

Documentación del Proyecto de Horarios

Janer Alberto Vega Jácome

2 de septiembre de 2024

1. Requerimientos

1.1. Idea general

El proyecto de horarios tiene como objetivo desarrollar una aplicación web que permita a los usuarios crear, gestionar y visualizar horarios de manera eficiente. Utilizando Spring Boot para el backend y Angular para el frontend, esta aplicación ofrecerá una interfaz intuitiva y funcionalidades avanzadas para la gestión de horarios.

1.2. Objetivos

1.2.1. Objetivo general

Desarrollar un sistema de gestión de horarios académicos para la Escuela de Ingeniería de Sistemas e Informática de la UIS que automatice y optimice la planificación, asignación y visualización de horarios y considerando las restricciones de recursos (aulas, profesores) con el fin de mejorar la eficiencia y transparencia del proceso académico.

1.2.2. Objetivos específicos

1. **Interfaz de Usuario:** Diseñar una interfaz gráfica de usuario (GUI) intuitiva y fácil de usar que permita a los usuarios (administradores, profesores y estudiantes) interactuar con el sistema de manera eficiente, proporcionando módulos específicos para la gestión de usuarios, registro académico y diseño de horarios.
2. **Gestión de Usuarios:** Implementar un módulo de gestión de usuarios que permita el registro, autenticación, autorización y administración de perfiles de usuarios (administradores, profesores, estudiantes, técnicos, auxiliares, secretarías), asignando roles y permisos adecuados para cada tipo de usuario.
3. **Registro Académico:** Desarrollar un sistema de registro académico que permita la gestión de información relevante para la generación de horarios, incluyendo:
 - Asignaturas: Nombre, código, créditos, tipo (teórica, práctica, laboratorio).
 - Profesores: Nombre, identificación, disponibilidad horaria, preferencias de asignaturas.
 - Aulas: Nombre, capacidad, tipo (aula de clase, laboratorio), disponibilidad horaria.
 - Grupos: Identificación, número de estudiantes, asignaturas asociadas.
4. **Algoritmo de Diseño de Horarios:** Diseñar e implementar un algoritmo eficiente y flexible para la generación automática de horarios académicos que considere:
 - Restricciones de recursos: Disponibilidad de aulas y profesores en diferentes horarios.
 - Restricciones académicas: Carga horaria de profesores, número máximo de estudiantes por grupo.
 - Métricas de optimización: Minimizar conflictos de horarios, maximizar la utilización de recursos, equilibrar la carga horaria de profesores.
5. **Generación de Reportes:** Desarrollar un módulo de generación de reportes que permita visualizar y exportar horarios en diferentes formatos (PDF, Excel) para:
 - Profesores: Horario individual por profesor.
 - Aulas: Horario de ocupación de cada aula.
 - Asignaturas: Horario de cada asignatura, incluyendo grupos y profesores.
 - Horario General: Horario completo de la Escuela, mostrando todas las asignaturas, grupos, profesores y aulas.

1.3. Usuarios

El sistema de gestión de horarios está diseñado para ser utilizado por los siguientes grupos de usuarios dentro de la Escuela de Ingeniería de Sistemas de la UIS:

1.3.1. Estudiantes

Los estudiantes son los principales beneficiarios del sistema. Utilizarán la aplicación para consultar sus horarios de clases, incluyendo aulas, profesores y horarios de cada materia.

1.3.2. Profesores

Los profesores también son usuarios clave del sistema. Utilizarán la aplicación para:

- Consultar sus horarios de clases, incluyendo aulas y grupos asignados.
- Reportar cambios o ajustes en sus horarios.
- Visualizar la disponibilidad de aulas para programar actividades adicionales.

1.3.3. Personal administrativo

El personal administrativo de la Escuela utilizará el sistema para:

- Crear y gestionar los horarios académicos, asignando cursos, profesores y aulas.
- Realizar cambios y ajustes en los horarios según sea necesario.
- Generar reportes y estadísticas sobre la utilización de aulas y recursos.
- Gestionar los permisos de acceso de los usuarios al sistema.

1.4. Requisitos generales

1.4.1. Funcionalidad

El sistema debe permitir la creación, visualización, modificación y gestión de horarios académicos para la Escuela de Ingeniería de Sistemas e Informática de la UIS.

1.4.2. Usabilidad

La interfaz de usuario debe ser intuitiva, fácil de usar y accesible para todos los usuarios (administradores, profesores y estudiantes).

1.4.3. Rendimiento

El sistema debe ser capaz de generar horarios de manera eficiente, incluso para un gran número de asignaturas, profesores y estudiantes.

1.4.4. Seguridad

El sistema debe proteger la información sensible de los usuarios y garantizar que solo los usuarios autorizados puedan acceder y modificar los datos.

1.4.5. Escalabilidad

El sistema debe ser diseñado para adaptarse al crecimiento futuro de la Escuela, permitiendo la adición de nuevas asignaturas, profesores y estudiantes.

1.4.6. Compatibilidad

El sistema debe ser compatible con los navegadores web modernos (Chrome, Firefox, Safari, Edge) y funcionar en diferentes dispositivos (computadoras, tabletas, teléfonos móviles).

1.5. Requisitos funcionales

1.5.1. Gestión de usuarios

- El sistema debe permitir el registro de nuevos usuarios (administradores, profesores, estudiantes).
- El sistema debe permitir la autenticación de usuarios mediante nombre de usuario y contraseña.
- El sistema debe asignar roles y permisos a los usuarios según su tipo (administrador, profesor, estudiante).
- El sistema debe permitir la modificación y eliminación de perfiles de usuario.

1.5.2. Registro académico

- El sistema debe permitir el registro de asignaturas, incluyendo nombre, código, créditos, prerequisites y tipo.
- El sistema debe permitir el registro de profesores, incluyendo nombre, identificación, disponibilidad horaria y preferencias de asignaturas.
- El sistema debe permitir el registro de aulas, incluyendo nombre, capacidad, tipo y disponibilidad horaria.
- El sistema debe permitir el registro de grupos, incluyendo identificación, número de estudiantes y asignaturas asociadas.

1.5.3. Diseño de horarios

- El sistema debe generar automáticamente horarios académicos que cumplan con las restricciones de recursos y académicas.
- El sistema debe permitir la modificación manual de los horarios generados.
- El sistema debe detectar y notificar conflictos de horarios (choques entre bloques de horas, solapamiento de horarios).
- El sistema debe permitir la optimización de los horarios según diferentes criterios (minimizar conflictos, maximizar la utilización de recursos, equilibrar la carga horaria de profesores).

1.5.4. Visualización de horarios

- El sistema debe permitir la visualización de horarios individuales por profesor.
- El sistema debe permitir la visualización de horarios de ocupación de aulas.
- El sistema debe permitir la visualización de horarios por asignatura, incluyendo grupos y profesores.
- El sistema debe permitir la visualización de un horario general de la Escuela.

1.5.5. Generación de reportes

- El sistema debe permitir la exportación de horarios en formato PDF.
- El sistema debe permitir la exportación de horarios en formato Excel.

1.6. Información de autoría

- Este proyecto es desarrollado por estudiantes de la Escuela de Ingeniería de Sistemas e Informática de la Universidad Industrial de Santander (UIS), como parte de una auxiliatura académica.
- Los derechos de autor del código fuente y la documentación del proyecto pertenecen a la Escuela de Ingeniería de Sistemas e Informática de la Universidad Industrial de Santander (UIS).

1.7. Alcances y limitaciones

1.7.1. Alcances

- El sistema se enfocará en la gestión de horarios académicos para la Escuela de Ingeniería de Sistemas e Informática de la UIS.
- El sistema permitirá la generación automática de horarios, considerando restricciones y preferencias.
- El sistema ofrecerá diferentes vistas de horarios y la posibilidad de exportarlos en varios formatos.

1.7.2. Limitaciones

- El sistema no gestionará otros aspectos de la vida académica, como calificaciones, asistencia o inscripciones.
- El sistema no se integrará inicialmente con otros sistemas de la universidad.
- El algoritmo de diseño de horarios puede no encontrar soluciones óptimas en todos los casos, especialmente si las restricciones son muy complejas o contradictorias.

1.8. Herramientas de desarrollo

1. IntelliJ IDEA Ultimate

- **Tipo:** Entorno de desarrollo integrado (IDE)
- **Justificación:** Para desarrollar el backend de la aplicación en Spring Boot, aprovechando sus herramientas de desarrollo Java y su integración con frameworks de Spring.

2. Visual Studio Code (VS Code)

- **Tipo:** Editor de código fuente
- **Justificación:** Para desarrollar el frontend de la aplicación en Angular, utilizando sus extensiones para Angular y TypeScript, así como sus herramientas de depuración y pruebas.

3. MySQL Workbench

- **Tipo:** Herramienta de administración de bases de datos
- **Justificación:** Para diseñar y administrar la base de datos MySQL donde se almacenarán los datos de horarios, asignaturas, profesores, aulas, etc.

4. GitHub

- **Tipo:** Plataforma de alojamiento de código y colaboración
- **Justificación:** Para alojar el código fuente del proyecto, facilitar la colaboración entre los miembros del equipo y llevar un seguimiento de los cambios en el código a lo largo del tiempo.

5. Tailscale

- **Tipo:** Red privada virtual (VPN)

- **Justificación:** Facilitar la colaboración entre los miembros del equipo de desarrollo, permitiéndoles compartir recursos y trabajar en el proyecto de forma remota como si estuvieran en la misma red local.

6. Swagger UI

- **Tipo:** Herramienta de visualización y documentación de APIs REST
- **Justificación:** Permitir a los desarrolladores probar los endpoints de la API directamente desde la interfaz de usuario, agilizando el proceso de desarrollo y depuración. Proporcionar una documentación clara y completa de la API REST del sistema, facilitando su uso por parte del desarrollador frontend.

7. Ubuntu Server

- **Tipo:** Sistema operativo de servidor basado en Linux
- **Justificación:** Alojar el backend en un servidor dedicado, así el sistema está disponible las 24 horas del día, los 7 días de la semana, lo que garantiza que el desarrollador frontend podrá consumir las APIs en cualquier momento, lo que facilita su trabajo.

1.9. Planificación

La planificación detallada del proyecto, pasos principales y estimación de tiempos, se encuentra en un documento adjunto en esta carpeta.

1.10. Obtención e instalación

1.10.1. Prerequisitos

- Java Development Kit (JDK) 17
- Maven (al menos 3.9.6)
- Node.js (al menos 18.19.1)
- npm (al menos 10.8.2)
- Angular (18.1.0)
- Git
- MySQL 5.7 instalado y configurado (usuario «root» con contraseña «1234»)

1.10.2. Obtención del código fuente

1. Clonar el repositorio desde GitHub

```
git clone https://github.com/SrNe0/HorarioUIS.git
```

2. Cambiar a la rama «develop»

```
git checkout develop
```

1.10.3. Instalación y configuración del Backend (Spring Boot)

1. Entrar a MySQL (*contraseña «1234»*)

```
mysql -u root -p
```

2. Crear la base de datos

```
CREATE DATABASE horario_uis;
```

3. Ir al directorio del Backend

```
cd HorarioUIS/Backend
```

4. Compilar el proyecto con Maven

```
mvn clean install
```

Esto creará un archivo .jar en la carpeta target.

5. Ejecutar el archivo .jar

```
java -jar nombre_del_archivo.jar
```

El backend se ejecutará por defecto en `http://localhost:8080`

1.10.4. Intalación y configuración del Frontend (Angular)

En el directorio del proyecto:

1. Instalar Angular y demás dependencias

```
sudo npm install
```

2. Ir al directorio del Frontend

```
cd Frontend
```

3. Iniciar el servidor de desarrollo

```
ng serve
```

El frontend se ejecutará por defecto en `http://localhost:4200`

2. Arquitectura del sistema

2.1. Descripción

El sistema de gestión de horarios sigue una arquitectura cliente-servidor, donde el frontend y el backend están separados y se comunican a través de una API REST. Adicionalmente, se utiliza una base de datos para el almacenamiento persistente de los datos.

1. **Angular:** El frontend se desarrolla utilizando el framework Angular, que organiza el código en componentes, servicios y módulos.

- **Componentes:** Representan las diferentes partes de la interfaz de usuario (vistas de horarios, vista de login, etc.).
- **Servicios:** Módulos que manejan la lógica de negocio en el frontend y se comunican con el backend a través de HTTP.
- **Módulos:** Agrupan componentes y servicios relacionados.

2. **Spring Boot:** El backend se construye con Spring Boot, que sigue una arquitectura de capas:

- **Controladores:** Reciben las solicitudes del frontend, interactúan con los servicios y devuelven las respuestas.
- **Servicios:** Implementan la lógica, como la generación de horarios, la validación de datos y el acceso a la base de datos.
- **Repositorios:** Se encargan de la interacción con la base de datos, utilizando JPA (Java Persistence API) para mapear las entidades del modelo de datos a tablas en la base de datos.
- **Entidades:** Clases que representan las tablas en la base de datos y son utilizadas para mapear los datos entre la aplicación y la base de datos.

3. Base de datos

Se utiliza una base de datos MySQL para almacenar los datos del sistema, como información de usuarios, asignaturas, profesores, disponibilidad horaria, aulas y horarios.

4. Integración y comunicación entre las capas

- **API RESTful:** El backend de Spring Boot expone una serie de endpoints RESTful que el frontend en Angular consume. Esta comunicación se realiza a través de HTTP, donde el frontend envía solicitudes y recibe respuestas en formato JSON.
- **Seguridad:** La autenticación y autorización se manejan en la capa de negocio, asegurando que solo usuarios autenticados puedan acceder a ciertos recursos, utilizando JWT (JSON Web Tokens).

2.2. Módulos principales

▪ Frontend:

• Módulo de interfaz de usuario (UI)

- **Descripción:** Este módulo maneja la interfaz de usuario de la aplicación. Su propósito es permitir que los usuarios interactúen con el sistema de manera amigable y eficiente.
- **Responsabilidad:** Mostrar datos y capturar las entradas del usuario.
- **Restricciones:** No tiene acceso directo a la lógica de negocio o a la base de datos; solo interactúa con los datos que le son provistos a través del módulo de comunicación.
- **Dependencias:** Depende del *Módulo de comunicación HTTP* para obtener y enviar datos al backend.
- **Implementación:** Este módulo se implementa en archivos `.html`, `.ts` y `.css` dentro de la carpeta `src/app`, dividido en varios componentes.

- **Módulo de comunicación HTTP**
 - **Descripción:** Este módulo maneja todas las solicitudes HTTP que el frontend envía al backend. Su objetivo es interactuar con el backend para obtener y enviar datos.
 - **Responsabilidad:** Enviar solicitudes HTTP y procesar las respuestas.
 - **Restricciones:** No debe contener lógica de presentación ni lógica de negocio, solo debe encargarse de la comunicación.
 - **Dependencias:** Depende del ***Módulo UI*** para recibir solicitudes de datos y enviar respuestas, y de los servicios REST expuestos por el backend en Spring Boot.
 - **Implementación:** Este módulo se implementa en archivos `.ts` dentro de la carpeta `src/app/services`.
- **Backend:**
 - **Módulo de controladores**
 - **Descripción:** Este módulo actúa como intermediario entre el frontend y la lógica de negocio. Maneja las solicitudes HTTP entrantes y devuelve las respuestas adecuadas.
 - **Responsabilidad:** Recibir solicitudes HTTP, delegarlas a los servicios adecuados, y retornar respuestas.
 - **Restricciones:** No debe contener lógica de negocio; solo delega tareas a los servicios.
 - **Dependencias:** Depende del ***Módulo de servicios*** para realizar la lógica de negocio y del ***Módulo de comunicación HTTP*** para recibir solicitudes desde el frontend.
 - **Implementación:** Este módulo se implementa en clases Java con anotaciones `@RestController`, ubicadas en el paquete `uis.horariouis.controller`.
 - **Módulo de servicios**
 - **Descripción:** Este módulo contiene la lógica de negocio de la aplicación. Es responsable de procesar los datos y aplicar las reglas de negocio antes de enviarlos a los controladores o repositorios.
 - **Responsabilidad:** Implementar la lógica de negocio y realizar transformaciones en los datos.
 - **Restricciones:** No debe interactuar directamente con la interfaz de usuario ni con la base de datos; estas interacciones deben pasar por los controladores y repositorios, respectivamente.
 - **Dependencias:** Depende del ***Módulo de repositorios*** para acceder a la base de datos.
 - **Implementación:** Este módulo se implementa en clases Java dentro del paquete `uis.horariouis.service`.
 - **Módulo de repositorios**
 - **Descripción:** Este módulo maneja la persistencia de los datos en la base de datos. Proporciona abstracciones para realizar operaciones CRUD sobre la base de datos.
 - **Responsabilidad:** Interactuar con la base de datos a través de JPA para realizar operaciones CRUD.
 - **Restricciones:** No debe contener lógica de negocio, solo acceso a datos.
 - **Dependencias:** Depende del ***Módulo de gestión de datos*** en MySQL para almacenar y recuperar datos.
 - **Implementación:** Este módulo se implementa en interfaces de Java que extienden `JpaRepository`, ubicadas en el paquete `uis.horariouis.repository`.
 - **Módulo de seguridad**
 - **Descripción:** Este módulo se encarga de la autenticación y autorización de usuarios dentro de la aplicación.
 - **Responsabilidad:** Proteger los recursos de la aplicación mediante autenticación y autorización.
 - **Restricciones:** No debe gestionar la lógica de negocio, solo asegurar que las solicitudes sean realizadas por usuarios autenticados y autorizados.

- **Dependencias:** Depende de JWT para la autenticación.
 - **Implementación:** Este módulo se implementa en clases Java con configuraciones de seguridad, ubicadas en el paquete `uis.horariouis.security`.
- **Base de datos:**
 - **Módulo de gestión de datos**
 - **Descripción:** Este módulo es responsable de gestionar el almacenamiento y recuperación de datos en la base de datos MySQL.
 - **Responsabilidad:** Almacenar, recuperar y manipular datos según las operaciones realizadas por el backend.
 - **Restricciones:** Solo maneja datos estructurados según el esquema de la base de datos.
 - **Dependencias:** Depende de las configuraciones de la base de datos y de la conexión establecida por el *Módulo de repositorios*.
 - **Implementación:** Este módulo se implementa en la estructura de la base de datos, incluyendo tablas, índices, y procedimientos almacenados.

2.3. Implementación

2.3.1. Seguridad en el Frontend

La seguridad en el proyecto se implementa a través de varios mecanismos clave que trabajan juntos para proteger la aplicación y asegurar que solo los usuarios autenticados y autorizados puedan acceder a los recursos sensibles. A continuación, se detalla cómo se han implementado estos mecanismos:

1. Autenticación con JWT (JSON Web Tokens)

- a) **Generación del token:** Cuando un usuario inicia sesión, el backend autentica sus credenciales y genera un token JWT. Este token contiene la información del usuario y su tiempo de expiración, y es firmado digitalmente para evitar manipulaciones.
- b) **Almacenamiento:** El token JWT generado es enviado al frontend y almacenado en el `localStorage` del navegador bajo la clave `token_user`. Este token se utiliza para autenticar todas las solicitudes subsiguientes del usuario.

2. Interceptores para la seguridad de las solicitudes

El interceptor `authApiInterceptor` agrega automáticamente el token JWT a cada solicitud HTTP que se envía al backend:

- a) **Intercepción de Solicitudes:** El interceptor intercepta cada solicitud HTTP y, si el token JWT está presente en `localStorage`, lo agrega en el encabezado `Authorization` de la solicitud.
- b) **Validación por el Backend:** El backend valida el token en cada solicitud para asegurar que sea válido y que el usuario esté autorizado para acceder al recurso solicitado. Si el token es inválido o ha expirado, el backend rechaza la solicitud con un error de autenticación.

3. Protección de rutas con Guards

Para proteger las rutas dentro de la aplicación y asegurar que solo los usuarios autenticados puedan acceder a ciertas secciones, se utiliza un guard:

- a) **Guard de Autenticación (`loginGuard`):** Este guard verifica si el usuario tiene un token JWT válido almacenado en el `localStorage`. Si el token existe, se permite el acceso a la ruta. Si no, el usuario es redirigido a la página de login. Este guard está implementado en la configuración de rutas (en el archivo `app.routes.ts`) de la aplicación, asegurando que todas las rutas críticas estén protegidas.

4. Protección de componentes y datos sensibles

- a) **Componentes protegidos:** El uso de los componentes críticos como `ModifyComponent` y `NewComponent` están protegidos mediante el uso de guards, asegurando que solo los usuarios autorizados puedan realizar modificaciones o crear nuevos registros.
- b) **Manejo Seguro de Datos:** Los datos sensibles, como contraseñas y tokens, son manejados con precaución en toda la aplicación. Las contraseñas nunca se almacenan en texto plano, y los tokens se mantienen en `localStorage` solo durante la duración de la sesión del usuario.

5. Respuesta y manejo de errores

En caso de que una solicitud no sea autenticada correctamente:

- a) **Manejo de errores:** El interceptor también maneja los errores que pueden ocurrir durante la comunicación con el backend. Si ocurre un error de autenticación, se muestra un mensaje de error.

2.3.2. Seguridad en el Backend

El sistema de seguridad está diseñado con Spring Security. La seguridad se implementa mediante el uso de JWT (JSON Web Tokens) para la autenticación y configuración de acceso basada en roles.

1. **Configuración de seguridad** En la clase `SecurityConfigurer.java` se definen las reglas para la autenticación y autorización:

- **Autenticación y autorización:**

- Se permite el acceso público a ciertas rutas, como la de autenticación (`/api/security/authenticate`).
- Se restringe el acceso a otras rutas según los roles de usuario (`USER`, `ADMIN`). Solo los usuarios con rol `ADMIN` pueden realizar operaciones `POST`, `PUT`, y `DELETE`.
- Se habilita la autenticación básica HTTP, lo cual es útil para acceder a la documentación Swagger.

- **Gestión de sesiones:**

- El sistema utiliza una política de sesión sin estado (`STATELESS`), lo que significa que no se mantiene una sesión en el servidor.

- **Cross-Origin Resource Sharing (CORS)**

- Se configuran las reglas de CORS para permitir que el frontend interactúe con el backend desde orígenes específicos.

2. **JSON Web Tokens (JWT)** Se utiliza JWT para manejar la autenticación de los usuarios:

- **Generación y Validación de Tokens:**

- La clase `JwtUtil.java` se encarga de generar los tokens JWT cuando un usuario se autentica exitosamente. También valida los tokens y extrae información relevante como el nombre de usuario, la fecha de expiración. Puede extraer un reclamo en específico o todos los reclamos del token JWT.

- **Filtro de solicitudes:**

- El filtro `JwtRequestFilter.java` intercepta cada solicitud HTTP para verificar si contiene un token JWT válido en el encabezado de autorización. Si el token es válido, se establece la autenticación en el contexto de seguridad.

3. **Detalles del usuario y servicios**

- **CustomUserDetails.java:**

- Esta clase implementa la interfaz `UserDetails` de Spring Security y proporciona los detalles de autenticación y autorización del usuario, como el nombre de usuario y la contraseña.

- **CustomUserDetailsService.java:**

- Este servicio se encarga de cargar los detalles del usuario desde la base de datos utilizando el repositorio `UsuarioRepository`. Si el usuario es encontrado, se retornan sus detalles encapsulados en un objeto `CustomUserDetails`.