**Estructura básica de una API en paquetes**

Paquetes básicos:

|  |  |  |
| --- | --- | --- |
| Context | Ahí se pondría algunas configuraciones necesarias para el proyecto ejemplo swagger |  |
| [Controller](#Controller2) | Ahí van los controladores |  |
| [Exceptions](#exceptions2) | Aquí se administran las excepciones |  |
| [Mapper](#mapper2) | Aquí va el mapeo de las entidades | Dentro va la interface IMapper que sería una genérico y los mapeos necesarios con nombres bien explícitos ejemplos TaskDTOToTask |
| [Persistence-principal](#persistence2).  Entity  Repository | Aquí todo lo necesario para la persistencia en la base de datos. | Entity: las entidades, POO  Repository:los repositorios de JpaRepositoty o crudRepository |
| [Service](#service2) | Los servicios de la aplicación | A partir de la inyección de repository creo los servicios para respetar las capas |
| [DTO](#dto2) | Data Objet Transfer | Creo un objeto de transferencia con los atributos que me interesan de las entidades , debe tener getter y setter y constructores |
| Util | Auxiliar para clases necesarias |  |

Se crean los controladores desde las capas mas bajas en este proyecto seria repository, luego service y por ultimo controller

Nota:

@PatchMapping("/updateAsFinished/{id}")

Se usa cuando se modifica un solo valor de una entidad!!!

[Controller](#Controller1)

@RestController  
@RequestMapping("/tasks")  
public class TaskController {  
 private TaskService taskService;  
  
 public TaskController(TaskService taskService) {  
 this.taskService = taskService;  
 }  
  
 @PostMapping("/create")  
 public ResponseEntity<?> createTask(@RequestBody TaskDTO taskDTO) {  
 Task task = taskService.createTask(taskDTO);  
 URI url= ServletUriComponentsBuilder.*fromCurrentRequest*()  
 .path("/{id}").buildAndExpand(task.getId()).toUri();  
  
 return ResponseEntity.*created*(url).body(taskDTO);  
 }  
 @GetMapping  
 public ResponseEntity<Page<?>> listTasks(){  
 List<Task> taskList = taskService.findAll();  
 Page<Task> taskPage=new PageImpl<>(taskList);  
 return ResponseEntity.*ok*(taskPage);  
 }  
  
 @GetMapping("/findByStatus/{status}")  
 public ResponseEntity<Page<?>> listByStatus(@PathVariable TaskStatus status){  
 List<Task> taskListStatus = taskService.findAllByStatus(status);  
 Page<Task> taskPageStatus=new PageImpl<>(taskListStatus);  
 return ResponseEntity.*ok*(taskPageStatus);  
 }  
  
 @PatchMapping ("/updateAsFinished/{id}")  
 public ResponseEntity<?> updateAsFinished(@PathVariable Long id){  
 taskService.updateTaskAsFinished(id);  
 return ResponseEntity.*ok*("Task modified successfully");  
 }  
 @DeleteMapping("/deleteById/{id}")  
 public ResponseEntity<?> deleteById(@PathVariable Long id){  
 taskService.deleteById(id);  
 return ResponseEntity.*ok*("Task deleted successfully");  
 }  
  
}

Exceptions

Con dos clases una creada para manejar las excepciones particulares de la API

Y otra casi genérica

Class creada

import org.springframework.http.HttpStatus;  
@Data  
public class TodoExceptions extends RuntimeException{  
 private String message;  
 private HttpStatus httpStatus;  
  
 public TodoExceptions(String message, HttpStatus httpStatus) {  
 super(message);  
 this.message = message;  
 this.httpStatus= httpStatus;  
 }  
}

ExceptionHandler

@ControllerAdvice  
public class ExceptionHandler extends ResponseEntityExceptionHandler {  
 @org.springframework.web.bind.annotation.ExceptionHandler(value = {TodoExceptions.class})  
 protected ResponseEntity<Object> handleConflict(  
 TodoExceptions ex, WebRequest request){  
 String bodyOfResponse = ex.getMessage();  
  
 return handleExceptionInternal(ex,bodyOfResponse,new HttpHeaders(),ex.getHttpStatus(),request);  
  
 }  
}

Mapper

Con una interface para mapear (genérica)

public interface IMapper <I,O>{  
 public O map(I in);  
  
}

y una clase que hace el mapeo en este caso TaskDTOToTask que implementa la interface IMapper

@Component  
public class TaskDTOToTask implements IMapper<TaskDTO,Task>{  
 @Override  
 public Task map(TaskDTO taskDTO) {  
 Task task=new Task();  
 task.setTitle(taskDTO.getTitle());  
 task.setDescription(taskDTO.getDescription());  
 task.setEta(taskDTO.getEta());  
 task.setCreationDate(LocalDateTime.*now*());  
 task.setFinished(false);  
 task.setTaskStatus(TaskStatus.*ON\_TIME*);  
 return task;  
 }  
}

Persistence-principal.

Entity

Las entidades de nuestra API

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
@Entity  
@Table(name = "task")  
public class Task {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
 private String title;  
 private String description;  
 private LocalDateTime creationDate;  
 private LocalDateTime eta;  
 private Boolean finished;  
 private TaskStatus taskStatus;  
}

En este caso en particular también tenemos un Enum

public enum TaskStatus {  
 *ON\_TIME*,  
 *LATE*}

Repository

Los repositorios de JpaRepository o CrudRepository ,etc. En este ejemplo vemos dos métodos creados para actuar sobre la DB uno con la configuración de spring jpa (sintaxis particular-> findAllBy….) y la otra con nativeQuery, recordemos que el repository es la primera capa , luego sigue el service y al final el controller.

@Repository  
public interface TaskRepository extends JpaRepository<Task,Long> {  
 public List<Task> findAllByTaskStatus(TaskStatus taskStatus);  
  
 @Modifying  
 @Query(value = "UPDATE TASK SET FINISHED=true WHERE ID=:id",nativeQuery = true)  
 public void markTaskAsFinished(@Param("id") Long id);  
}

Service

Aqui coloco toda la lógica para los endPoint, algunas comprobaciones, el manejo de las excepciones el mapeo etc con iyecciones de dependencias o new excepciones

@Service  
public class TaskService {  
 private final TaskRepository taskRepository;  
 private TaskDTOToTask taskDTOToTask;  
 public TaskService(TaskRepository taskRepository, TaskDTOToTask taskDTOToTask) {  
 this.taskRepository = taskRepository;  
 this.taskDTOToTask = taskDTOToTask;  
 }  
 public Task createTask(TaskDTO taskDTO){  
 Task task = taskDTOToTask.map(taskDTO);  
 return taskRepository.save(task);  
 }  
 public List<Task> findAll(){  
 return taskRepository.findAll();  
 }  
  
 public List<Task> findAllByStatus(TaskStatus status){  
 return taskRepository.findAllByTaskStatus(status);  
 }  
 @Transactional  
 public void updateTaskAsFinished(long id){  
 Optional<Task> taskOptional = taskRepository.findById(id);  
 if (!taskOptional.isPresent()){  
 throw new TodoExceptions("Task ID does not exist", HttpStatus.*BAD\_REQUEST*);  
 }  
 taskRepository.markTaskAsFinished(id);  
 }  
 public Optional<Task> findById(Long id){  
 return taskRepository.findById(id);  
 }  
 public void deleteById(Long id){  
 Optional<Task> taskOptional = taskRepository.findById(id);  
 if (!taskOptional.isPresent()){  
 throw new TodoExceptions("Task ID does not exist", HttpStatus.*BAD\_REQUEST*);  
 }  
 this.taskRepository.deleteById(id);  
 }  
}

El DTO

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class TaskDTO {  
 private String title;  
 private String description;  
 private LocalDateTime eta;  
  
}

El main en este caso tiene la configuración del swagger (no se si es lo mejor, pero funciona)

@SpringBootApplication  
public class TodoAppApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(TodoAppApplication.class, args);  
 }

@Bean  
 public OpenAPI customOpenAPI() {  
 return new OpenAPI()  
 .info(new Info()  
 .title("Sertec APP Lista de Tareas")  
 .version("0.1")  
 .description("Sample app Spring Boot 3 with Swagger")  
 .termsOfService("http://swagger.io/terms/")  
 .license(new License().name("Apache 2.0").url("http://springdoc.org")));  
 }  
}

El application.properties

#crea las entidades en la DB create,create-drop,validate, update, none  
spring.jpa.hibernate.ddl-auto=create-drop  
spring.datasource.url=jdbc:mysql://localhost:3306/todoApp2  
spring.datasource.username=root  
spring.datasource.password=Sertec2580$  
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver  
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect  
spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.format\_sql=true  
#springdoc  
springdoc.api-docs.enabled=true  
springdoc.swagger-ui.enabled=true  
springdoc.swagger-ui.path=/doc/swagger-ui.html

Archivo pom.xml

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.1.1</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.sertecApp</groupId>  
 <artifactId>todoApp</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>todoApp</name>  
 <description>api rest todo app </description>  
 <properties>  
 <java.version>17</java.version>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>com.mysql</groupId>  
 <artifactId>mysql-connector-j</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <optional>true</optional>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springdoc</groupId>  
 <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>  
 <version>2.0.2</version>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 <configuration>  
 <excludes>  
 <exclude>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 </exclude>  
 </excludes>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>