Trabajo Fin de Máster

Alumno: Iñigo Sanz Delgado
Competencia: Desarrollo Full Stack
Profesores: Cliff, José Luis, Javier, Rubén

Índice

Índice de Imágenes

1. Introducción

Este documento recoge la parte técnica del desarrollo de la aplicación web GetaBreak, la cual sirve para solicitar, administrar y gestionar las vacaciones de los empleados de una empresa. La herramienta está pensada para contribuir y simplificar el proceso desde que un empleado solicita los días libres hasta que esta es revisada y aprobada por los responsables correspondientes.

En el backend se ha empleado el patrón de arquitectura hexagonal para mantener un diseño limpio, escalable y fácil de mantener. Se ha hecho uso de la separación en capas, uso de DTOs y MapStruct para agilizar el proceso de trabajo entre los servicios y controladores.

La aplicación implementa distintos perfiles de usuarios: empleados (normales), encargados de departamento y responsables de RRHH. Cada uno de ellos accede a funcionalidades especificas según su rol. El sistema permite a los empleados solicitar vacaciones a través de un formulario, visualizar el calendario compartido de su departamento y consultar el estado de sus solicitudes. Por otra parte, los encargados pueden aprobar o rechazar las solicitudes de su equipo, mientras que RRHH hace una validación final de estas y gestiona los usuarios/empleados desde un panel de administración.

2. Tecnologías y frameworks utilizados

Para el desarrollo de la aplicación se ha utilizado un conjunto de tecnologías tanto en el backend como en el frontend con el fin de aportar sencillez, rendimiento y escalabilidad.

Backend

Tecnologías utilizadas:

Patrón Arquitectura Hexagonal

"La arquitectura Hexagonal propone que nuestro dominio sea el núcleo de las capas y que este no se acople a nada externo. En lugar de hacer uso explícito y mediante el principio de inversión de dependencias nos acoplamos a contratos (interfaces o puertos) y no a implementaciones concretas".

Lo que hace es definir puertos de entrada y salida, interfaces y adaptadores. Para que así los otros módulos de la aplicación como la UI o la BBDD puedan implementarlos y comunicarse con la capa donde esta el negocio sin que esta deba saber de dónde la están utilizando.

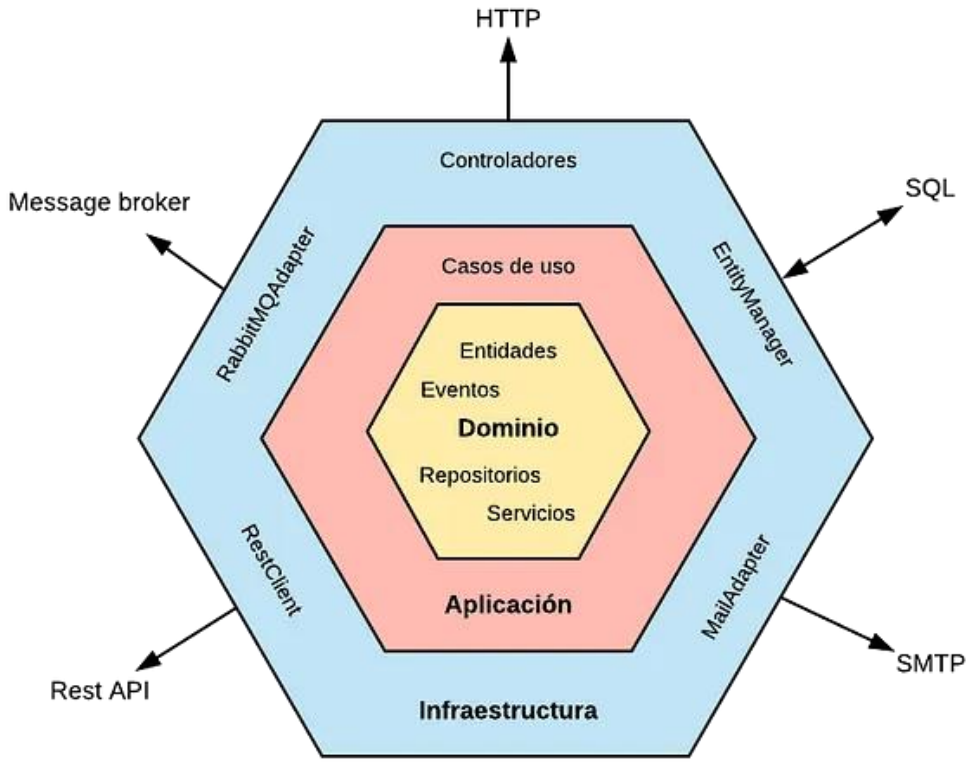

Figura 1 - Arquitectura Hexagonal

En resumidas cuentas, lo podemos plantear como:

- A la izquierda el lado del usuario
- La lógica de negocio en el centro
- A la derecha el lado del servidor

Spring Boot

Java Spring Boot es una herramienta que sirve para acelerar y simplificar el desarrollo de microservicios y aplicaciones web con Spring Framework gracias a principalmente tres funcionalidades:

1. Configuración automática
2. Enfoque de configuración obstinado
3. Capacidad de crear configuraciones autónomas

La configuración automática significa que las aplicaciones se inicializan con dependencias predefinidas que no es necesario configurar manualmente.

El enfoque obstinado se refiere a que Spring Boot añade y configura dependencias de iniciador, en función de las necesidades del proyecto. Según su criterio elige que paquetes instalar y que valores predeterminados utilizar.

Y la capacidad de crear configuraciones autónomas, que permite a los desarrolladores crear aplicaciones que simplemente se ejecuta, sin depender de un servidor web externo, integrando un servidor web como Tomcat o Netty embebido durante la inicialización.

Lombok

Es una biblioteca de código abierto para Java que elimina la necesidad de escribir código repetitivo, comúnmente llamado código boilerplate.

Lombok genera este código mediante anotaciones, como podrían ser los Getter y Setter o constructores (@Getter y @Setter).

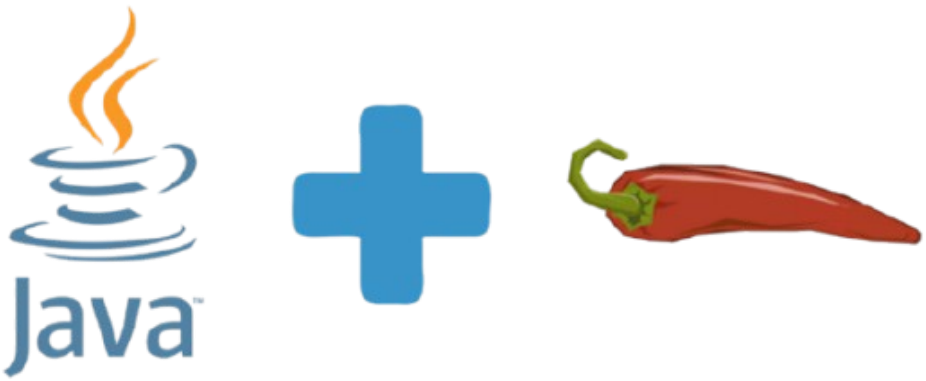

Figura 2 - Java Lombok

MapStruct

Librería Java que permite realizar mapeos automáticos entre objetos. Se utiliza principalmente para convertir objetos de entidades (modelos) a DTOs (Data Transfer Objects) y viceversa.

- Rápido
- Mantenible
- Seguro en compilación
- Bidireccional (por ejemplo de Command a DTO y de DTO a Command)

Maven

Es una herramienta para la gestión de dependencias en proyectos, sirve también como herramienta de compilación y documentación.

Entre otras cosas, permite:

- Gestionarlas dependencias del proyecto.
- Compilar el código fuente.
- Empaquetar el código en archivos .jar o .zip
- Instalar paquetes en repositorios tanto locales como remotos.
- Generar documentación.
- Gestionar el ciclo de vida de la aplicación

Figura 3 - Maven

Frontend

Tecnologías utilizadas:

Angular

Es un framework basado en JavaScript desarrollado por Google que sirve para crear aplicaciones web de una sola página o ASP.

Incluye herramientas integradas como:

- Enrutamiento o gestión de URLs
- Inyección de dependencias
- Formularios reactivos o plantillas
- Servicios HTTP para consumir APIs
- Testing

Angular sigue el patrón de arquitectura MVC (Modelo – Vista – Controlador).

TypeScript

Es un lenguaje de programación basado en JavaScript que implementa algunas mejoras respecto a este último. Realmente es un superset de JavaScript.

Las principales diferencias son:

- JavaScript tiene un tipado dinámico y TypeScript tipado estático, es decir que se deben declarar los tipos.
- En TypeScript los errores se encuentran en tiempo de compilación.
- Necesita compilarse a JavaScript.
- Mayor escalabilidad y robustez.
- Soporte para clases y Módulos.

Base de Datos

Como BBDD se ha utilizado lo siguiente:

MongoDB

"MongoDB es una base de datos de documentos que ofrece una gran escalabilidad y flexibilidad, y un modelo de consultas e indexación avanzado".

Es una base de datos NoSQL, es decir que, a diferencia de por ejemplo MySQL, esta es no relacional. Almacena los datos en documentos similares a JSON dentro de colecciones.

Principales características:

- No utiliza tablas.
- Se pueden guardar documentos con distintas estructuras en la misma colección.
- Fácil de escalar en múltiples servidores.
- Tiene un lenguaje de consultas propio, MQL – Mongo Query Lenguage.
- Soporta índices simples y compuestos.
- Mayor velocidad en las operaciones con los datos
- Documentos anidados.

Figura 4 - MongoDB

Modelo – Vista – Controlador

MVC se basa en separar la lógica de negocio (Modelo), la presentación (Vista) y la interacción con el usuario (Controlador), facilitando el mantenimiento y la escalabilidad.

El modelo representa los datos y la lógica del negocio, como podrían ser las entidades, DTOs o servicios.

La vista es la capa de muestra la información al usuario, pero en este caso no la renderizamos en el backend, si no que es Angular quien se encarga de eso. Podríamos decir que se utiliza el patrón MVC desacoplado, donde el backend actúa como proveedor de datos y el frontend como capa de presentación.

Los Controladores contienen la lógica de los archivos, en este caso los controladores rest o los component.ts de Angular.

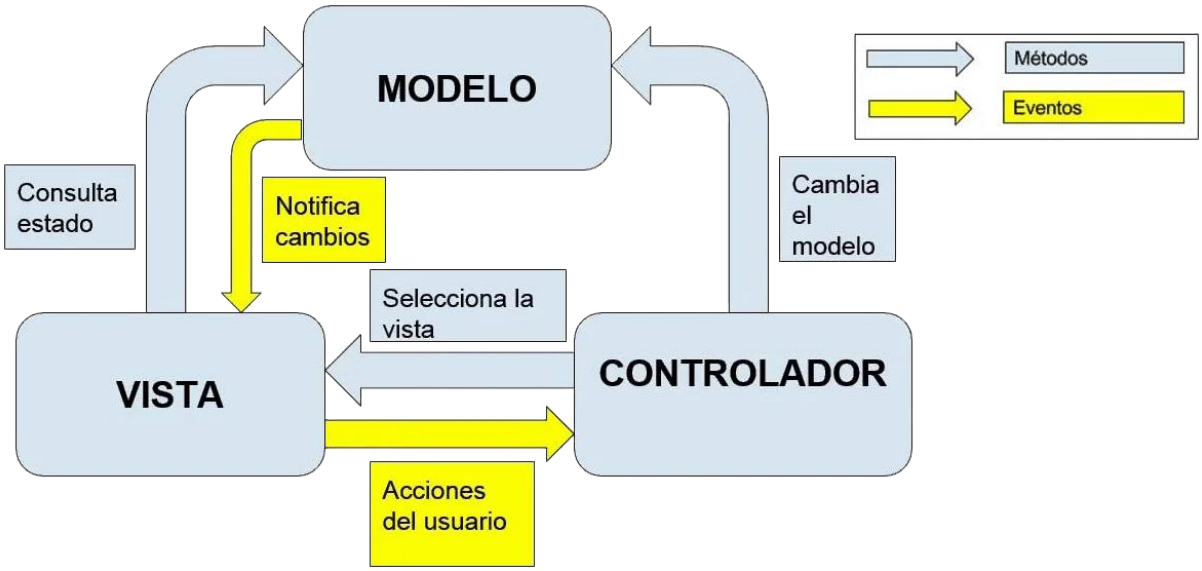

Figura 5 - MVC

Resumen Tecnologías y Herramientas

Backend

- Lenguaje: Java
- Framework principal: Spring Boot
- Arquitectura: Arquitectura Hexagonal (puertos y adaptadores)
- Persistencia (ORM): Spring Data MongoDB
- Mapping: MapStruct
- Controladores (REST): Spring Web (@RestController)
- Gestión de dependencias: Maven
- Excepciones: Clases personalizadas
- Paginación: Implementada en empleados y solicitud de vacaciones
- Autenticación y Seguridad: Pendiente de implementar con Spring Security
- Pruebas: Pendiente de implementar con Junit / Mockito

Frontend

- Lenguaje: TypeScript
- Framework: Angular
- Estilos: CSS personalizado
- Plantillas: HTML
- Formularios: Formularios Reactivos (FormBuilder, Validators)
- Servicios HTTP: HttpClient
- Routing: Rutas anidadas
- Paginación: Implementada en los empleados y solicitudes

Otras herramientas

- Base de Datos: MongoDB
- IDE Backend: Eclipse Spring Tool Suite (STS)
- IDE Frontend: Visual Studio Code
- Control: Github Desktop
- Documentación: Javadoc (a falta de implementar Swagger)
- Despliegue: Entorno local
- Gestión de BBDD: MongoDB Compass
- Pruebas endpoints: Postman

Diagrama de Arquitectura y Componentes

Como se ha mencionado anteriormente, la arquitectura de la aplicación se basa en el patrón Hexagonal – Puertos y Adaptadores, con el objetivo de lograr un diseño modular, mantener cada componente desacoplado y siendo fácil de mantener.

La distribución de la aplicación es la siguiente:

- Frontend (Angular): Es la aplicación cliente desde la que los usuarios interactúan, esta consume los endpoints expuestos por el backend.
- Backend (Spring Boot):
    - Capa de Aplicación: contiene la lógica que orquesta los casos de uso.
        - Puertos de entrada (input): Interfaces que definen que operaciones se pueden ejecutar, como registrar empleados o aprobar solicitudes de vacaciones.
        - Puertos de salida (output): Interfaces que declaran que servicios necesita el caso de uso, como guardar algo en la BBDD.
        - Servicios: Son las implementaciones de los casos de uso, contienen la lógica e interactúan con la capa de dominio y la capa de infraestructura.
    - Capa de Dominio:
        - Entidades: Como Employee, Vacation, etc.
        - Commands: Que representan acciones concretas del negocio.
        - Reglas: Excepciones de dominio y clases de utilidad.
    - Capa de Infraestructura: Implementa las necesidades externas.
        - Controladores REST: Expone la API para que sea consumida.

- Adaptadores de persistencia: Implementan los puertos de salida para MongoDB.
- Mappers: Transforma los datos que se utilizan (DTOs, Commands, Entidades).

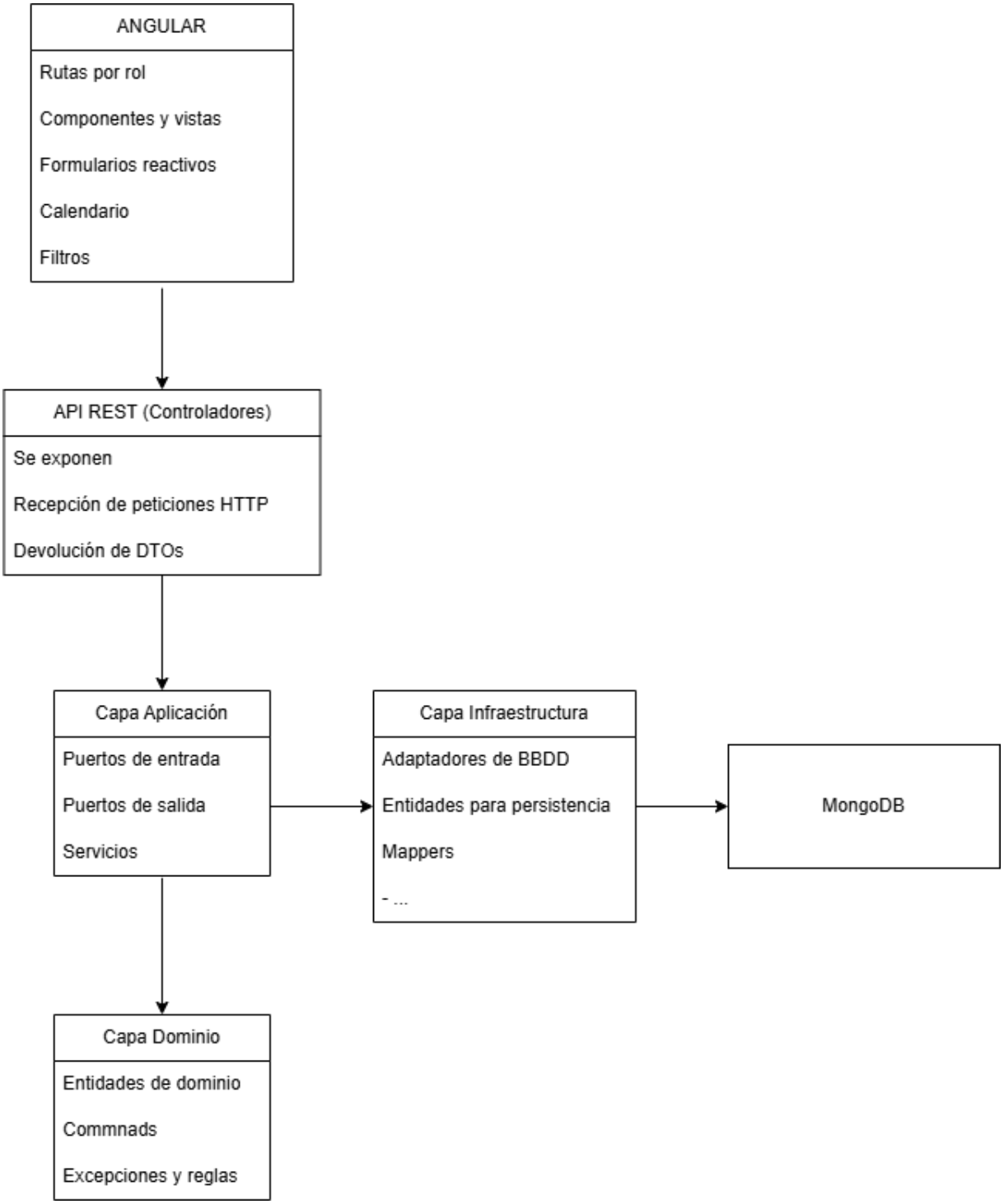

Figura 6 - Diagrama Arquitectura y Componentes

Diagrama de estructura y relación de clases

Clases implementadas en el backend

El siguiente diagrama muestra las clases principales implementadas en el backend, enfocándose en las relaciones entre entidades.

Capa de Aplicación

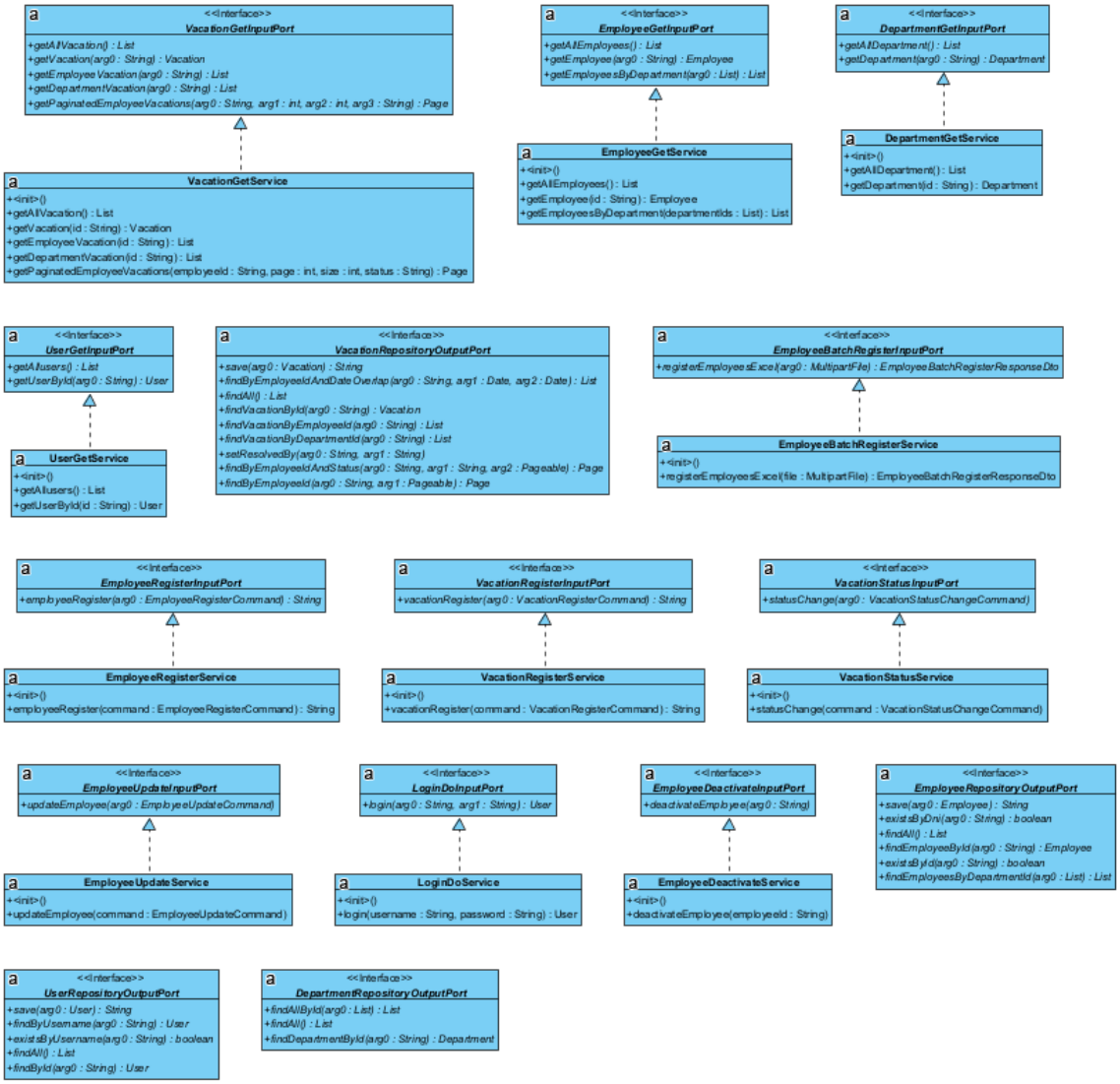

Figura 7 - UML Capa de Aplicación

Capa de Dominio

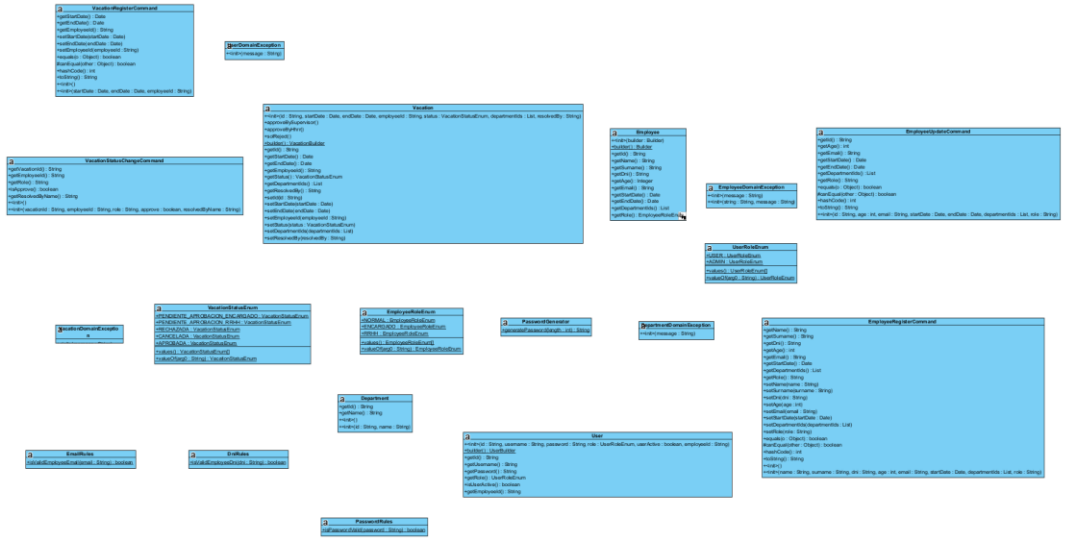

Figura 8 - UML Capa de Dominio

Capa de Infraestructura

Figura 9 - UML Capa de Infraestructura

Clases implementadas en el frontend

Se muestran los diagramas por carpetas de uso y el diagrama final:

Core

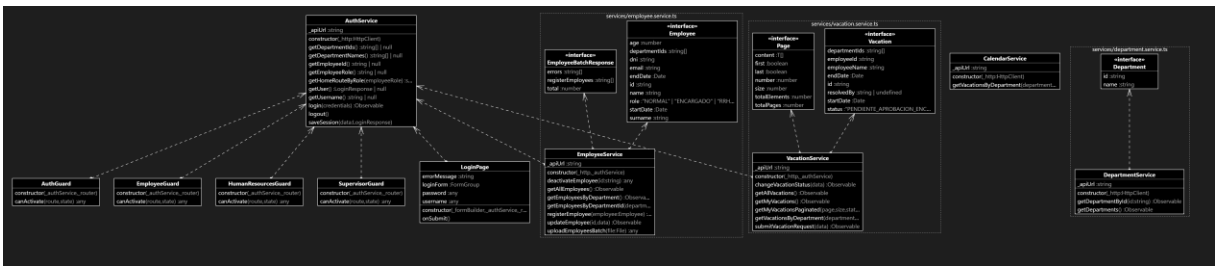

Figura 10 - UML Frontend Core

Modules

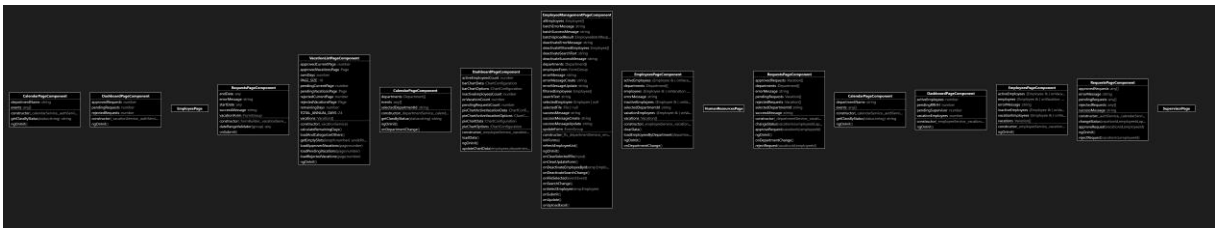

Figura 11 - UML Frontend Modules

Shared

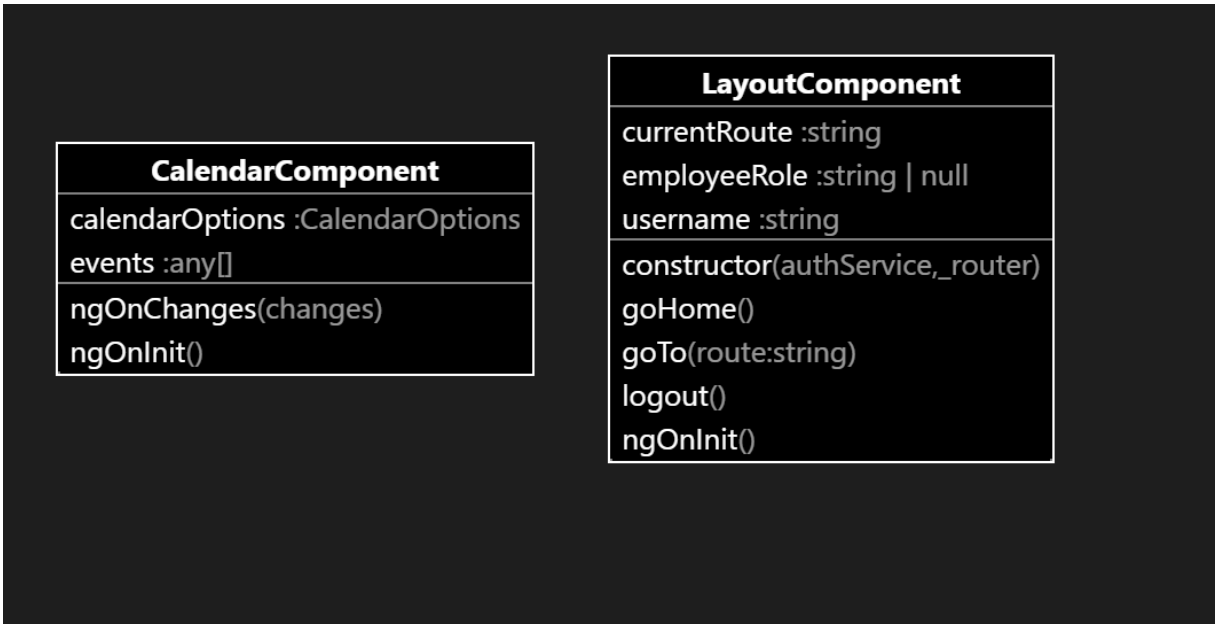

Figura 12 - UML Frontend Shared

All

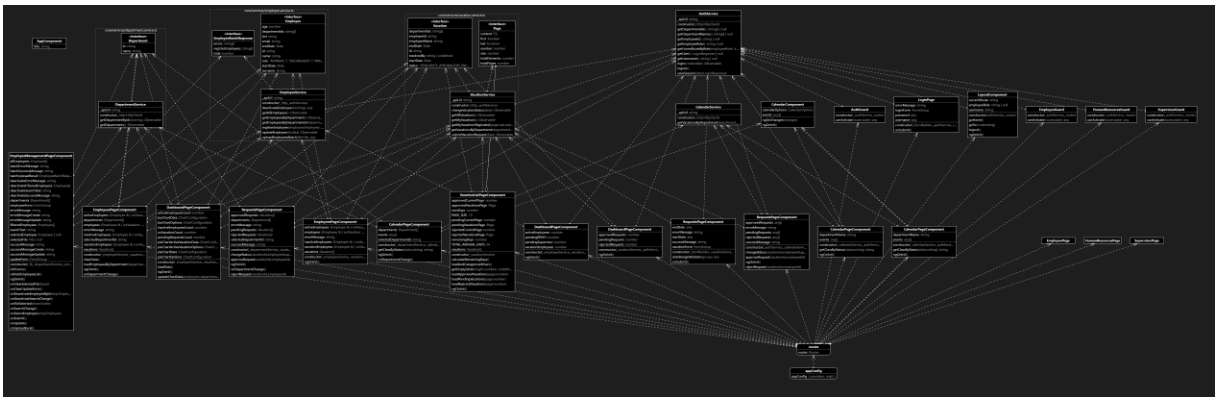

Figura 13 - UML Frontend Global

Procedimiento para Desarrollar Componentes Backend

API Rest con Spring Boot

Para cada entidad de dominio se implementa un controlador, modularizando lo máximo posible la aplicación y siguiendo el estándar CRUD:

- Vacation ↔ VacationController
- Employee ↔ EmployeeController
- Department ↔ DepartmentController
- User ↔ UserController
- LoginController

Endpoints – LoginController

Login

Método:	login
Endpoint:	POST /login
Descripción:	Permite autenticar a un usuario en el sistema a partir de sus credenciales. Si el usuario y la contraseña son válidos, devuelve un DTO con los datos del usuario, incluyendo su rol y los departamentos asociados.
Parámetros:	- Request: Objeto LoginRequestDto que contiene el nombre de usuario y la contraseña del usuario que desea autenticarse.
Excepciones:	- 400 BAD REQUEST: Si el usuario o la contraseña no son válidos, o no se encuentra el usuario.
Respuestas:	- 200 OK: Devuelve un objeto LoginResponseDto con los datos del usuario autenticado, incluyendo: o userId o username o role o userActive o employeeId o employeeRole o departmentNames

	○ departmentIds - 400 BAD REQUEST: Si las credenciales son incorrectas.

Endpoints – UserController

User

Método:	getAllUsers
Enpoint:	GET /user
Descripción:	Obtiene la lista completa de usuarios registrados en el sistema. Devuelve un DTO por usuario, incluyendo información sobre el mismo.
Parámetros:	- No requiere.
Excepciones:	- 204 NO CONTENT: Si no hay usuarios registrados en el sistema.
Respuestas:	- 200 OK: Lista de objetos USerResponseDto representando los usuarios activos.

Método:	getUser
Endpoint:	GET /user/{user-id}
Descripción:	Devuelve los datos de un usuario especifico a partir de su ID. Si el usuario está vinculado a un empleado, se incluirá también su rol.
Parámetros:	- user-id: Identificador del usuario a consultar.
Excepciones:	- 404 NOT FOUND: Si no existe ningún usuario con el ID proporcionado.
Respuestas:	- 200 OK: Objeto UserResponseDto con la información del usuario encontrado: ○ id ○ username ○ role ○ userActive ○ employeeId ○ employeeRole - 404 NOT FOUND: Si no se encuentra el usuario en el sistema.

Endpoints – EmployeeController

Employee

Método:	employeeRegister
Endpoint:	POST /employee
Descripción:	Registra un nuevo empleado en el sistema a partir de los datos proporcionados en la solicitud.
Parámetros:	- request: EmployeeRequestDto con los datos del nuevo empleado.
Respuestas:	- 201 CREATED: Registro exitoso. Devuelve la URI del recurso creado.

Método:	employeeRegisterExcel
Endpoint:	POST /employee/batch-upload
Descripción:	Permite registrar múltiples empleados mediante la carga de un archivo Excel.
Parámetros:	- File: Archivo Excel (multipart) que contiene los empleados que se van a registrar.
Respuestas:	- 200 OK: Todos los empelados fueron registardos correctamente. - 207 MULTI-STATUS: Algunos empleados se registraron, otros no. - 400 BAD REQUEST: Ningún empleado fue registrado. - 500 INTERNAL SERVER ERROR: Error al procesar el archivo.

Método:	getAllEmployees
Endpoint:	GET /employee
Descripción:	Obtiene la lista de todos los empleados registrados en el sistema.
Parámetros:	- No requiere.
Respuestas:	- 200 OK: Lista de EmployeeResponseDto. - 204 NO CONTENT: No hay empleados registrados.

Método:	getEmployee
Endpoint:	GET /employee/{employee-id}
Descripción:	Obtiene la información de un empleado especifico a partir de su ID.
Parámetros:	- employee-id: identificador del empleado.
Respuestas:	- 200 OK: Objeto EmployeeResponseDto. - 404 NOT FOUND: Empleado no encontrado.

Método:	getEmployeesByDepartment
Endpoint:	GET /employee/department/{department-id}
Descripción:	Obtiene todos los empleados asociados a uno o varios departamentos.
Parámetros:	- department-id: Lista de identificadores de departamento.
Respuestas:	- 200 OK: Lista de EmployeeResponseDto. - 204 NO CONTENT: No hay empleados en los departamentos solicitados.

Método:	updateEmployee
Endpoint:	PUT /employee/{employee-id}
Descripción:	Actualiza los datos modificables de un empleado existente.
Parámetros:	- employee-id: identificador del empleado a actualizar. - request: Objeto EmployeeUpdateRequestDto con los nuevos datos.
Respuestas:	- 200 OK: Actualización realizada con éxito.

Método:	deactivateEmployee
Endpoint:	PATCH /employee/{employee-id}/deactivate
Descripción:	Da de baja un empleado del sistema.
Parámetros:	- employee-id: Identificador del empleado a dar de baja.
Respuestas:	- 200 OK: El empleado fue desactivado correctamente.

Método:	getPaginatedEmployeesByDepartment
Endpoint: **Descripción:**	GET /employee/department/{department-id}/paginated Devuelve una lista paginada de empleados que pertenecen a un departamento específico.
Parámetros:	- department-id: Identificador del departamento del que se desean obtener los empleados. - page: Número de página (por defecto 0). - size: Número de elementos por página (por defecto 4).
Respuestas:	- 200 OK: Página de resultados Page<EmployeeResponseDto> con los empleados encontrados.

Endpoints – DepartmentController

Department

Método:	getDepartments
Endpoint:	GET /department
Descripción:	Obtiene la lista completa de departamentos registrados en el sistema.
Parámetros:	- No requiere.
Respuestas:	- 200 OK: Lista de objetos DepartmentResponseDto. - 204 NO CONTENT: No hay departamentos registrados en el sistema.

Método:	getDepartment
Endpoint:	GET /department/{department-id}
Descripción:	Devuelve los datos de un departamento específico a partir de su ID.
Parámetros:	- department-id: Identificador del departamento que se desea consultar.
Respuestas:	- 200 OK: Objeto DepartmentResponseDto con los datos del departamento. - 404 NOT FOUND: Si no se encuentra ningún departamento con ese ID.

Endpoints – VacationController

Vacation

Método:	vacationRegister
Endpoint:	POST /vacations
Descripción:	Registra una nueva solicitud de vacaciones en el sistema.
Parámetros:	- request: Objeto VacationRequestDto con los datos de la solicitud.
Respuestas:	- 201 CREATED: Vacaciones registradas exitosamente. Devuelve la URI del nuevo recurso.

Método:	getAllVacations
Endpoint:	GET /vacations
Descripción:	Recupera todas las colitiudes de vacaciones registradas en el sistema.
Parámetros:	- No requiere.
Respuestas:	- 200 OK: Lista de objetos VacationResponseDto. - 204 NO CONTENT: No hay vacaciones registradas.

Método:	getVacation
Endpoint:	GET /vacations/{vacation-id}
Descripción:	Obtiene los datos de una solicitud de vacaciones específica a partir de su ID.
Parámetros:	- vacation-id: Identificador de la solicitud.
Respuestas:	- 200 OK: Objeto VacationResponseDto. - 404 NOT FOUND: No se encontró la solicitud de vacaciones.

Método:	getVacationsOfEmployee
Endpoint:	GET /vacations/employee/{employee-id}
Descripción:	Recupera todas las solicitudes de vacaciones de un empleado.
Parámetros:	- employee-id: Identificador del empleado.
Respuestas:	- 200 OK: Lista de VacationResponseDto. - 204 NO CONTENT: El empleado no tiene solicitudes registradas.

Método:	getVacationsOfDepartment
Endpoint:	GET /vacations/department/{department-id}
Descripción:	Obtiene todas las vacaciones de los empleados de un departamento.
Parámetros:	- department-id: Identificador del departamento.
Respuestas:	- 200 OK: Lista de VacationResponseDto. - 204 NO CONTENT: No hay vacaciones registradas en el departamento indicado.

Método:	getPaginatedVacationsEmployee
Endpoint:	GET /vacations/employee/{employee-id}/paginated
Descripción:	Obtiene las vacaciones de un empleado de forma paginada, con opción de filtrar por el estado de las vacaciones.
Parámetros:	- employee-id: Identificador del empleado. - page: Número de página. - size: Número de elementos por página. - status (optional): Estado de las vacaciones.
Respuestas:	- 200 OK: Página de resultados Page<VacationResponseDto>.

Método:	getPaginatedVacationsDepartment
Endpoint:	GET /vacations/department/{department-id}/paginated
Descripción:	Devuelve una lista paginada de solicitudes de vacaciones asociadas a un departamento. También permite aplicar un filtro opcional por estado.
Parámetros:	- department-id: Identificador del departamento del que se quieren obtener las vacaciones. - page: Número de página. - size: Número de elementos por página. - status (optional): Estado de las vacaciones.
Respuestas:	- 200 OK: Página de resultados Page<VacationResponseDto>.

Método:	vacationStatusChange
Endpoint:	PATCH /vacations/{vacation-id}/status
Descripción:	Cambia el estado de una solicitud de vacaciones.

Parámetros:	- vacation-id: Identificador de la solicitud.
	- request:
	  o employeeId
	  o role (del que resuelve la solicitud)
	  o isApprove
	  o resolvedByName

Respuestas:	- 200 OK: El estado fue actualizado correctamente.

Estructura de paquetes del Backend

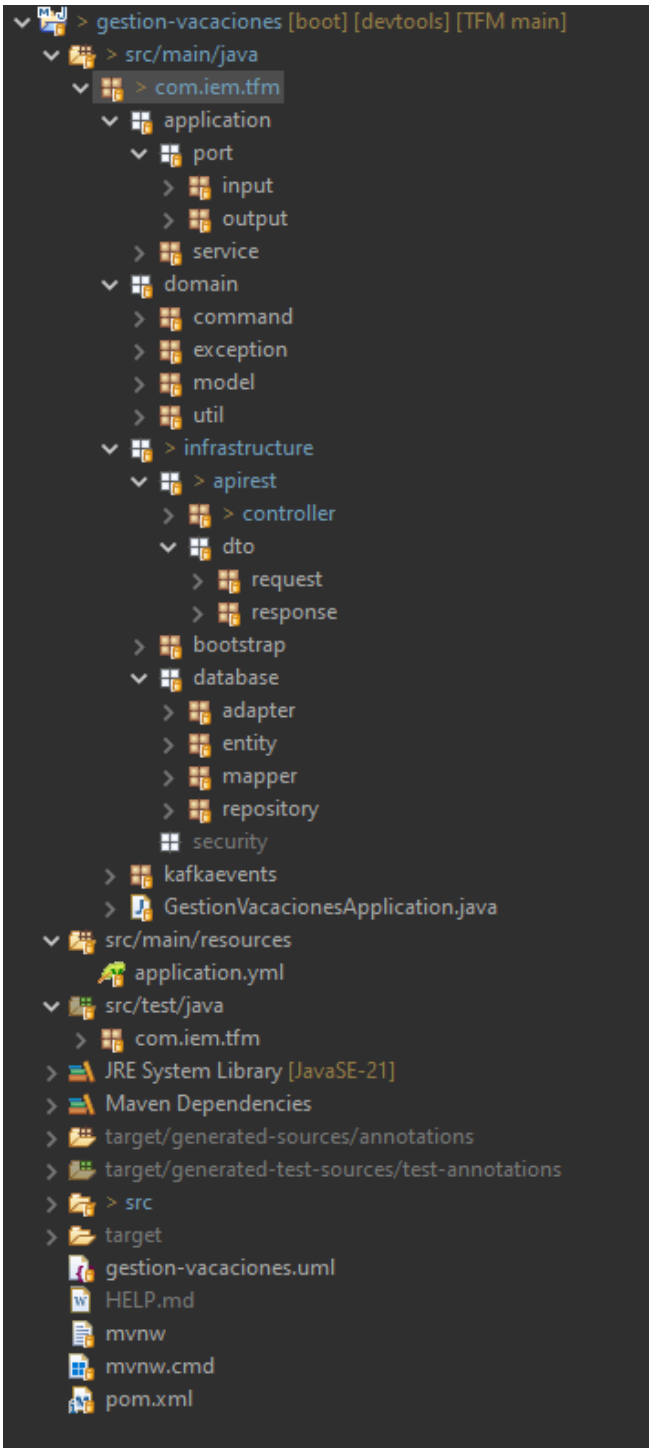

Figura 14 - Estructura Paquetes Backend

Frontend con Angular

La parte frontend de este proyecto se ha desarrollado utilizando Angular, empleado una estructura modular basado en los roles de la aplicación. El propósito principal es ofrecer una interfaz intuitiva para los empleados, supervisores y responsables de RRHH.

Las tecnologías remarcables que se han utilizado son las siguientes:

- RxJS: Utilizado para la comunicación reactiva entre componentes, como los filtros.
- Formularios Reactivos: Implementados en las vistas que requieren validación de datos.
- CSS personalizado: Los estilos de la aplicación se han implementado sin depender de frameworks externos. Se ha hecho uso de styles.css para implementar estilos globales.

Estructura del Proyecto

El código utiliza la siguiente estructura:

core – Elementos comunes e indispensables

- auth/pages: Página de login.
- guards: Protección de las rutas según el rol.
- services: Servicios generales que se responsabilizan de las llamadas HTTP y consumen los endpoints de la API REST.

modules – Se dividen por roles de usuario y cada uno agrupa las vistas correspondientes

- employee: Incluye calendar-page, dashboard, requests y vacation-list.
- supervisor: Incluye vista para gestionar empleados y las solicitudes de su departamento, más un dashboard y la página calendario del departamento.
- humar-resources: Dispone de un panel de administrador más completo, solicitudes globales, calendario globar y dashboard.

Shared – Componentes reutilizables

- calendar: Módulo visual del calendario compartido, implementado con la versión gratuita de FullCalendar.
- layout: Contenedor general de la interfaz que incluye cabecera, navegación, distribución de cierto contenido y footer.

Integración con API REST

Los servicios son los que se encargan de gestionar la comunicación con el backend a través de peticiones HTTP. Cada entidad de la aplicación cuenta con su propio servicio dedicado.

Los servicios se inyectan en los componentes correspondientes para mantener la lógica desacoplada ayudando en la mantenibilidad del código.

Control de Acceso y Rutas

El acceso a las rutas se restringe según el rol de cada usuario, que realmente es el rol que tienen como empleados dentro de la empresa. Se integra con el AuthService, que expone la sesión activa y su rol.

Los guards utilizados son:

- auth.guard.ts: Verifica que el usuario este autenticado antes de permitir el acceso a cualquier ruta privada, siendo el primer filtro que se aplica tras realizar el login.

- Employee.guard.ts: Restringe el acceso a las vistas del empleado, asegurando que solo usuarios con rol EMPLOYEE puedan acceder.
- Supervisor.guard.ts: Permite el acceso a rutas del supervisor si únicamente el usuario autentica tiene el rol ENCARGADO.
- Humar-resources.guard.ts: Solo permite acceder a las vistas de recursos humanos si el usuario autenticado tiene el rol RRHH.

Con esto permitimos un control en la navegación de la aplicación, seguridad en la interfaz y simplicidad en la gestión de los roles.

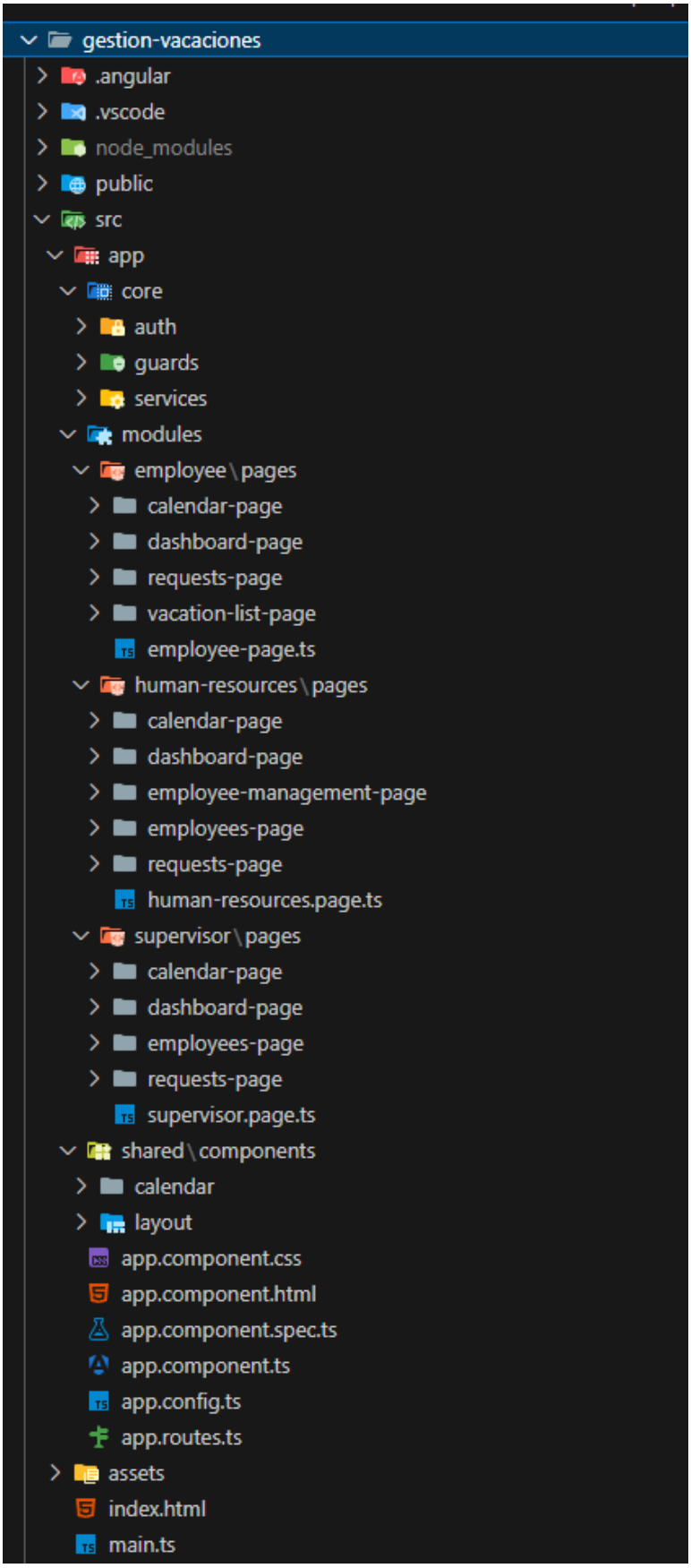

Figura 15 - Estructura Carpetas Frontend

Anotaciones Importantes durante el Desarrollo

Durante el desarrollo del proyecto han surgido diferentes decisiones, retos técnicos y mejoras que han sido determinantes para alcanzar una solución estable y bien estructurada. En este apartado se recogen las anotaciones más relevantes a nivel técnico y de diseño.

Se definieron las entidades principales (Employee, Department, User, Vacation) siguiendo los principios de la arquitectura hexagonal. En un primer momento no se iban a implementar excepciones dentro del dominio, pero se decidió incluirlas para reforzar las validaciones internas y evitar la creación de objetos que no cumplan con las reglas del negocio. Estas excepciones personalizadas (EmployeeDomainException, UserDomainException, etc.) se implementaron en el método build() del patrón Builder, asegurando que cualquier creación de entidad se haga correctamente.

En cuanto a la capa de aplicación, se optó por el uso de Commands en lugar de DTOs directos. Esto permite encapsular los datos de entrada, validar antes de ejecutar la lógica del caso de uso, y mantener un bajo acoplamiento entre capas. Se crearon puertos de entrada y salida para cada caso de uso, y se mantuvo la coherencia creando adaptadores que conectan los controladores con los servicios de aplicación.

Durante la integración de MapStruct surgieron problemas relacionados con el patrón Builder. Por defecto, MapStruct intenta crear objetos mediante un constructor vacío, lo que provocaba errores. La solución fue forzar a MapStruct a utilizar el builder definido, o realizar mapeos manuales en los casos más complejos.

En las pruebas realizadas con Postman se validaron correctamente las reglas del dominio. Se utilizaron IDs de tipo String para compatibilidad con MongoDB y se crearon departamentos ficticios de manera temporal para permitir probar el flujo de alta de empleados. Más adelante se conectaron correctamente con datos reales mediante mapeos completos.

Para los departamentos se decidió precargarlos al iniciar la aplicación, creando una clase tipo Bootstrap que los crea solo si no existen ya en la base de datos. Esto garantiza su disponibilidad sin duplicar datos.

Durante el desarrollo del endpoint de actualización de empleados (PUT) surgió un problema funcional: si un empleado cambia de departamento, sus solicitudes de vacaciones anteriores seguían asociadas al antiguo. Se modificó el servicio para que al actualizar el departamento del empleado, se actualicen también sus vacaciones correspondientes.

También se añadió un endpoint para dar de baja a un empleado. Esta acción no elimina sus vacaciones, sino que las mantiene en el histórico. En el frontend se añadió una lógica visual para mostrar que ese empleado se encuentra de baja.

Se implementó la carga masiva de empleados mediante ficheros Excel utilizando Apache POI. Como buena práctica, se gestionaron las excepciones con bloques try/catch, ya que el manejo de archivos puede lanzar errores controlados.

En la parte de login se detectó que el backend solo devolvía el rol del usuario (UserRoleEnum), lo cual no era suficiente para controlar las vistas. Se modificó el LoginResponseDto para incluir

también el rol del empleado (EmployeeRoleEnum), permitiendo así que Angular muestre correctamente las vistas según el perfil.

Se trabajó el layout global del frontend para mantener coherencia visual. Se creó un componente layout reutilizable y se definieron estilos globales. También se integraron guards para proteger las rutas según el rol.

Por último, se detectaron puntos de mejora como la implementación futura de Swagger para documentar la API, y el uso de PATCH en lugar de PUT en algunos endpoints para simplificar actualizaciones. En el frontend se anotó como mejora pendiente la necesidad de hacer que los filtros ignoren las tildes al buscar.

Referencias

En este apartado se menciona el contenido y páginas utilizadas para documentarse y redactar la memoria del proyecto.

LA PRINCIPAL REFERENCIA SON LOS APUNTES UTILIZADOS Y PROPORCIONADOS POR LA ESCUELA – IEM BUSINESS DIGITAL SCHOOL

https://medium.com/@oliveraluis11/arquitectura-hexagonal-con-spring-boot-parte-1-57b797eca69c - Patrón Arquitectura Hexagonal

https://www.ibm.com/es-es/topics/java-spring-boot - Java Spring Boot

https://vicentesg.com/ahorrar-tiempo-y-esfuerzo-en-desarrollos-java-con-lombok/ - Java Lombok

https://chatgpt.com/ - MapStruct

https://www.campusmvp.es/recursos/post/java-que-es-maven-que-es-el-archivo-pom-xml.aspx?srsltid=AfmBOopXO4ub-1u6SOlNdxEyNAEnf8fio6Fmr2Qp0cXE6D8s0K6PJp0o - Maven

https://talently.tech/blog/que-es-angular/ - Angular

https://www.unir.net/revista/ingenieria/que-es-typescript/ - TypeScript

https://www.mongodb.com/es/company/what-is-mongodb - MongoDB

https://www.arquitecturajava.com/el-modelo-vista-controlador-y-sus-responsabilidades/ - Modelo Vista Controlador

https://chatgpt.com/ - Preguntas generales

https://github.com/features/copilot - Documentación del código