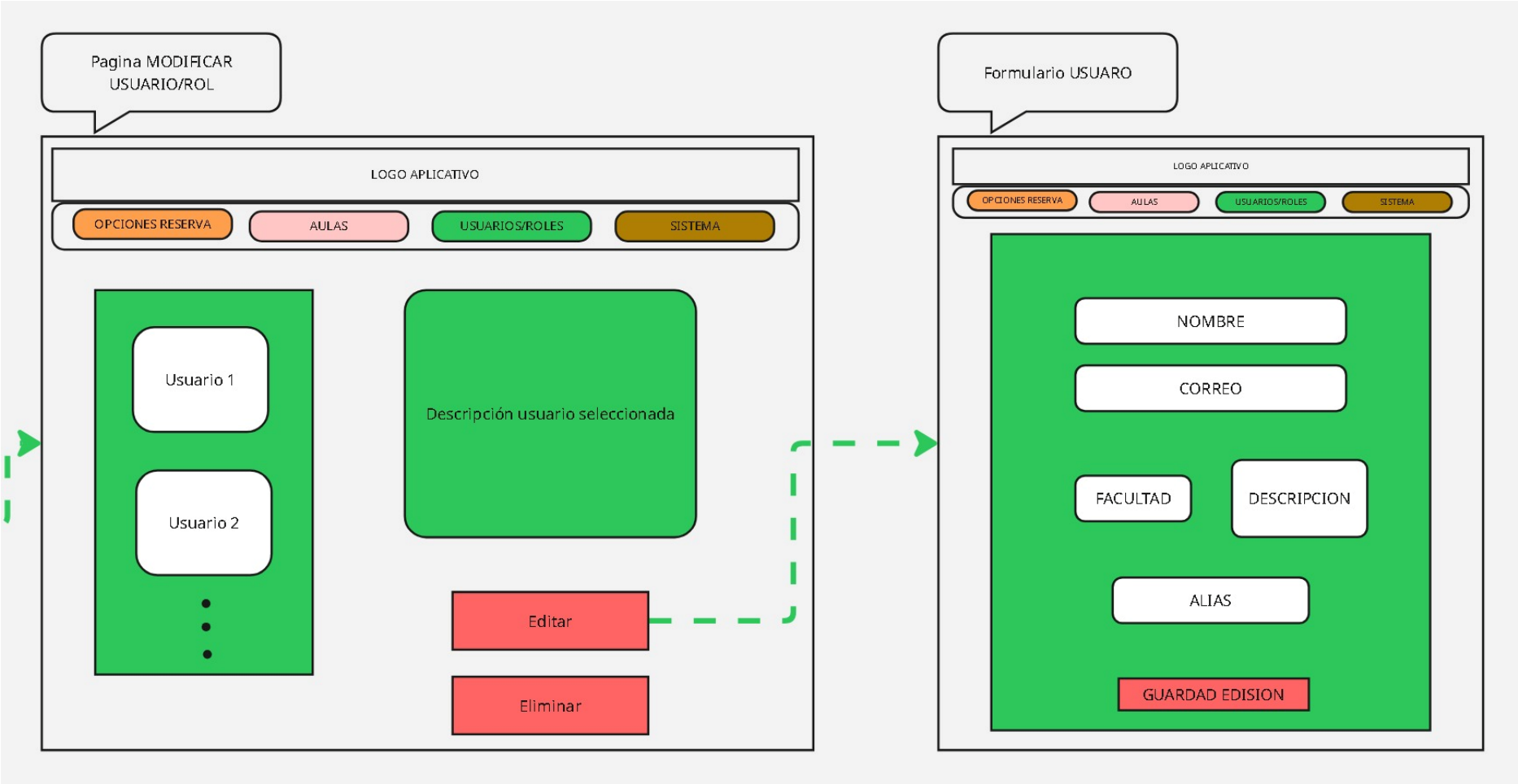


Figura 14: Tablero administrador configuración USUARIO.

Figura 15: Tablero edición USUARIO.

Requisitos mínimos de Hardware

A continuación, se especifica los requerimientos mínimos del hardware en los equipos donde se iniciará el aplicativo:

- Procesador: Core i5 o superior.
- Memoria RAM: al menos de 8GB
- Disco duro: disponible 260GB o mas

La aplicación de escritorio:

- Deberá de ser capaz de funcionar en un equipo servidor dedicado de la PUCE dentro de una intranet, o en la nube a futuro por el internet en caso de ser requerido.
- Deberá tener una interfaz amigable y fácil de usar.
- Funcionará con un ejecutable para escritorio en Windows.
- Realizará tareas específicas para la asignación de aulas y solo las consideradas dentro de los requerimientos y alcance de este documento, y cualquier otro requerimiento adicional se considerará fuera de alcance de las especificaciones
- Será desarrollado con las herramientas provistas por la Dirección de Informática de la PUCE.
- Deberá ser seguro para acceso a la información usando claves y usuarios.

Consideraciones Éticas y de Privacidad

Durante el desarrollo de este proyecto se tomó en cuenta que el sistema maneja datos propios de la universidad: nombres de docentes, horarios, facultades, materias y solicitudes internas. Esta situación obligó a definir desde el inicio cómo proteger esa información y cómo darle un uso responsable.

En conversaciones con los técnicos de la DI lo primero que se estableció fue la restricción de acceso. No cualquier persona puede entrar al sistema ni ver toda la información. Los permisos están separados por roles y cada usuario solo puede acceder a lo que le corresponde de acuerdo al permiso por rol. Las contraseñas no se almacenan en texto plano, sino que se cifran con bcrypt, y cada sesión tiene un token único generado por Sanctum. Esta decisión se tomó porque la DI maneja datos que, aunque no son extremadamente sensibles, sí son de uso interno y no deben estar expuestos.

También se verificó que el sistema cumpla con las disposiciones de la PUCE. Todo dato ingresado se usa exclusivamente para la asignación de aulas, no para otro fin. El sistema no recolecta más información de la necesaria, solo solicita lo que hace falta para completar una reserva o para que el administrador gestione los usuarios.

Un aspecto que se consideró importante fue la trazabilidad.

El sistema implementa una tabla audit_log que registra automáticamente cada acción que afecta los datos: quién realizó la acción, qué cambió y cuándo ocurrió. Un ejemplo del registro sería:

- Usuario: tech@informática.puce.edu.ec

- Acción: UPDATE en tabla reservations

- ID del registro afectado: 234

- Detalles del cambio: status modificado de "Pendiente" a "Reservada"

- Timestamp: 2026-02-05 10:23:15

Los técnicos de la DI informaron que se desea reporte de reserva de aula por docente y sus horas del día, por lo cual se estableció la funcionalidad de reportes y el historial de reservas.

Así, estas funcionalidades fueron pensadas para que los técnicos puedan planificar mejor el uso de las aulas. No deben usarse para juzgar el desempeño de docentes ni para hacer comparaciones entre facultades de manera negativa.

Por último, el sistema cuenta con respaldos periódicos para recuperarse de fallos, como la sincronización local-first que mantiene una copia en SQLite. Esto asegura que la información no se pierda en caso de una caída del servidor principal.

Capítulo II

Construcción del Sistema

Metodología del desarrollo

Framework del desarrollo web

Para la construcción del sistema se pensó por una separación entre lo que es el backend y el frontend, algo que al inicio del proyecto se evaluó entre dos opciones: usar Laravel con Blade (que es el motor de plantillas propio de Laravel) o usar Laravel con Vue.js mediante Inertia.js. Se optó por la segunda opción porque se buscaba que la interfaz fuera más llamativa y dinámica, sin recargas de página cada vez que el usuario hace alguna acción, algo que es importante en un sistema donde los técnicos y gestores van a estar consultando y creando reservas de forma continua.

Backend: Laravel Framework

Laravel 11.x fue el framework elegido para el backend, principalmente porque proporciona muchas funcionalidades ya resueltas que de otra forma se hubiese tenido que programar desde cero. Lo que se usa por defecto del framework es:

Eloquent ORM: Es el sistema que Laravel usa para trabajar con la base de datos sin tener que escribir SQL directo. Se definieron modelos como Classroom, Reservation, User, Teacher, Subject, Role, Semester y Report. Cada modelo representa una tabla y establece sus relaciones con las demás. Por ejemplo, el modelo Reservation tiene relaciones con Classroom (a qué aula pertenece), Teacher (qué docente la solicitó), Subject (qué materia se va a impartir) y Semester (en qué periodo académico cae).

Routing: Las rutas del sistema se organizaron en módulos separados para mantener orden. Las rutas públicas van en un archivo, las protegidas en otro, las de

sincronización en otro y las de reportes en otro. Esto se realizó porque al inicio se tenía todo en un solo archivo y se volvió difícil de manejar.

Controladores: Cada entidad del sistema tiene su controlador. Para la parte API se tienen controladores como ClassroomController, ReservationController, SemestralReservationController, UserController, RoleController, entre otros. Los controladores web se encargan de renderizar las páginas Vue usando Inertia.

Validación de Datos: Se usaron las validaciones que trae Laravel tanto del lado del servidor como FormRequests personalizados, para asegurar que los datos que llegan al backend cumplan con las reglas del negocio antes de procesarlos.

Frontend: Vue.js 3 con Inertia.js

Para la interfaz se utilizó Vue.js 3 porque permitió crear componentes reutilizables, por ejemplo, el componente DataTable que se usa en varias páginas para mostrar listados de reservas, usuarios y aulas, o el componente FormModal que se reutiliza en todos los formularios de creación y edición. La conexión entre Laravel y Vue se hace mediante Inertia.js, que en la práctica lo que hace es que Laravel envía los datos directamente a los componentes Vue como si fueran “props”, sin necesidad de crear endpoints JSON separados ni usar fetch o axios para obtenerlos. Esto simplificó bastante el desarrollo porque no fue necesario manejar dos sistemas de rutas (uno en Laravel y otro en Vue).

Inertia mantiene toda la navegación del lado del servidor, así que la seguridad y los permisos se siguen validando en Laravel, pero la experiencia para el usuario es la de una aplicación de página única (SPA) donde la página no se recarga completamente al navegar.

Herramientas de Construcción y Estilos

Vite: Bundler y herramienta de desarrollo next-generation que "proporciona una experiencia de desarrollo superior mediante una retroalimentación instantánea" (Vite, s.f.), compilando los archivos Vue.js y CSS con recarga en caliente (hot reload) para mejorar la experiencia de desarrollo.

Tailwind CSS: Framework de estilos CSS basado en utilidades que "permite construir diseños modernos sin salir de HTML" (Tailwind CSS, s.f.), permitiendo diseñar interfaces responsivas de forma rápida y garantizando coherencia visual en toda la aplicación.

Axios: Cliente HTTP utilizado para realizar solicitudes asincrónicas desde el frontend hacia el backend, facilitando la comunicación con los endpoints API del servidor (Axios, s.f.).

Autenticación y Seguridad

Para la autenticación se usa Laravel Sanctum en modo SPA, que trabaja con cookies de sesión junto con protección CSRF para prevenir ataques de falsificación de solicitudes. Cuando un usuario inicia sesión, Sanctum genera un token único que queda asociado a ese usuario y que se valida en cada petición posterior. Se eligió Sanctum sobre Passport u otro generador de token porque es más liviano y porque no se necesita autenticación OAuth, solo un sistema de tokens simples para la aplicación de escritorio. Se complementa con Laravel Jetstream que proporcionó funcionalidades de gestión de perfiles de usuario ya construidas, además del soporte para autenticación de dos factores que se dejó preparado para activarse en caso de que la DI lo requiera más adelante. (Laravel, s.f.),

Conexión entre Backend y Frontend

La arquitectura del sistema implementa el patrón MVC proporcionado por Laravel y Arquitectura por capas ya que se usa NativePHP sincronizando con PostgreSQL. A continuación, se muestra el proceso de conexión entre el Backend y Frontend.

Solicitud del Usuario: El usuario interactúa con los componentes Vue.js en el aplicativo de escritorio.

Seguridad: Los datos pasan por varios servicios de protección que son Security Headers para ver la integridad de los datos, Sanctum para verificar los pedidos estén autenticados. CheckPermission que sirve para verificar que permiso tiene el rol que trata de ingresar.

Envío de Datos: Vue.js captura los datos y los envía al servidor Laravel mediante Inertia.js.

Procesamiento en Backend: Laravel recibe la solicitud, valida los datos utilizando sus reglas de validación, consulta o modifica la base de datos SQLite de nativePHP mediante Eloquent y sincroniza la base de datos PostgreSQL, aplicando la lógica de negocio del sistema.

Respuesta y Actualización: Laravel responde con los datos procesados, que Inertia.js utiliza para actualizar reactivamente los componentes Vue.js sin necesidad de recargar el aplicativo.

Esta arquitectura garantiza un sistema eficiente, seguro y fácil de mantener, proporcionando a los usuarios finales una experiencia fluida y responsiva (Pressman &

Maxim, 2014). Para ver el diagrama respecto a la conexión entre Backend y Frontent revise el Anexo 1.

Metodología del trabajo

El desarrollo del Sistema de Gestión para la Asignación de Aulas en la DI de la PUCE se llevará a cabo mediante la metodología ágil SCRUM, la cual permite organizar el trabajo de forma colaborativa, iterativa y estructurada. Esta metodología facilita la división del proyecto en ciclos cortos de desarrollo denominados sprints, lo que posibilita la entrega progresiva de incrementos funcionales del sistema.

Según (Drumond, 2025), "Scrum es uno de los marcos de metodología ágil más populares, que ayuda a los equipos a abordar proyectos complejos dividiendo el trabajo en ciclos iterativos más pequeños llamados sprints, fomentando la colaboración, la transparencia y la mejora continua". En este contexto, SCRUM resulta adecuada para el proyecto debido a la necesidad de validar constantemente los requerimientos funcionales y operativos de la Dirección de Informática.

1. Justificación de la Metodología

Desde el inicio del proyecto se tenía claro que los requerimientos podían ir cambiando conforme avanzara el desarrollo, sobre todo porque se dependía de las indicaciones de la DI y del tutor. Por eso se decidió usar SCRUM en lugar de una metodología más rígida como cascada. La idea era poder entregar avances funcionales cada semana y recibir retroalimentación antes de seguir con lo siguiente. Esto resultó útil, por ejemplo, en el Sprint 2 se identificó que la lógica de reservas semestrales era más compleja de lo que se había pensado y se pudieron ajustar las tareas del Sprint 3 sin que eso afectara el cronograma general. También sirvió para organizar el trabajo entre dos personas, ya que cada sprint tenía tareas asignadas de forma clara y ambos desarrolladores podían avanzar en paralelo.

2. Roles de Scrum

El equipo de trabajo se organizará de acuerdo con los roles definidos en el marco de trabajo SCRUM, adaptados al contexto académico del proyecto:

- Product Owner: Representado por la DI responsable de definir y priorizar los requerimientos del sistema, así como de establecer los criterios de disponibilidad y asignación de aulas. De acuerdo con (Drumond, 2025), el Product Owner es quien comprende en mayor medida las necesidades del negocio y del usuario final.
- Scrum Master: Rol asumido por el Ing. José Luis Galarraga, tutor del proyecto, quien se encargará de guiar el correcto uso de la metodología SCRUM, asegurar el cumplimiento de los objetivos académicos y facilitar el desarrollo ordenado del proyecto.
- Equipo de Desarrollo: Conformado por Armando Estévez y Nelson Guijarro, quienes asumirán el rol de desarrolladores, siendo responsables del diseño, implementación y pruebas del sistema, utilizando tecnologías de backend y frontend.

3. Artefactos

Durante el desarrollo del proyecto se emplearán los siguientes artefactos:

- Product Backlog: Lista priorizada de todas las funcionalidades del sistema, tales como el inicio de sesión, registro de solicitudes, historial de reservas, validación de facultades y visualización dinámica de la disponibilidad de aulas. Este artefacto será gestionado y actualizado por el Product Owner.
- Sprint Backlog: Conjunto de historias de usuario y tareas seleccionadas del Product Backlog para ser desarrolladas durante un sprint específico, definidas por el equipo de desarrollo.

- Incremento de Software: Producto funcional resultante de cada sprint, que representará una versión estable y utilizable del sistema, lista para ser evaluada en el entorno de la DI.

4. Sprint

El proyecto se estructurará en ciclos de trabajo denominados sprints, con una duración fija de una semana. Al inicio de cada sprint se llevará a cabo la Sprint Planning, donde se definirá el alcance y las actividades a desarrollar durante el ciclo.

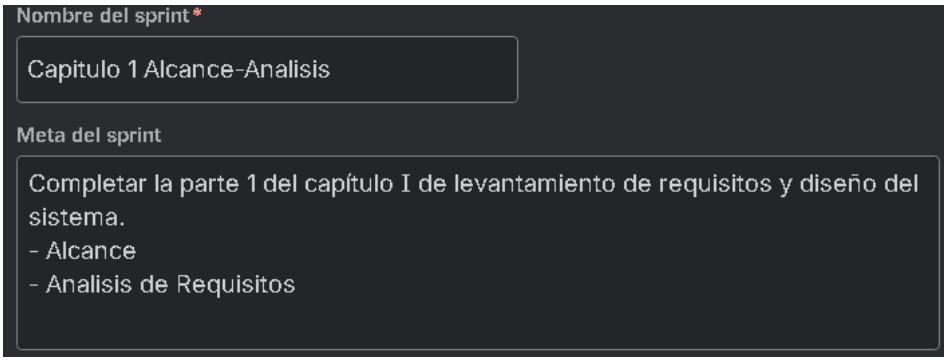


Figura 16: Sprint 1- Sprint Planning

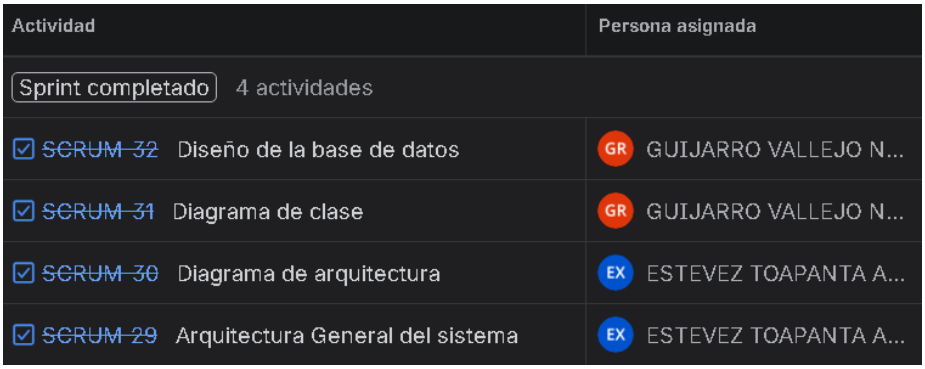


Figura 17: Actividades de Sprint 1

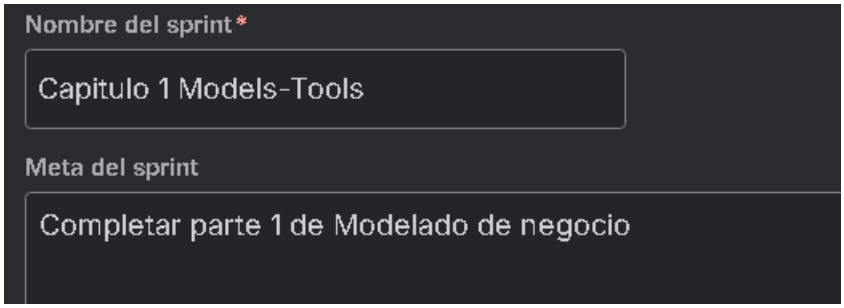


Figura 18: Sprint 2 - Sprint Planning

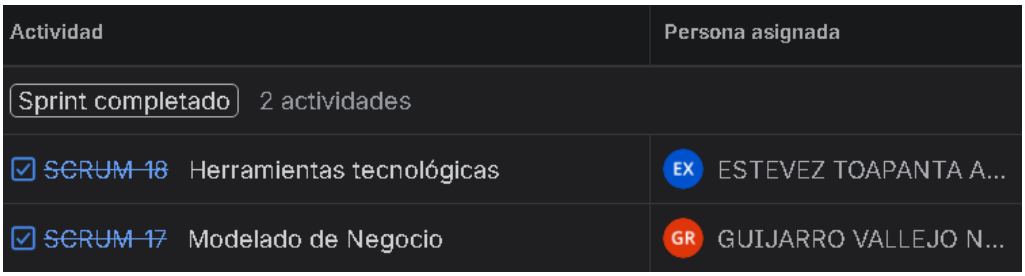


Figura 19: Actividades de Sprint 2

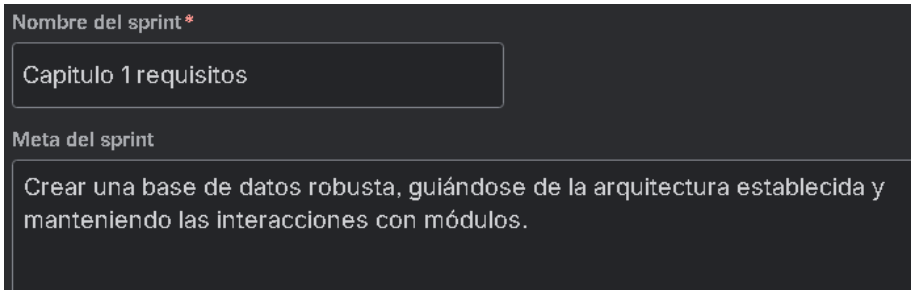


Figura 20: Sprint 3 - Sprint Planning

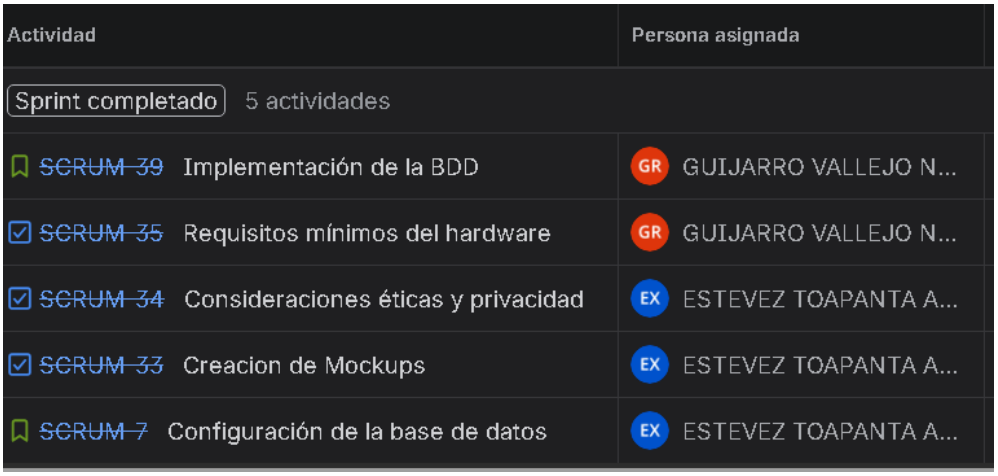


Figura 21: Actividades de Sprint 3

Figura 22: Sprint 5 – Sprint Planning

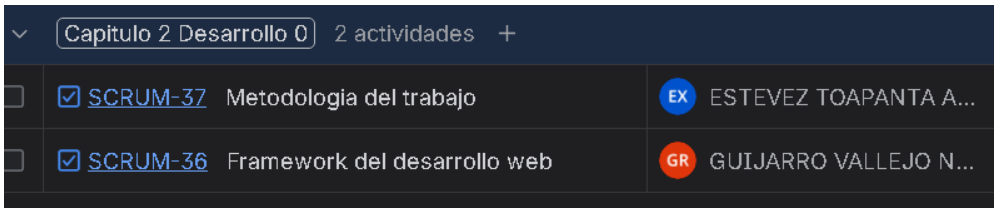


Figura 23: Sprint 4 - Sprint Planning

Durante la ejecución del sprint, se realizará el desarrollo técnico, en este caso este documento para el Sprint 1. A continuación se realiza el Sprint Retrospective que indica que estamos haciendo bien, que hay que cambiar y mejorar.

Ejemplo de desarrollo de Sprint Retrospective del Sprint del Capitulo 1.

Retrospective: Capitulo 1 requisitos

De ESTEVEZ TOAPANTA ARMANDO XAVIER | Ver visualizaciones | Añade una reacción

Reflexiona sobre trabajos anteriores e identifica oportunidades de mejora siguiendo las instrucciones de la estrategia de retrospectiva.

Resumen

Fecha	21 dic 2025
Equipo	Desarrollo
Participantes	Armando Estevez y Nelson Guijarro

Retrospectiva

Empezar a hacer	Dejar de hacer	Seguir haciendo
• Recopilar los requisitos nombrados por los tecnicos de la DI de la PUCE. • Mensionar que hace y como se hace.	• Aumentar deuda tecnica esperando llenar luego. • Dejar imcompleto cada requisito.	• Detalle de cada requisito. • Organizacion de cada uno. • Comunicacion con el equipo.

Figura 24: Sprint 4 - Sprint Planning

Implementación del Back-End

Para la implementación del Back-End se utilizará el modelo de base de datos ya definido en la Figura 5 y 6.

Se indica el ejemplo de cómo se realiza una entidad en PHP-Laravel, en este caso la tabla “Users”

```php
public function up(): void
{
    Schema::create(table: 'users', callback: function (Blueprint $table): void {
        $table->id();
        $table->string(column: 'name');
        $table->string(column: 'email')->unique();
        $table->timestamp(column: 'email_verified_at')->nullable();
        $table->string(column: 'password');
        $table->string(column: 'description')->nullable(value: true);
        $table->string(column: 'department')->nullable();
        $table->string(column: 'profile_photo_path', length: 2048)->nullable();


        $table->rememberToken();
        $table->timestamps();

        /**
         * Reference the migrations.
         */
        $table->unsignedBigInteger(column: 'role_id');
        $table->unsignedBigInteger(column: 'faculty_id')->nullable();

        $table->foreignId(column: 'current_team_id')->nullable();

        $table->foreign(columns: 'role_id')
            ->references(columns: 'id')
            ->on(table: 'role')
            ->onDelete(action: 'cascade');

        $table->foreign(columns: 'faculty_id')
            ->references(columns: 'id')
            ->on(table: 'faculty')
            ->onDelete(action: 'set null');


    });

    Schema::create(table: 'password_reset_tokens', callback: function (Blueprint $table): void {
        $table->string(column: 'email')->primary();
        $table->string(column: 'token');
        $table->timestamp(column: 'created_at')->nullable();
    });
}

/**
 * Reverse the migrations.
 */
public function down(): void
{
    Schema::dropIfExists(table: 'users');
    Schema::dropIfExists(table: 'password_reset_tokens');

}
```

Figura 25: Ejemplo de código para la entidad Users

Laravel indica a estas tablas como migraciones, la figura 24 contiene las relaciones hacia demás entidades, como indica la Figura 5 y 6 la relación de Users es con Roles de “Un rol para muchos usuarios”, Facultad de “Facultad con varios gestores”.

API Rest

Para la implementación de nuestra API Rest en laravel, se usa el paquete de instalación Api, que permite realizar peticiones HTTP y probar los End-Points. Este paquete se debe configurar en la carpeta bootsrap/app.php que proporciona la configuración de varios acciones en Laravel.

El proyecto lleva una organización por End-Points.

├── /api/health	→ Health check (monitoreo)
├── /api/user	→ Usuario autenticado (auth)
├── /api/auth/*	→ Autenticación (públicos)
├── /api/public/*	→ Lecturas públicas (GET)
├── /api/protected/*	→ CRUD protegido (auth + permisos)
├── /api/sync/*	→ Sincronización (headers especiales)
├── /api/reports/*	→ Reportería (auth + permisos)
└── /api/sync/status	→ Monitoreo de sincronización

Se aclarar que también se usa End-Point para mostrar los componentes Vue en NativePHP que es la aplicación de escritorio, esto se hace en web.php

─ /	→ Página de inicio (pública)
─ /dashboard	→ Panel principal (auth)
─ /admin/*	→ Administración (auth + permisos)
─ /faculty/*	→ Gestión de facultad (auth + permisos)
─ /technician/*	→ Panel técnico (auth + permisos)

Posteriormente gracias al ORM (ELOQUENT) que nos proporciona Laravel se pude conectar a la base de datos configurando un archivo ".env"

```
# AQUI INGRESAS LA CONTRASEÑA DE TU BASE
#TAMBIEN LE CAMBIAS EL NOMBRE DE LA BASE
DB_CONNECTION=pgsql
DB_HOST=127.0.0.1
DB_PORT=5432
DB_DATABASE=Asignacion_Aulas_PUCE
DB_USERNAME=postgres
DB_PASSWORD=123456789
```

Figura 26: Variables de entorno que usa ELOQUENT

En los controladores se usa "use Illuminate\Support\Facades\Http;" que permite utilizar los métodos de peticiones, en este caso según Laravel Facades permite acceder a los servicios y clases del contenedor de inyección de dependencias.

```
$response = Http::post(url: 'http://asignacionaulaspuce.test/api/register', data: [
    'name' => $input['name'],
    'email' => $input['email'],
    'password' => $input['password'],
    'description' => $input['description'],
    'role_id' => $input['role_id'],
]);
```

Figura 27: Petición post usando servicio Http de Facades

Posteriormente se realiza las rutas usando Route que proporciona Facades, se muestra el ejemplo de route/api/sync para sincronizaicon con PostgreSQL.

```
Route::middleware(middleware: ['sync-without-auth'])->prefix(prefix: 'sync')->group(callback: function (): void {

    // ================================================================
    // RECURSOS CON CONTROLADORES SYNC DEDICADOS
    // ================================================================

    // Facultades
    Route::post(uri: '/faculties', action: [FacultySyncController::class, 'store'])->name(name: 'sync.faculties.store');
    Route::put(uri: '/faculties/{id}', action: [FacultySyncController::class, 'update'])->name(name: 'sync.faculties.update');
    Route::delete(uri: '/faculties/{id}', action: [FacultySyncController::class, 'destroy'])->name(name: 'sync.faculties.dest

    // Docentes
    Route::post(uri: '/teachers', action: [TeacherSyncController::class, 'store'])->name(name: 'sync.teachers.store');
    Route::put(uri: '/teachers/{id}', action: [TeacherSyncController::class, 'update'])->name(name: 'sync.teachers.update');
    Route::delete(uri: '/teachers/{id}', action: [TeacherSyncController::class, 'destroy'])->name(name: 'sync.teachers.destro

    // Materias
    Route::post(uri: '/subjects', action: [SubjectSyncController::class, 'store'])->name(name: 'sync.subjects.store');
    Route::put(uri: '/subjects/{id}', action: [SubjectSyncController::class, 'update'])->name(name: 'sync.subjects.update');
    Route::delete(uri: '/subjects/{id}', action: [SubjectSyncController::class, 'destroy'])->name(name: 'sync.subjects.destro

    // Reportes
    Route::post(uri: '/reports', action: [ReportSyncController::class, 'store'])->name(name: 'sync.reports.store');
    Route::put(uri: '/reports/{id}', action: [ReportSyncController::class, 'update'])->name(name: 'sync.reports.update');
    Route::delete(uri: '/reports/{id}', action: [ReportSyncController::class, 'destroy'])->name(name: 'sync.reports.destroy');
```

Figura 27: Creación de End-Points

Seguridad y Autenticación

La implementación de la seguridad cumple con 6 capas.

Autenticación en el inicio se sesión: Se valida en LoginController, valida las credenciales como email y contraseña para otorgar acceso al sistema.

El propósito es asegurar que solo el usuario con los legítimos permisos pueda acceder al sistema. Cuenta con bloque en caso de que haya varios fallos al intentar acceder, esto previene varios ataques de múltiples solicitudes como los famosos DDOS.

La contraseña siempre es hasheada, usa el algoritmo bcrypt para poder generar esto hash de contraseña

```php
// 3. Buscar usuario con relaciones
$user = User::where('email', $email)
    ->with('role.permissions')  // Eager load
    ->first();

// 4. Verificar credenciales (Hash::check = timing-safe)
if (!$user || !Hash::check($request->password, $user->password)) {

    // Registrar intento fallido
    $this->recordLoginAttempt($email, $ipAddress, false, $userAgent);

    // Verificar si dispara bloqueo
    $attemptsRemaining = $this->getAttemptsRemaining($email);

    if ($attemptsRemaining === 0) {
        AuditLog::logAuthEvent(
            AuditLog::ACTION_ACCOUNT_LOCKED,
            $email,
            'Account locked after 5 failed attempts'
        );
    }
```

Figura 28: Control de bloqueo por varios intentos de sesión.

Se usa token para las sesiones, como se mencionó anteriormente Laravel usa Sanctum token con CSRF esto proporciona seguridad antes sesiones falsas. Es decir, cada que alguien inicia sesión el servicio de Laravel Sanctum crea un registro en la tabla de tokens que tiene por defecto y los tokens que cree sean únicos para las futuras solicitudes.

```php
// 5. Generar token Sanctum
$token = $user->createToken('api_token')->plainTextToken;
```

Figura 28: Generación de tokens,

```php
public function up(): void
{
    Schema::create(table: 'personal_access_tokens', callback: function (Blueprint $table)
        $table->id();
        $table->morphs(name: 'tokenable');
        $table->text(column: 'name');
        $table->string(column: 'token', length: 64)->unique();
        $table->text(column: 'abilities')->nullable();
        $table->timestamp(column: 'last_used_at')->nullable();
        $table->timestamp(column: 'expires_at')->nullable()->index();
        $table->timestamps();
    });
}
```

Figura 29: Tabla que almacena los tokens de usuarios.

Para una vista general de el proceso de Seguridad y Autenticación revisar el Anexo 2.

Implementación de funcionalidades especificas

Existe tres funcionalidades que se destacan en el proyecto pues ayudan al usuario en general a navegar en la aplicación de escritorio, generar reportes en PDFs y control por de permisos por roles, está claro que existen más funciones dentro de los requisitos funcionales aún se destaca estas tres por su implementación.

Búsqueda Global

Esta funcionalidad permite al usuario buscar mediante varios parámetros como: nombre del docente, aula, fecha, facultad, materia, tipo de reserva. Además, este buscador esta creado para que muestre las coincidencias por tipeo, es decir, mientras se escribe va mostrando lo que coincida. En general se usa para buscar la reserva.

```
export function useReservationFilters() {
  // Estado de filtros
  const filters = ref({
    search: '',           // Búsqueda global
    type: 'all',          // all | semestral | ocasional
    status: 'all',        // all | Reservada | Cancelada | disponible
    classroom_id: null,   // ID del aula específica
    teacher_id: null,     // ID del profesor
    subject_id: null,     // ID de la materia
    date: null,           // Fecha específica
    from: null,           // Rango desde
    to: null,             // Rango hasta
    day_of_week: null,    // 0-6 (Domingo-Sábado)
  });
```

Figura 30: definición de estado de filtros.

```
const applyLocalSearch = (reservations) => {
  if (!filters.value.search) {
    return reservations;
  }

  const query = filters.value.search.toLowerCase();

  return reservations.filter(r =>
    r.classroom?.class_code?.toLowerCase().includes(query) ||
    r.classroom?.name?.toLowerCase().includes(query) ||
    r.subject?.name?.toLowerCase().includes(query) ||
    r.subject?.code?.toLowerCase().includes(query) ||
    r.teacher?.name?.toLowerCase().includes(query) ||
    r.start_time?.toLowerCase().includes(query) ||
    r.end_time?.toLowerCase().includes(query) ||
    r.modality?.toLowerCase().includes(query) ||
    r.status?.toLowerCase().includes(query)
  );
};
```

Figura 31: Búsqueda local en texto.

Generación de PDFs

En esta funcionalidad se puede visualizar los reportes en cada apartado, aulas, docentes, horario y espacio para que firme el docente. El usuario con los permisos en su rol adecuados podrá generar el reporte en PDF si lo desea de acuerdo con los campos mencionados anteriormente. Para ello se usa la librería jsPDF lo cual permite realizar el PDF y guardarlo en su dispositivo.

```
const generatePDF = async (reportData) => {
  try {
    const doc = new jsPDF()
    const pageWidth = doc.internal.pageSize.getWidth()
    const pageHeight = doc.internal.pageSize.getHeight()
    const margin = 15
    let yPosition = margin
```

Figura 32: Generar el PDF desde los datos de los campos nombrados.

```
  // Guardar y descargar PDF
  const fileName = `${reportData.title}_${new Date().toISOString().split('T')[0]}.pdf`
  doc.save(fileName)

  return true
} catch (error) {
  console.error('Error generando PDF:', error)
  throw error
}
```

Figura 33: Guardar y descargar PDF.

Permisos por Rol

Esta funcionalidad tiene el fin de entregar autorización de realizar acciones en varias gestiones de la aplicación de escritorio, tales como, CRUD completo de Roles, CRUD completo de Usuarios, CRUD completo de Aulas, Reportes, ver docentes, facultades, materias.

```
public function handle(Request $request, Closure $next, ...$permissions): Response
{
    $user = $request->user();

    // Verificar que el usuario esté autenticado
    if (!$user) {
        Log::warning(message: 'CheckPermission: Unauthenticated user', context: [
            'route' => $request->route()?->getName() ?? $request->path(),
            'ip' => $request->ip(),
        ]);
        abort(code: Response::HTTP_UNAUTHORIZED, message: 'Usuario no autenticado');
    }

    // Asegurar que el rol y permisos estén cargados
    if (!$user->relationLoaded(key: 'role')) {
        $user->load(relations: 'role.permissions');
    }

    // Verificar que el usuario tenga un rol asignado
    if (!$user->role) {
        Log::error(message: 'CheckPermission: Usuario sin rol asignado', context: [
            'user_id' => $user->id,
            'user_email' => $user->email,
        ]);
        abort(code: Response::HTTP_FORBIDDEN, message: 'Usuario sin rol asignado');
    }
```

Figura 34: Verificación de permisos.

Figura 35: Se obtiene permisos en cache para el rol.

Reservas Semestrales (Recurrentes)

Una de las funcionalidades más largas del sistema es el manejo de las reservas semestrales. A diferencia de una reserva ocasional, que se hace para un solo día, la reserva semestral se registra una única vez, pero el sistema se encarga de generar automáticamente todas las reservas individuales que corresponden a las fechas del semestre según el día de recurrencia seleccionado.

Por ejemplo, si un docente requiere un aula todos los martes de 10:00 a 12:00 durante el semestre 2024-2, el gestor solo debe completar el formulario indicando la asignatura, el docente, el aula, el horario y seleccionar la modalidad “semestral” con el día de recurrencia “martes”. A partir de esta información, el sistema calcula todas las fechas que coinciden con ese día dentro del período activo del semestre y crea una reserva para cada una de ellas.

El reto de esta funcionalidad es que, antes de confirmar la operación, el sistema debe verificar que no existan conflictos de horario en cada una de las fechas generadas. Si en alguno de los martes el aula ya se encuentra ocupada en ese mismo horario, el sistema cancela todo el proceso e informa al usuario la fecha específica en la que se presenta el conflicto, evitando así inconsistencias en la planificación.

Este proceso se gestiona mediante el servicio SemestralReservationService, que trabaja en conjunto con ReservationValidationService para realizar las validaciones de manera eficiente y ordenada.

Al final, todas las reservas creadas a partir de una reserva semestral quedan asociadas a una reserva principal mediante el campo parent_reservation_id. Esto permite que, si es necesario cancelar una serie completa de reservas, se pueda hacer desde la reserva principal sin tener que eliminarlas una por una, facilitando la administración del sistema.

Patrones de diseño aplicados

Durante el desarrollo surgió la necesidad de aplicar ciertos patrones de diseño para mantener el código organizado, sobre todo en las partes más grandes del sistema.

El patrón que ayudo fue el de herencia con clase abstracta en la capa de sincronización. Como cada entidad del sistema (aulas, usuarios, reservas, docentes, materias, facultades, roles y reportes) necesita sincronizarse entre SQLite y PostgreSQL, al inicio se empezó a escribir la lógica de sincronización por separado para cada una. Se identificó que el código se estaba repitiendo: todas tenían que verificar si el registro existe, comparar versiones, manejar errores y encolar reintentos. Por ello se creó una clase AbstractSyncService que contiene toda esa lógica común y cada servicio de sincronización específico solo define qué modelo usa y cómo se mapean los campos. Esto ahorró tiempo considerable y redujo errores.

También se usó el patrón Factory para crear instancias de los servicios de sincronización. En lugar de que el código que procesa la cola de sincronización tenga que saber qué servicio usar para cada tipo de entidad, el SyncServiceFactory recibe el tipo (por ejemplo "classroom" o "reservation") y devuelve el servicio correcto. Esto

hace que agregar una nueva entidad al sistema de sincronización sea tan simple como crear su servicio y registrarlo en la fábrica.

Además, se implementaron Traits para reutilizar funcionalidades transversales que necesitan varias clases. El trait Auditable se agrega a los modelos que requieren registro de auditoría y automáticamente registra las acciones de creación, actualización y eliminación. El trait ApiResponse estandariza el formato de las respuestas JSON de toda la API para que siempre tengan la misma estructura. El trait ValidatesReservationData centraliza las reglas de validación que se usan tanto al crear como al editar reservas, evitando tener esas reglas duplicadas en dos controladores diferentes. Por último, ClearsReportCache invalida el caché de reportes cuando se modifica algún dato que afecta las estadísticas.

Gestión de semestres y validación de períodos

Una funcionalidad que llevo más tiempo y resultó más complejo de lo esperado fue el manejo de los semestres académicos. El modelo Semester no solo almacena las fechas de inicio y fin del período, sino que también controla un rango específico en el que se permite crear reservas (reservation_start y reservation_end). Esto fue un requerimiento de la DI porque no quieren que se hagan reservas fuera del período habilitado. El servicio de reservas consulta al semestre activo antes de permitir cualquier creación, y si la fecha de la reserva cae fuera del rango o no hay un semestre activo, el sistema rechaza la solicitud con un mensaje claro.

Implementación de Frontend

En la implementación de las funcionalidades en el Frontend se usó Vue3.js en conjunto con Tailwind CSS para un aspecto más moderno y más ágil. Se separa por componentes y por páginas para una mayor organización.

Se hace énfasis en el uso de useApi.js que centraliza el cliente HTTP. Se agrupa cada petición estableciendo su errores. A continuación se indica de forma parcial como se integra en este archivo cada petición:

```
const get = async (url, params = {}) => { loading.value = true …}

const post = async (url, payload) => {  loading.value = true …}

const put = async (url, payload) => { loading.value = true …}

const del = async (url) => { loading.value = true …}
```

El proceso que sigue el usuario es: Presionar en iniciar sesión, se aparece la opción de ingresar el correo y su contraseña, al momento en que el usuario ingrese sus datos y presione en iniciar sesión estos datos pasaran por validación de cliente-side que luego se usa la petición “post” para enviar a la ruta log-in los datos de ingreso, posteriormente se validara el token CSFR generado gracias a Laravel, si sobrepasa el tiempo de espera indicara un error y se mostrará que la sesión esta expirada, en el caso de que no suceda eso se manda un OK en el frontend y se busca internamente al usuario por su email esto se hace en la base de datos local que usa nativePHP, al momento de retornar el usuario se verifica que la contraseña este hasheada, luego pasa por la ruta de verificar autenticación y se almacena la cookie de sesión, pasa a crear el token único para ese usuario y almacenarlo, luego se redirige al dashboar de bienvenida y se actualiza los props y ya ha iniciado sesión correctamente.

Para el registro cada una de las contraseñas no se almacenan en texto plano sino que se usa el hasheo de las mismas, y se proporciona el método de check para verificarlo, se continua con el proceso del log-in mencionado anteriormente.

Al ingresar se usa el patrón RBAC (Control de acceso basado en roles), para mas panorama acerca de este proceso revisar el Anexo 3.

El proceso es: Usuario ingresa luego se verifica el rol y posteriormente los permisos de ese rol. Esto se puede revisar en las Figuras 34 y 35.

En la navegación, de acuerdo con sus permisos en su rol puede desplazarse por las páginas para poder: Crear Reserva, Ver reserva, Ver Reserva Generales, Ver Panel de Control en el cual se indican varias gestiones de acuerdo con sus permisos. A continuación, se muestra el formulario de Crear Reserva.

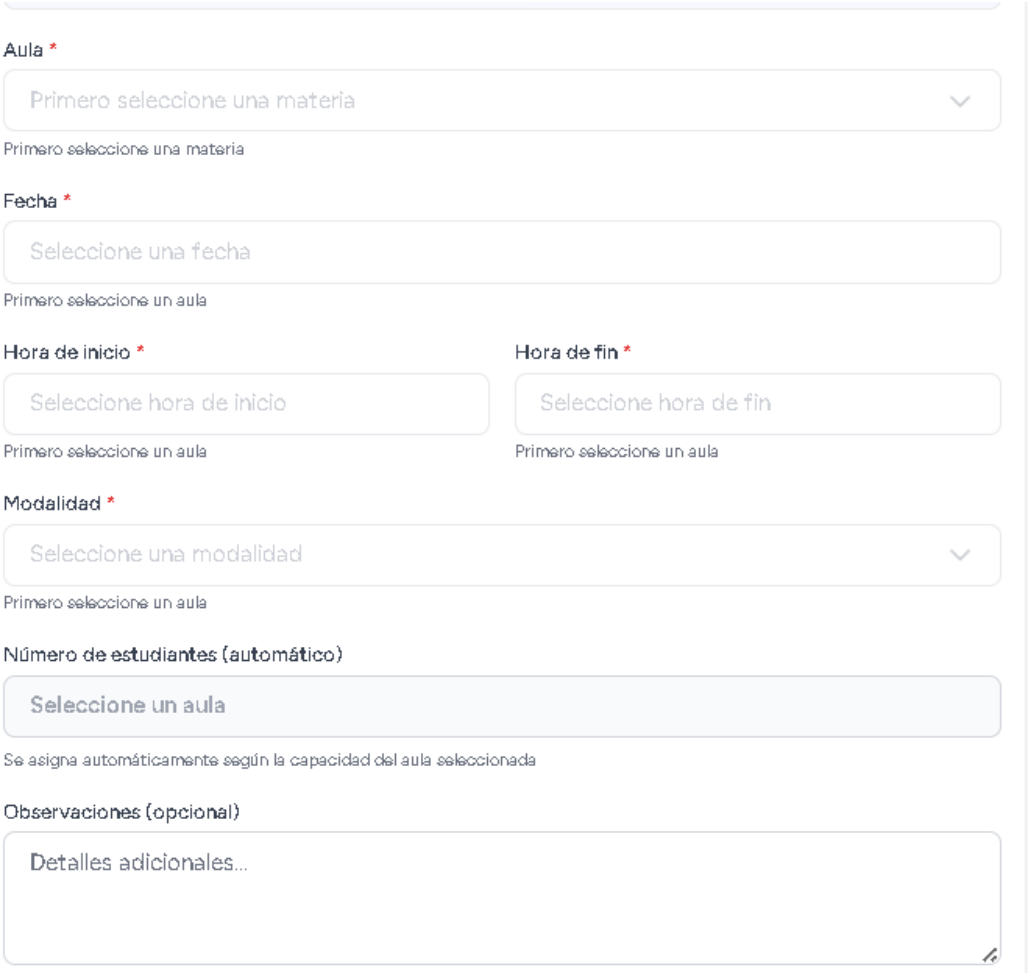


Figura 36: Muestra de formulario de reserva.

Este formulario restringe cada entrada para que el gestor o técnico vaya colocando los datos de acuerdo con la petición del docente.

El panel de administración muestra todas las gestiones.

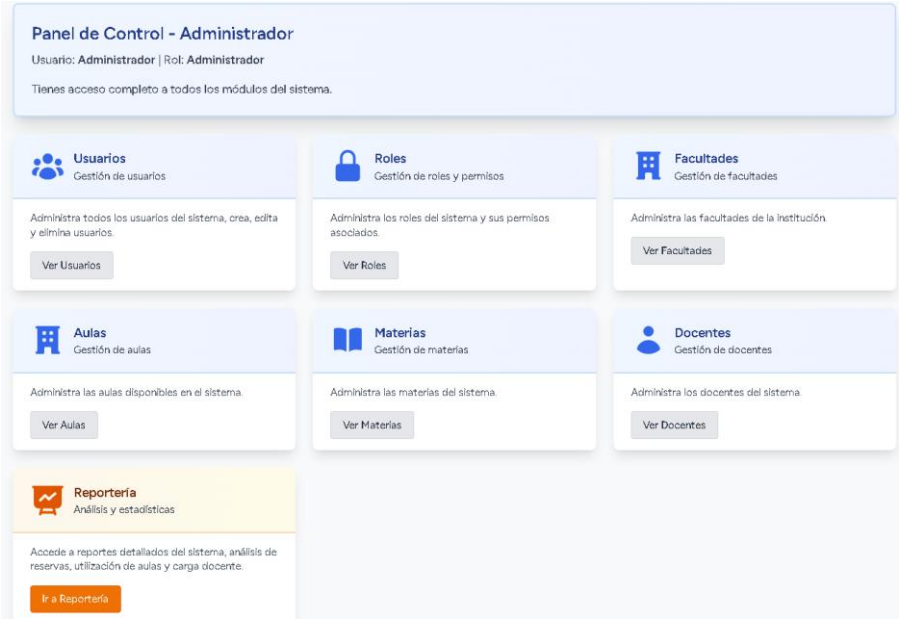


Figura 37: Panel de control.

El panel de calendario es donde todos los usuarios tienen acceso y pueden visualizar las reservas y realizar las búsquedas por en el filtro global.

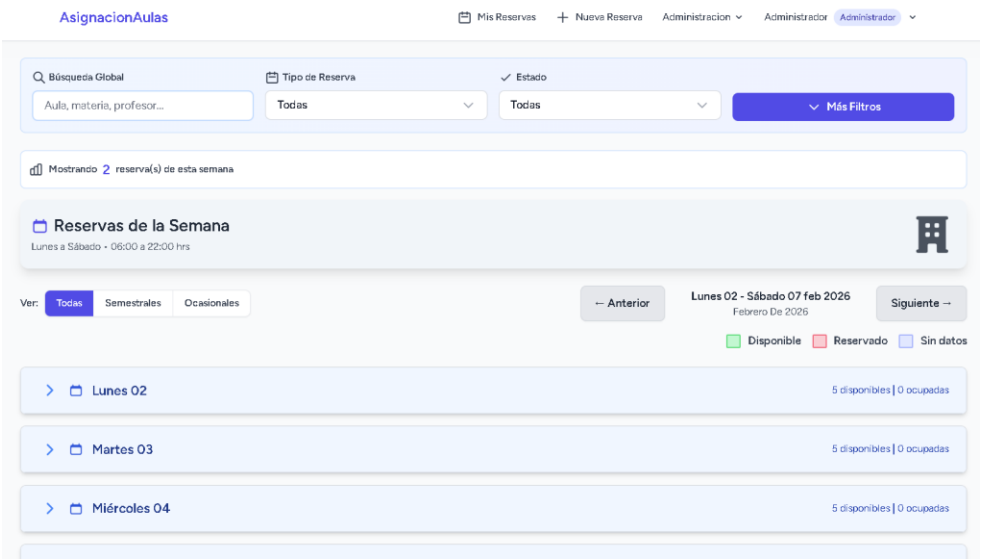


Figura 38: Panel de control.

Cuando se realiza una reserva las reservas aparecen en este panel, se puede visualizar la materia y el profesor que la imparte además de mostrar que esta reservado ese horario.

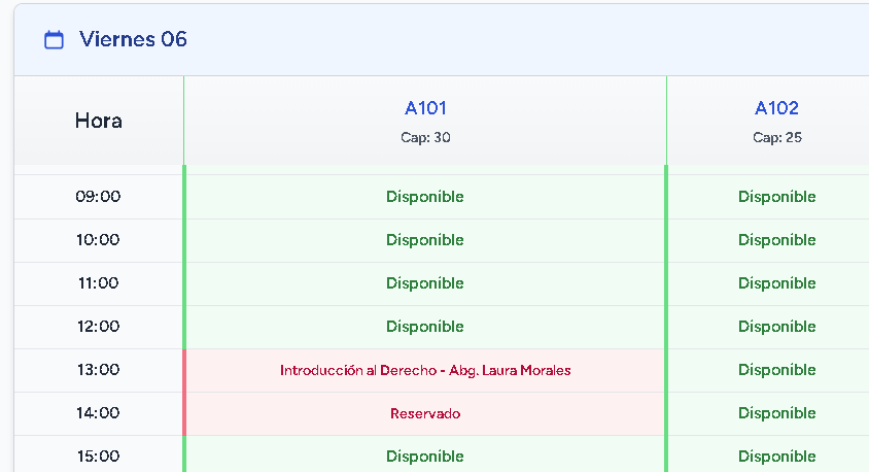


Figura 39: Panel de control.

Como se menciono en el inicio de la implementación del frontend existe un aparatado que muestra las reservas que ha hecho ese usuario, para el caso del administrador y el técnico podrán ver todas las reservas creadas, pero para el gestor podrá ver únicamente sus reservas creadas en este panel. Se indica como ejemplo como es este panel para el administrador.

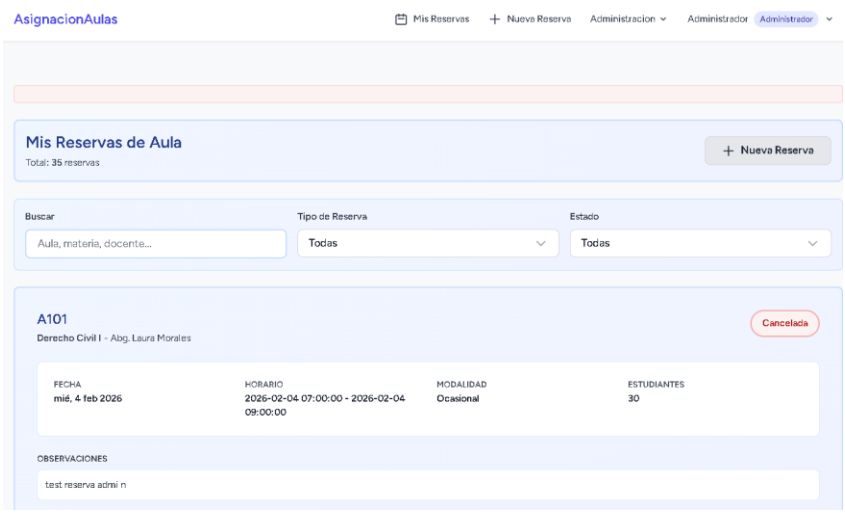


Figura 40: Panel de control.

Capítulo III

Pruebas y Estabilización

Introducción

Las pruebas del Sistema de Asignación de Aulas se realizaron para verificar que todo funcione correctamente antes de entregar la aplicación de escritorio. Cada prueba simula lo que haría un usuario real en un entorno de producción. Se realizaron tres tipos de pruebas: unitarias para probar cada componente por separado y asegurar que las relaciones entre modelos, las validaciones y los cálculos internos funcionen correctamente; de integración para verificar que los componentes trabajen bien entre sí cuando se hacen peticiones HTTP completas con middleware y base de datos; y de rendimiento para medir si los tiempos de respuesta del sistema están dentro de lo aceptable.

Pruebas Unitarias

Estas pruebas verifican que cada clase funcione correctamente de forma aislada, sin depender de otras partes del sistema. Se probaron los modelos Eloquent para confirmar que las relaciones (por ejemplo que una Reservation pertenece a un Classroom y a un Teacher), los campos configurados y los castings de datos estén bien definidos. También se probaron los servicios de negocio como ReservationService y CacheService para verificar que la lógica de validación de conflictos y el manejo de caché funcionen según lo esperado.

Pruebas de Integración

En estas pruebas simulan una petición HTTP: desde que llega al servidor, pasa por los middlewares de seguridad y permisos, llega al controlador, ejecuta la lógica de negocio y devuelve una respuesta. Todo esto se realiza contra una base de datos SQLite en memoria que replica la estructura de la base real. Esto permitió detectar problemas que no aparecen en las pruebas unitarias, como errores en los middlewares de permisos o en la forma en que los controladores validan los datos de entrada.

Pruebas de Rendimiento

Las pruebas de rendimiento ayudan a medir el tiempo de ejecución de las peticiones, verificar el soporte de fuljo de alta carga y detecta cuellos de botella.

Plan de Pruebas Generales

El objetivo de las pruebas del Sistema de Gestión de Asignación de Aulas tiene como objetivo principal validar la funcionalidad, estabilidad y seguridad de la aplicación de escritorio y su API REST en diferentes escenarios de uso, roles de usuario y condiciones de conectividad. Se busca identificar y corregir errores que puedan afectar la gestión de reservas de aulas, el control de acceso basado en roles y la sincronización de datos entre la base local y el servidor.

Prueba por módulos: Cada módulo del sistema (autenticación, reservas, aulas, seguridad y permisos, sincronización local-first, reportes, gestión de usuarios y gestión de semestres/roles) fue sometido a pruebas específicas que validan su funcionamiento individual y su integración con otros componentes.

Pruebas automatizadas: Se implementó un enfoque de pruebas automatizadas utilizando dos frameworks de testing complementarios: PHPUnit para el backend (Laravel 11) y Vitest para el frontend (Vue 3). A diferencia de las pruebas manuales, las

pruebas automatizadas se ejecutan de forma repetible e inmediata ante cada cambio en el código, lo que permite detectar regresiones al instante, validar reglas de negocio.

Cobertura: Las pruebas cubren los casos de uso principales y los flujos alternativos identificados en la fase de diseño. La cobertura abarca las tres capas de la aplicación: modelos de datos, servicios de negocio y controladores/API.

Criterios de aceptación

Funcionalidad. Todas las funcionalidades del sistema deben funcionar según las especificaciones definidas en los casos de uso: la gestión de reservas de aulas, la asignación por roles, la detección de conflictos de horario, la sincronización local-servidor y la generación de reportes deben producir los resultados esperados en el 100% de los escenarios probados.

Usabilidad. La interfaz debe ser intuitiva y permitir que usuarios como gestores y técnicos de la PUCE y DI respectivamente, sin experiencia técnica puedan completar tareas básicas como crear una reserva, consultar disponibilidad de aulas y generar un reporte en menos de 3 intentos. La navegación debe adaptarse al rol del usuario (Administrador, Técnico, Gestor de Facultad).

Rendimiento. Las operaciones críticas del sistema deben completarse dentro de los tiempos establecidos:

- El listado de reservas en menos de 500 milisegundos
- La detección de conflictos de horario en menos de 100 milisegundos
- La creación de reservas en menos de 3 segundos
- El inicio de sesión en menos de 500 milisegundos y la recuperación de datos cacheados en menos de 1 segundo

- Los tiempos de respuesta de la API REST no deben exceder los 500 milisegundos en conexiones estándar.

Compatibilidad. El sistema debe funcionar correctamente como aplicación de escritorio en Windows (x64) mediante NativePHP 2/Electron 38.

Seguridad. Los mecanismos de autenticación y autorización deben prevenir accesos no autorizados en el 100% de los casos de prueba. El sistema de Control de Acceso Basado en Roles (RBAC) debe bloquear correctamente las acciones no permitidas para cada rol, el administrador debe tener bypass completo, y los headers de seguridad OWASP (X-Content-Type-Options, X-Frame-Options, X-XSS-Protection, Referrer-Policy, Permissions-Policy y Content-Security-Policy) deben estar presentes en todas las respuestas HTTP. El cifrado debe emplear AES-256-CBC y las contraseñas deben almacenarse con BCrypt.

Pruebas Unitarias

Modulo Autenticación y Seguridad

Prueba 1: Inicio de sesión válido.

Objetivo: Verificar que los usuarios puedan iniciar sesión con credenciales válidas a través de la API REST

Datos de entrada: Usuario registrado (test@example.com), contraseña correcta (password123)

Pasos para seguir:

1. Crear un usuario de prueba con correo y contraseña conocidos en la base de datos

2. Enviar petición POST a /api/login con email y password en formato JSON
3. Verificar que la respuesta tenga código HTTP 200
4. Verificar que la respuesta JSON contenga las claves user (con id, name, email), token y status
5. Verificar que el campo status sea "success"

Resultado esperado: El sistema retorna código 200 con un token de acceso Sanctum, los datos del usuario autenticado y el estado "success"

Estado: Aprobado

Prueba 2: Inicio de sesión con credenciales inválidas.

Objetivo: Verificar que el sistema rechace credenciales incorrectas y registre el intento fallido

Datos de entrada: Usuario registrado (test@example.com), contraseña incorrecta (wrongpassword)

Pasos para seguir:

1. Crear un usuario de prueba en la base de datos
2. Enviar petición POST a /api/login con email correcto y contraseña incorrecta
3. Verificar que la respuesta tenga código HTTP 401
4. Verificar que el JSON contenga success: false

5. Verificar que se registre un intento fallido en la tabla login_attempts con successful: false
6. Verificar que se cree un registro de auditoría con acción LOGIN_FAILED

Resultado esperado: El sistema retorna código 401, registra el intento fallido en la auditoría y en login_attempts

Estado: Aprobado

Prueba 3: Registro de auditoría en inicio de sesión.

Objetivo: Verificar que cada inicio de sesión exitoso genere un registro en el log de auditoría

Datos de entrada: Usuario registrado, credenciales correctas

Pasos para seguir:

1. Crear un usuario de prueba en la base de datos
2. Enviar petición POST a /api/login con credenciales correctas
3. Consultar la tabla audit_log en la base de datos
4. Verificar que exista un registro con action: LOGIN y entity_type: auth

Resultado esperado: El sistema crea un registro de auditoría que documenta el acceso exitoso, incluyendo el tipo de acción y la entidad

Estado: Aprobado

Prueba 4: Control de acceso RBAC — bypass de administrador.

Objetivo: Verificar que un usuario administrador tenga acceso a todos los recursos sin restricción de permisos

Datos de entrada: Usuario con rol Administrador (role_id = 1)

Pasos para seguir:

1. Crear un usuario con rol Administrador
2. Autenticarlo en el sistema
3. Enviar petición GET a /api/users (requiere permiso leer:usuarios)
4. Verificar que la respuesta tenga código HTTP 200

Resultado esperado: El middleware detecta que el usuario es administrador y permite el acceso sin verificar permisos individuales

Estado: Aprobado

Prueba 5: Control de acceso RBAC — denegación sin permiso.

Objetivo: Verificar que un usuario sin el permiso requerido no pueda realizar la acción protegida

Datos de entrada: Usuario con rol "Gestor de Facultad" que tiene permiso leer:aulas pero NO crear:aulas

Pasos para seguir:

1. Crear roles Administrador y Gestor de Facultad con permisos diferenciados
2. Asignar solo el permiso leer:aulas al rol Gestor
3. Crear un usuario con rol Gestor y autenticarlo

4. Enviar petición POST a /api/classrooms con datos de nueva aula
5. Verificar que la respuesta tenga código HTTP 403 (Forbidden)

Resultado esperado: El middleware CheckPermission bloquea la petición con código 403 porque el usuario no posee el permiso crear:aulas

Estado: Aprobado

Módulo de Reservas

Prueba 1: Creación de reserva ocasional exitosa.

Objetivo: Verificar que un usuario autenticado pueda crear una reserva ocasional con datos válidos

Datos de entrada: Usuario administrador autenticado, aula con capacidad 30, docente, materia, fecha futura (no domingo), horario 10:00-12:00, 20 estudiantes

Pasos para seguir:

1. Crear datos de prueba: facultad, roles, usuario admin, aula, docente, materia y semestre activo
2. Autenticar al usuario administrador
3. Enviar petición POST a /api/reservations con classroom_id, teacher_id, subject_id, start_time, end_time y number_of_student
4. Verificar que la respuesta tenga código HTTP 201
5. Verificar que el JSON contenga modality: Ocasional y status: Reservada
6. Verificar que el semester_id asignado corresponda al semestre activo

Resultado esperado: El sistema crea la reserva con modalidad Ocasional, estado Reservada y la asigna automáticamente al semestre activo

Estado: Aprobado

Prueba 2: Detección de conflicto de horario en misma aula.

Objetivo: Verificar que el sistema detecte y rechace una reserva que se solapa con otra existente en la misma aula

Datos de entrada: Reserva existente (10:00-12:00, Aula BIZ01), nueva reserva (11:00-13:00, misma aula, diferente docente)

Pasos para seguir:

1. Crear una reserva en el aula BIZ01 de 10:00 a 12:00
2. Intentar crear otra reserva en la misma aula de 11:00 a 13:00 con un docente diferente
3. Verificar que hasTimeConflict() retorne true

Resultado esperado: El sistema detecta la superposición horaria y retorna conflicto, impidiendo la doble reserva del aula

Estado: Aprobado

Prueba 3: Detección de conflicto de profesor entre aulas diferentes.

Objetivo: Verificar que un profesor no pueda estar asignado a dos aulas diferentes en el mismo horario

Datos de entrada: Docente reservado en Aula 1 (10:00-12:00), intento de reserva en Aula 2 (11:00-13:00) con mismo docente

Pasos para seguir:

1. Crear una reserva del docente en el aula BIZ01 de 10:00 a 12:00
2. Crear un aula diferente (BIZ03)
3. Verificar hasTeacherConflict() para el mismo docente en horario 11:00-13:00
4. Confirmar que retorna true (conflicto detectado)

Resultado esperado: El sistema detecta que el profesor ya está asignado en otra aula durante ese horario y rechaza la nueva reserva

Estado: Aprobado

Prueba 4: Rechazo de reserva en día domingo.

Objetivo: Verificar que el sistema no permita crear reservas en días domingo

Datos de entrada: Fecha que sea domingo, horario 10:00-12:00, datos válidos de aula y docente

Pasos para seguir:

1. Calcular la próxima fecha que sea domingo
2. Invocar canBeCreated() con esa fecha y datos válidos
3. Verificar que el resultado tenga valid: false
4. Verificar que el mensaje contenga la palabra "domingos"

Resultado esperado: El sistema rechaza la reserva con un mensaje indicando que no se permiten reservas en domingos

Estado: Aprobado

Prueba 5: Cancelación de reserva por el propietario.

Objetivo: Verificar que el propietario de una reserva pueda cancelarla proporcionando un motivo

Datos de entrada: Reserva existente creada por un usuario regular, motivo de cancelación

Pasos para seguir:

1. Crear una reserva asignada a un usuario regular
2. Autenticar al usuario propietario
3. Enviar petición POST a /api/reservations/{id}/cancel con cancellation_reason
4. Verificar código HTTP 200 y mensaje "Reserva cancelada exitosamente"
5. Verificar que el estado de la reserva cambie a "Cancelada"
6. Verificar que cancelled_by contenga el ID del usuario y cancellation_reason contenga el motivo

Resultado esperado: La reserva se cancela correctamente, se registra quién la canceló y el motivo

Estado: Aprobado

Prueba 6: Rechazo de cancelación por usuario no propietario.

Objetivo: Verificar que un usuario que no sea propietario ni administrador no pueda cancelar una reserva ajena

Datos de entrada: Reserva creada por el administrador, intento de cancelación por usuario regular

Pasos para seguir:

1. Crear una reserva asignada al usuario administrador
2. Autenticar a un usuario regular (no propietario)
3. Enviar petición POST a /api/reservations/{id}/cancel
4. Verificar que la respuesta tenga código HTTP 403

Resultado esperado: El sistema deniega la cancelación con código 403 porque el usuario no es propietario ni administrador

Estado: Aprobado

Prueba 7: Listado público de reservas excluye canceladas.

Objetivo: Verificar que el endpoint público de listado de reservas no incluya reservas con estado Cancelada

Datos de entrada: Una reserva cancelada y una reserva activa en la base de datos

Pasos para seguir:

1. Crear una reserva con status: Cancelada
2. Crear una reserva con status: Reservada
3. Enviar petición GET a /api/reservations sin autenticación
4. Verificar código HTTP 200
5. Recorrer los resultados y verificar que ningún elemento tenga status: Cancelada

Resultado esperado: El listado público solo muestra reservas activas, filtrando automáticamente las canceladas

Estado: Aprobado

Prueba 8: Cambio de estado a Pendiente por administrador.

Objetivo: Verificar que un administrador pueda cambiar el estado de una reserva a Pendiente

Datos de entrada: Reserva con estado Reservada, usuario administrador autenticado

Pasos para seguir:

1. Crear una reserva con status: Reservada
2. Autenticar al usuario administrador
3. Enviar petición PUT a /api/reservations/{id}/set-pending
4. Verificar código HTTP 200
5. Verificar que el estado de la reserva cambie a "Pendiente"

Resultado esperado: El estado de la reserva se actualiza correctamente de Reservada a Pendiente

Estado: Aprobado

Modulo Reportes y Estadísticas

Prueba 1: Conteo de reservas por modalidad.

Objetivo: Verificar que el servicio de reportes cuente correctamente las reservas por modalidad (Ocasional/Semestral)

Datos de entrada: 2 reservas ocasionales y 1 semestral en un semestre específico

Pasos para seguir:

1. Crear 2 reservas con modalidad Ocasional y 1 con modalidad Semestral en el semestre de prueba
2. Invocar getReservationsByModality() con el ID del semestre
3. Verificar que el resultado contenga las claves occasional y semestral
4. Verificar que occasional sea 2 y semestral sea 1

Resultado esperado: El servicio retorna el conteo exacto por modalidad, excluyendo reservas canceladas

Estado: Aprobado

Prueba 2: Resumen general del dashboard con todas las métricas.

Objetivo: Verificar que el resumen del dashboard contenga las 15 métricas requeridas

Datos de entrada: Datos de prueba con aulas, docentes y reservas en la base de datos

Pasos para seguir:

1. Invocar getSummary() del servicio de reportes
2. Verificar la presencia de las 15 claves: total_reservations, reservations_today, reservations_this_month, reservations_this_year, active_classrooms, total_classrooms, classroom_utilization_percentage, active_teachers, total_students_scheduled, average_students_per_reservation, occasional_reservations, semestral_reservations, pending_reservations, active_reservations, cancelled_reservations

Resultado esperado: El resumen contiene todas las métricas requeridas para el dashboard administrativo

Estado: Aprobado

Módulo de Sincronización

Prueba 1: Sincronización con headers válidos sin autenticación.

Objetivo: Verificar que el middleware permita peticiones de sincronización con el secreto compartido correcto

Datos de entrada: Headers X-Skip-Sync: true y X-Sync-Secret con el valor correcto configurado en el servidor

Pasos para seguir:

1. Configurar el secreto de sincronización en la configuración del sistema
2. Enviar petición POST a /api/sync/roles con los headers X-Skip-Sync: true y X-Sync-Secret: [secreto correcto]
3. Verificar que la respuesta NO tenga código 401

Resultado esperado: El middleware permite la petición de sincronización sin requerir autenticación de usuario, validando únicamente el secreto compartido

Estado: Aprobado

Prueba 2: Rechazo de sincronización con secreto incorrecto.

Objetivo: Verificar que el middleware rechace peticiones de sincronización con un secreto inválido

Datos de entrada: Header X-Skip-Sync: true, X-Sync-Secret con valor incorrecto

Pasos para seguir:

1. Configurar el secreto de sincronización correcto en el servidor
2. Enviar petición POST a /api/sync/roles con X-Sync-Secret: wrong-secret-value
3. Verificar que la respuesta tenga código HTTP 401
4. Verificar que el JSON contenga error: Unauthorized

Resultado esperado: El middleware rechaza la petición con código 401 porque el secreto no coincide con el configurado en el servidor

Estado: Aprobado

Prueba 3: Rechazo de petición sin autenticación ni headers de sincronización.

Objetivo: Verificar que las rutas de sincronización rechacen peticiones sin credenciales ni headers de sync

Datos de entrada: Petición sin autenticación y sin headers especiales

Pasos para seguir:

1. Enviar petición POST a /api/sync/roles sin headers de autenticación ni sincronización
2. Verificar que la respuesta tenga código HTTP 401

Resultado esperado: El sistema rechaza la petición porque no tiene ni autenticación de usuario (Sanctum) ni el secreto de sincronización válido

Estado: Aprobado

Módulo de Rendimiento

Prueba 1: Rendimiento del listado de reservas.

Objetivo: Verificar que el listado de 100 reservas se complete en menos de 500 milisegundos

Datos de entrada: 100 reservas creadas en la base de datos, usuario administrador autenticado

Pasos para seguir:

1. Crear 100 reservas distribuidas en días futuros diferentes
2. Autenticar al usuario administrador
3. Registrar el tiempo de inicio con microtime(true)
4. Enviar petición GET a /api/reservations
5. Registrar el tiempo de fin y calcular la diferencia en milisegundos
6. Verificar que el tiempo sea menor a 500ms

Resultado esperado: El listado de 100 reservas se completa en menos de 500 milisegundos, demostrando que el endpoint es eficiente para el volumen esperado

Estado: Aprobado

Prueba 2: Rendimiento de la detección de conflictos.

Objetivo: Verificar que la detección de conflictos entre 200 reservas se complete en menos de 100 milisegundos

Datos de entrada: 200 reservas en la misma aula distribuidas en 200 días, consulta de conflicto en un día específico

Pasos para seguir:

1. Crear 200 reservas para la misma aula en días diferentes
2. Registrar el tiempo de inicio
3. Invocar getConflictingReservations() para el día 50 en horario 10:00-12:00
4. Registrar el tiempo de fin y calcular la diferencia
5. Verificar que el tiempo sea menor a 100ms

Resultado esperado: La detección de conflictos opera eficientemente incluso con un volumen alto de reservas, confirmando que los índices de base de datos y las consultas SQL están optimizados

Estado: Aprobado

Prueba 3: Eficiencia del eager loading en listado de usuarios.

Objetivo: Verificar que el listado de 50 usuarios no genere más de 10 consultas SQL (detección de N+1)

Datos de entrada: 50 usuarios creados con roles y permisos asignados

Pasos para seguir:

1. Crear 50 usuarios con roles asignados
2. Activar el contador de consultas SQL con DB::enableQueryLog()
3. Enviar petición GET a /api/users como administrador
4. Obtener el número de consultas ejecutadas
5. Verificar que sea menor a 10

Resultado esperado: El sistema utiliza eager loading para cargar las relaciones (rol, permisos) en pocas consultas, evitando el problema N+1 que generaría 50+ consultas

Estado: Aprobado

Resumen de Tests

La fase de pruebas se llevó a cabo durante un período de dos semana, usando ejecución automatizada con PHPUnit para el backend de Laravel y Vitest para los componentes del frontend en Vue.js. Todas las pruebas corren sobre una base de datos SQLite en memoria que se recrea en cada ejecución, lo que garantiza que los resultados sean reproducibles y no dependan de datos previos.

Las pruebas se organizaron en las siguientes categorías:

Pruebas unitarias funcionales: Cubren la lógica de los modelos Eloquent, los servicios de negocio y las operaciones CRUD de las principales entidades del sistema (reservas, aulas, usuarios, roles, semestres, reportes).

Pruebas de seguridad: Validan los mecanismos de autenticación, el control de acceso RBAC, los headers de seguridad OWASP y el manejo de tokens.

Pruebas de compatibilidad y consolidación: Verifican la integración correcta entre los distintos módulos del sistema y la consistencia de los datos.

Pruebas de rendimiento: Miden tiempos de respuesta en operaciones críticas como el listado de reservas, la detección de conflictos y la carga de usuarios con sus relaciones.

Pruebas de sincronización: Verifican la comunicación entre SQLite y PostgreSQL, incluyendo la validación de headers de sincronización y el rechazo de peticiones no autorizadas.

Tasa de éxito

La gran mayoría de las pruebas se ejecutan correctamente sin errores, abarcando los módulos de reservas, reportes, aulas, sincronización, rendimiento, modelos de datos, servicios de negocio y los composables del frontend. Las incidencias identificadas durante las pruebas fueron menores y se registraron para seguimiento, sin afectar la funcionalidad principal del sistema.

Fortalezas

- Rendimiento estable en cada una de las acciones que haga el usuario.
- Seguridad óptima y robusta para el proceso de asignación de aulas.
- Perdida de datos escasa gracias a la sincronización y base de datos SQLite siempre lista.
- Permisos bien establecidos para acceso a gestiones.

Conclusiones

El desarrollo del Sistema de Gestión para la Asignación de Aulas permitió cumplir el objetivo general del proyecto: construir una aplicación de escritorio funcional que automatiza el proceso de asignación de aulas en la Dirección de Informática de la PUCE. Con este sistema se reemplaza el flujo manual que antes se realizaba mediante archivos Excel y correos electrónicos, reduciendo errores y tiempos de gestión.

Respecto a los objetivos específicos, se implementó un sistema completo que permite crear, editar, buscar y cancelar reservas, diferenciando entre modalidad ocasional y semestral. Las reservas semestrales representaron uno de los mayores retos del proyecto, ya que fue necesario generar automáticamente múltiples reservas recurrentes y validar posibles conflictos en cada fecha, una complejidad que no se había previsto inicialmente con tanto detalle.

El módulo de validación y asignación automática funciona de manera correcta, detectando conflictos de horario tanto por aula como por docente. Este aspecto fue uno de los más valorados por la DI durante las revisiones, ya que evita que se asigne un aula ocupada o que un docente quede programado en dos espacios al mismo tiempo.

Los reportes y el historial de reservas se implementaron mediante la generación de archivos PDF y un panel de estadísticas que muestra métricas como reservas por modalidad, uso de aulas y actividad por período. Esta información permite a los técnicos planificar de mejor forma el uso de los espacios académicos.

El control de acceso basado en roles se definió con los cuatro perfiles planteados y permisos específicos configurables por el administrador. Las pruebas realizadas confirmaron que el sistema restringe correctamente las acciones no autorizadas, fortaleciendo la seguridad de la aplicación.

En cuanto a la interfaz, se diseñó pensando en que los gestores de facultad y técnicos puedan completar sus tareas en pocos pasos. El uso de Vue.js con Inertia proporciona una navegación fluida, evitando recargas innecesarias de página y mejorando la experiencia de uso.

Finalmente, la arquitectura local-first con sincronización entre SQLite y PostgreSQL garantiza que la aplicación funcione incluso cuando existen problemas de conexión con el servidor. Además, las pruebas automatizadas se ejecutan correctamente y cubren los flujos principales del sistema, el control de accesos y el rendimiento de las operaciones más críticas.

Recomendaciones

Se recomienda implementar herramientas de linting y formateo automático (ESLint/Prettier para Vue y PHP CS Fixer para Laravel).

Considerar la implementación de notificaciones en tiempo real usando WebSockets o Laravel Echo, para que cuando un gestor cree una reserva, los técnicos reciban una notificación inmediata sin necesidad de recargar su pantalla.

Si se requiere integrar el sistema con la plataforma SARS de la PUCE, sería necesario desarrollar una capa de adaptadores que consuma la API de SARS, similar a la que ya existe con ExternalApiDataService para la obtención de datos de docentes, materias y facultades.

Referencias bibliográficas

contributors, W. (s.f.). *Modelo–vista–controlador*. Obtenido de Wikipedia, The Free Encyclopedia: https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador

Drumond, C. (25 de 12 de 2025). *ATLASSIAN*. Obtenido de ¿Qué es scrum? Desglose del marco de trabajo ágil: https://www.atlassian.com/es/agile/scrum

Hernández, U. (22 de 02 de 2015). *MVC (Model, View, Controller) explicado*. Obtenido de Codigofacilito: https://codigofacilito.com/articulos/mvc-model-view-controller-explicado

Jain, A. (22 de mayo de 2025). *¿Qué es el análisis de requisitos? Proceso y técnicas*. Obtenido de Visure Solutions; Visure Solutions, Inc: https://visuresolutions.com/es/gu%C3%ADa-de-limosna/an%C3%A1lisis-de-requisitos-a/

LatamTech. (s.f.). *Qué es la arquitectura en capas: descubre sus ventajas y ejemplos*. Obtenido de LATAMTECH-SOFTWARE-CLOUD-SUPPORT: https://latamtech-sac.com/que-es-la-arquitectura-en-capas-descubre-sus-ventajas-y-ejemplos/

Sommerville, I. (2011). *INGENIERÍA DE SOFTWARE - Novena edición.* México: PEARSON EDUCACIÓN.

Axios. (s.f.). Axios - Promise based HTTP client for the browser and node.js. Recuperado de https://axios-http.com/

Patrones de diseño. (n.d.). Refactoring.guru. Retrieved February 8, 2026, from https://refactoring.guru/es/design-patterns

Inertia.js. (s.f.). Inertia.js Official Guide. Recuperado de https://inertiajs.com/

Laravel. (s.f.). Laravel Documentation. Recuperado de https://laravel.com/docs/11.x

Pressman, R. S., & Maxim, B. R. (2014). Software Engineering: A Practitioner's Approach (8ª ed.). McGraw-Hill Education.

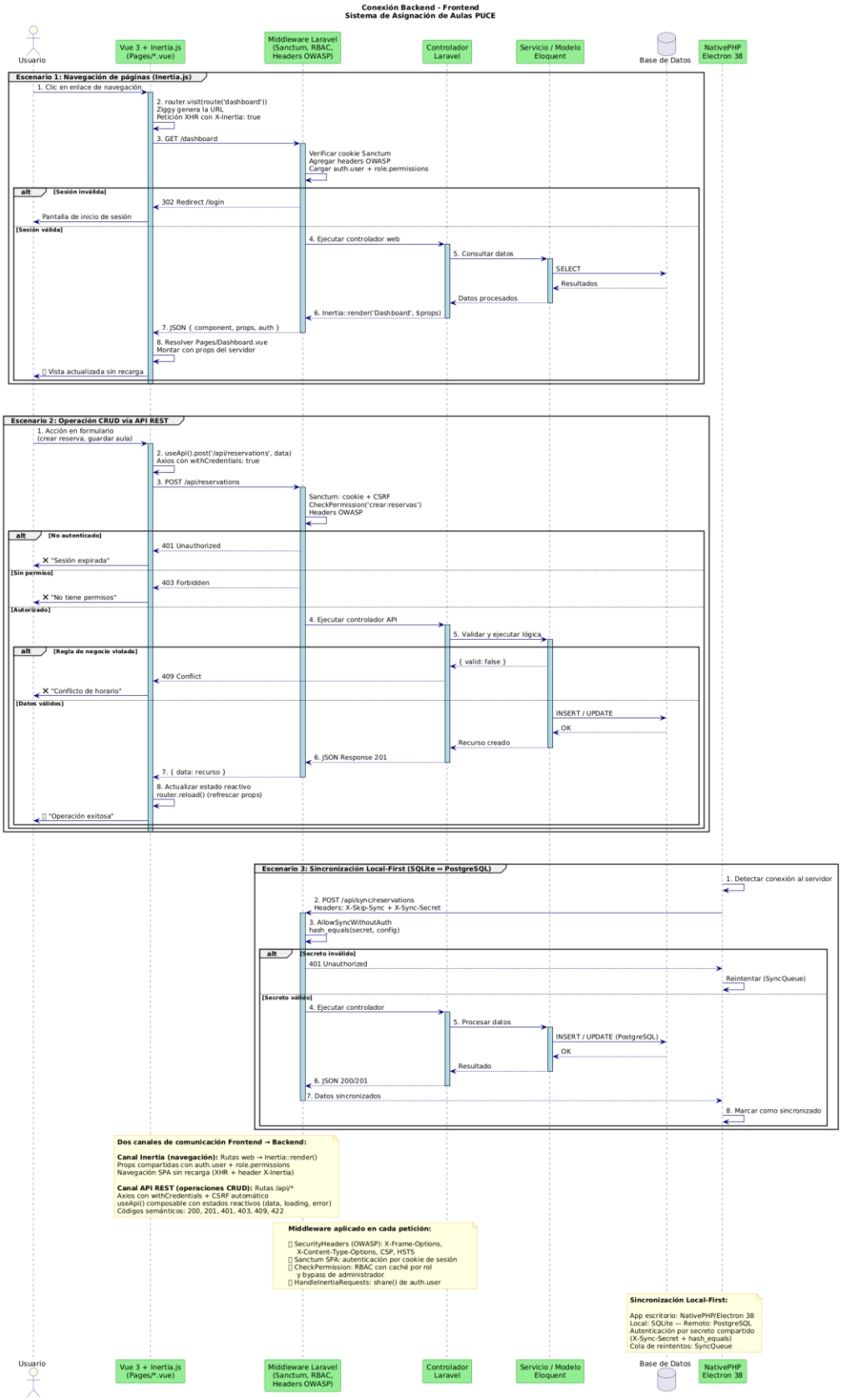
Tailwind CSS. (s.f.). Tailwind CSS - Rapidly build modern websites without leaving your HTML. Recuperado de https://tailwindcss.com/

Vite. (s.f.). Vite - Next Generation Frontend Tooling. Recuperado de https://vitejs.dev/

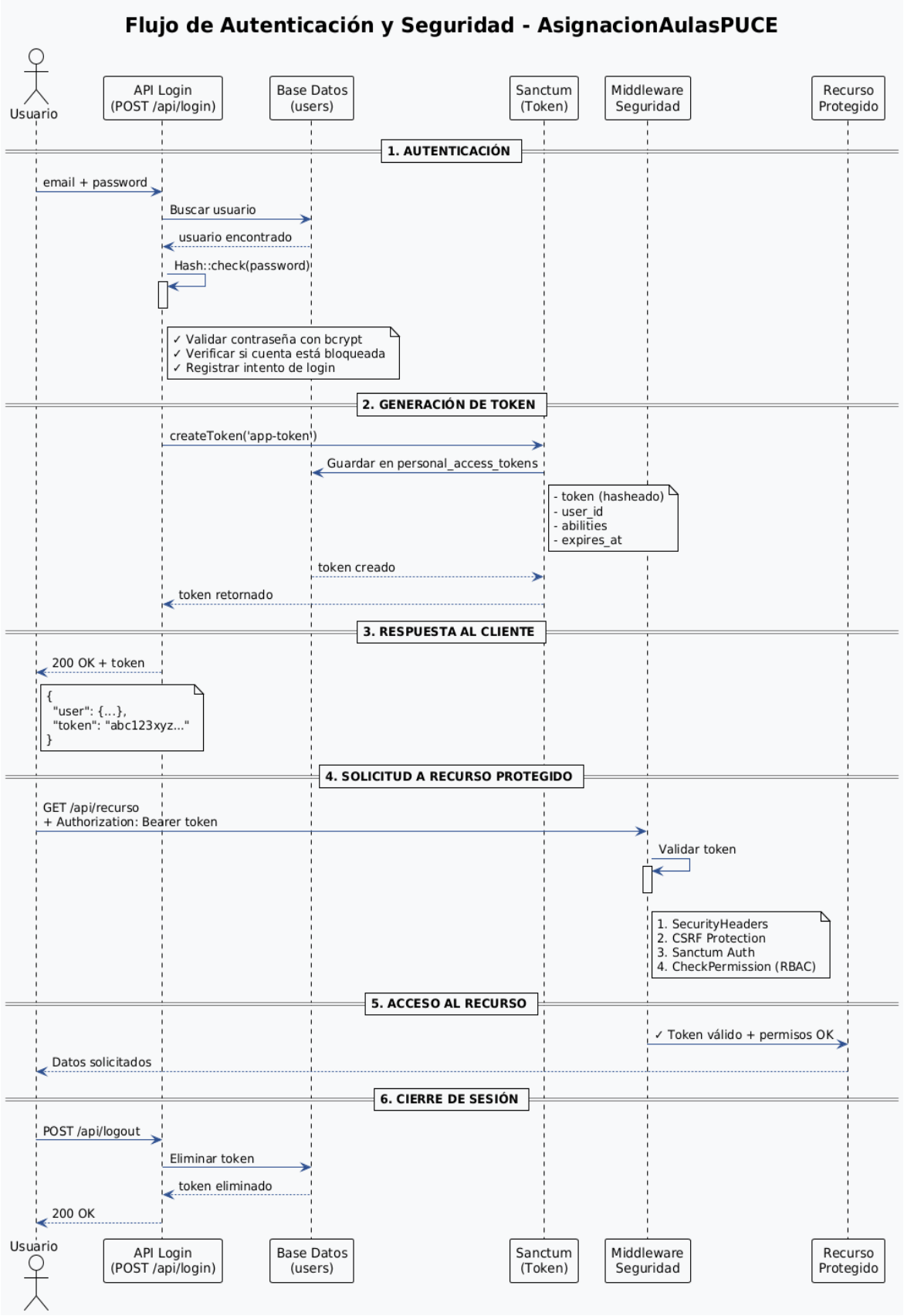
Vue.js. (s.f.). Vue.js 3 Guide. Recuperado de https://vuejs.org/guide/

Anexos

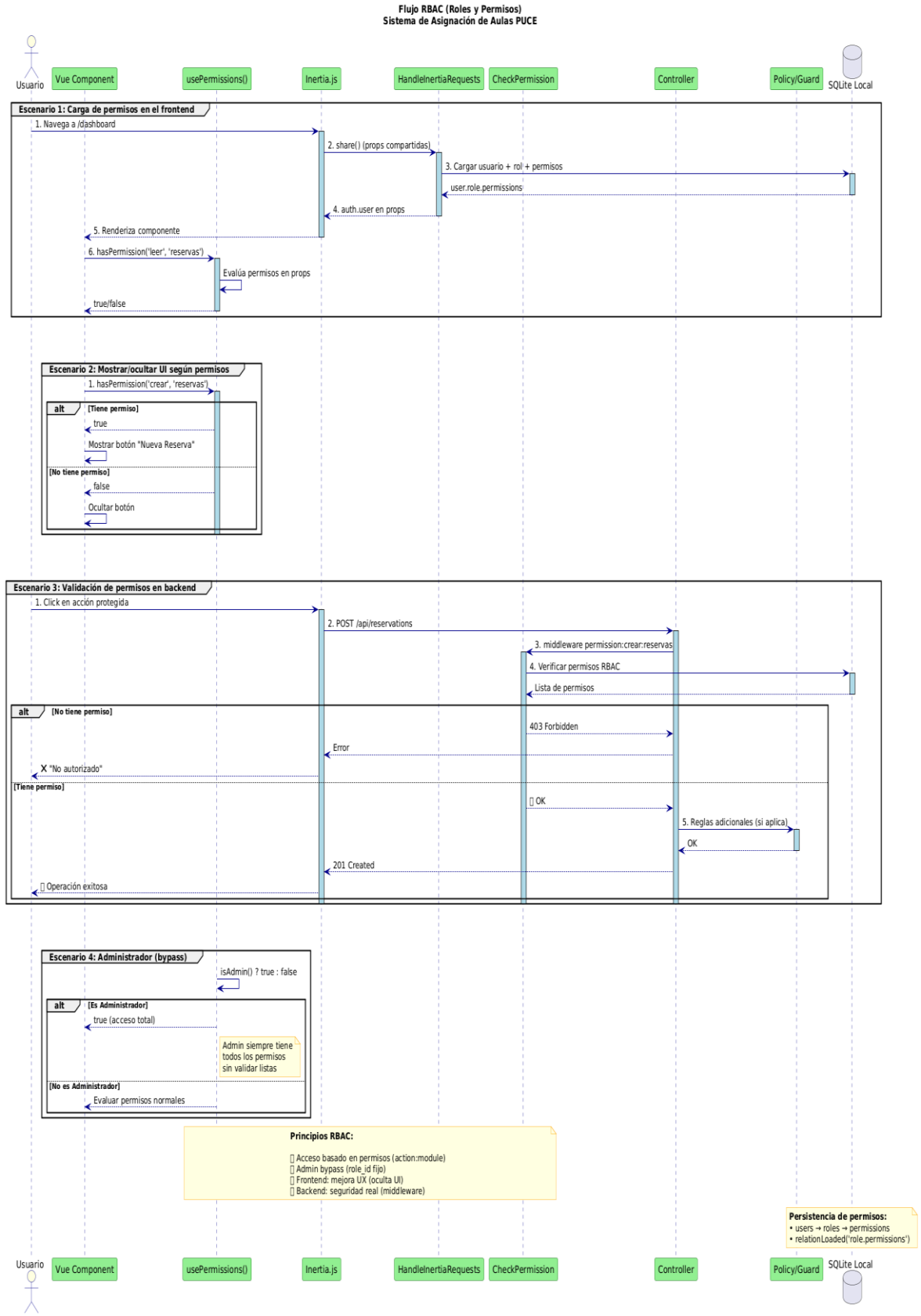
Anexo 1



Anexo 2


Flujo de Autenticación y Seguridad - AsignacionAulasPUCE
Usuario
API Login (POST /api/login)
Base Datos (users)
Sanctum (Token)
Middleware Seguridad
Recurso Protegido
1. AUTENTICACIÓN
email + password
Buscar usuario
usuario encontrado
Hash::check(password)
✓ Validar contraseña con bcrypt
✓ Verificar si cuenta está bloqueada
✓ Registrar intento de login
2. GENERACIÓN DE TOKEN
createToken('app-token')
Guardar en personal_access_tokens
- token (hasheado)
- user_id
- abilities
- expires_at
token creado
token retornado
3. RESPUESTA AL CLIENTE
200 OK + token
{
"user": {...},
"token": "abc123xyz..."
}
4. SOLICITUD A RECURSO PROTEGIDO
GET /api/recurso
+ Authorization: Bearer token
Validar token
1. SecurityHeaders
2. CSRF Protection
3. Sanctum Auth
4. CheckPermission (RBAC)
5. ACCESO AL RECURSO
✓ Token válido + permisos OK
Datos solicitados
6. CIERRE DE SESIÓN
POST /api/logout
Eliminar token
token eliminado
200 OK
Usuario
API Login (POST /api/login)
Base Datos (users)
Sanctum (Token)
Middleware Seguridad
Recurso Protegido

Anexo 3


Flujo RBAC (Roles y Permisos)
Sistema de Asignación de Aulas PUCE
Usuario
Vue Component
usePermissions()
Inertia.js
HandleInertiaRequests
CheckPermission
Controller
Policy/Guard
SQLite Local
Escenario 1: Carga de permisos en el frontend
1. Navega a /dashboard
2. share() (props compartidas)
3. Cargar usuario + rol + permisos
user.role.permissions
4. auth.user en props
5. Renderiza componente
6. hasPermission('leer', 'reservas')
Evalúa permisos en props
true/false
Escenario 2: Mostrar/ocultar UI según permisos
1. hasPermission('crear', 'reservas')
alt
[Tiene permiso]
true
Mostrar botón "Nueva Reserva"
[No tiene permiso]
false
Ocultar botón
Escenario 3: Validación de permisos en backend
1. Click en acción protegida
2. POST /api/reservations
3. middleware permission:crear:reservas
4. Verificar permisos RBAC
Lista de permisos
alt
[No tiene permiso]
403 Forbidden
Error
"No autorizado"
[Tiene permiso]
OK
5. Reglas adicionales (si aplica)
OK
201 Created
Operación exitosa
Escenario 4: Administrador (bypass)
isAdmin() ? true : false
alt
[Es Administrador]
true (acceso total)
Admin siempre tiene todos los permisos sin validar listas
[No es Administrador]
Evaluar permisos normales
Principios RBAC:
Acceso basado en permisos (action:module)
Admin bypass (role_id fijo)
Frontend: mejora UX (oculta UI)
Backend: seguridad real (middleware)
Persistencia de permisos:
• users → roles → permissions
• relationLoaded('role.permissions')