Desarrollo de un sistema de optimización de la programación académica para la universidad del valle seccional Buga

David Steven Arce Franco

Universidad del Valle
Facultad de ingeniería
Escuela de ingeniería de sistemas y computación
Tuluá, Valle Del Cauca
2025

Desarrollo de un sistema de optimización de la programación académica para la universidad del valle seccional Buga

David Steven Arce Franco
Código 202067533
david.arce@correounivalle.edu.co

Director
Héctor Fabio Ocampo Arbeláez
hector.ocampo@correounivalle.edu.co

Universidad del Valle
Facultad de ingeniería
Escuela de ingeniería de sistemas y computación
Tuluá, Valle del Cauca
2025

Tabla de contenido

Índice de figuras

Índice de cuadros

Resumen

El problema estudiado es la gestión ineficiente de horarios y recursos en instituciones educativas, lo que puede provocar conflictos de horarios y una utilización subóptima de los recursos disponibles. Se estudia con el fin de mejorar la eficiencia y la organización en la programación académica.

El sistema utiliza algoritmos avanzados para optimizar la distribución de clases, estos algoritmos tienen en cuenta múltiples factores, como la disponibilidad de docentes, requerimientos de asignaturas y características de los salones.

Proporcionan una programación académica óptima, minimizando conflictos de horarios y maximizando la utilización de recursos disponibles, a través de una interfaz que permita a los administradores ingresar información relevante.

Los resultados obtenidos demuestran que el Sistema de Optimización de Programación Académica es eficaz para mejorar la gestión de horarios y recursos en instituciones educativas. Esto significa que se puede lograr una distribución equitativa de las clases y una utilización más eficiente de los recursos disponibles, lo que contribuye a un funcionamiento más fluido y productivo.

Palabras claves
Sistema, optimización, programación, algoritmos, seccional, teoría, académica

1. Introducción

En las organizaciones, incluyendo las educativas, poseen sistemas de información los cuales alimentan a grandes bases de datos. Estas bases de datos con gran volumen de información tienden a convertirse en un gran problema para las organizaciones que no poseen una arquitectura de aplicaciones adecuada y de datos correctamente definida[1].

El problema de la planificación de horarios de universidades y escuelas ha sido estudiado en detalle y la cantidad de trabajos publicados sobre este tema ha ido aumentado a lo largo de los años. La importancia de la programación de un buen horario radica en el impacto que tiene sobre la organización laboral de los docentes y el uso eficiente de recursos pedagógicos para la enseñanza que reciben los estudiantes, como, por ejemplo, salas de informática, laboratorios de ciencias o clases de apoyo escolar[2].

Algunas instituciones educativas han implementado sistemas de horarios a través de aplicaciones informáticas y han logrado de esta manera optimizar la gran mayoría o la totalidad los procesos de generación y gestión de horarios, estos sistemas reducen significativamente largos tiempos de trabajo y horas hombre invertidas en estas actividades [3].

Sin embargo, para alcanzar su máximo potencial, es importante adaptar estas soluciones a las necesidades y especificidades de cada institución educativa. En este sentido, el Sistema de Optimización de Programación Académica Buga representa un ejemplo de cómo estas tecnologías se pueden adaptar y optimizar para cumplir con los requerimientos específicos de una sede universitaria o centro educativo en particular.

Se desarrolló el Sistema de Programación con el objetivo de mejorar y optimizar tanto los horarios de los profesores como la asignación de aulas. Utilizando algoritmos avanzados, esta plataforma tiene como objetivo optimizar la asignación de clases, teniendo en cuenta varios factores como la disponibilidad de profesores, los requerimientos específicos de cada asignatura y las características de los espacios físicos disponibles. A través de una interfaz de usuario, los administradores podrán ingresar información relevante, lo que permitirá que el sistema cree de forma semiautomática un cronograma académico óptimo.

Además de facilitar la asignación de horarios, este sistema también ofrece la capacidad de gestionar eficazmente los cambios y ajustes del cronograma. Su capacidad para adaptarse rápidamente a nuevas solicitudes o eventos inesperados, reprogramando horarios de forma semiautomática, ayuda a minimizar las interrupciones y garantizar actividades de aprendizaje continuas y fluidas en la seccional Buga. En este sentido, el sistema de optimización de la programación académica no es sólo una herramienta tecnológica avanzada, sino también un aliado estratégico en la gestión efectiva de los recursos educativos.

2. Formulación del problema

2.1. Descripción del problema

En toda institución de educación superior, al inicio de cada período académico se presenta la necesidad de asignar y coordinar los recursos económicos, materiales y humanos en beneficio de los estudiantes[13]. La problemática de la programación manual de horarios y sus consecuencias negativas no se limita al ámbito educativo. Diversas áreas enfrentan retos similares al momento de gestionar horarios y asignar recursos. Un ejemplo claro es la programación en la entrega de paquetes de las transportadoras. La optimización de rutas y la coordinación de entregas en múltiples puntos generan desafíos similares a los de la programación académica, como la coordinación compleja de horarios de entregas y transportistas, la escasa consideración de la disponibilidad de vehículos y rutas, y la dificultad para adaptarse rápidamente a cambios y ajustes en la programación.

En todas las instituciones de educación, el proceso de creación de un horario académico es un desafío que se debe sortear semestre a semestre. Este proceso no es simple, ya que está sujeto a restricciones físicas, reglamentarias y legales entre otras[14]. La programación manual, tanto en el ámbito educativo como en otros, conlleva una pérdida significativa de tiempo y recursos. El proceso manual de coordinación, ajuste y resolución de conflictos consume horas de trabajo que podrían dedicarse a otras tareas más importantes. Además, la naturaleza manual del proceso aumenta la probabilidad de errores, lo que puede generar retrasos, confusiones y problemas aún mayores.

Este no es un proceso simple, ya que si se analiza con profundidad, se descubre que existe una gama de factores que se transforman en restricciones que deben ser consideradas al momento de generar un horario académico. Así, encontramos restricciones físicas elementales tales como por ejemplo que una sala no puede ser utilizada por más de un curso a la vez, o que un docente no puede realizar más de una clase en el mismo momento; restricciones reglamentarias, como que un alumno con sus asignaturas al día, en el ámbito universitario, no debe tener coincidencia de horario entre dos de sus cursos[14]. La universidad del valle seccional Buga no es ajena a esta problemática, enfrenta dificultades significativas en la planificación y asignación de horarios académicos debido a la falta de un sistema semiautomatizado de gestión. Actualmente, el proceso se realiza manualmente, lo que conlleva a una serie de desafíos, como la coordinación compleja de horarios de docentes y asignaturas, la escasa consideración de la disponibilidad de salones y la dificultad para adaptarse rápidamente a cambios y ajustes en la programación. Esta situación genera conflictos de horarios entre docentes y asignaturas, la utilización poco efectiva de espacios físicos, inequidad en la distribución de clases y un aumento en el tiempo y esfuerzo dedicado a la programación manual.

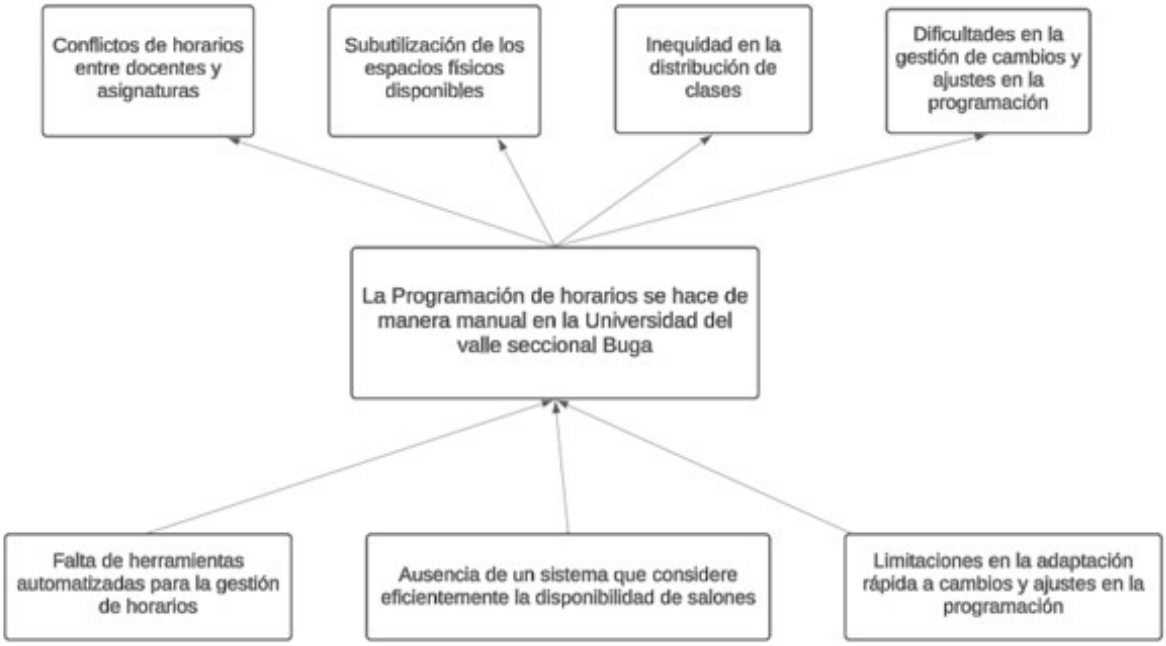

Figura 1: Árbol de problemas

2.2. Definición del problema

¿Cómo la implementación de un sistema de optimización para la programación de horarios en la Universidad del Valle Seccional Buga podría apoyar el proceso manual actual, mejorar la eficiencia y disminuir errores?

3. Marco de referencia

3.1. Marco teórico.

Teoría General de Sistemas (TGS)

La teoría general de los sistemas ha ido ganando terreno conforme la sociedad se va desarrollando. Existen una gran cantidad de investigadores que han contribuido para armar una bibliografía amplia al respecto, autores como Ludwig Von Bertalanffy, considerado el padre de la teoría, y Oscar Johansen Bertoglio constituyen el punto de partida del presente estudio[4]. En el contexto del Sistema de Optimización de Programación Académica, la TGS proporciona un marco conceptual para comprender la interacción entre los diversos elementos involucrados en la gestión de horarios y recursos en instituciones educativas.

Teoría de la optimización convexa

La optimización matemática es el estudio sobre cómo hacer la mejor elección cuando estamos limitados por un conjunto de requerimientos. Cuando hablamos en concreto de optimización convexa nos referimos a minimizar funciones convexas reales definidas para una variable contenida dentro de un subconjunto convexo de un espacio vectorial. Las técnicas abarcadas por este campo de estudio son importantes en aplicaciones de ingeniería porque al plantear un problema de diseño en forma convexa se puede identificar la estructura de la solución óptima, que suele revelar aspectos importantes de dicho diseño[5].

Teoría de la evolución y los algoritmos genéticos

El algoritmo genético es una técnica de búsqueda basada en la teoría de la evolución de Darwin, que ha cobrado tremenda popularidad alrededor del mundo durante los últimos años. Esta técnica se basa en los mecanismos de selección que utiliza la naturaleza, de acuerdo con los cuales los individuos más aptos de una población son los que sobreviven, al adaptarse más fácilmente a los cambios que se producen en su entorno[6].

3.2. Estado del Arte

En el ámbito de la gestión educativa, la optimización de la programación académica es un tema de gran relevancia que ha sido abordado mediante diferentes enfoques y tecnologías. En el estado del arte actual, se observa un creciente interés en el desarrollo de sistemas de optimización de programación académica que emplean algoritmos avanzados y tecnologías emergentes para mejorar la eficiencia y la organización en instituciones educativas.

Tendencias y Avances Recientes

Optimización mediante algoritmos genéticos: Durante las últimas décadas ha habido un creciente interés en algoritmos basados en el principio de la evolución (supervivencia del más apto). Entre los algoritmos evolutivos más conocidos se incluyen los algoritmos genéticos, programación evolucionaria, estrategias evolutivas y programación genética. El conjunto de estas técnicas se agrupa bajo el nombre de computación evolucionaria[15].

Entre los algoritmos genéticos están los siguientes:

Algoritmo Genético Simple: El algoritmo genético simple aplica a problemas de optimización de parámetros continuos, utiliza una regla de supervivencia probabilística[15].

Algoritmos de Nichos Paralelos: Los algoritmos de nichos paralelos forman y mantienen subpoblaciones dentro del espacio de una población única, mediante la disminución de la competencia entre puntos distantes en el espacio de búsqueda.

A continuación, se mencionan algunos trabajos que implementan algoritmos avanzados y tienen que ver con la programación automática o semiautomática de horarios.

Oñate, J. Tullmo, D. (2017), con el tema de titulación 'Optimización Basada En Colonia De Hormigas Para Resolver Problemas Asignación De Horarios En La Universidad Técnica Estatal De Quevedo'[17]

sustentado en la universidad con el mismo nombre. Proyecto donde insisten con la finalidad de disminuir la insatisfacción de los docentes y estudiantes durante el proceso de gestión de horarios. Obtuvieron información del distributivo suministrado por un ingeniero encargado de la gestión horaria de los distributivos en la Universidad Técnica Estatal de Quevedo. Los métodos aplicados fueron: Método de observación, ya que por pertenecer a la institución como estudiantes tuvieron la posibilidad de analizar en persona la situación, como instrumento realizaron entrevistas a los encargados de la elaboración de horarios. Se aplicó el algoritmo Optimización Colonia de Hormigas con una función heurística que compruebe al docente con más conveniente al planificar su horario dependiendo de la prioridad académica. Como conclusión obtuvieron que los resultados arrojados no son cien por ciento exactos; sino, más aproximadas a la solución.

Castrillón, O. (2019). Sobre un artículo con titulado "Combinación entre Algoritmos Genéticos y Aleatorios para la Programación de Horarios de Clases basado en Ritmos Cognitivos"[18]

Se tiene que, en el caso de estudio obtenido de la Universidad Nacional de Colombia, Departamento de Ingeniería Industrial, Campus la Nubia, Manizales-Colombia, el autor desarrolló un método basado en algoritmos evolutivos (genéticos y aleatorios) para programar los horarios de clases en una universidad. Realiza el análisis de diferentes técnicas empleadas para lograr solucionar el problema. Con los resultados de la investigación realizó la comparativa de diferentes metodologías para establecer cuál es la más eficiente para llegar a proponer una metodología basada en la combinación de dos algoritmos evolutivos y aleatorios, para la resolución de las restricciones planteadas. En la solución del problema empleó cuatro multidimensionales arreglos de datos, horarios, profesores y salones. Finalmente, los autores concluyen que los algoritmos evolutivos son más eficaces que otras técnicas aplicadas con un porcentaje del 19.5 de exactitud en la programación de horarios de la institución.

Viñas, S., Rodríguez, N., Corona, E. Jiménez, A, en enero de 2019 como autores del artículo denominado "Avances en Ciencias e Ingeniería", con el tema "Software para la generación automática de horarios académicos"[19]

Quienes cumplieron con la elaboración de un sistema gestor de horario con el fin de presentar un prototipo de software para dar solución a los problemas de optimización de horarios en la Universidad Mexiquense del Bicentenario (UMB) en el departamento de la Unidad de Estudios Superiores Villa Victoria (UESVV) del Estado de México, México. Investigación realizada de un estudio sobre las diferentes técnicas de Inteligencia Artificial (IA), El estudio fue realizado como una investigación de campo través de la entrevista a las autoridades de la Unidad de Estudios Superiores Villa Victoria, con lo que consiguieron recolectar información para realizar el análisis de las técnicas de generar horarios en la institución. El software aplica Algoritmos Genéticos (AG) con la configuración de los parámetros del algoritmo genético generado: tamaño de población, el método de selección (elitismo), el porcentaje de cruce y mutación, y una cantidad de torneo para la selección. Cumpliendo con las restricciones señaladas, generando horarios en un lapso de tiempo corto. Los autores recomiendan que el software desarrollado puede ser fácilmente adaptado en otras instituciones de educación superior.

Modelos de Programación Entera para el Problema de Asignación de Horarios en Cursos Universitarios[7]

Propone un modelo de programación entera (IP), que aborda el problema de programación de horarios conformado por variables asociadas con franjas horarias, asignatura y docente asignado. También se incluyen parámetros como número de franjas horarias mínimas y máximas a impartir por el docente, tiempo de franja horaria, disponibilidad de docen-

te, salones disponibles y costo estimado de insatisfacción generado por el horario asignado.

Aplicación de algoritmos genéticos
El algoritmo genético utiliza el concepto de evolución biológica integrando tres operadores: selección, cruce y mutación para la creación de la población y la selección de las mejores soluciones.

3.3. Antecedentes

UniTime

UniTime es un sistema integral de programación educativa que admite el desarrollo de horarios de cursos y exámenes, la gestión de cambios en estos horarios, el uso compartido de salas con otros eventos y la programación de estudiantes en clases individuales. Es un sistema distribuido que permite a múltiples administradores de horarios universitarios y departamentales coordinar esfuerzos para crear y modificar un horario que satisfaga sus diversas necesidades organizativas y al mismo tiempo permita minimizar los conflictos de cursos de los estudiantes. Puede usarse solo para crear y mantener el cronograma de clases y/o exámenes de una escuela, o conectarse con un sistema de información estudiantil existente.[8]

SOPA (Sistema de optimización de la programación académica)

Tabla 1: Características de proyectos antecedentes

Características	SOPA	Unitime
Desarrollo y Origen	Desarrollado específicamente para la seccional Buga, con enfoque en la adaptación a las necesidades específicas de una institución educativa.	Desarrollado como un esfuerzo colaborativo entre profesores, estudiantes y personal de universidades de América del Norte y Europa, con el propósito de beneficiar a una amplia gama de instituciones educativas.
Algoritmos de Optimización	Utiliza algoritmos avanzados de optimización para distribuir clases, teniendo en cuenta factores como la disponibilidad de docentes, características de las asignaturas y capacidad de los salones.	También emplea algoritmos de optimización para desarrollar horarios de cursos y exámenes, minimizando los conflictos de horarios entre estudiantes y maximizando la utilización de recursos disponibles.
Plataforma	Web	Web
Licencia	Código abierto	Código abierto

3.4. Marco Conceptual

Algoritmos genéticos: Los algoritmos genéticos son métodos sistemáticos para la resolución de problemas de búsqueda y optimización que aplican a estos los mismos métodos de la evolución biológica: selección basada en la población, reproducción sexual y mutación[9].

Programación de restricciones: La programación de restricciones es una tecnología software utilizada para la descripción y posterior resolución efectiva de grandes y complejos problemas, particularmente combinatorios, de muchas áreas de la vida real. Muchos de estos problemas pueden modelarse como problemas de satisfacción de restricciones (CSPs) y resolverse usando técnicas de programación de restricciones[10].

Restricciones: Incluyen limitaciones como la disponibilidad de docentes, las capacidades de las aulas y las preferencias de los estudiantes.

Algoritmos de programación lineal, entrera y mixta: Los modelos de optimización basados en programación lineal, entera y mixta son ampliamente utilizados en problemas reales para formular modelos que contribuyen eficientemente en la toma de decisiones en todos los niveles organizacionales, lo cual tiene extensa contribución en la reducción de costos operativos[11].

4. Alcance del Proyecto

4.1. Declaración del alcance

El desarrollo de este proyecto se centrará en el diseño, desarrollo e implementación de un sistema web de programación académica para la seccional Buga de la Universidad del Valle. El sistema tendrá como objetivo optimizar los horarios de los profesores y la asignación de aulas, los requerimientos específicos de cada asignatura y las características de los espacios físicos disponibles. El sistema ofrecerá una interfaz de usuario que permitirá a los administradores ingresar información relevante y generar de forma semiautomática un cronograma académico óptimo.

El sistema permitirá realizar reprogramaciones a partir de algún cambio para minimizar interrupciones y garantizar la continuidad de las actividades de aprendizaje.

La capacidad de adaptación del sistema estará sujeta a la precisión de los datos ingresados por los administradores y a la disponibilidad de recursos informáticos adecuados.

Restricciones y supuestos

- La integración del sistema debe ser compatible con las plataformas y bases de datos ya existentes en la Universidad del Valle.
- La precisión del cronograma académico generado dependerá de la exactitud y actualización de los datos ingresados, como la disponibilidad de los profesores, la capacidad y características de las aulas, y los requerimientos específicos de cada curso.
- Aunque los algoritmos genéticos y de programación por restricciones son avanzados, su rendimiento puede verse afectado por la complejidad del problema, como la cantidad de cursos, profesores y aulas a gestionar.
- El sistema debe cumplir con las normativas y políticas institucionales de la Universidad del Valle, incluyendo aspectos de privacidad y manejo de datos sensibles.

4.2. Objetivos

4.2.1. Objetivo General

Desarrollar un sistema de optimización para la Universidad del Valle, Seccional Buga, que genere una programación de horarios semiautomática, reduciendo así el tiempo de planificación y minimizando los errores en la asignación de clases y recursos.

4.2.2. Objetivos Específicos

1. Analizar los procesos actuales de la programación académica en la seccional Buga para identificar áreas de mejora y requerimientos específicos del sistema.

2. Diseñar una arquitectura de software adecuada que permita la integración de algoritmos de optimización y una interfaz de usuario para los administradores.

3. Desarrollar los módulos y funcionalidades del Sistema de Programación Académica, incluyendo algoritmos de optimización, gestión de cambios y ajustes.

4. Implementar pruebas y validaciones que permitan evaluar el rendimiento del software y los algoritmos de optimización en escenarios reales.

4.2.3. Resultados esperados

Tabla 2: Resultados esperados

Objetivo Específico	Entregable
Analizar detalladamente los procesos actuales de la programación académica en la seccional Buga para identificar áreas de mejora y requerimientos específicos del sistema.	▪ Informe de Análisis de Procesos y Requerimientos.
Diseñar una arquitectura de software adecuada que permita la integración de algoritmos de optimización y una interfaz de usuario para los administradores.	▪ Documento detallado que describe diseño de la arquitectura se software.
Desarrollar los módulos y funcionalidades del Sistema de Programación Académica, incluyendo algoritmos de optimización, gestión de cambios y ajustes.	▪ Proyecto web del Sistema de Programación Académica completo.
Implementar pruebas y validaciones que permitan evaluar el rendimiento del software y los algoritmos de optimización en escenarios reales.	▪ Informe de Pruebas y Validaciones.

4.2.4. Marco lógico de la propuesta

Metas	Indicadores	Fuentes de verificación	Supuestos
Identificar áreas de mejora en la programación académica actual y definir los requerimientos específicos del nuevo sistema.	▪ Lista de áreas de mejora identificadas. ▪ Requerimientos específicos detallados.	Informe de Análisis de Procesos y Requerimientos.	Acceso a documentación y datos relevantes de la programación actual. Aceptación y aprobación del informe por parte de la dirección.
Diseñar la arquitectura de software del nuevo sistema e integrar algoritmos de optimización.	▪ Prototipo de la interfaz de usuario desarrollado. ▪ Diagramas de arquitectura creados.	Documento de Diseño de Arquitectura de Software.	Aprobación del diseño por parte del desarrollador y la dirección. Disponibilidad de recursos técnicos para el desarrollo del prototipo.
Desarrollar módulos esenciales del sistema e implementar algoritmos de optimización. Incorporar funcionalidades de gestión de cambios y ajustes.	▪ Módulos desarrollados y funcionales. ▪ Algoritmos de optimización implementados. ▪ Funcionalidades de gestión de cambios activas.	▪ Pruebas de funcionalidad realizadas. ▪ Retroalimentación de usuarios sobre los módulos desarrollados.	▪ Herramientas de desarrollo adecuadas. ▪ Colaboración continua de los usuarios finales para pruebas y Retroalimentación.

Metas	Indicadores	Fuentes de verificación	Supuestos
Implementar pruebas y validaciones que permitan evaluar el rendimiento del software y los algoritmos de optimización en escenarios reales.	▪ Informe de pruebas y validaciones completado. ▪ Resultados de pruebas unitarias y de integración satisfactorios y retroalimentación positiva sobre la usabilidad del sistema.	▪ Informe de Pruebas y Validaciones. ▪ Reportes de pruebas unitarias, de integración y de rendimiento.	▪ Disponibilidad de un entorno de pruebas adecuado. ▪ Recursos suficientes para realizar ajustes según los resultados de las pruebas.

Tabla 3: Marco lógico de la propuesta

5. Levantamiento de requerimientos y análisis de los procesos

En esta sección se presentan los requerimientos del sistema dividiendolos en funcionales y no funcionales. Además de identificar y analizar los procesos de la programación académica en la seccional buga.

5.1. Requisitos funcionales

El sistema debe ofrecer una experiencia integral orientada a la gestión académica y que reduzca la carga operativa de los usuarios y mejorar la eficiencia en la planificación horaria. Para lograrlo es necesario implementar las siguientes funcionalidades:

- **Autenticación y sesiones:** El sistema debe permitir iniciar y cerrar sesión mediante credenciales únicas (usuario y contraseña), garantizando el acceso seguro a la plataforma. El proceso debe seguir buenas práticas de seguridad y almacenamiento cifrado de contraseñas.

- **Gestión de roles:** El sistema debe contar con un control de acceso basado en roles para acceder al sistema, con los siguientes perfiles definidos:
  Administrador: Accede a todas las funcionalidades del sistemas y gestiona todos los datos de entrada como lo son las asignaturas, las aulas de clase, la información de los docentes, la disponibilidad de los docentes, el periodo acádemico, los programas acádemicos, la sede y los horarios generados.
  Docente: Accede únicamente a la funcionalidad de consulta y registro de su disponibilidad horaria y puede visualizar el horario asignado.

- **CRUD (create, read, update, delete) sobre entidades principales:** Se desarrollarán interfaces y servicios para realizar operaciones CRUD sobre las entidades clave del sistema. Esto incluye la creación, conculta, actualización y eliminación de los siguientes registros:
  Docentes: Almacena toda la información personal de los docentes, incluyendo nombre completo, número de identificación, correo electrónico y número de contacto.
  Asignaturas: Se almacena toda la información correspondiente con las asignaturas, incluyendo el nombre, código, intensidad horaria, nombre del docente asignado, el aula correspondiente y el programa acádemico al que pertenece.
  Disponibilidad: Registra la disponibilidad horaria que tiene cada docente.
  Sedes: En este registro se almacenará el nombre de la sede, por ejemplo, sede principal.
  Aulas: Incluye información del tipo de aula (laboratorio, general, etc.), y la capacidad.
  Programas: Registra los programas acádemicos ofrecidos por la seccional.

Periodo académico: Aquí solo se almacena el periodo acádemico, como por ejemplo 2025-1.

- **Generación de horarios:** Desde la interfaz de administración, se dispondrá de una funcionalidad para ejecutar el modelo de optimización. Cuando se ejecute el modelo creado este devolverá una solución factible u óptima para el conjunto de datos de entrada (docentes, asignaturas, disponibilidad, aulas, etc.).
- **Visualización de resultados:** El sistema debe permitir la visualización del horario generado mediante una interfaz tabular estructurada. Esta vista debe mostrar, de forma clara y ordenada, la distribución horaria por asignatura, aula, docente y día.
- **Exportación:** El usuario podrá exportar el horario generado como una hoja de cálculo. La exportación debe conservar la estructura tabular.

5.2. Requisitos no funcionales

- **Rendimiento:** El sistema ofrecerá tiempos de respuesta adecuados al momento de generar horarios con gran volumen de datos de entrada y también con varios usuarios activos.
- **Escalabilidad:** La arquitectura del sistema debe estar diseñada para adaptarse al crecimiento en la cantidad de entidades (docentes, asignaturas, programas, sedes, aulas, etc.), datos de entrada y número de usuarios sin degradar el rendimiento general.
- **Seguridad:** Se implementará un sistema de autenticación robusto mediante cifrado y control de acceso basado en roles.
- **Mantenibilidad:** El sistema debe estar diseñado con buenas prácticas de programación para faciliar el mantenimiento y la incorporación de nueva funcionalidades.
- **Fiabilidad:** el sistemas debe garantizar la integridad de los datos generados, incluso si hay un fallo el proceso pueda reanudarse.
- **Compatibilidad:** La aplicación web debe ser compatible con la mayoria de navegadores, asegurando que los usuarios puedan utilizarla independientemente de qué navegador usen o sistema operativo.
- **Usabilidad:** la interfaz debe ser intuitiva, permitiendo a usuarios ingresar información de manera sencilla, al igual que ejecutar y revisar el resultado final.
- **Disponibilidad:** el sistema debe estar disponible en todo momento para los usuarios autorizados.

5.3. Análisis de los procesos e identificación de mejoras

Para este análisis se identifican los involucrados, herramientas, criterios, restricciones y desafíos que conlleva los procesos de la programación académica.

Metodología
Para comprender en profuncidad estos procesos, se realizaron entrevistas (en el anexo A se muestra la evidencia de las preguntas realizadas en un documento compartido) a los principales involucrados: Secretaría académica, coordinadores académicos y monitores. A partir de dichas entrevistas se identificaron los tiempos, los pasos, condiciones y herramientas bajo las cuales se realiza la programación, así como los principales retos que enfrentan con la ejecución manual del proceso.

5.3.1. Roles y actores involucrados

- **Secretaría académica**: Remite las plantillas en una hoja de cálculo (anualizada y semestralizada), además que consolidad la información de todos los programas, valida la versión final de la programación y publica los horarios.
- **Coordinadores académicos**: Reciben las plantillas y se encargan de diligenciar la información correspondiente a su programa.
- **Monitores**: Apoyan en la digitalización de la programación, en realizar ajustes y en la verificación de cruces de horarios.
- **Docentes**: Proveen su disponibilidad horaria.
- **Recursos tecnológicos**: Es el área encargada de la asignación de aulas.

5.3.2. Datos de entrada

Para llevar a cabo la programación, se requieren los siguientes insumos:

- **Disponibilidad horaria de los docentes**: Incluye horarios base ya registrados y confirmación actualizada con cada docente.
- **Listado de asignaturas por cohorte**: Contiene el nombre de la asignatura, código e intensidad horaria.
- **Maya curricular y proyección por cohorte**: Permite asegurar que los cursos se programen en el semestre correspondiente.
- **Tipo de salón y capacidad**: Considera el tipo de aula como salas de sistemas, laboratorios y salones generales.
- **Restricciones operativas**: Se debe tener en cuenta las restricciones para la programación las cuales son: como máximo 4 horas continuas por bloque, cursos de 5 horas divididos en dos días, los días festivos se tratan como días normales y un docente no puede tener más de 20 horas semanales.

5.3.3. Descripción del proceso

1. **Recepción de plantillas**
   Secretaría académica envía dos documentos de excel por cada programa académico, uno es la programación anualizada y el otro es la semestralizada. La programación anualizada es la proyección de lo que se va a programar todo el año siguiente, que contiene la categoría de los profesores, la cantidad de horas por semana y por semestre y la asignatura, el tipo de vinculación, el semestre académico y la jornada.
   El otro documento que se envía a cada dependencia es para realizar la programación semestralizada, esta se realiza en base a la programación anualizada y es donde se programan los horarios de clase de manera manual. Este documento contiene las asignaturas por periodo junto con la intensidad horaria, el docente asignado, el día y la hora.

2. **Carga de información en una hoja de cálculo por programa académico**
   Cada programa académico recibe las plantillas y procede a diligenciar la información según el semestre correspondiente. La digitación de los datos incluye:

   - Listado de asignaturas por cohorte.
   - Intensidad horaria de cada asignatura.
   - Categoría y tipo de vinculación de los docentes asignados.

   Una vez completadas las plantillas, estas se devuelven a secretaría académica para su consolidación. En caso de detectar cruces de horario u otras inconsistencias, los documentos son reenviados al programa respectivo para su correción.

3. **Confirmación de disponibilidad docente**
   Cada programa académico consulta la disponibilidad horaria de cada docente mediante diversos canales de comunicación, como correo electrónico, llamadas telefónicas o mensajería instantanea (por ejemplo, Whatsapp). Aunque muchos docentes cuentan ya con un horario base predefinido, se realiza una confirmación constante para evitar posibles conflitos.

4. **Ajustes y validaciones**
   Los ajustes se realizan cada vez que secretaría académica encuentra inconsistencias o cruces en la programación. Los ajustes se realizan manualmente teniendo en cuenta las observaciones hechas.
   Despué de realizar los ajustes necesarios, secretaría académica valida la versión final de la programación.

5. **Asignación de aulas**
   Una vez que la programación académica fue aprovada el siguiente paso es la asignación de aulas, de esta se encarga el área de recursos tecnológicos de la seccional teniendo en cuenta la cantidad de estudiantes, el tipo de aula ya sea laboratorio o general y la capacidad.

6. **Publicación de los horarios**
   Una vez aprobados los horarios, estos se cargan en el sistema SIRA (sistema de información de admiciones y registro acádemico), permitiendo su consulta por parte de los estudiantes y docentes.

5.3.4. Principales desafíos

El proceso presenta diversas limitaciones, entre las cuales se destacan:

- **Fragmentación y gestión manual**: Todo el procedimiento se realiza de forma manual mediante mediante hojas de cálculo, lo que incrementa el riesgo de errores de transcripción y obliga a realizar reajustes de manera frecuenta. Además, el proceso involucra tres dependencias diferentes, esto hace que el proceso se vuelva más lento y extenso.
- **Falta de centralización**: No existe una base de datos unificada de disponibilidad de docentes; la información se recopila por múltiples canales (email, wharsapp, llamadas), dificultando su gestión.
- **Retrasos en la programación**: El proceso manual implica demoras significativas para tener la programación lista, especialmente debido a los cruces de horarios que requieren correcciones frecuentes.
- **Duración del proceso**: El proceso completo desde la entraga de las plantillas hasta la publicación de los horarios tiene una duración aproximada de dos meses. Durante este tiempo, pueden realizarse hasta cuatro ciclos de reajustes antes de llegar a una versión definitiva.

5.3.5. Requerimientos para una solución mejorada

Con base en este análisis, se identifican los siguientes requerimientos para optimizar el proceso:

- **Base de datos centralizada**: Un sistema único donde se registre la disponibilidad horaria de los docentes, junto con sus datos personales, asignaturas asociadas, sedes, aulas y programas académicos.
- **Generador semiautomático de horarios**: Un algoritmo que sea capaz de proponer horarios iniciales teniendo en cuenta los datos de entrada y también validando la disponibilidad del docente, las restricciones de franjas horarias y no exceder la carga máxima de horas por docente.
- **Interfaz web de gestión**: Una plataforma amigable que permita el ingreso y edición de los datos de entrada de forma eficiente y centralizada.

Conclusiones
El análisis evidenció que el proceso presenta deficiencias en automatización, centralización

de datos y validación de restricciones. Esto se traduce en tiempo significativos de ejecución (hasta 2 meses), múltiples ciclos de reajustes (entre 2 y 4), y una gestión poco eficiente de la disponibilidad docente y la asignación de aulas. La implementación de herramientas tecnológicas permitirá reducir estos tiempos, mejorar la precisión del proceso y facilitar la toma de decisiones administrativas.

6. Diseño de la arquitectura de software, algoritmos de optimización e interfaz de usuario

6.1. Arquitectura

Se diseñó la arquitectura del sistema y se definieron los componentes principales. También se diseñó la interfaz de usuario considerando la integración con el algoritmo de optimización y la facilidad de uso para los administradores. El diseño de la interfaz de usuario es esencial porque representa el punto de contacto entre el usuario y el sistema. Una buena interfaz mejora la experienca del usuario. En esta sección se mostrarán los wireframes iniciales y en los anexos B se mostraran la interfaz completa.

Para integrar un algoritmo de optimización robusto y ofrecer una interfaz de usuario se realizó una arquitectura en capas que reutilice y extienda la base del frontend en React y el backend en FastAPI, esta arquitectura se compone de los siguientes elementos principales:

Frontend
En las responsabilidades del frontend tenemos:

- interfaz de administración para gestionar los datos de entrada (docentes, disponibilidad, asignaturas, sedes, programas académicos y periodos)
- ejecutar la corrida del algoritmo de optimización y mostrar el resultado en forma de tabla de horarios y también exportar a excel.

Además de gestionar y llamar a la API REST de FastAPI para autenticación, gestión de docentes, asignaturas, disponibilidad y gestión de horarios.

Backend
En las responsabilidades del backend tenemos:

- Exponer endpoints para catálogos maestros (docentes, disponibilidad, asignaturas, sedes, programas y periodos)
- Gestionar la disponibilidad del docente (lectura y escritura de franjas libres)
- Definir y almacenar los parámetros globales de optimización (horas máximas, tipos de bloques permitidos)
- Recibir la solicitud de `'ejecutar optimización'` para poner en marcha el algoritmo de optimización.

En el Anexo C se pueden evidenciar todos los endpoints del backend

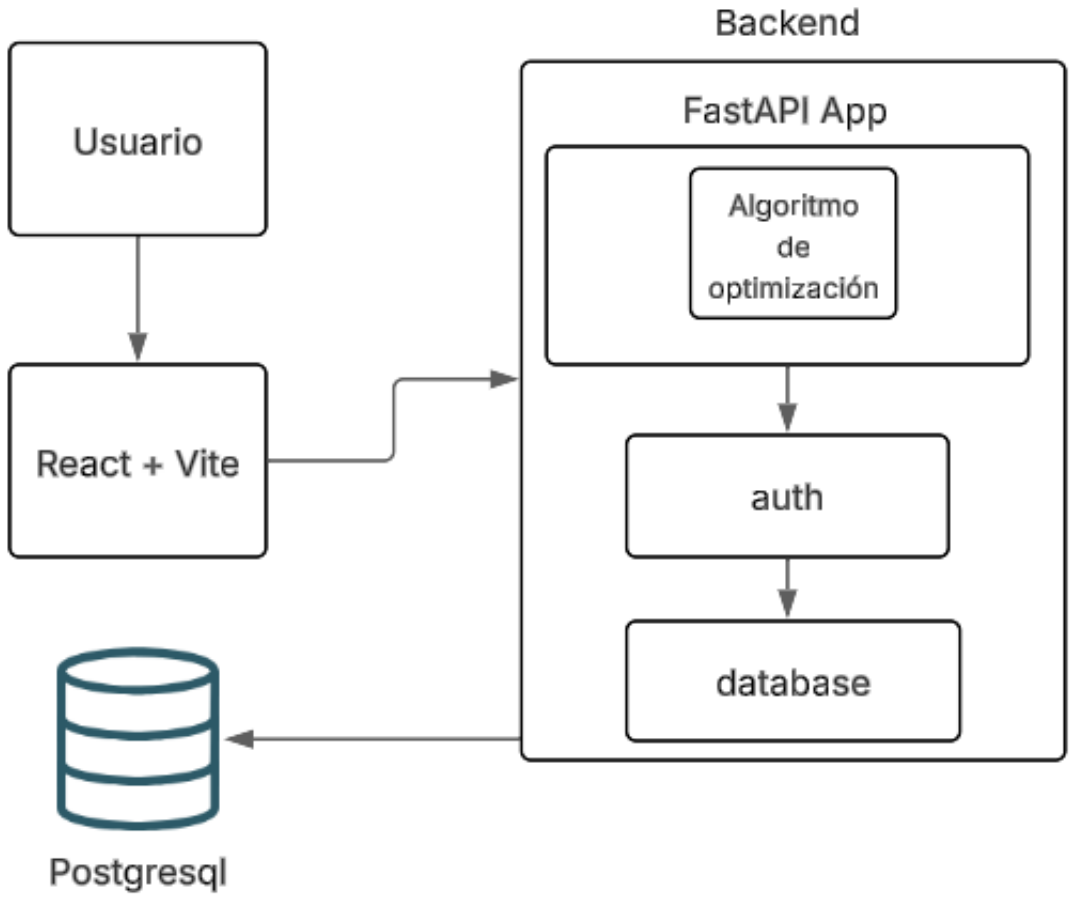

Figura 2: Esquema general de la arquitectura de la aplicación

Algoritmo de optimización
En las responsabilidades del algoritmo están:

- Manipular desde la base de datos todos los datos requeridos como las asignaturas y la disponibilidad docente para así mostrar una solución.

Base de datos relacional
Se utiliza PostgresSQL como base de datos relacional para este proyecto.

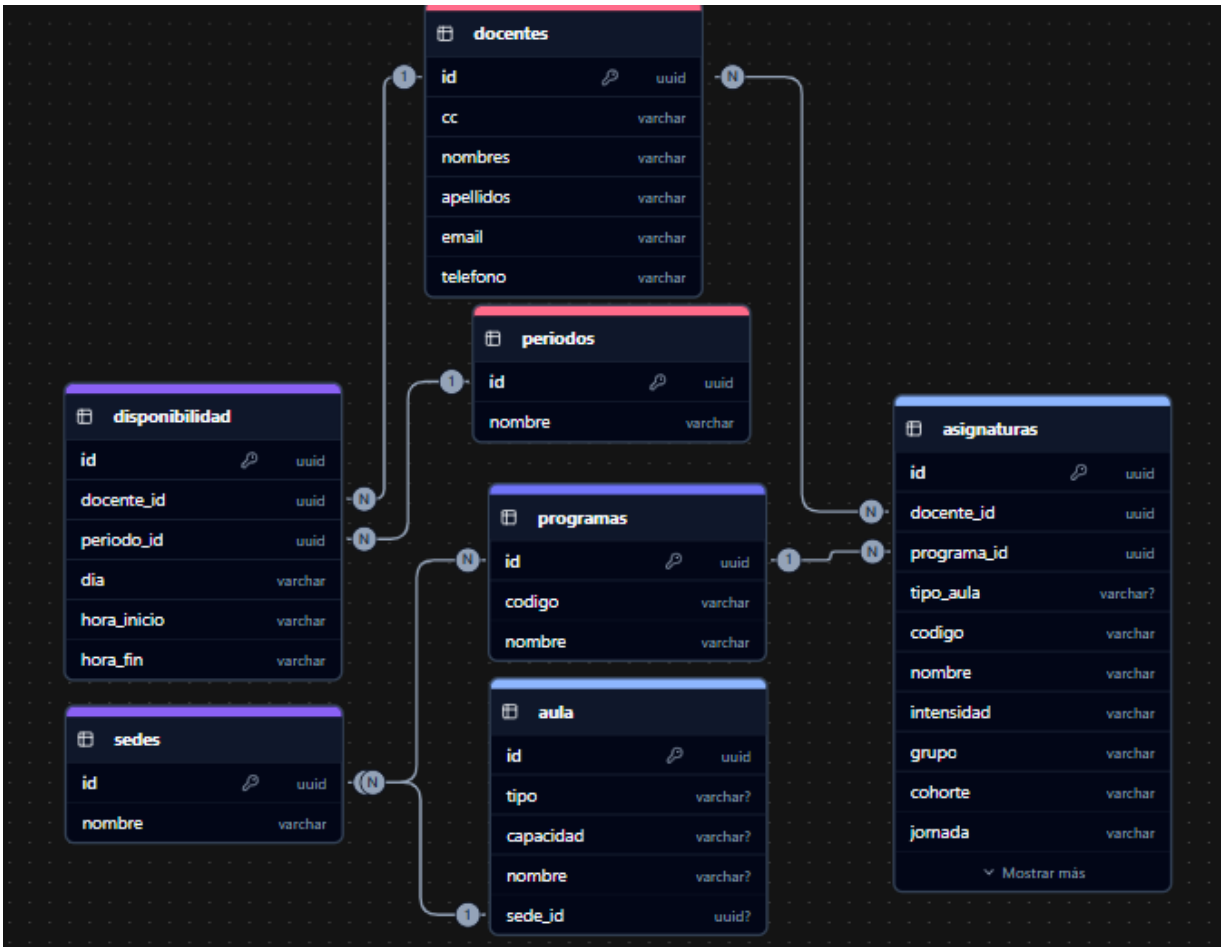

Figura 3: Modelo relacional

Este modelo relacional representa la base de datos para el software de programación de horarios.

Entidades principales:

Docentes: información personal del docente (nombres, apellidos, email, teléfono, cc).

Disponibilidad: registra los días y horas en que un docente está disponible. También se relaciona con docentes y periodos

Periodos: Define los distintos periodos académicos (semestre)

sedes: Lista de sedes disponibles

Programas: Información de los programas académicos

Asignaturas: Detalles de las asignaturas Relacionada con docentes y programas

Aulas: Contiene las aulas de clase.

6.1.1. Diseño de interfaces de usuario

Wireframes

Los wireframes muestran el diseño inicial del aplicativo web.

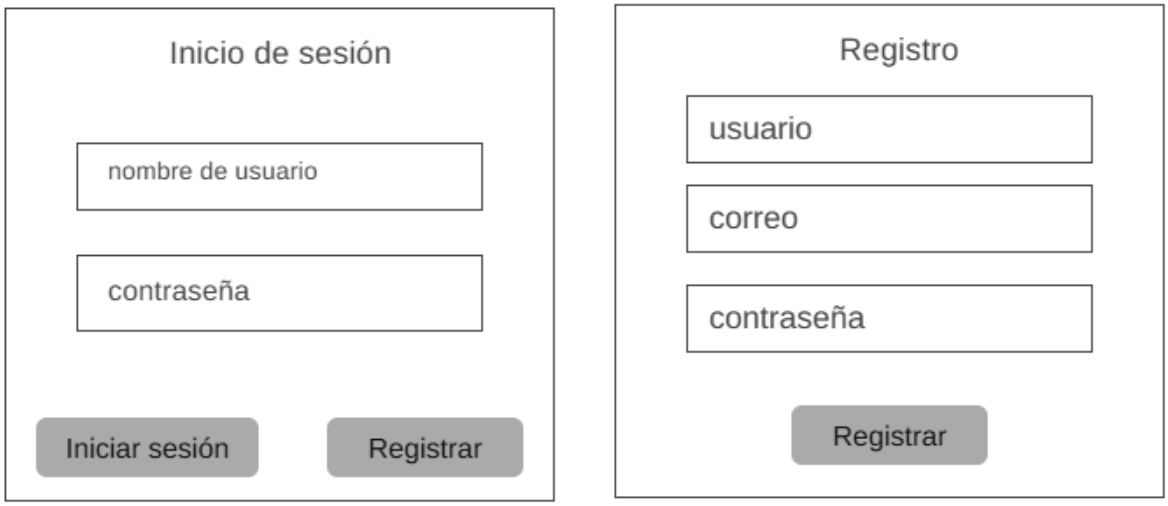

Figura 4: Wireframe del inicio de sesión y registro

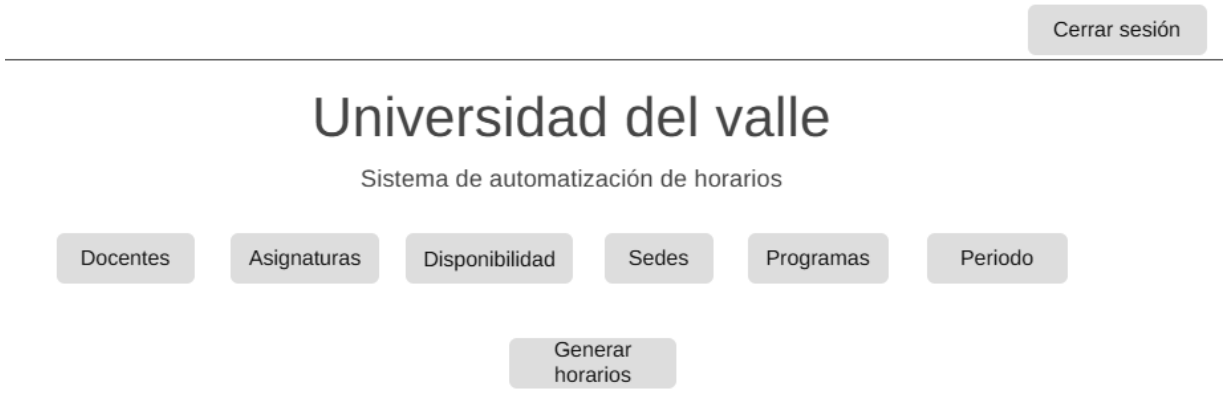

Figura 5: Wireframe de la página principal

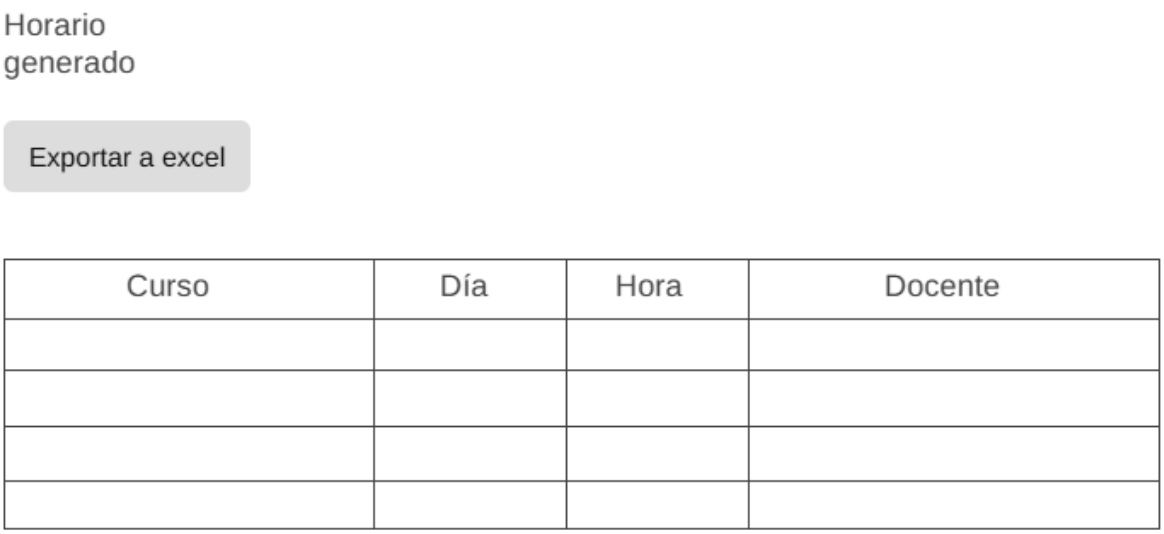

Figura 6: Wireframe de como se vería el horario generado

7. Implementación

Módulos

En el backend (FastAPI) se utilizaron los siguientes modelos y esquemas: docente, asignatura, sede, programa, periodo y disponibilidad.

En el frontend (React) al igual que en el back se utilizaron los siguientes componentes principales: docente, asignatura, sede, programa, periodo y disponibilidad.

Módulo de optimización

Contiene la lógica de construcción del módelo de optimización que permite traducir la disponibilidad de docentes y bloques permitidos.
El tipo de solución que se usa es un modelo de programación por restriciones resuelto con el solver CP-SAT de OR-tools, en este caso se emplea para encontrar cualquier solución factible que cumpla todas las restriciones.

Restricciones del algortimo de optimización

1. **Cobertura de intensidad (carga horaria)**: Para cada curso, la suma total de horas asignadas a cada curso debe ser igual a su intensidad.
2. **Un solo bloque por día y curso**: No asignar más de un bloque por día a la misma asignatura, es decir, que un curso no se repita más de una vez en el mismo día.
3. **Estructura por intensidad**: Las asignaturas con intensidad menor o igual a 4 horas deben asignarse en un solo día y las que son mayores o iguales a 5 deben dividirse en dos días.
4. **No solapamiento de docentes**: Un mismo profesor no puede impartir dos clases simultáneamente.
5. **Exclusión de la franja de almuerzo**: No se programa clases en el intervalo 12:00-13:00.
6. **Asignación única de aula por bloque**: Asignar solo una aula de clase para cada bloque.
7. **No solapamiento en el aula**: Un mismo salón no puede albergar dos clases al mismo tiempo.
8. **Límite de carga docente**: Un docente no puede tener más de 20 horas asignadas.

Definiciones

$\mathcal{C}$ Conjunto de cursos (c)

$\mathcal{D}$ Conjunto de días lectivos $\{\text{Lu}, \text{Ma}, \text{Mi}, \text{Ju}, \text{Vi}\}$

$\mathcal{S}$ Conjunto de *slots* de 30 min

$\mathcal{A}$ Conjunto de aulas (a)

$\mathcal{B}_{cd}$ Bloques factibles del curso c en el día d

L_{cdb} Conjunto de slots del bloque $b \in \mathcal{B}_{cd}$

q_c Número de slots requeridos por el curso c

teach(c) Docente que imparte el curso c

A_{pd} Slots donde el docente p está disponible el día d

$H_{\text{máx}}$ Máximo de slots semanales por docente (40)

L^{lunch} Slots prohibidos (12:00–13:00)

Variables de decisión

$$x_{cdb} = \begin{cases} 1, & \text{si el curso } c \text{ se dicta en el bloque } b \text{ del día } d, \\ 0, & \text{en otro caso,} \end{cases}$$

$$y_{cdba} = \begin{cases} 1, & \text{si el bloque } (c, d, b) \text{ se imparte en el aula } a, \\ 0, & \text{en otro caso.} \end{cases}$$

Modelos mátematicos

Cobertura de intensidad
$$\sum_{d\in\mathcal{D}}\sum_{b\in\mathcal{B}_{cd}} |L_{cdb}|\, x_{cdb} = q_c \qquad \forall c \in \mathcal{C} \tag{1}$$

Máx. un bloque por día
$$\sum_{b\in\mathcal{B}_{cd}} x_{cdb} \le 1 \qquad \forall c \in \mathcal{C},\ d \in \mathcal{D} \tag{2}$$

Cursos ≤ 4 h en un día
$$\sum_{d\in\mathcal{D}} \min\Big(1, \sum_{b} x_{cdb}\Big) = 1 \qquad \forall c:\ q_c \le 8 \tag{3}$$

No solape de docente
$$\sum_{c:\mathrm{teach}(c)=p}\ \sum_{\substack{b\in\mathcal{B}_{cd}\\ s\in L_{cdb}}} x_{cdb} \le 1 \qquad \forall p,\ d \in \mathcal{D},\ s \in \mathcal{S} \tag{4}$$

Almuerzo (12–13 h)
$$x_{cdb} = 0 \qquad \forall (c,d,b):\ L_{cdb} \cap L^{\mathrm{lunch}} \neq \varnothing \tag{5}$$

Asignación única de aula
$$\sum_{a\in\mathcal{A}} y_{cdba} = x_{cdb} \qquad \forall c,d,b \tag{6}$$

No solape en el aula
$$\sum_{(c,b):\, s\in L_{cdb}} y_{cdba} \le 1 \qquad \forall a \in \mathcal{A},\ d \in \mathcal{D},\ s \in \mathcal{S} \tag{7}$$

Máx. 20 h por docente
$$\sum_{c:\mathrm{teach}(c)=p}\ \sum_{d\in\mathcal{D}}\sum_{b\in\mathcal{B}_{cd}} |L_{cdb}|\, x_{cdb} \le H_{\max} \qquad \forall p \tag{8}$$

$$x_{cdb},\ y_{cdba} \in \{0,1\}. \tag{9}$$

$$\begin{aligned}\min\ \mathrm{OBJ} \;=\; & \underbrace{\alpha \sum_{c}\sum_{d}\big(\delta_{cd}\big)}_{\text{compactar días por curso}} \;+\; \underbrace{\beta \sum_{c}\sum_{d}\sum_{b}\ \sum_{s\in L_{cdb}\cap \mathcal{S}^{\mathrm{late}}} x_{cdb}}_{\text{evitar slots tardíos}} \\ & +\; \underbrace{\gamma \sum_{c}\sum_{d}\sum_{b}\ \sum_{a\in\mathcal{A}_c} U_{ca}\, y_{cdba}}_{\text{ajustar tamaño de aula (menos holgura)}} \;+\; \underbrace{\delta \sum_{c}\Big(\sum_{a\in\mathcal{A}_c} r_{ca} - 1\Big)}_{\text{consistencia de aula por curso}} .\end{aligned} \tag{10}$$

7.1. pseudocodigo

Algoritmo 1: ScheduleGenerator

Entrada: *teacher_availabilities*,;
courses,;
classrooms
Salida: *schedule*

Función *ValidarTeacherAvailability()*:
  para cada *docente p* **hacer**
    si *suma_intensidades*(*p*) > *horas_disponibles*(*p*) **entonces**
      devolver falso;
  devolver verdadero;
Función *ParseAvailability()*:
  Convertir cada rango `HH:MM-HH:MM` → índices de slots;
Función *GeneratePossibleTimeBlocks*(*c*, *d*):
  devolver lista de bloques consecutivos compatibles con la disponibilidad del docente de *c*;
si *ValidarTeacherAvailability()* = ***falso*** **entonces**
  devolver error "Disponibilidad insuficiente";
ParseAvailability();
`/* Construir modelo CP-SAT */`
Inicializar modelo M;
para cada *curso c* **hacer**
  para cada *día d* **hacer**
    bloques ← GeneratePossibleTimeBlocks(c, d);
    para cada *bloque b* ∈ *bloques* **hacer**
      Crear variable binaria x_{cdb};
      para cada *aula a compatible con c* **hacer**
        Crear variable binaria y_{cdba};
restricciones #1–#8 M;
`/* Resolver modelo */`
estado ← CpSolver.Solve(M);
si *estado* ∉ {*OPTIMAL, FEASIBLE*} **entonces**
  devolver "No se encontró solución viable";
`/* Extraer solución */`
para cada $x_{cdb} = 1$ **hacer**
  Determinar aula a con $y_{cdba} = 1$;
  Convertir L_{cdb} → rango "HH:MM–HH:MM";
  Insertar entrada en *schedule*;
devolver *schedule*;

7.2. Documentación de la API de programación de horarios

1. **Contexto y alcance**
   Esta API expone recursos para administrar docentes, aulas, asignaturas, horarios, sedes, programas, disponibilidades y periodos académicos, además de un generador de horarios basado en restricciones. Está construida con FastAPI.

2. **Catálogo de endpoints**
   Generador

   - **POST** /generar-horarios: Ejecuta el generador y retorna el horario calculado como JSON.

   Docentes

   - **POST** /docentes: Crea un docente.
   - **GET** /docentes: Lista todos los docentes.
   - **GET** `/docentes/{docente_id}`: Obtiene un docente por ID.
   - **PUT** `/docentes/{docente_id}`: Actualiza un docente por ID.
   - **DELETE** `/docentes/{docente_id}`: Elimina un docente por ID.

   Sedes

   - **POST** /sedes: Crea una sede.
   - **GET** /sedes: Lista todas las sedes.
   - **GET** `/sedes/{sede_id}`: Obtiene una sede por ID.
   - **PUT** `/sedes/{sede_id}`: Actualiza una sede.
   - **DELETE** `/sedes/{sede_id}`: Elimina una sede.

   Programas

   - **POST** /programas: Crea un programa académico.
   - **GET** /programas: Lista todos los programas.
   - **GET** `/programas/{programa_id}`: Obtiene un programa
   - **PUT** `/programas/{programa_id}`: Actualiza un programa
   - **DELETE** `/programas/{programa_id}` Elimina un programa

   Asignaturas

   - **POST** /asignaturas: Crea una asignatura (permite asociar docentes, programas y aulas).
   - **GET** /asignaturas: Lista todas las asignaturas.
   - **GET** `/asignaturas/{asignatura_id}`: Obtiene una asignatura por ID.

- **PUT** `/asignaturas/{asignatura_id}`: Actualiza una asignatura (incluye docentes y aula).
- **DELETE** `/asignaturas/{asignatura_id}`: Elimina una asignatura.

Periodos

- **POST** /periodos: Crea un periodo académico.
- **GET** /periodos: Listo periodos.
- **GET** `/periodos/{periodo_id}`: Obtiene un periodo.
- **PUT** `/periodos/{periodo_id}`: Actualiza un periodo.
- **DELETE** `/periodos/{periodo_id}`: Elimina un periodo.

Disponibilidad

- **POST** /disponibilidades: Crea disponibilidad de un docente (permite asocias docentes y periodo).
- **GET** /disponibilidades: Lista disponibilidades.
- **GET** `/disponibilidades/{disponibilidad_id}`: Obtiene una disponibilidad.
- **PUT** `/disponibilidades/{disponibilidad_id}`: Actualiza disponibilidad.
- **DELETE** `/disponibilidades/{disponibilidad_id}`: Elimina disponibilidad.

Aulas

- **POST** /aulas: Crea un aula (permite asociar sedes).
- **GET** /aulas: Lista aulas.
- **GET** `/aulas/{aula_id}`: Obtiene un aula.
- **PUT** `/aulas/{aula_id}`: Actualiza un aula.
- **DELETE** `/aulas/{aula_id}`: Elimina un aula.

7.3. Modulos funcionales

- **Módulo de autenticación en el frontend**
  Contiene el flujo de autentiación que lo compone el registro, el inicio de sesion que envía las credenciales y si son válidas redirige a la aplicación, las sesiones de los usuarios y el cierre de sesión.

- **Módulos para el manejo de datos (CRUD)**
  Se crean los módulos para la creación, lectura, actualización y eliminación de los datos de entrada.
  Las rutas del sistema son:

- /: Home
- /**login** : Formulario de inicio de seción.
- /**register** : Formulario para el registro de usuarios.
- /**docentes**: Manejo de docentes.
- /**aulas**: Manejo de aulas.
- /**asignaturas**: Asignaturas.
- /**disponibilidades**: Disponibilidad.
- /**periodos**: Manejo de los periodos académicos.
- /**programas**: Registro de los programas.
- /**sedes**: Registro de las sedes.

7.4. Interfaz de usuario del aplicativo

Acontinuación se evidencia la interfaz de usuario del home y de cada uno de los modulos para registrar los datos de entrada.

Visual de la página de inicio donde se observan 7 botones que redireciona a otra página que corresponde a la entrada de datos.

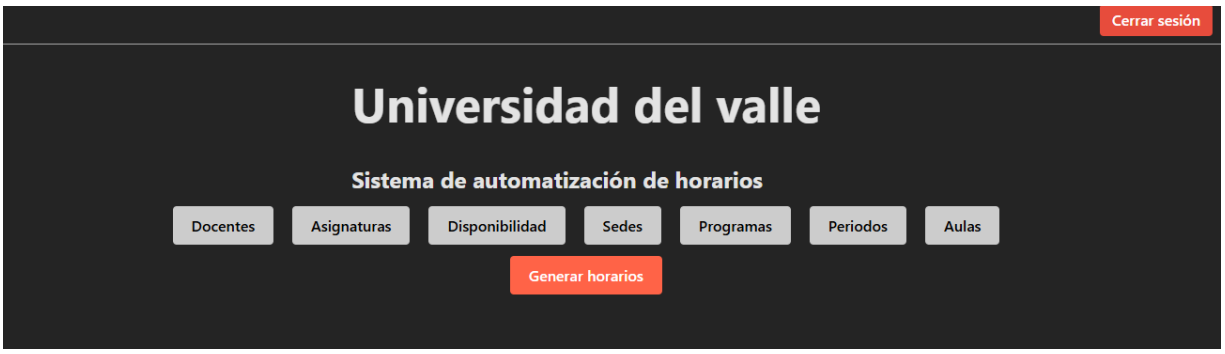

Figura 7: Home

En esta sección se registran los docentes con sus datos personales

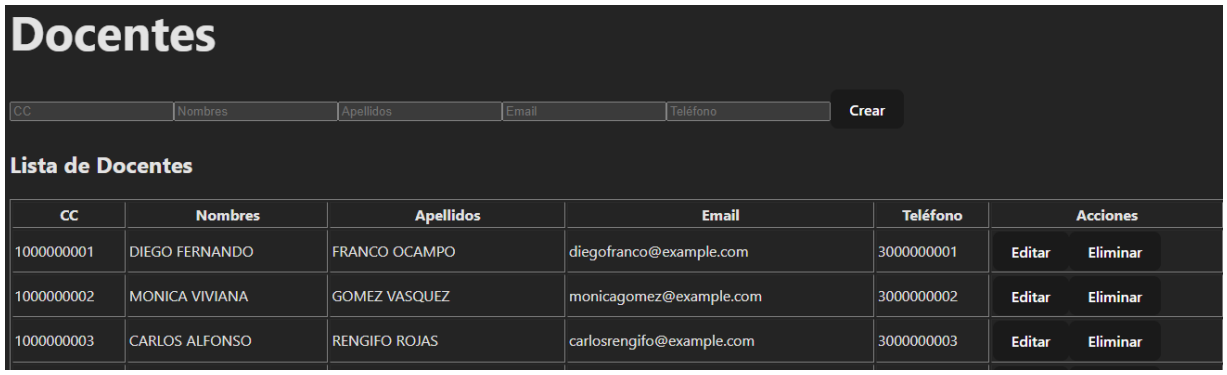

Figura 8: Registro de docentes

Aqui se registran las asignaturas con el docente y programa que se le vaya a asignar

Figura 9: Registro de asignaturas

En esta página se registra la disponibilidad horaria del docente, se pueden registrar uno o varios días.

Figura 10: Registro disponibilidad

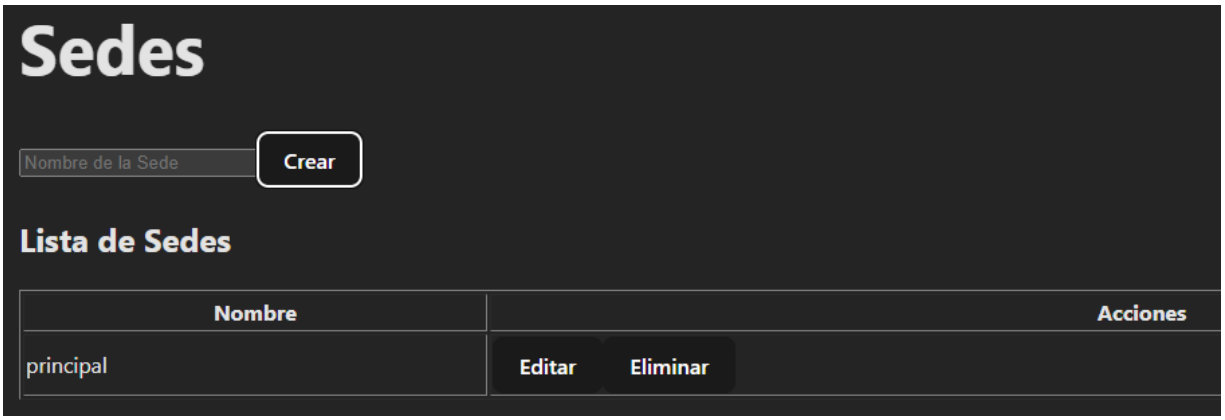

Figura 11: Registro de sedes

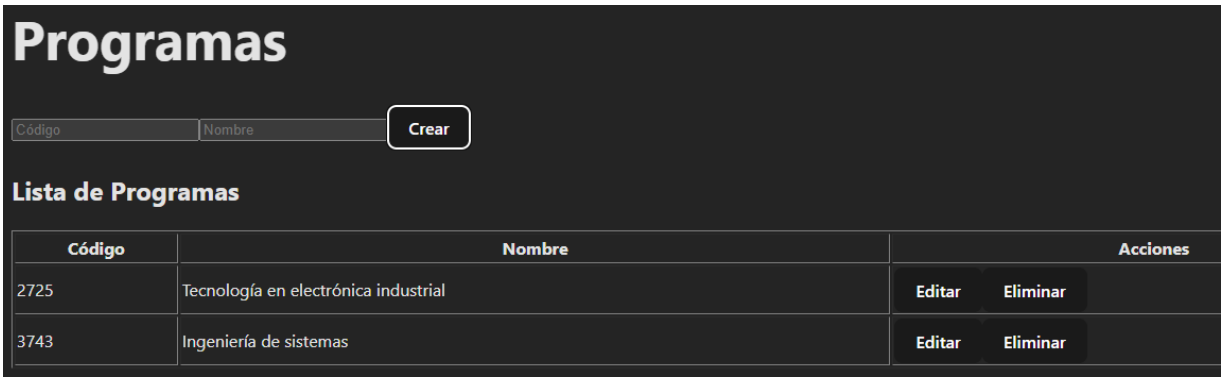

Figura 12: Registro de programas académicos

Figura 13: Aulas de clase

Figura 14: Registro de periodos académicos

Aquí podemos observar que al dar clic en el botón de generar horarios el algortimo tomará todas las variables de entrada y mostrará un horario generado en base a esto, donde se puede ver el curso, el día, la franja horaria y el docente asignado. También cuenta con la opción de exportar a excel.

Figura 15: Horario generado

8. Pruebas y validaciones

8.1. Pruebas unitarias en la autenticación

Este apartado se documenta las pruebas unitarias construidas para los componentes de interfaz `LoginForm` y `RegisterForm`, así como para el módulo de servicios `authService`. Las pruebas validan el flujo de autenticación en escenarios de éxito y error, garantizando el correcto llamado a servicios, el manejo de estados en la interfaz y las redirecciones esperadas.

Configuración y herramientas

Tipo de pruebas: unitarias (componente y módulo).
Herramientas: Vitest (runner), React Testing Library (interacción de usuario), JSDOM (entorno DOM), `@testing-library/user-event` (eventos).
Aislamiento y limpieza:

- Se mockean los módulos `authService` y el hook `useNavigate` de `react-router-dom` para controlar respuestas y asertar redirecciones.
- Se limpia el estado de los mocks antes de cada caso de prueba.

Prueba unitaria: `LoginForm` — inicio de sesión exitoso

Objetivo. Verificar que, ante credenciales válidas, el componente invoca `authService.login` con los parámetros correctos y redirige a `/dashboard`.
Tipo de prueba. Unitaria (componente React aislado).
Herramientas utilizadas. Vitest, React Testing Library, JSDOM, `@testing-library/user-event`.

Entorno y aislamiento.

- Mock del módulo `../../services/authService` para controlar la resolución de `login`.
- Mock del hook `useNavigate` para verificar la navegación mediante un espía.

Datos simulados.

- Usuario: `demo`
- contraseña: `secret`.
- Respuesta del servicio: `{ access_token: 'fake' }`.

Flujo validado.

1. Render del componente `<LoginForm />` envuelto en `MemoryRouter`.
2. Escritura en los campos Usuario y Contraseña.

3. Activación del envío mediante click en el botón Login.
4. Aserciones:
    - `login('demo','secret')` es llamado exactamente una vez.
    - `navigate('/dashboard')` es invocado.

Resultados esperados.

- El servicio recibe las credenciales suministradas.
- Se ejecuta la redirección a `/dashboard`.
- No se muestra el mensaje de error.

Criterios de aceptación.

- Al menos una aserción sobre llamada al servicio y otra sobre la navegación.

Prueba unitaria: `LoginForm` — credenciales inválidas

Objetivo. Verificar que, ante un error de `authService.login`, el componente muestre el mensaje de error y **no** navegue.
Tipo de prueba. Unitaria (componente React).
Herramientas. Vitest, React Testing Library.
Flujo validado.

1. Render de `<LoginForm />` en `MemoryRouter`.
2. Completar campos de usuario y contraseña.
3. Enviar el formulario.
4. Aserciones:
    - El DOM contiene el texto Usuario o contraseña incorrectos.
    - La función `navigate` no se invoca.

Resultados esperados.

- Mensaje de error visible en el DOM.
- Cero redirecciones.

Riesgo mitigado. Garantiza que un fallo de autenticación no lleve al usuario a una vista protegida.

Prueba unitaria: `RegisterForm` — registro exitoso

Objetivo. Verificar que, con datos válidos, se invoque `authService.register` con `(username, email, password)`, se muestre el mensaje de éxito y se redirija a `/login`.
Tipo de prueba. Unitaria (componente React).
Herramientas. Vitest, React Testing Library.
Flujo validado.

1. Render de `<RegisterForm />` envuelto en `MemoryRouter`.
2. Rellenar Usuario, Correo y Contraseña.
3. Click en Registrarse.
4. Aserciones:
   - `register('nuevo','nuevo@correo.com','123456')` es llamado una vez.
   - El componente muestra ¡Registro exitoso! Redirigiendo al login....
   - `navigate('/login')` es invocado.

Resultados esperados.

- Se registra y se redirige al login.
- Mensaje de confirmación visible para el usuario.

Prueba unitaria: `RegisterForm` — error de backend (detail string)

Objetivo. Validar que, si el backend retorna un error con `response.data.detail` (cadena), el componente lo muestre y **no** navegue.
Tipo de prueba. Unitaria (componente React).
Herramientas. Vitest, React Testing Library.
Aislamiento.

- Mock de `register` que *rechaza* con `{ response: { data: { detail: 'El usuario ya existe' } } }`.
- Mock de `useNavigate` y verificación de no invocación.
- Limpieza de mocks entre casos.

Flujo validado.

1. Render de `<RegisterForm />`.
2. Completar campos de formulario.
3. Enviar.

4. Aserciones:
    - Se muestra El usuario ya existe en el DOM.
    - `navigate` no se invoca.

Resultados esperados.

- Mensaje de error visible.
- Sin redirecciones no deseadas.

Pruebas unitarias (servicio): `authService.login` y `authService.register`

Objetivo. Asegurar que las funciones de servicio construyen correctamente la solicitud HTTP y retornan la estructura esperada.
Tipo de prueba. Unitaria pura (módulo).
Herramientas. Vitest; mock de `axios`.

Caso a) `login`

Flujo validado.

1. `axios.post` resuelve `{ data: { access_token: 't0k3n' } }`.
2. Se llama `login('demo','secret')`.
3. Aserciones:
    - `axios.post` fue llamado con la URL de login y el payload esperado.
    - El valor retornado contiene `access_token: 't0k3n'`.

Caso b) `register`

Flujo validado.

1. `axios.post` resuelve `{ data: { id: 7 } }`.
2. Se llama `register('u','e@e.com','p')`.
3. Aserciones:
    - `axios.post` fue llamado con la URL de registro y el cuerpo `{ username, email, password }`.
    - El valor retornado incluye `id: 7`.

Resultados esperados.

- Las funciones devuelven `data` (no el *response* completo).

- `axios.post` se invoca una vez por llamada, con parámetros correctos.

En la siguiente imagen se muestra la ejecución de los test.

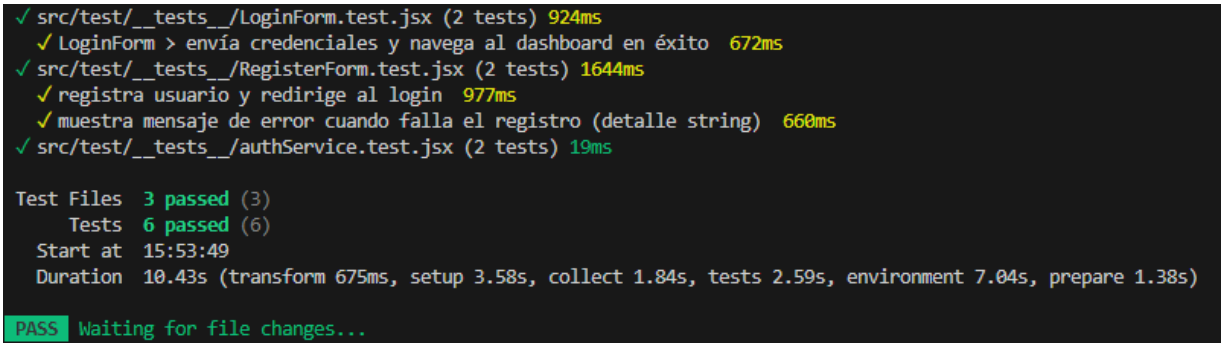

Figura 16: Test de autenticación

8.2. Pruebas unitarias en los módulos CRUD

En esta sección se documentan las pruebas unitarias desarrolladas para los componentes de gestión de datos: `Asignaturas`, `Aulas`, `Disponibilidad`, `Docentes`, `Periodos`, `Programas` y `Sedes`.
Todas las pruebas se ejecutan con Vitest y React Testing Library, utilizando JSDOM como entorno DOM simulado. Para aislar el comportamiento, se *mockea* el módulo de acceso HTTP `api.js` (métodos `get`, `post`, `put`, `delete`) y se limpia el estado de los mocks con `vi.clearAllMocks()` antes de cada caso.

Prueba unitaria: `AsignaturasCRUD` — carga inicial y mapeos

Objetivo: Verificar que, al montar el componente, se consultan los recursos (`asignaturas`, `programas`, `docentes`, `aulas`) y se renderiza la tabla aplicando los mapeos de nombres de programa y docentes.
Tipo: Unitaria (componente).
Herramientas: Vitest, React Testing Library.
Flujo validado:

1. Se simula la respuesta de `GET` para listas de asignaturas, programas, docentes y aulas.
2. Se renderiza el componente y se espera la presencia de la fila de ejemplo.
3. Se verifica que el nombre del programa y los nombres de los docentes se muestren correctamente.

Resultados esperados: La tabla contiene la asignatura cargada, con celdas de programa y docentes mapeadas por sus identificadores.

Prueba unitaria: `AsignaturasCRUD` — creación y edición

Objetivo: Asegurar que el formulario envía un `POST` con los campos mínimos y que, al editar, se ejecuta un `PUT` a la URL esperada.
Tipo: Unitaria.
Herramientas: Vitest, React Testing Library.

Flujo validado:

1. Se completan campos requeridos del formulario (código, nombre, programa, docente) y se envía.
2. Se aserta que `api.post` recibe el cuerpo con los valores digitados (incluyendo el arreglo de docentes).
3. Se selecciona una fila con Editar, se modifica el nombre y se envía.

Resultados esperados: `POST` y `PUT` invocados una sola vez cada uno, con los datos coherentes y la URL correcta.

Prueba unitaria: `AulaCRUD` — listado, creación y actualización

Objetivo: Validar el mapeo del nombre de la sede, la creación de un aula y la actualización con la convención de URL que incluye `/id/`.
Tipo: Unitaria.
Herramientas: Vitest, React Testing Library.

Flujo validado:

1. Se simula `GET` de aulas y de sedes y se comprueba el render del listado con el nombre de sede mapeado.
2. Se completa el formulario y se envía; se aserta `api.post` con el cuerpo esperado.
3. Se pulsa Editar, se modifica el nombre y se envía; se aserta `api.put` a `/aulas/{id}/`.

Resultados esperados: El mapeo de sede es correcto; `POST` y `PUT` son llamados con rutas y cuerpos válidos.

Prueba unitaria: `DisponibilidadCRUD` — render y persistencia

Objetivo: Verificar que la disponibilidad lista mapea el nombre del docente y del periodo, y que el flujo de creación y edición dispara `POST`/`PUT` con los campos (`docente_id`, `periodo_id`, día, horas).
Tipo: Unitaria.
Herramientas: Vitest, React Testing Library.

Flujo validado:

1. Se simula `GET` de disponibilidades, docentes y periodos y se verifica el render con mapeos.

2. Se completa el formulario con día y horas y se envía (`POST`).

3. Se edita la fila y se cambia el día, enviando (`PUT`) a `/disponibilidades/{id}`.

Resultados esperados: `POST`/`PUT` reciben los datos normalizados (cadenas en selects y horas `HH:MM`) y la URL coincide con la del componente.

Prueba unitaria: `DocentesCRUD` — creación y edición

Objetivo: Corroborar que el formulario de docentes persiste datos básicos y permite su edición.
Tipo: Unitaria.
Herramientas: Vitest, React Testing Library.

Flujo validado:

1. Se simula `GET` para listar docentes y se verifica el render.

2. Se completa el formulario y se envía (`POST`); se valida el cuerpo enviado.

3. Se edita una fila existente y se envía (`PUT` a `/docentes/{id}`) con la modificación.

Resultados esperados: Llamadas HTTP coherentes y mutación del estado de edición (Actualizar).

Prueba unitaria: `PeriodosCRUD` — creación y actualización

Objetivo: Asegurar la persistencia del nombre del periodo y su actualización.
Tipo: Unitaria.
Herramientas: Vitest, React Testing Library.

Flujo validado:

1. Se lista el periodo existente y se comprueba el render.

2. Se crea un nuevo periodo (`POST`) y se valida el cuerpo.

3. Se edita el existente (`PUT` a `/periodos/{id}`).

Resultados esperados: `POST`/`PUT` llamados una vez y con payload `{nombre}`.

Prueba unitaria: `ProgramasCRUD` — creación y edición

Objetivo: Validar la creación y edición de programas con código y nombre.
Tipo: Unitaria.
Herramientas: Vitest, React Testing Library.

Flujo validado:

1. Se renderiza el listado inicial (`GET`).

2. Se crea un programa (`POST`) con `código` y `nombre`.

3. Se edita un programa (`PUT` a `/programas/{id}`) cambiando el nombre.

Resultados esperados: Estructuras de payload correctas y rutas coherentes.

Prueba unitaria: `SedesCRUD` — creación, edición y eliminación

Objetivo: Corroborar que el módulo permite crear, editar y eliminar sedes.
Tipo: Unitaria.
Herramientas: Vitest, React Testing Library.
Flujo validado:

1. Se visualiza la sede existente tras el `GET` inicial.
2. Se crea una nueva sede (`POST`) y se valida el cuerpo.
3. Se edita una sede (`PUT` a `/sedes/{id}`) cambiando el nombre.
4. Se elimina la sede con `DELETE` a la URL del recurso.

Resultados esperados: Llamadas HTTP de `POST`, `PUT` y `DELETE` con parámetros y rutas esperadas; botón de acción cambia entre Crear/Actualizar según el estado.

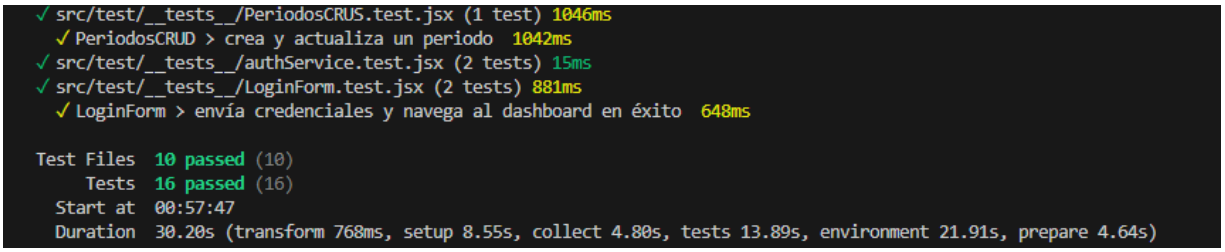

Figura 17: Test de los registros CRUD

Conclusión

Las pruebas unitarias descritas aseguran la correcta interacción de los formularios CRUD con la capa de servicios en el frontend, incluyendo carga inicial, mapeos de entidades relacionadas, y flujos de creación, edición y eliminación. El aislamiento mediante mocks de `api.js` y la limpieza de estado entre casos garantizan reproducibilidad y reducen el riesgo de regresiones en la interfaz de usuario.

A continuación se realizan pruebas de rendimiento al algoritmo de optimización.

variables de entrada
Para estas variables de entrada se tomará como ejemplo el programa de tecnología en electrónica diurno que constan de 23 docentes y 32 asignaturas.

Los siguientes son muestras de los datos, en el siguiente link se pueden observar los datos de entrada y de salida completos: https://pastebin.com/J4F1uZRB
Disponibilidad docente:

```
    teacher_availabilities_json = """
{
  "FRANCO OCAMPO DIEGO FERNANDO": {
        "Lunes": ["07:30-14:00"],
        "Martes": ["07:00-12:30"],
        "Miércoles": ["07:30-16:30"],
        "Jueves": ["14:00-18:00"]
    },
    "GOMEZ VASQUEZ MONICA VIVIANA": {
        "Lunes": ["15:30-20:00"],
        "Martes": ["12:30-18:00"],
        "Miércoles": ["08:00-12:30"],
        "Jueves": ["08:00-11:30"],
        "Viernes": ["08:00-12:30"]
    }
}
"""
```

Asignaturas:

```
    courses_json = """
{
  "Matemáticas básicas": {
  "teacher": "RENGIFO ROJAS CARLOS ALFONSO",
  "intensity": 3,
  "students": 60,
  "room_type": "general",
  "semester": 1},
  "Introducción a las tecnologías": {
  "teacher": "FRANCO OCAMPO DIEGO FERNANDO",
  "intensity": 2,
  "students": 40,
  "room_type": "general",
  "semester": 1}
```

```
}
"""
```

Salida:

```
Curso: Matemáticas básicas
  Día: Lunes, Hora: 09:00-12:00, Docente: RENGIFO ROJAS CARLOS ALFONSO, Aula: salón5,
Curso: Introducción a las tecnologías
  Día: Martes, Hora: 10:00-12:00, Docente: FRANCO OCAMPO DIEGO FERNANDO, Aula: salón1,
```

Se observa que se cumplen con las condiciones y restricciones dadas al algoritmo de optimización.

Para calcular y verificar el rendimiento del algoritmo se usó una función que toma tres variables de entrada, las cuales son disponibilidad, asignaturas y la cantidad de ejecuciones. Se guarda en una lista los tiempos de ejecución teniendo en cuenta el inicio y final, la duración final se toma restando el tiempo final con el inicial.

Se obtuvieron los siguientes resultados teniendo en cuenta que se está ejecutando en un entorno local con un computador que tiene las siguietens características: procesador intel gama baja de 2 núcleos de septima generación y una memoria RAM de 12gb.
Para una sola ejecución vemos que tiempo de respuesta es de apróximadamente 3 segundos.

```
Resultados de rendimiento (1 ejecuciones):
  Mínimo: 2.8340 s
  Promedio: 2.8340 s
  Máximo: 2.8340 s
```

Ahora para 100 ejecuciones vemos que el tiempo de respuesta es relativamente bajo.

```
Resultados de rendimiento (100 ejecuciones):
  Mínimo: 1.9910 s
  Promedio: 2.2882 s
  Máximo: 4.8530 s
```

A partir de los resultados obtenidos, se puede concluir que el rendimiento del sistema es eficiente, incluso ejecutandose en un entorno local y con un hardware que no es muy potente. Estos resultados sugieren que el algoritmo puede manejar múltiples ejecuciones sin degradación significativa.

9. Conclusiones

La implementación de un modelo de programación por restricciones integrado en una arquitectura cliente–servidor basada en React y FastAPI, consiguió convertir la asignación de horarios en un proceso coherente y redujo significativamente los cruces entre cursos y aulas, así como la necesidad de intervenciones manuales repetitivas. Al mantener las reglas estructuradas y constantes, incluso frente a cambios en la disponibilidad de docentes, aulas y recursos, el sistema puede generar el horarios completo sin requerir reconstrucciones desde cero. Esto se traduce a una gestión operativa más eficiente y a la disminución de errores.

Se logró transformar la programación de horarios en un proceso ágil y confiable, el sistema genera el horario de un programa académico en menos de cinco segundos y puede completar la planificación de todos los programas en menos de un minuto, lo que supone una reducción superior al 90 % frente a un ciclo manual que se extendía cerca de dos meses. Esto fue posible al trasladar las reglas y restricciones institucionales a un esquema formal de decisión y emplear un procedimiento de resolución que evalúa de manera simultánea múltiples combinaciones, descartando de forma sistemática aquellas que provocarían cruces de horarios o sobrecargas de trabajo. En términos prácticos, esto significa menos iteraciones y ciclos de trabajo, mayor consistencia en el cumplimiento de criterios, capacidad de reajustar rápidamente ante cambios y una mejora en el proceso, liberando tiempo a las areas encargadas de la programación para realizar solo validaciones en lugar de tareas repetitivas de edición.

La adopción de una metodología ágil basada en Kanban, implementada a través de la herramienta Trello, permitió consolidar una gestión de proyecto más estructurada, adaptable y centrada en el cumplimiento de los objetivos funcionales del sistemas. Esto se logró al organizar visualmente las tareas en un flujo compuesto por columnas como "por hacer", 'en ejecución' y "hecho", lo que facilitó la priorización dinámica de actividades según su complejidad. El beneficio de este enfoque fue una ejecución más eficiente, permitiendo incorporar ajustes técnicos y funcionales.

La solución demostró solidez, capaz de generar horarios académicos ante escenarios complejos y diversos. Eso fue posible gracias a una correcta interpretación de las restricciones y reglas que maneja la univerdidad, como las franjas horarias, la disponibilidad docente, el tipo de aula y su capacidad. Además, la integración de estas restricciones a un modelo de optimización permite evaluar simultaneamente múltiples criterios en cada asignación. La incorporación de este modelo dentro de una arquitectura escalable permite no solo resolver de forma sistemática los casos reales de asignación, si no también la posibilidad de extender el sistema con nuevas variables, restricciones o funcionalidades sin comprometer su estabilidad.

En resumen, el proyecto no solo implementa una solución tecnológica funcional, sino que representa un avance significativo en la transformación digital de la gestión académica, al sustituir un proceso tradicionalmente manual, extenso y vulnerable a errores, por un siste-

ma automatizado, ágil y adaptable. Al integrar modelos de programación por restricciones se logró optimizar de forma consistente la asignación de horarios y reducir drásticamente los tiempos operativos. Esto no solo mejora la eficiencia, sino que redefine la manera en que se planifica y organiza uno de los procesos más críticos del calendario académico.

10. Trabajos futuros

Algunas opciones de mejora que puede tener el sistema de programación son:

- **Integración con inteligencia artificial**: La integración de una IA al sistema permitirá alimentar mejos al optimizador, explorar el espacio de soluciones de forma más inteligente, reaccionar a imprevistos con mínimo impacto, comunicar con claridad y aprender del ciclo completo.
- **Ampliación de restricciones y preferencias**: Modelar ventanas "preferidas/evitadas"para incluir preferencias horarias y restricciones de recursos (aulas, aforo, equipos especiales), topes de horas consecutivas, tiempos de traslado entre edificios o sedes y criterios de equidad para distribuir de forma balanceada las franjas poco deseadas.
- **Aplicación móvil**: Desarrollar una versión móvil para Android o iOS que permita realizar la programación académica.
- **Escalabilidad institucional**: Extender de la seccional buga al resto de seccionales, con calendarios, husos horarios, feriados y políticas propias.
- **Interfaz de usuario avanzada**: Añadir funcionalidades de 'Drag and Drop' para mover bloques de horarios y validar al instante, resaltando conflictos en cuanto el usuario suelte un bloque y sugiriendo automáticamente ubicaciones válidas cercanas. Además de vistas múltiples por programa, docente, aula, grupo con filtros y búsqueda avanzada.
- **Calendarios y distribución**: Añadir un calendario donde se pueda visualizar el día donde está asignada cada asignatura y que también se pueda descargar. Además, un control de versiones tipo 'historial de horarios' con etiquetas y comparación de cambios, permitiendo revertir a estados anteriores.
- **Reprogramación dinamica en tiempo real**: Implementar mecanismos para generar bloques nuevos automáticamente ante cambios imprevistos (ausencias de docentes, cambios de aula, etc.)
- **Notificaciones y comunicación**: Integrar notificaciones por correo electrónico o mensajería instantanea para informar a los docentes y estudiantes de ajustes en la programación. Además que el sistema cuente con la opción de solicitar la información de disponibilidad horaria a los docentes mediante correos electrónicos masivos.
- **Compatibilidad con sistemas institucionales**: Desarrollar una integración con el sistema de información académica (SIRA) para facilitar la toma de desiciones y los datos de entrada que requiere el sistema de programación.
- **Desplegar en la nube**: Desplegar el proyecto en una infraestructura en la nube.

Estos trabajos futuros permitirán evolucionar el sistema hacia una solución más robusta y alineada a las necesidades de la comunidad académica.

11. Anexos

Los siguientes anexos contienen la documentación visual, y evidencia de los entregables.

Anexo A:
link del documento que contiene las preguntas hechas en la entrevista:
`https://docs.google.com/document/d/1Fq5U3LpE5ReVcv_EZyEf44_uirWexONlVc-zHNVmqfM/edit?usp=sharing`

Anexo B:
Repositorios al código fuente:
Repositorio frontend: `https://github.com/david-arce/front-horarios`

Repositorio backend: `https://github.com/david-arce/back-horarios`

Anexo C: Enlace a las capturas de pantalla del swagger dondes están los endpoints:
`https://docs.google.com/document/d/1i3WzpaHNf7ZGsC73G20Gu66P8kvPnnxDr1YqN32hrN0/edit?usp=sharing`

Anexo D: Enlace a las capturas de pantalla del código de pruebas unitarias realizadas:
`https://docs.google.com/document/d/1aq2K6njmriJisd0Mx-942pMgWadf4ZBxLDDBbIR-wM8/edit?usp=sharing`

12. Referencias

[1] M. Flores Pichardo, "Revisión de Algoritmos Genéticos Aplicados al Problema de la Programación de Cursos Universitarios," *Programación matemática y software*, vol. 3, no. 1, 2011, doi: 10.30973/progmat/2011.3.1/5.

[2] G. A. Durán and N. Faillace, "Implementación de una heurística para la programación automática de horarios de una escuela secundaria."

[3] E. M. Rojas, M. Ramírez Ramírez, H. B. Ramírez Moreno, M. D. C. Salgado Soto, N. D. C. Osuna Millán, and L. M. Cerda Suarez, "Sistema de Gestión Académica a través del desarrollo de Modelo-Vista-Controlador," *Revista Ibérica de Sistemas e Tecnologias de Informação*, no. 17, 2019.

[4] V. A. Domínguez Ríos and M. Á. López Santillán, "Teoría General de Sistemas, un enfoque práctico," *TECNOCIENCIA Chihuahua*, vol. 10, no. 3, 2017, doi: 10.54167/tch.v10i3.174.

[5] E. Mejía, "Herramientas de optimización convexa y aplicaciones en telecomunicaciones," Universidad Carlos III de Madrid. Departamento de Teoría de la Señal y Comunicaciones, 2011.

[6] C. A. C. Coello, "Introducción a los Algoritmos Genéticos," *Soluciones Avanzadas. Tecnologías de Información y Estrategias de Negocios*, no. 17, 1995.

[7] C. A. Arango, H. Felizzola, A. Hualpa, P. Gomez, and C. Mora, "Modelos de Programación Entera para el Problema de Asignación de Horarios en Cursos Universitarios," *Revista de Métodos Cuantitativos para la Economía y la Empresa*, 2023, doi: 10.46661/rev.metodoscuant.econ.empresa.7354.

[8] "Unitime," `https://www.unitime.org/`.

[9] V. P. Torrez, "Algoritmos Genéticos (AG's)," *Investig Cienc*, 2008.

[10] L. Mandow, "Introducción a la Programación de Restricciones," *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, vol. 11, no. 36, 2007.

[11] Y. Bermúdez, "Aplicaciones de programación lineal, entera y mixta," *Ingeniería Industrial. Actualidad y Nuevas Tendencias*, vol. II, 2011.

[12] Colciencias, "DEPARTAMENTO ADMINISTRATIVO DE CIENCIA, TECNOLOGIA E INNOVACIÓN-COLCIENCIAS-CONVOCATORIA CIERRE DE BRECHAS TECNOLÓGICAS-2018 ANEXO 1. TECHNOLOGY READINESS LEVELS-TRL," 2016. [Online]. Available: `https://www.nasa.gov/directorates/heo/scan/engineering/technology/txt_accordion1.html`

[13] Vásquez, Augusto Cortez, "Sistema de apoyo a la generación de horarios basado en algoritmos genéticos.Revista de investigación de Sistemas e Informática

[14] Ahumada Ahumada, Jorge. 2014, "Generación de horarios académicos en INACAP utilizando algoritmos genéticos". Santiago, Chile: Universidad de Chile - Facultad de Ciencias Físicas y Matemáticas. https://repositorio.uchile.cl/handle/2250/131197

[15] Arranz de la Peña, J, Parra Truyol, A. (2007). Algoritmos genéticos. Universidad Carlos III, 1-8.

[16] T. E. I. .-. C. DEPARTAMENTO ADMINISTRATIVO DE CIENCIA. CONVOCATORIA CIERRE DE BRECHAS TECNOLOGICAS. ´ (2018), dirección: https : / / minciencias . gov . co / sites / default / files / upload / convocatoria / anexo _ 1 . _technology _ readiness _ levels_ - _trl.pdf (visitado 2018).

[17] Oñate, J. Tullmo, D., 'Optimización basada en colonia de hormigas para resolver problemas asignación de horarios en la Universidad Técnica Estatal De Quevedo'. Universidad Técnica Estatal De Quevedo - Facultad Ciencias De La Ingeniería Carrera Ingeniería En Sistemas. Quevedo, Ecuador.

[18] Castrillón, O. (2019), combinación entre Algoritmos Genéticos y Aleatorios para la Programación de Horarios de Clases basado en Ritmos Cognitivos. Universidad Nacional de Colombia, Departamento de Ingeniería Industrial, Campus la Nubia, Manizales-Colombia. Vol. 25(4), Pág. 51-62 (2014) doi: 10.4067/S0718-07642014000400008

[19] Viñas, S., Rodríguez, N., Corona, E. Jiménez, A. (2019), software para la generación automática de horarios académicos. Avances en Ciencias e Ingeniería. México. ISSN: 0718-8706

[20] programación de empleados https://developers.google.com/optimization/scheduling/employee_scheduling?hl=es-419