MANUAL TÉCNICO

ADU7EX01_EL

Eric Lachcik
2n Dam
05/05/2025
Accés a Dades

Index

Arquitectura de la Solución

Se describirán las tecnologías utilizadas dentro del proyecto y se dará a conocer que contiene cada carpeta dentro del proyecto.

Tecnologías utilizadas

- **Java 23** — Lenguaje de programación principal del proyecto.
- **Spring Boot** — Framework que facilita la creación de aplicaciones web y REST.
- **Maven** — Herramienta de gestión de dependencias y compilación.
- **MySQL** — Sistemas de gestión de bases de datos utilizados.
- **JPA** — Para la persistencia de entidades y acceso a la base de datos.
- **Thymeleaf** — Motor de plantillas para renderizar vistas HTML.
- **MongoDB** — Para la ingesta de datos no relacionales.

Capas del Proyecto

- Java - Carpeta Principal donde están las carpetas principales
    - Clases - Carpeta donde se encuentran las clases usadas para la base de datos donde el JPA las lee y crea las tablas sino están creadas en tu base de datos.
    - Configs - Carpeta donde se encuentra un clase para configurar el manejo de fechas del ThymeLeaf.
    - Controllers - Carpeta donde se encuentran los controladores necesarios para que la aplicación funcione, estos se encargan de recibir y responder a las peticiones HTTP.
    - Ingestors - Carpeta donde se encuentran los ingestors del la funcionalidad de Batch y Mongo.
        - Batch - En esta carpeta se encuentra la clase que maneja el procesado de archivos xml y json para después inyectar las reservas que se encuentran en ellas dentro de la base de datos.
        - Mongo - En esta carpeta se encuentra la clase que maneja el procesado de colección de Mongo, la cual los lee y después inyecta las reservas dentro de la base de datos.

- Repositorys - Carpeta donde se encuentran los repositorios de todas las clases del proyecto.

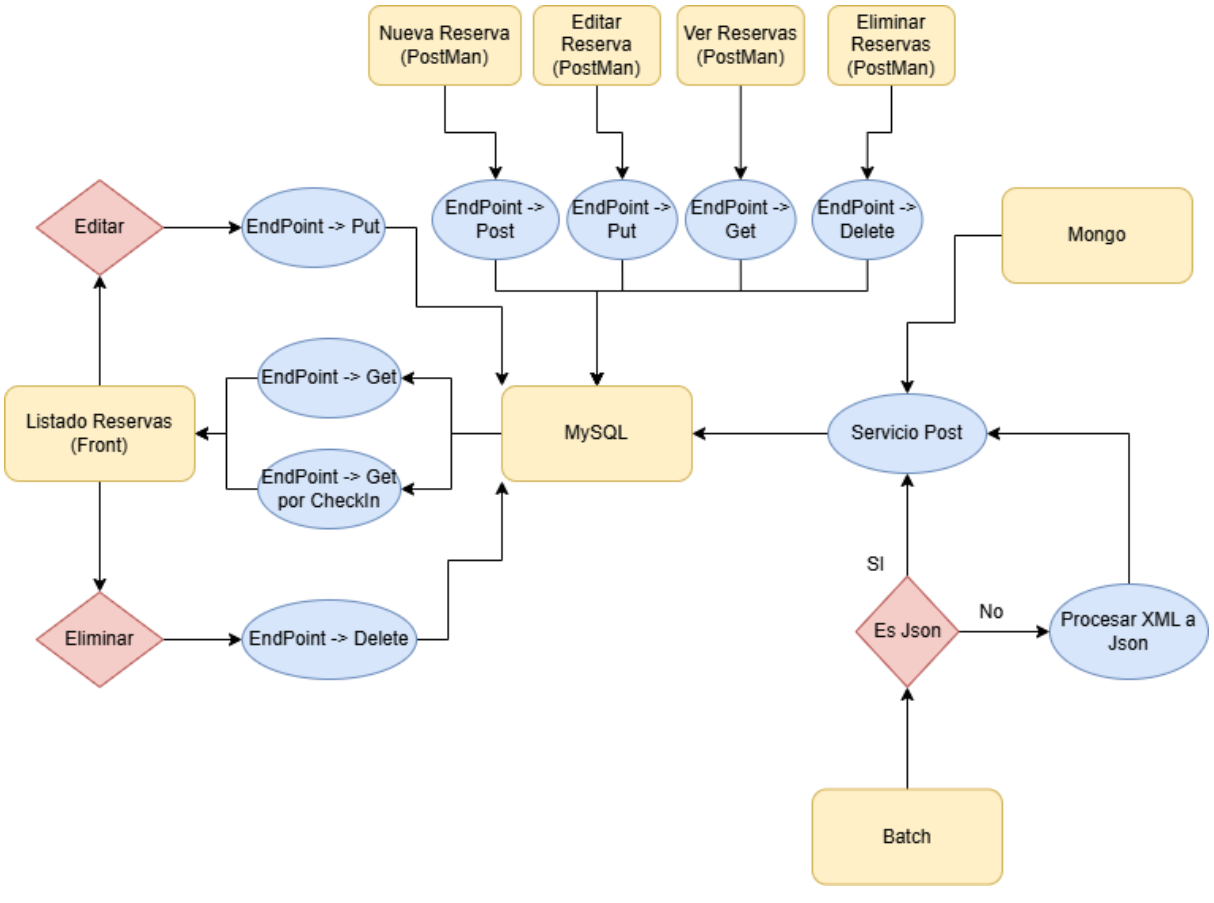
- Services - Esta carpeta contiene una clase que solo tiene un método que está en las clases de Batch y Mongo, el cual se encarga de hacer el POST con los datos procesados de las reservas a la base de datos.

- Resources - SubCarpeta que contiene la principal configuración del proyecto

  - Reservas - Estructura de Carpetas que forma parte de la inyección de Reservas mediante Batch
    - Correctes - Carpeta donde se almacenan las reservas que han tenido éxito en ser procesadas.
    - errores - Carpeta donde se almacenan las reservas que habran tenido algún error cuando se estaban procesando.
    - Pendents - Carpeta donde se almacenan las reservas antes de ser procesadas.

  - Templates - Carpeta donde se encuentran las diferentes carpetas correspondientes a las diferentes tipos de páginas que había en el FrontEnd.
    - reservas- Carpeta destinada a la página de Reservas con lista.html y editar.html.

  - Static - Carpeta necesaria, sino se producen errores dentro del proyecto.

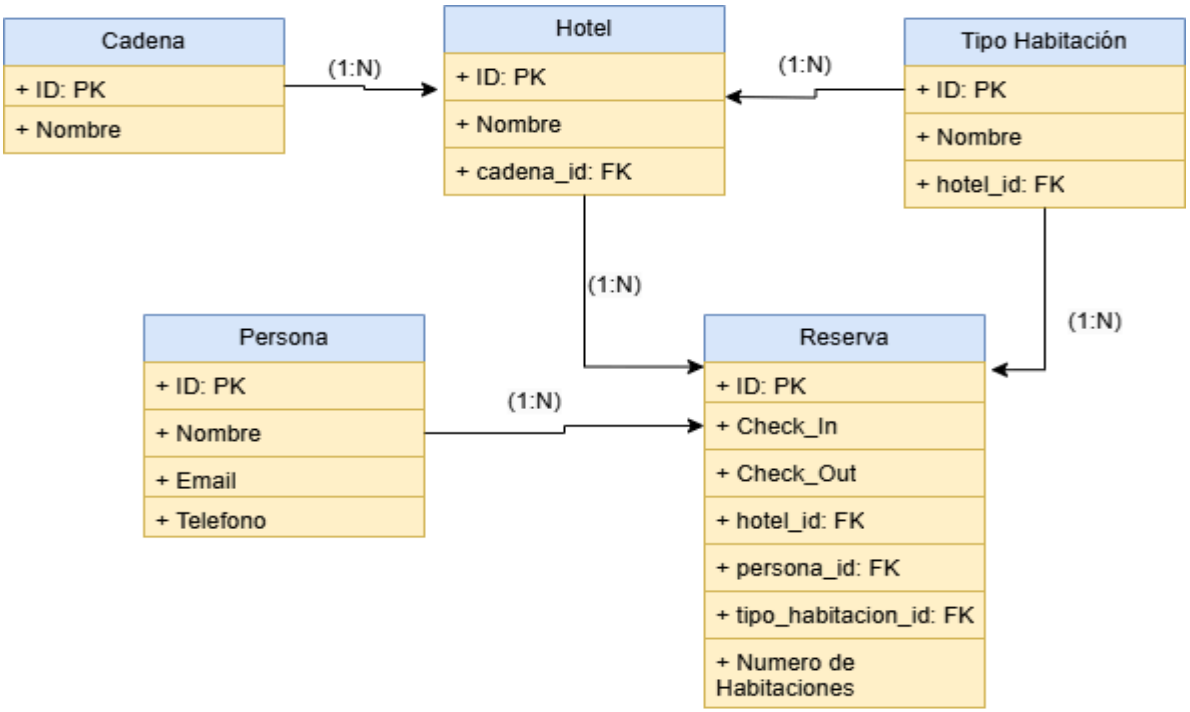


Definición de la estructura de Datos

Las DDL de las tablas dentro de mi base de datos es esta

```
CREATE TABLE cadena (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255) NOT NULL
);

CREATE TABLE hotel (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255) NOT NULL,
    cadena_id INT,
    FOREIGN KEY (cadena_id) REFERENCES cadena(id)
);

CREATE TABLE tipo_habitacion (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255) NOT NULL,
    hotel_id INT,
    FOREIGN KEY (hotel_id) REFERENCES hotel(id)
);

CREATE TABLE persona (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    telefono VARCHAR(50)
);


CREATE TABLE reserva (
    id INT AUTO_INCREMENT PRIMARY KEY,
    check_in DATE NOT NULL,
    check_out DATE NOT NULL,
    hotel_id INT,
    persona_id INT,
    num_habitaciones INT,
    tipo_habitacion_id INT,
    FOREIGN KEY (hotel_id) REFERENCES hotel(id),
    FOREIGN KEY (persona_id) REFERENCES persona(id),
    FOREIGN KEY (tipo_habitacion_id) REFERENCES tipo_habitacion(id));
```

Y asi seria la Entidad relacion de cada tabla



Definición de los métodos de consulta

Cadena Controller

Método: getAllCadenas()

- **Ruta / Endpoint**: GET /cadenas
- **Descripción**: Recupera la lista completa de todas las cadenas almacenadas en la base de datos.
- **Respuesta**: List<Cadena> en formato JSON
- **Lógica Interna:** `return cadenaRepository.findAll();`

Método: crearCadena()

- **Ruta / Endpoint:** POST /cadenas
- **Descripción:** Crea y persiste una nueva cadena a partir de los datos enviados en el cuerpo de la petición.
- **Respuesta:** ResponseEntity<Cadena> con el objeto creado y código HTTP 201 Created
- **Lógica interna:**

```
Cadena nouCadena = cadenaRepository.save(cadena);
return new ResponseEntity<>(nouCadena, HttpStatus.CREATED);
```

Hotel Controller

Método: getAllHoteles()

- **Ruta / Endpoint:** GET /hoteles
- **Descripción:** Recupera la lista completa de todos los hoteles almacenados en la base de datos.
- **Respuesta:** List<Hotel> en formato JSON
- **Lógica interna:** `return hotelRepository.findAll();`

Método: crearHotel()

- **Ruta / Endpoint:** POST /hoteles
- **Descripción:** Crea y persiste un nuevo hotel a partir de los datos enviados en el cuerpo de la petición.
- **Respuesta:** ResponseEntity<Hotel> con el objeto creado y código HTTP 201 Created.
- **Lógica interna:**
```
Hotel nouHotel = hotelRepository.save(hotel);
return new ResponseEntity<>(nouHotel, HttpStatus.CREATED);
```

Persona Controller

Método: getAllPersonas()

- **Ruta / Endpoint:** GET /personas
- **Descripción:** Recupera la lista completa de todas las personas almacenadas en la base de datos.
- **Respuesta:** List<Persona> en formato JSON
- **Lógica interna:** `return personaRepository.findAll();`

Método: crearPersona()

- **Ruta / Endpoint:** POST /personas
- **Descripción:** Crea y persiste una nueva persona a partir de los datos enviados en el cuerpo de la petición.
- **Respuesta:** ResponseEntity<Persona> con el objeto creado y código HTTP 201 Created.
- **Lógica interna:** `Persona nouPersona =personaRepository.save(persona);`

Reserva Controller

Método: getAllReservas()

- **Ruta / Endpoint:** GET /reservas
- **Descripción:** Recupera la lista completa de todas las reservas almacenadas en la base de datos.
- **Respuesta:** List<Reserva> en formato JSON
- **Lógica interna:** `return reservaRepository.findAll();`

Método: getReservasByCheckIn()

- **Ruta / Endpoint:** GET /reservas/checkin/{fecha}
- **Descripción:** Recupera todas las reservas que coinciden con la fecha de check-in indicada.
- **Respuesta:** List<Reserva> en formato JSON
- **Lógica interna:** `return reservaRepository.findByCheckIn(fecha);`

Método: getReservaById()

- **Ruta / Endpoint:** GET /reservas/{id}
- **Descripción:** Recupera una reserva específica a partir de su ID.
- **Respuesta:**
  - Reserva en formato JSON si se encuentra.
  - 404 Not Found si no existe.
- **Lógica interna:**

```
Optional<Reserva>reserva=reservaRepository.findById(id);

return reserva.map(ResponseEntity::ok).orElseGet(() ->
ResponseEntity.notFound().build());
```

Método: createReserva()

- **Ruta / Endpoint:** POST /reservas
- **Descripción:** Crea y persiste una nueva reserva a partir de los datos enviados en el cuerpo de la petición.
- **Respuesta:** Reserva en formato JSON
- **Lógica interna:** `return reservaRepository.save(reserva);`

Método: updateReserva()

- **Ruta / Endpoint:** PUT /reservas/{id}
- **Descripción:** Actualiza una reserva existente con los datos proporcionados.
- **Respuesta:**
    - Reserva actualizada en formato JSON si se encuentra.
    - 404 Not Found si el ID no existe.

- **Lógica interna:**

```
return reservaRepository.findById(id).map(reserva->
{reserva.setCheckIn(reservaDetails.getCheckIn());
reserva.setCheckOut(reservaDetails.getCheckOut());
reserva.setHotel(reservaDetails.getHotel());
reserva.setPersona(reservaDetails.getPersona());
reserva.setNumHabitaciones(reservaDetails.getNumHabitaciones());
reserva.setTipoHabitacion(reservaDetails.getTipoHabitacion());
returnResponseEntity.ok(reservaRepository.save(reserva));}).orEls
eGet(() -> ResponseEntity.notFound().build());
```

Método: deleteReserva()

- **Ruta / Endpoint:** DELETE /reservas/{id}
- **Descripción:** Elimina una reserva específica a partir de su ID.
- **Respuesta:**
    - 200 OK si se elimina correctamente.
    - 404 Not Found si no existe.
- **Lógica interna:**

```
return reservaRepository.findById(id).map(reserva -> {
reservaRepository.delete(reserva);
return ResponseEntity.ok().build();
}).orElseGet(() -> ResponseEntity.notFound().build());
```

Reserva View

Método: listarReservas()

- **Ruta / Endpoint:** GET /vistas/reservas/listar
- **Descripción:** Muestra la vista de la lista de todas las reservas registradas en el sistema.
- **Respuesta:** Renderiza la vista reservas/lista con el título "Listado de Reservas".
- **Lógica interna:**

```
model.addAttribute("titulo", "Listado de Reservas");
return "reservas/lista";
```

Método: mostrarFormularioEdicion()

- **Ruta / Endpoint:** GET /vistas/reservas/editar/{id}
- **Descripción:** Muestra un formulario para editar una reserva específica.
- **Respuesta:** Renderiza la vista reservas/editar con los datos de la reserva seleccionada y listas de hoteles, personas y tipos de habitación para facilitar la edición.
- **Lógica interna:**

```
Reserva reserva = reservaRepository.findById(id)
.orElseThrow(() -> new IllegalArgumentException("Reserva no
encontrada"));
model.addAttribute("reserva", reserva);
model.addAttribute("hoteles", hotelRepository.findAll());
model.addAttribute("personas", personaRepository.findAll());
model.addAttribute("tiposHabitacion",
tipoHabitacionRepository.findAll());

return "reservas/editar";
```

Método: actualizarReserva()

- **Ruta / Endpoint:** PUT /vistas/reservas/editar/{id}
- **Descripción:** Procesa el formulario de edición de una reserva y actualiza sus datos en la base de datos.
- **Respuesta:** Redirige al listado de reservas (/vistas/reservas/listar).
- **Lógica interna:**

```
reservaRepository.findById(id).ifPresent(reserva ->
{reserva.setCheckIn(reservaActualizada.getCheckIn());
reserva.setCheckOut(reservaActualizada.getCheckOut());
reserva.setNumHabitaciones(reservaActualizada.getNumHabitaciones(
));
reserva.setHotel(reservaActualizada.getHotel());
reserva.setTipoHabitacion(reservaActualizada.getTipoHabitacion())
;reservaRepository.save(reserva);});

return "redirect:/vistas/reservas/listar";
```

Tipo Habitación Controller

Método: getAllTiposHabitacion()

- **Ruta / Endpoint:** GET /tipoHabitaciones
- **Descripción:** Recupera la lista completa de todos los tipos de habitación almacenados en la base de datos.
- **Respuesta:** List<TipoHabitacion> en formato JSON
- **Lógica interna:** `return tipoHabitacionRepository.findAll();`

Método: crearTipo()

- **Ruta / Endpoint:** POST /tipoHabitaciones
- **Descripción:** Crea y persiste un nuevo tipo de habitación a partir de los datos enviados en el cuerpo de la petición.
- **Respuesta:** ResponseEntity<TipoHabitacion> con el objeto creado y código HTTP 201 Created.
- **Lógica interna:**
```
TipoHabitacion nouTipoHabitacion = tipoHabitacionRepository.save(tipohabitacion);
return newResponseEntity<>(nouTipoHabitacion,HttpStatus.CREATED);
```