DAGSS

Spring

ÓSCAR LESTÓN CASAIS | ESTEBAN CANAL BAUTISTA

Contenido

Introducción

Este proyecto consta en la parte backend en forma de Api Rest de un sistema de gestión en el ámbito de la medicina. De las historias de usuario suministradas se trabajaron las historias de administradores, médicos y pacientes. Sin embargo, al tener en cuenta la existencia de medicamentos, recetas y prescripciones nos vimos obligados a implementar también algunas historias de farmacias.

El Api Rest es capaz de completar con éxito las operaciones básicas de un Api Rest como son consulta, listado, creación, eliminado y actualización de recursos para las entidades básicas, así como para entidades como citas o prescripciones. Por otra banda delegamos responsabilidades en la lógica principal del programa como en la parte de FrontEnd para operaciones como la ordenación de los resultados o la gestión de la agenda del médico.

Estructura

Para la implementación dividimos el proyecto de la siguiente forma:

- Paquete Controladores
  - AdministradorController.java
  - FarmaciaController.java
  - PacienteController.java
  - UsuarioController.java
  - CentroSaludController.java
  - MedicamentoController.java
  - PrescripcionController.java
  - CitaController.java
  - MedicoController.java
  - RecetaController.java
- Paquete Daos
  - AdministradorDAO.java
  - FarmaciaDAO.java
  - PacienteDAO.java
  - UsuarioDAO.java
  - CentroSaludDAO.java
  - MedicamentoDAO.java
  - PrescripcionDAO.java
  - CitaDAO.java
  - MedicoDAO.java
  - RecetaDAO.java
- Paquete Entidades
  - Administrador.java
  - Farmacia.java
  - Paciente.java
  - Usuario.java
  - CentroSalud.java
  - Medicamento.java
  - Prescripcion.java
  - Cita.java
  - Medico.java
  - Receta.java
- Paquete Servicios
  - Genéricos
    - GenericoServicio.java
    - GenericoServicioImpl.java
  - AdministradorServicioImpl.java
  - FarmaciaServicioImpl.java
  - PacienteServicioImpl.java
  - UsuarioServicioImpl.java
  - CentroSaludServicioImpl.java
  - MedicamentoServicioImpl.java
  - PrescripcionServicioImpl.java
  - CitaServicioImpl.java
  - MedicoServicioImpl.java
  - RecetaServicioImpl.java
- Clase Principal
  - RecetasApplication.java

La clase principal está en el directorio raíz y ejecuta 2 funciones, crearCosas() y recuperarCosas() que testean que los objetos DAO funcionen de forma correcta introduciendo y recuperando datos de la base de datos.

Recopilamos 10 entidades JPA en el paquete entidades donde se encuentran también EstadoCita.java, EstadoGeneral.java, EstadoReceta.java y Rol.java que son enumerados que usan las entidades. También incluimos la clase Dirección que usaremos como atributo compuesto y no como entidad.

En el paquete DAOS tenemos agrupadas todas las interfaces DAO estas extienden JpaRepository<K, T>. Esto nos permite por un lado que JPA automatice las consultas de CRUD al mismo tiempo que nos ofrece un punto de expansión para operaciones como la consulta de todos los médicos que forman un centro de salud[1].

En el paquete Controladores incluimos cada uno de los RestControllers encargados de despachar las peticiones entrantes. Decidimos crear una clase por cada entidad.

[1] *Interface MedicoDAO { findByCentroSaludAsign(CentroSalud)}*

En el paquete Servicios incluimos una clase ServicioImplementacion por cada entidad la cual se crea como protección para los DAO ante posibles cambios en los controladores. Todas las implementaciones implementan la interfaz GenericoServicio<K,T>. Se crea también GenericoServicioImpl cuya idea inicial era reducir la redundancia de código producida por las implementaciones de los servicios (En todas se repiten los métodos propios del CRUD) y limitar las implementaciones especificas de servicios a aquellos que necesiten funciones específicas.[2] Finalmente no se utilizó para nuestra implementación.

Descripción del API

Administradores:

- Listar todos los administradores de la base de datos: GET /api/administrador
- Buscar administrador por ID (login): GET /api/administrador/{login}[3]
- Eliminar un administrador por ID (login): DELETE /api/administrador
- Crea un administrador: POST /api/administrador
    - Incluir JSON con los datos de Administrador y Usuario String *login*, String *password*, String *nombre*, String *email*
- Editar un administrador: PUT /api/administrador/{login}
    - Incluir JSON con los datos a editar de Administrador y Usuario a editar String *login*, String *password*, String *nombre*, String *email*

Centros de Salud:

- Listar todos los Centros de Salud de la base de datos: GET /api/centroSalud
- Crea un centroSalud: POST /api/ centroSalud
    - Incluir JSON con los datos de centroSalud String *nombre*, String *teléfono*, String *email*, Dirección *dirección*
- Eliminar un centro de salud: DELETE /api/centroSalud/{id}[4]
- Buscar centro de salud por id: GET /api/centroSalud/{id}
- Editar un centro de salud: PUT /api/centroSalud/{id}

Citas:

- Listar todas las citas de la base de datos: GET /api/cita
- Crear una nueva cita: POST /api/cita
    - Incluir JSON con los datos de la cita[5].
- Eliminar una cita: DELETE /api/cita/{id}
- Buscar cita por id: GET /api/cita/{id}
- Editar una cita: PUT /api/cita/{id}
- Listar las citas que tiene un paciente: GET /api/cita/paciente/{id}
- Listar las citas que tiene un médico: GET /api/cita/medico/{id} – NO IMPLEMENTADO (análogo a paciente)

Farmacias:

- Listar todas las farmacias de la base de datos: GET /api/farmacia
- Crear una nueva farmacia: POST /api/farmacia
    - Incluir JSON con los datos de la farmacia y usuario (ya que el farmacéutico-farmacia puede acceder al sistema).
- Eliminar una farmacia: DELETE /api/farmacia/{id}
- Buscar farmacia por id: GET /api/farmacia/{id}
- Editar una farmacia: PUT /api/farmacia/{id}
- Listar farmacias por nombre del establecimiento: GET/api/farmacia/establecimiento/{nombreEstablecimiento}
- Buscar farmacia por ID (login): GET /api/farmacia/login/{login}

---

[2] *Entiéndase una función especifica como aquella que no es "búsqueda por identificador, listar todos, editar por id, eliminar por id o crear por id"*

[3] *Login es un string único independiente del rol que identifica al usuario*

[4] *Id es numérico generado secuencialmente*

[5] *Idealmente para identificar al paciente y al médico usaríamos la clave, en nuestra implementación hace falta pasar todos los atributos del objeto*

- Listar farmacias por nombre de la provincia: GET /api/farmacia/provincia/{provincia}

Medicamentos:

- Listar todos los medicamentos de la base de datos: GET /api/medicamento
- Crea un medicamento: POST /api/ medicamento
    - Incluir JSON con los datos de medicamento
- Eliminar un medicamento: DELETE /api/medicamento/{id}[6]
- Buscar medicamento por id: GET /api/medicamento/{id}
- Editar un medicamento: PUT /api/medicamento/{id}
- Listar medicamento por nombre del fabricante: GET /api/ medicamento/fabricante/{fabricante}
- Listar medicamento por familia: GET /api/ medicamento/familia/{familia}
- Listar medicamento por nombre comercial: GET /api/ medicamento/nombreComercial/{nombreComercial}
- Listar medicamento por principio activo: GET /api/ medicamento/principioActivo/{principioActivo}

Prescripciones:

- Filtrar prescripciones por usuario mediante ID (login): GET /api/prescripcion/paciente/{id}
- Editar una prescripción: PUT /api/prenscripcion/{id}
- Buscar una prescripción mediante su ID {id}: GET /api/prenscripcion/{id}
- Borrar una prescripción: DELETE /api/prenscripcion/{id}
- Obtener todas las prescripciones: GET /api/prenscripcion

Pacientes:

- Editar un paciente mediante su ID (login): PUT /api/paciente/{login}
- Buscar un paciente por su ID {login}: GET /api/paciente/{login}
- Eliminar un paciente: DELETE /api/paciente/{login}
- Añadir un paciente: POST /api/paciente
- Obtener un listado de pacientes: GET /api/paciente

Médicos:

- Filtrar médicos mediante un Nombre(nombre): GET /api/medico/nombre/{nombre}
- Buscar un médico mediante su ID(Login): GET /api/medico/login/{login}
- Filtrar los médicos por centro de salud asignado: GET /api/medico/centroSalud/{centro}
- Edita un medico identificado por su login: PUT /api/medico/{login}
- Eliminar un medico identificado por su login: DELETE /api/medico/{login}
- Crear un médico: POST /api/medico/
    - JSON con todos los datos de médico y usuario

Tecnologías Utilizadas

Para la ejecución del proyecto utilizamos Java 11 y VS Code con las siguientes extensiones:

Pivotal.vscode-boot-dev-pack -> Para Spring Boot

vscjava.vscode-java-pack -> Para Java y Maven

[6] *Id es numérico generado secuencialmente*

Realización

En la realización del proyecto fuimos un grupo de 2 personas. Todas las tareas fueron elaboradas utilizando programación XP. Ambos usamos el IDE "VS Code" y mediante la extensión "Live Share" trabajamos en simultaneidad.

La primera etapa la dedicamos a crear todas las entidades JPA de forma que la base de datos se inicializase correctamente cada vez que iniciásemos la aplicación. Después de eso fueron los DAO que inicialmente dejamos extendiendo JpaRepository sin añadir funciones extra. A continuación, creamos los servicios que no tenían una función consolidada, pero sabíamos que no queríamos conectar los objetos DAO con los controladores directamente. En esa misma etapa se crean los controladores para dar soporte a las operaciones básica. En este punto probamos que el api se comporte correctamente y solucionamos errores que no se detectaron antes de estas pruebas.[7] Finalmente y con el conocimiento adquirido en las pruebas implementamos algunas funciones para satisfacer las historias de usuario que no satisfacen las operaciones que se heredan por defecto de JpaRepository.

Posible expansión

Como es normal en los proyectos de software, estos tienden a expandirse o requerir modificaciones. En este caso no es distinto, algunos posibles cambios son la adicción de más entidades como enfermeras o la especialización de centros médicos en clínicas o hospitales. Al añadir entidades es inevitable tener que definir sus características en un fichero. Sin embargo, sabemos que todas las entidades que se sumen a la base de datos tendrán una interfaz con sus operaciones, de las cuales cinco son comunes en todos los casos y un controlador. Ambos comportamientos son muy similares y aunque no queda reflejado en esta implementación en particular que presentamos, son candidatos para aplicar diversos patrones de diseño para evitar la redundancia de código (Visible en nuestro paquete de servicios).

Inicialmente pensamos en crear una interfaz genérica e instanciar a esta desde los controladores dándole la forma deseada en función del DAO, pero esta opción no nos permitía definir una interfaz propia para la implementación que añada métodos específicos como podría ser filtrar por fecha para una receta o buscar por número de colegiado para un médico. Una posible solución es aplicar un patrón template method. Por el lado de los controladores tenemos exactamente el mismo problema y al cual podríamos aplicar la misma solución.

Conclusiones

En general hemos podido ver que utilizar un framework, aunque sea para algo de juguete como esto, tiene unas ventajas que nos han gustado mucho como el echo de tener toda la infraestructura de la base de datos implícita en las propias clases de java abstrayendo toda la parte de SQL. Pero por otro lado si es cierto que entre tantas herramientas en varias ocasiones nos vimos entorpecidos para hacer cambios que sin utilizar un framework hubiesen sido cambios mínimos y sencillos de realizar. Parece que una vez más es la experiencia la que dirá en que escenarios utilizar un framework es algo beneficioso y cuando no y si se decide usarlo de qué modo hacerlo.

[7] Utilizamos Talent API Tester

```java
package es.uvigo.dagss.recetas.servicios;

import java.util.List;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import es.uvigo.dagss.recetas.daos.UsuarioDAO;

import es.uvigo.dagss.recetas.entidades.Usuario;

import es.uvigo.dagss.recetas.servicios.Genericos.GenericoLogeablesServicio;

@Service

public class UsuarioServicioImpl implements GenericoServicio<Usuario, String> {

   @Autowired

   UsuarioDAO dao;


   @Override

   @Transactional(readOnly = true)

   public List<Usuario> buscarTodos() {

      return dao.findAll();

   }


   @Override

   @Transactional(readOnly = true)

   public Optional<Usuario> buscarPorId(String login) {

      return dao.findById(login);

   }


   @Override

   public void eliminar(String login) {

      dao.deleteById(login);

   }

            […]

}
```

```
package es.uvigo.dagss.recetas.servicios;

[…]

public abstract class UsuarioServicioImpl implements GenericoServicio<Usuario, String>  {

[…Aquí se invocarían a los métodos comúnes…]

}
```

```
package es.uvigo.dagss.recetas.servicios;


import es.uvigo.dagss.recetas.entidades.Usuario;

import es.uvigo.dagss.recetas.servicios.Genericos.GenericoServicio;

@Service

public class UsuarioServicioImpl extends GenericoServicio<Usuario, String> {

[…Aquí se implementarían los métodos específicos …]


}
```