Documentación del Proyecto de Horarios

Janer Alberto Vega Jácome

4 de septiembre de 2024

1. Arquitectura del sistema

1.1. Decripción

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).

1.2. 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 desdel 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.

1.3. Implementación

1.3.1. 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.

1.3.2. 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`.
4. **Gestión de roles y permisos**