**INSTITUTO DE EDUCACIÓN SUPERIOR CIBERTEC**

**DIRECCIÓN ACADÉMICA**

**CARRERAS PROFESIONALES**

**CURSO : 4691** Lenguaje de programación II

**NOTA**

**PROFESOR :** YOVANI EDY QUINTEROS CAMAPAZA

**SEMESTRE :** 2024-33

**CICLO :** Cuarto

**SECCION :** T4FA

**GRUPO :** 01

**FECHA : 23/10/2024**

|  |
| --- |
| **ALUMNO (A) : Gallegos Yanarico, Jarem Joseph** |

**EXAMEN FINAL DE LABORATORIO**

**Consideraciones generales:**

* **Se considerará el orden, la claridad de las respuestas y las buenas prácticas**. **En cada pregunta consigne su repuesta (Código, comentarios, observaciones y resultados en captura de imágenes) adicionalmente la URL en GitHub.**
* Las preguntas deben resolverse de acuerdo con los conceptos discutidos o analizados en clase. Para ello, aplicará su propio criterio para dar una solución a los problemas planteados
* Para el desarrollo de la siguiente aplicación cree el proyecto con el nombre **LPII\_EF\_APELLIDO\_NOMBRE** y cree la estructura de carpetas necesarias

|  |
| --- |
| **LOGRO DE LA EVALUACION:**  Al término de la evaluación, el alumno implementa proceso de registro y listado de una tabla empleando JPA, realizando un correcto mapeado y una vista para su funcionalidad. |

**Consolidado**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Pregunta** | **Puntaje** | | **Llenar solo en caso de Recalificación justificada** | |
| **Máximo** | **Obtenido** | **Sustento** | **Puntaje** |
| **1** | **20** |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| **Nota Recalificada** | | | |  |

**Contexto:**

Un sistema de gestión para una clínica odontológica es una aplicación que facilita la administración y automatización de las actividades diarias de una clínica dental. Este tipo de sistema tiene como objetivo mejorar la eficiencia en la gestión de citas, expedientes médicos, inventarios de insumos, facturación y comunicación con los pacientes. Se encaga a Ud la implementación de una parte de uno de los módulos:

**Gestión de Pacientes**

* **Registro de pacientes:** El sistema permite registrar y mantener un historial detallado de los pacientes, incluyendo información personal (nombre, dirección, teléfono, correo electrónico) y datos médicos (historial odontológico, alergias, tratamientos previos).
* **Historial médico:** Guarda el historial completo de cada paciente, como los tratamientos recibidos, fecha, diagnostico, radiografías, y medico que atendió.

**Pregunta 1**

1. Crear un proyecto Spring Boot; Crear el modelo y crear el repositorio con las clases necesarias para la gestión de registro de pacientes. Suba su proyecto a Github y envíe la URL.

**Respuesta**: <https://github.com/JaremGallegos/ExamenFinal.git>

**(Explicación de Integración GitHub y Desarrollo del proyecto en las siguientes páginas)**

**Paso 1: Creación de un Proyecto Spring**

Personalmente, tengo problemas en el uso de Spring Tool Suite al crear proyectos y desarrollarlos, por lo que decidí utilizar Spring Initializr y Eclipse.

Como se puede observar en la captura, se añadieron las dependencias de Thymleaf, Spring Web, Spring Data JPA y el conector MySQL. Además, se integró los datos del proyecto correctamente, bajo el artefacto y el nombre.

**Captura de pantalla de computadora

Descripción generada automáticamente**

**Paso 2: Configuración del archivo properties**

Tras la realización de este primer paso, se abrió el proyecto generado en Eclipse y se realizó la configuración inicial del proyecto, teniendo que declararse en el archivo .properties, el nombre de usuario, contraseña, el conector, url y el acción hibernate para crear la base de datos.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Código del archivo .properties:**

spring.application.name=jaremgallegos

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=**jdbc:mysql:**//localhost:3306/dbodontologia

spring.datasource.username=root

spring.datasource.password=minombreesjarem1

spring.jpa.hibernate.ddl-auto=create

**Paso 2: Creación de capas de arquitectura**

Como se menciona en la propuesta se desea una arquitectura de servicio e integrado junto con un patrón mvc. De esta manera se comienza a crear los paquetes que albergarán a las clases para la creación del sistema de gestión. De esta manera definimos el paquete modelo, servicio, repositorio y controlador.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Paso 3: Creación de las clase modelo**

Tras la creación de los paquetes, se necesita de poder crear las clases que servirán para el mapeo de la entidad de JPA a una tabla de la base de datos. De esta manera, através de la identificación del caso propuesto, se identificaron a 7 entidades, las cuales se lograron tras uan normalización de las entidades padres. Estas son: Alergia, Diagnostico, Historial Odontológico, Odontólogo, Paciente, Radiografía y Tratamiento.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Tras la creación de estas clases que servirán como mapeo de la base de datos, será necesario integrar dentro de las dependencias .pom la librería lombok, esto permitirá a que nos podamos enfocar más en las relaciones entre las diversas entidades y sin la necesidad de preocuparnos en la agregación de métodos de acceso debido al encapsulamiento.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

De esta manera, a partir de los atributos mencionados en el caso propuesto y algunos deducidos se obtuvo el siguiente código de cada clase:

* **Clase Alergia**

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**package com.cibertec.examenfinal.jaremgallegos.modelo;**

**import java.util.List;**

**import jakarta.persistence.Entity;**

**import jakarta.persistence.GeneratedValue;**

**import jakarta.persistence.GenerationType;**

**import jakarta.persistence.Id;**

**import jakarta.persistence.ManyToMany;**

**import lombok.AllArgsConstructor;**

**import lombok.Getter;**

**import lombok.NoArgsConstructor;**

**import lombok.Setter;**

**import lombok.ToString;**

**@Getter**

**@Setter**

**@NoArgsConstructor**

**@AllArgsConstructor**

**@ToString**

**@Entity**

**public class Alergia {**

**@Id**

**@GeneratedValue(strategy = GenerationType.IDENTITY)**

**private Long id;**

**private String nombre;**

**private String descripcion;**

**@ManyToMany(mappedBy = "alergias")**

**private List<Paciente> pacientes;**

**}**

* **Clase Diagnostico**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**package** com.cibertec.examenfinal.jaremgallegos.modelo;

**import** **java.time.LocalDate**;

**import** **jakarta.persistence.Entity**;

**import** **jakarta.persistence.GeneratedValue**;

**import** **jakarta.persistence.GenerationType**;

**import** **jakarta.persistence.Id**;

**import** **jakarta.persistence.JoinColumn**;

**import** **jakarta.persistence.ManyToOne**;

**import** **lombok.AllArgsConstructor**;

**import** **lombok.Getter**;

**import** **lombok.NoArgsConstructor**;

**import** **lombok.Setter**;

**import** **lombok.ToString**;

**@Getter**

**@Setter**

**@NoArgsConstructor**

**@AllArgsConstructor**

**@ToString**

**@Entity**

**public** **class** **Diagnostico** {

**@Id**

**@GeneratedValue**(strategy = GenerationType.IDENTITY)

**private** Long id;

**private** String descripcion;

**private** LocalDate fecha;

**@ManyToOne**

**@JoinColumn**(name = "paciente\_id")

**private** Paciente paciente;

**@ManyToOne**

**@JoinColumn**(name = "odontologo\_id")

**private** Odontologo odontologo;

**@ManyToOne**

**@JoinColumn**(name = "tratamiento\_id")

**private** Tratamiento tratamiento;

}

* **Clase HistorialOdontologico**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**package** com.cibertec.examenfinal.jaremgallegos.modelo;

**import** **java.time.LocalDate**;

**import** **java.util.List**;

**import** **jakarta.persistence.Entity**;

**import** **jakarta.persistence.GeneratedValue**;

**import** **jakarta.persistence.GenerationType**;

**import** **jakarta.persistence.Id**;

**import** **jakarta.persistence.JoinColumn**;

**import** **jakarta.persistence.ManyToOne**;

**import** **jakarta.persistence.OneToMany**;

**import** **lombok.AllArgsConstructor**;

**import** **lombok.Getter**;

**import** **lombok.NoArgsConstructor**;

**import** **lombok.Setter**;

**import** **lombok.ToString**;

**@Getter**

**@Setter**

**@NoArgsConstructor**

**@AllArgsConstructor**

**@ToString**

**@Entity**

**public** **class** **HistorialOdontologico** {

**@Id**

**@GeneratedValue**(strategy = GenerationType.IDENTITY)

**private** Long id;

**private** String descripcion;

**private** LocalDate fecha;

**@ManyToOne**

**@JoinColumn**(name = "odontologo\_id")

**private** Odontologo odontologo;

**@ManyToOne**

**@JoinColumn**(name = "paciente\_id")

**private** Paciente paciente;

**@OneToMany**(mappedBy = "historialOdontologico")

**private** List<Radiografia> radiografias;

}

* **Clase Odontólogo**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**package** com.cibertec.examenfinal.jaremgallegos.modelo;

**import** **java.util.List**;

**import** **jakarta.persistence.Entity**;

**import** **jakarta.persistence.GeneratedValue**;

**import** **jakarta.persistence.GenerationType**;

**import** **jakarta.persistence.Id**;

**import** **jakarta.persistence.OneToMany**;

**import** **lombok.AllArgsConstructor**;

**import** **lombok.Getter**;

**import** **lombok.NoArgsConstructor**;

**import** **lombok.Setter**;

**import** **lombok.ToString**;

**@Getter**

**@Setter**

**@NoArgsConstructor**

**@AllArgsConstructor**

**@ToString**

**@Entity**

**public** **class** **Odontologo** {

**@Id**

**@GeneratedValue**(strategy = GenerationType.IDENTITY)

**private** Long id;

**private** String nombre;

**private** String especialidad;

**private** String telefono;

**private** String correo;

**private** String direccion;

**@OneToMany**(mappedBy = "odontologo")

**private** List<HistorialOdontologico> historiales;

**@OneToMany**(mappedBy = "odontologo")

**private** List<Diagnostico> diagnosticos;

}

* **Clase Paciente**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**package** com.cibertec.examenfinal.jaremgallegos.modelo;

**import** **java.time.LocalDate**;

**import** **java.util.List**;

**import** **jakarta.persistence.CascadeType**;

**import** **jakarta.persistence.Column**;

**import** **jakarta.persistence.Entity**;

**import** **jakarta.persistence.GeneratedValue**;

**import** **jakarta.persistence.GenerationType**;

**import** **jakarta.persistence.Id**;

**import** **jakarta.persistence.JoinColumn**;

**import** **jakarta.persistence.JoinTable**;

**import** **jakarta.persistence.ManyToMany**;

**import** **jakarta.persistence.OneToMany**;

**import** **lombok.AllArgsConstructor**;

**import** **lombok.Getter**;

**import** **lombok.NoArgsConstructor**;

**import** **lombok.Setter**;

**import** **lombok.ToString**;

**@Getter**

**@Setter**

**@NoArgsConstructor**

**@AllArgsConstructor**

**@ToString**

**@Entity**

**public** **class** **Paciente** {

**@Id**

**@GeneratedValue**(strategy = GenerationType.IDENTITY)

**private** Long id;

**private** String nombre;

**private** String direccion;

**private** String telefono;

**private** String correo;

**private** String sexo;

**@Column**(name = "fecha\_nacimiento")

**private** LocalDate fechaNacimiento;

**@OneToMany**(mappedBy = "paciente", cascade = CascadeType.ALL)

**private** List<HistorialOdontologico> historiales;

**@OneToMany**(mappedBy = "paciente", cascade = CascadeType.ALL)

**private** List<Diagnostico> diagnosticos;

**@ManyToMany**

**@JoinTable**(

name = "paciente\_alergia",

joinColumns = **@JoinColumn**(name = "paciente\_id"),

inverseJoinColumns = **@JoinColumn**(name = "alergia\_id")

)

**private** List<Alergia> alergias;

**@OneToMany**(mappedBy = "paciente", cascade = CascadeType.ALL)

**private** List<Tratamiento> tratamientos;

}

* **Clase Radiografía**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**package** com.cibertec.examenfinal.jaremgallegos.modelo;

**import** **java.time.LocalDate**;

**import** **jakarta.persistence.Entity**;

**import** **jakarta.persistence.GeneratedValue**;

**import** **jakarta.persistence.GenerationType**;

**import** **jakarta.persistence.Id**;

**import** **jakarta.persistence.JoinColumn**;

**import** **jakarta.persistence.ManyToOne**;

**import** **lombok.AllArgsConstructor**;

**import** **lombok.Getter**;

**import** **lombok.NoArgsConstructor**;

**import** **lombok.Setter**;

**import** **lombok.ToString**;

**@Getter**

**@Setter**

**@NoArgsConstructor**

**@AllArgsConstructor**

**@ToString**

**@Entity**

**public** **class** **Radiografia** {

**@Id**

**@GeneratedValue**(strategy = GenerationType.IDENTITY)

**private** Long id;

**private** String url;

**private** LocalDate fecha;

**private** String tipo;

**@ManyToOne**

**@JoinColumn**(name = "historial\_odontologico\_id")

**private** HistorialOdontologico historialOdontologico;

}

* **Clase Tratamiento**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**package** com.cibertec.examenfinal.jaremgallegos.modelo;

**import** **java.util.List**;

**import** **jakarta.persistence.Entity**;

**import** **jakarta.persistence.GeneratedValue**;

**import** **jakarta.persistence.GenerationType**;

**import** **jakarta.persistence.Id**;

**import** **jakarta.persistence.JoinColumn**;

**import** **jakarta.persistence.ManyToOne**;

**import** **jakarta.persistence.OneToMany**;

**import** **lombok.AllArgsConstructor**;

**import** **lombok.Getter**;

**import** **lombok.NoArgsConstructor**;

**import** **lombok.Setter**;

**import** **lombok.ToString**;

**@Getter**

**@Setter**

**@NoArgsConstructor**

**@AllArgsConstructor**

**@ToString**

**@Entity**

**public** **class** **Tratamiento** {

**@Id**

**@GeneratedValue**(strategy = GenerationType.IDENTITY)

**private** Long id;

**private** String nombre;

**private** String descripcion;

**private** String duracion;

**private** **double** costo;

**@ManyToOne**

**@JoinColumn**(name = "paciente\_id")

**private** Paciente paciente;

**@OneToMany**(mappedBy = "tratamiento")

**private** List<Diagnostico> diagnosticos;

}

**Paso 4: Generación de tablas a partir del mapeado**

Tras la creación de atributos de cada una de las clases, al ejecutar hibernete, este podrá mapear todas las clases y crear tablas para la gestión en la base de datos. De esta manera se generarán las siguientes tablas:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Cabe mencionar que para poder mapear cada una de las clases en esquemas de tablas, se debió de poder haber creado previamente la base de datos, de esta manera hibernete solo se encargará de poder crear las tablas y establecer los atributos de cada entidad. Por otro lado, tras la creación de las tablas, es necesario poder cambiar el parámetro hibernte.ddl-auto a update, de esta manera se evitará en un futuro que se borren datos tras la ejecución del proyecto:

Texto, Aplicación

Descripción generada automáticamente

**Paso 5: Creación de Repositorio en GitHub**

Dado de que se quiere poder manipular las versiones de nuestro proyecto, es importante poder crear un repositorio en GitHub para poder modificar y recuperar sin la necesidad de haber guardado de manera manual un proyecto backup.

Texto

Descripción generada automáticamente

Tras el ingreso del nombre del repositorio y una breve descripción, se crea el repositorio.

Captura de pantalla de un celular

Descripción generada automáticamente

Tras la creación, se ingresa a la consola de git, para poder ejecutar los comandos y poder realizar un commit. Primeramente, buscamos el directorio raíz de nuestro proyecto.

Texto

Descripción generada automáticamente

Tras esto, se ingresa el comando de git init

Texto

Descripción generada automáticamente

Luego ejecutamos el comando “git add .” para incluir todos nuestros archivos al commit.

Texto

Descripción generada automáticamente

Luego ejecutamos el comando git commit -m “mensaje” para poder lanzar nuestro primer commit.

Texto

Descripción generada automáticamente

Tras realizar esto, se ejecuta el comando de git branch -M main, para poder cambiar el nombre de la rama principal “master” a “main”. Además, indicamos con **git remote add origin** el repositorio en donde se desea guardar el proyecto.

Texto

Descripción generada automáticamente

Finalmente, para poder agregar nuestro commit a nuestro repositorio, utilizamos el comando “git push” para poder subir nuestro proyecto a la rama principal

Texto

Descripción generada automáticamente

De esta manera se comprueba el cargado de nuestro proyecto a nuestro repositorio

Captura de pantalla con la imagen de una pantalla

Descripción generada automáticamente

<https://github.com/JaremGallegos/ExamenFinal.git>

1. Crear el controlador y la capa servicio que permita realizar las siguientes gestiones:
   1. Registro de pacientes y lista de pacientes
   2. Registro de atenciones y listas de atenciones
   3. Registro de historial medico de pacientes y por paciente

**Paso 1: Creación de capas de arquitectura**

Dado que ya se creó el paquete de modelo, para poder desarrollar con la arquitectura planteada, será necesario poder crear los paquetes Repository, Service y Controller para poder crear la lógica y acciones dentro del sistema planteado.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

De esta manera se plantea la creación de las siguientes clases Repository:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Clases Repository creadas:**

**package** com.cibertec.examenfinal.jaremgallegos.repositorio;

**import** **org.springframework.data.jpa.repository.JpaRepository**;

**import** **org.springframework.stereotype.Repository**;

**import** **com.cibertec.examenfinal.jaremgallegos.modelo.Diagnostico**;

**@Repository**

**public** **interface** **DiagnosticoRepository** **extends** JpaRepository<Diagnostico, Long> {

}

**public** **interface** **OdontologoRepository** **extends** JpaRepository<Odontologo, Long> {

}

**@Repository**

**public** **interface** **PacienteRepository** **extends** JpaRepository<Paciente, Long> {

}

**@Repository**

**public** **interface** **TratamientoRepository** **extends** JpaRepository<Tratamiento, Long>{

}

**@Repository**

**public** **interface** **HistorialOdontologicoRepository** **extends** JpaRepository<HistorialOdontologico, Long> {

List<HistorialOdontologico> **findByPacienteId**(Long pacienteId);

}

De esta manera se plantea la creación de las siguientes clases Service:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

**Clases Service creadas:**

**package** com.cibertec.examenfinal.jaremgallegos.servicio;

**import** **java.util.List**;

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

**import** **org.springframework.stereotype.Service**;

**import** **com.cibertec.examenfinal.jaremgallegos.modelo.Diagnostico**;

**import** **com.cibertec.examenfinal.jaremgallegos.repositorio.DiagnosticoRepository**;

**@Service**

**public** **class** **DiagnosticoService** {

**@Autowired**

**private** DiagnosticoRepository diagnosticoRepository;

**public** List<Diagnostico> **listarDiagnosticos**() {

**return** diagnosticoRepository.findAll();

}

**public** Diagnostico **registrarDiagnostico**(Diagnostico diagnostico) {

**return** diagnosticoRepository.save(diagnostico);

}

}

**@Service**

**public** **class** **HistorialOdontologicoService** {

**@Autowired**

**private** HistorialOdontologicoRepository historialRepository;

**public** List<HistorialOdontologico> **listarHistoriales**() {

**return** historialRepository.findAll();

}

**public** List<HistorialOdontologico> **listarHistorialesPorPaciente**(Long pacienteId) {

**return** historialRepository.findByPacienteId(pacienteId);

}

**public** HistorialOdontologico **registrarHistorial**(HistorialOdontologico historial) {

**return** historialRepository.save(historial);

}

}

**@Service**

**public** **class** **OdontologoService** {

**@Autowired**

**private** OdontologoRepository odontologoRepository;

**public** List<Odontologo> **listarTodos**() {

**return** odontologoRepository.findAll();

}

**public** Optional<Odontologo> **obtenerPorId**(Long id) {

**return** odontologoRepository.findById(id);

}

**public** Odontologo **guardar**(Odontologo odontologo) {

**return** odontologoRepository.save(odontologo);

}

**public** **void** **eliminar**(Long id) {

odontologoRepository.deleteById(id);

}

}

**@Service**

**public** **class** **PacienteService** {

**@Autowired**

**private** PacienteRepository pacienteRepository;

**public** List<Paciente> **listarPacientes**() {

**return** pacienteRepository.findAll();

}

**public** Paciente **registrarPaciente**(Paciente paciente) {

**return** pacienteRepository.save(paciente);

}

**public** Optional<Paciente> **obtenerPacientePorId**(Long id) {

**return** pacienteRepository.findById(id);

}

}

**@Service**

**public** **class** **TratamientoService** {

**@Autowired**

**private** TratamientoRepository tratamientoRepository;

**public** List<Tratamiento> **listarTratamientos**() {

**return** tratamientoRepository.findAll();

}

**public** Tratamiento **registrarTratamiento**(Tratamiento tratamiento) {

**return** tratamientoRepository.save(tratamiento);

}

}

De esta manera se plantea la creación de las siguientes clases Controller Rest y Vista, que se utilizarán para poder implementar la lógica de negocio:

Texto

Descripción generada automáticamente

**Clases Controller Rest creadas:**

**@RestController**

**@RequestMapping**("/api/diagnosticos")

**public** **class** **DiagnosticoController** {

**@Autowired**

**private** DiagnosticoService diagnosticoService;

**@GetMapping**

**public** List<Diagnostico> **listarDiagnosticos**() {

**return** diagnosticoService.listarDiagnosticos();

}

**@PostMapping**

**public** Diagnostico **registrarDiagnostico**(**@RequestBody** Diagnostico diagnostico) {

**return** diagnosticoService.registrarDiagnostico(diagnostico);

}

}

**@RestController**

**@RequestMapping**("/api/historiales")

**public** **class** **HistorialOdontologicoController** {

**@Autowired**

**private** HistorialOdontologicoService historialService;

**@GetMapping**

**public** List<HistorialOdontologico> **listarHistoriales**() {

**return** historialService.listarHistoriales();

}

**@GetMapping**("/paciente/{pacienteId}")

**public** List<HistorialOdontologico> **listarHistorialesPorPaciente**(**@PathVariable** Long pacienteId) {

**return** historialService.listarHistorialesPorPaciente(pacienteId);

}

**@PostMapping**

**public** HistorialOdontologico **registrarHistorial**(**@RequestBody** HistorialOdontologico historial) {

**return** historialService.registrarHistorial(historial);

}

}

**@RestController**

**@RequestMapping**("/api/odontologos")

**public** **class** **OdontologoController** {

**@Autowired**

**private** OdontologoService odontologoService;

**@GetMapping**

**public** List<Odontologo> **listarOdontologos**() {

**return** odontologoService.listarTodos();

}

**@GetMapping**("/{id}")

**public** ResponseEntity<Odontologo> **obtenerOdontologoPorId**(**@PathVariable** Long id) {

Optional<Odontologo> odontologo = odontologoService.obtenerPorId(id);

**return** odontologo.map(**ResponseEntity:**:ok)

.orElseGet(() -> ResponseEntity.notFound().build());

}

**@PostMapping**

**public** Odontologo **crearOdontologo**(**@RequestBody** Odontologo odontologo) {

**return** odontologoService.guardar(odontologo);

}

**@PutMapping**("/{id}")

**public** ResponseEntity<Odontologo> **actualizarOdontologo**(**@PathVariable** Long id, **@RequestBody** Odontologo detallesOdontologo) {

Optional<Odontologo> odontologo = odontologoService.obtenerPorId(id);

**if** (odontologo.isPresent()) {

Odontologo odontologoActualizado = odontologo.get();

odontologoActualizado.setNombre(detallesOdontologo.getNombre());

odontologoActualizado.setEspecialidad(detallesOdontologo.getEspecialidad());

odontologoActualizado.setTelefono(detallesOdontologo.getTelefono());

odontologoActualizado.setCorreo(detallesOdontologo.getCorreo());

odontologoActualizado.setDireccion(detallesOdontologo.getDireccion());

**return** ResponseEntity.ok(odontologoService.guardar(odontologoActualizado));

} **else** {

**return** ResponseEntity.notFound().build();

}

}

**@DeleteMapping**("/{id}")

**public** ResponseEntity<Void> **eliminarOdontologo**(**@PathVariable** Long id) {

**if** (odontologoService.obtenerPorId(id).isPresent()) {

odontologoService.eliminar(id);

**return** ResponseEntity.noContent().build();

} **else** {

**return** ResponseEntity.notFound().build();

}

}

}

**@RestController**

**@RequestMapping**("/api/pacientes")

**public** **class** **PacienteController** {

**@Autowired**

**private** PacienteService pacienteService;

**@Autowired**

**private** ReporteService reporteService;

**@GetMapping**

**public** List<Paciente> **listarPacientes**() {

**return** pacienteService.listarPacientes();

}

**@PostMapping**

**public** ResponseEntity<String> **registrarPaciente**(**@ModelAttribute** Paciente paciente) {

Paciente nuevoPaciente = pacienteService.registrarPaciente(paciente);

String mensaje = reporteService.exportReportePDF(nuevoPaciente);

**return** ResponseEntity.ok("Paciente registrado" + mensaje);

}

**@GetMapping**("/{id}")

**public** ResponseEntity<Paciente> **obtenerPacientePorId**(**@PathVariable** Long id) {

Optional<Paciente> pacienteOpt = pacienteService.obtenerPacientePorId(id);

**return** pacienteOpt.map(**ResponseEntity:**:ok)

.orElse(ResponseEntity.notFound().build());

}

}

**@RestController**

**@RequestMapping**("/api/atenciones")

**public** **class** **TratamientoController** {

**@Autowired**

**private** TratamientoService tratamientoService;

**@Autowired**

**private** PacienteService pacienteService;

**@GetMapping**

**public** List<Tratamiento> **listarTratamientos**() {

**return** tratamientoService.listarTratamientos();

}

**@PostMapping**

**public** Tratamiento **registrarTratamiento**(**@ModelAttribute** Tratamiento tratamiento) {

Optional<Paciente> paciente = pacienteService.obtenerPacientePorId(tratamiento.getPaciente().getId());

**if** (paciente.isPresent()) {

tratamiento.setPaciente(paciente.get());

**return** tratamientoService.registrarTratamiento(tratamiento);

} **else** {

**throw** **new** **RuntimeException**("Paciente no encontrado");

}

}

}

**Clases Controller Vista creadas:**

**@Controller**

**@RequestMapping**("/historiales")

**public** **class** **HistorialOdontologicoVistaController** {

**@Autowired**

**private** DiagnosticoService diagnosticoService;

**@GetMapping**

**public** String **listarHistoriales**(Model model) {

model.addAttribute("historialOdontologico", **new** HistorialOdontologico());

model.addAttribute("diagnosticos", diagnosticoService.listarDiagnosticos());

**return** "historial";

}

}

**@Controller**

**@RequestMapping**("/odontologos")

**public** **class** **OdontologoVistaController** {

**@Autowired**

**private** OdontologoService odontologoService;

**@GetMapping**

**public** String **listarOdontologos**(Model model) {

List<Odontologo> odontologos = odontologoService.listarTodos();

model.addAttribute("odontologos", odontologos);

**return** "odontologos/lista";

}

**@GetMapping**("/nuevo")

**public** String **mostrarFormularioDeNuevoOdontologo**(Model model) {

model.addAttribute("odontologo", **new** Odontologo());

**return** "odontologos/formulario";

}

**@PostMapping**

**public** String **guardarOdontologo**(**@ModelAttribute** Odontologo odontologo) {

odontologoService.guardar(odontologo);

**return** "redirect:/odontologos";

}

**@GetMapping**("/editar/{id}")

**public** String **mostrarFormularioDeEditarOdontologo**(**@PathVariable** Long id, Model model) {

Odontologo odontologo = odontologoService.obtenerPorId(id).orElseThrow(() -> **new** IllegalArgumentException("Odontologo no encontrado: " + id));

model.addAttribute("odontologo", odontologo);

**return** "odontologos/formulario";

}

**@PostMapping**("/editar/{id}")

**public** String **actualizarOdontologo**(**@PathVariable** Long id, **@ModelAttribute** Odontologo odontologo) {

odontologo.setId(id);

odontologoService.guardar(odontologo);

**return** "redirect:/odontologos";

}

**@GetMapping**("/eliminar/{id}")

**public** String **eliminarOdontologo**(**@PathVariable** Long id) {

odontologoService.eliminar(id);

**return** "redirect:/odontologos";

}

}

**@Controller**

**@RequestMapping**("/pacientes")

**public** **class** **PacienteVistaController** {

**@Autowired**

**private** PacienteService pacienteService;

**@GetMapping**

**public** String **listarPacientes**(Model model) {

List<Paciente> pacientes = pacienteService.listarPacientes();

model.addAttribute("pacientes", pacientes);

model.addAttribute("paciente", **new** Paciente());

**return** "pacientes";

}

}

**@Controller**

**@RequestMapping**("/atenciones")

**public** **class** **TratamientoVistaController** {

**@Autowired**

**private** TratamientoService tratamientoService;

**@Autowired**

**private** PacienteService pacienteService;

**@GetMapping**

**public** String **listarTratamientos**(Model model) {

model.addAttribute("tratamientos", tratamientoService.listarTratamientos());

model.addAttribute("nuevoTratamiento", **new** Tratamiento());

model.addAttribute("pacientes", pacienteService.listarPacientes());

**return** "atenciones";

}

}

**La creación de todos los controladores modelos y demás se crearon en el proyecto, si no se muestran se debe a la gran cantidad de elementos**

**Paso 2: Creación de vista para poder registrar y listar un paciente**

Para el logro de la creación de la vista, se integró archivos html con Bootstrap dentro de la carpeta templates. Por otro lado también se fragmento la parte de cabecera y footer para poder integrarlo como componente.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

De esta manera se creó el siguiente dashboard:

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

En este caso se integrarán datos para poder registrar a un paciente

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

**Paso 3: Creación de vista para poder registrar y listar una atención**

De la misma forma con el caso anterior, para poder registrar una atención, se tiene que haber creado un paciente, de esta manera para mayor facilidad, se estableció una barra de opciones.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Para continuar con el caso anterior, se seleccionará al paciente recientemente registrado

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Como se puede observar tras el ingreso y asignación del paciente se crea la cita del paciente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Paso 4: Registrar historial médico de pacientes por pacientes**

Para poder realizar la gestión del historial médico del paciente, primeramente, se deberá de poder registrar un odontólogo y el diagnóstico.

Interfaz de usuario gráfica, Texto, Aplicación, Tabla

Descripción generada automáticamente

Lo que se pretende es poder registrar el historial médico del paciente al ingresar una descripción, seleccionar una fecha y asignar a un paciente, diagnóstico, odontólogo a cargo y la dirección de una radiografía

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Tras haber realizado esto, se podrá mostrar el registro del historial médico, juntamente también será capaz de poder ingresar a la radiografía.

Una pantalla de televisión encendida con la imagen de una mano

Descripción generada automáticamente con confianza media

**[Extras]**

Dado a que es necesario asignar un odontólogo y un diagnóstico, se crearon módulos para cada uno

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

1. Al pulsar la opción **registrar** deberá guardar los datos de los pacientes y en caso de éxito, se debe generar una **constancia PDF** (constanciaApellido.jasper) del paciente registrado (mascota).

**Paso 1: Creación de Reporte Jasper**

Para poder realizar un reporte Jasper, primeramente, se debe conectar con la base de datos, tras la realización de este paso, se crea el proyecto Jasper y se integra la sentencia SQL para el listado de los pacientes (SELECT \* FROM paciente)

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Tras esto se diseño los datos que se quiere poner en el reporte, en este caso se desea integrar los datos de ID, Nombre, Dirección y Correo

Tabla

Descripción generada automáticamente

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

Tras esto se copia el código XML generado del reporte diseñado, teniendo que integrarse dentro de un archivo plane para poder generar reportes

Una captura de pantalla de una computadora

Descripción generada automáticamente

Para poder lograr esto, es necesario poder integrar las dependencias Jasper para poder generar documentos pdf del reporte e integrarlo en Spring

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Tras esto, es necesario poder crear la clase de ReporteService para poder integrarlo dentro del método post que realizará el registro del paciente.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Dado que tratamos de apartar la lógica de negocio con la vista, en nuestro controlar Rest establecemos la verificación de creación

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Paso 2: Prueba de registro de paciente y generación de reporte**

Para uso prácticos se integran los datos del paciente Jose Maria, para poder generar un reporte, al registrar, deberá mostrar un mensaje de éxito y mostrarnos la dirección en donde se guardó el reporte pdf.

Interfaz de usuario gráfica, Aplicación, Tabla

Descripción generada automáticamente

Se generó con éxito

Texto

Descripción generada automáticamente con confianza media

Dirección del Reporte generado

Interfaz de usuario gráfica

Descripción generada automáticamente

Tras haber generado el reporte con éxito se muestra los datos actualizados de la constancia de registro del paciente.

Interfaz de usuario gráfica, Aplicación, Tabla

Descripción generada automáticamente

[Problema]

Al tratar de generar una constancia de registro personal, este no muestra los datos, generando un documento en blanco del paciente que se a registrado, por lo que se pretende poder corregir en ese sentido el uso de este reporte.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Rúbrica** | | | | |
| **Pregunta** | **Excelente** | **Bueno** | **Regular** | **Deficiente** |
| 1 | Implementa adecuadamente el proceso de registro de datos, muestra la constancia en PDF (18p) y  El reporte gráfico  (20 p) | Implementa los proceso de registro, listados con la estructura solicitada.  (14 p) | Implementa sólo la página principal, y el modelo, repositorio; mostrando los datos de la tabla (10p) | No implementa ninguna funcionalidad. |