

## Introducción

La API que se presenta está diseñada para gestionar diversos aspectos de un juego de rol, donde se involucran múltiples entidades como usuarios, personajes, monstruos, objetos y mapas. Cada componente o entidad del sistema tiene su propia API que permite realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para su manejo eficiente. Estas APIs están organizadas de manera que permiten una gestión modular y fácil de entender para las diferentes funcionalidades del juego.

## Objetivos

### 1. Gestión de Usuarios:

- **Usuario Controller:** Permitir crear, leer, actualizar y eliminar usuarios dentro del sistema.
- **Usuario Tipo Controller:** Gestionar los diferentes tipos de usuarios con sus respectivas operaciones CRUD.

### 2. Gestión de Personajes:

- **Personaje Controller:** Ofrecer operaciones CRUD para la creación y gestión de personajes dentro del juego.
- **Personaje Misiones Controller:** Permitir gestionar las misiones asociadas a los personajes, con operaciones CRUD.
- **Personaje Logros Controller:** Gestionar los logros obtenidos por los personajes.

### 3. Gestión de Clases, Habilidades y Estadísticas:

- **ClasePersonaje:** Gestionar las diferentes clases o roles de los personajes.
- **Habilidad:** Crear, leer, actualizar y eliminar habilidades que los personajes pueden aprender y utilizar.
- **Personaje Habilidades Controller:** Gestionar las habilidades que los personajes pueden adquirir.
- **EstadisticasGenerales:** Gestionar las estadísticas generales de los personajes, como fuerza, agilidad, inteligencia, etc.

### 4. Gestión de Monstruos y sus Habilidades:

- **Monstruo:** Permitir la gestión de monstruos, incluyendo su creación, actualización y eliminación.
- **MonstruoHabilidad:** Gestionar las habilidades de los monstruos.
- **MonstruoDrop:** Controlar los objetos que los monstruos pueden dejar como recompensa al ser derrotados.

- **MapaMonstruo:** Gestionar la distribución de monstruos en diferentes mapas.

#### 5. Gestión de Objetos y Efectos:

- **Objeto:** Gestionar los objetos o ítems que los personajes pueden usar o equipar.
- **TipoItem:** Clasificar y gestionar los diferentes tipos de objetos dentro del juego.
- **ObjetoEfecto:** Gestionar los efectos que los objetos tienen sobre los personajes o el entorno.

#### 6. Gestión de Grupos y Líderes:

- **Grupo:** Gestionar la creación, modificación y eliminación de grupos dentro del juego.
- **LiderGrupo:** Gestionar los líderes de los grupos, quienes tienen autoridad sobre el equipo.
- **TipoGrupo:** Organizar y clasificar los diferentes tipos de grupos.

#### 7. Gestión de NPCs (Personajes No Jugadores):

- **NPC:** Gestionar la creación y el comportamiento de los NPCs dentro del juego.
- **TipoNPC:** Clasificar y gestionar los diferentes tipos de NPCs.
- **NPCProducto:** Gestionar los productos que los NPCs pueden ofrecer en su tienda.

#### 8. Gestión de Mapas y Efectos:

- **Mapa:** Permitir la creación, modificación y eliminación de mapas dentro del juego.
- **TipoMapa:** Clasificar y gestionar los diferentes tipos de mapas.
- **MapaEfecto:** Gestionar los efectos que pueden ocurrir en los mapas.

#### 9. Gestión de Logs y Transacciones:

- **LogTransacciones:** Gestionar los logs relacionados con las transacciones realizadas dentro del juego.
- **LogPersonajeMonstruo:** Gestionar los logs relacionados con las interacciones entre personajes y monstruos.
- **LogUsuario:** Mantener un registro de las actividades de los usuarios en el sistema.

## Análisis del Problema

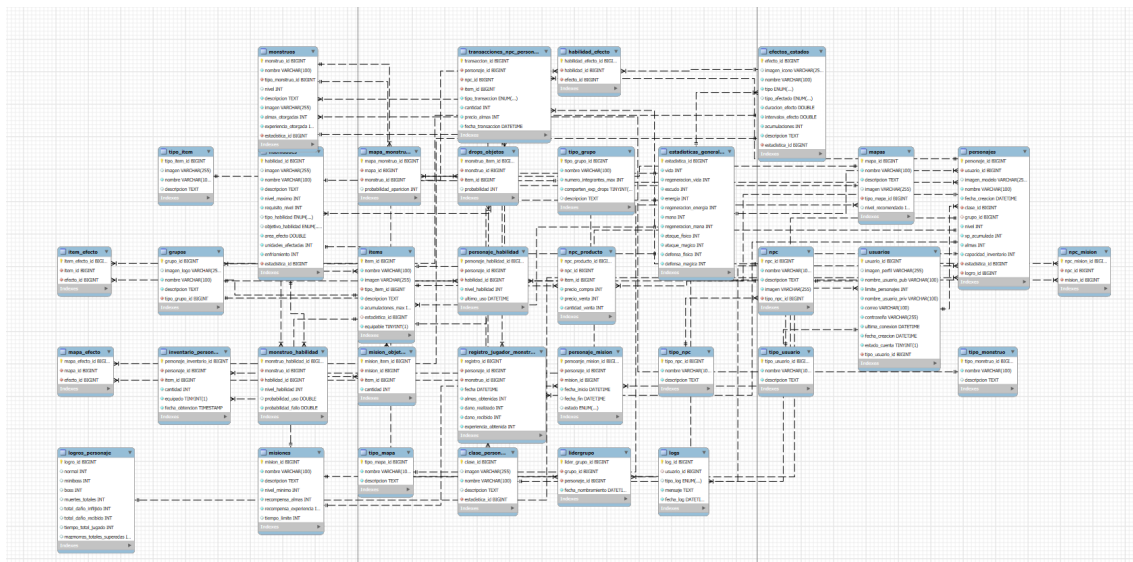
En el contexto de un videojuego de rol (RPG), gestionar las interacciones entre diversas entidades del juego —usuarios, personajes, monstruos, objetos, mapas, etc. -- se convierte en una tarea compleja. A medida que el juego crece en términos de contenido y funcionalidad, también lo hace la necesidad de tener un sistema eficiente y bien organizado para gestionar todos estos elementos de manera modular y escalable.

Los problemas comunes que suelen surgir en estos casos son falta de tiempo ,errores tipográficos o desconocimiento de tecnologías.

## Justificación del Proyecto

La creación de una serie de APIs bien estructuradas para gestionar todos los aspectos del juego de rol no solo resuelve los problemas mencionados, sino que también proporciona múltiples beneficios aprender a estructurar, ver por que se hace cada cosa y comerse la cabeza con errores por una letra.

## 1. Entidades Principales



**Usuarios (usuarios)**

- Representa a los jugadores del sistema.
- Atributos importantes: `usuario_id`, `nombre_usuario`, `correo`, `ultima conexion`, `tipo usuario`.

- Relacionado con personajes (un usuario puede tener múltiples personajes).

#### **Personajes (personajes)**

- Representa a los personajes jugables en el juego.
- Atributos: personaje\_id, usuario\_id, nombre, clase\_id, grupo\_id, nivel, experiencia, almas, capacidad\_inventario.
- Relacionado con usuarios, clase\_personaje, grupo, misiones, habilidades, y inventario\_personaje.

#### **Monstruos (monstruos)**

- Representa a los enemigos en el juego.
- Atributos: monstruo\_id, nombre, tipo\_monstruo\_id, nivel, ataque, defensa, experiencia\_otorgada.
- Relacionado con tipo\_monstruo, monstruo\_habilidad, y mapa\_monstruo.

#### **Misiones (misiones)**

- Representa las misiones dentro del juego.
- Atributos: mision\_id, nombre, descripcion, recompensa\_items, recompensa\_experiencia, tiempo\_limite.
- Relacionado con npc\_mision, personaje\_mision, y mision\_objetivo.

#### **Inventario de personajes (inventario\_personaje)**

- Representa los ítems que posee cada personaje.
- Atributos: inventario\_id, personaje\_id, item\_id, cantidad.
- Relacionado con items y personajes.

#### **Ítems (items)**

- Representa objetos del juego como armas, pociones, etc.
- Atributos: item\_id, nombre, habilidad\_id, efectos, equipable.
- Relacionado con tipo\_item, habilidad\_efecto, y mapa\_efecto.

#### **Habilidades (habilidad\_efecto)**

- Representa habilidades y efectos en personajes y monstruos.
- Atributos: habilidad\_id, nombre, efecto, duracion, unidad\_efecto.
- Relacionado con personaje\_habilidad, monstruo\_habilidad, y efectos\_estados.

## **2. Relaciones Clave**

- **usuarios → personajes (1:N)**  
Un usuario puede tener múltiples personajes, pero un personaje pertenece a un solo usuario.
- **personajes → inventario personaje (1:N)**  
Un personaje puede poseer varios ítems en su inventario.
- **personajes → misiones (N:M) a través de personajes mision**  
Un personaje puede estar involucrado en múltiples misiones, y una misión puede ser aceptada por varios personajes.
- **monstruos → monstruo habilidad (1:N)**  
Un monstruo puede tener múltiples habilidades.
- **mapas → mapa monstruo (1:N)**  
Un mapa puede contener varios monstruos.
- **ítems → tipo ítem (N:1)**  
Cada ítem pertenece a un solo tipo, pero un tipo de ítem puede agrupar múltiples ítems.
- **npc → npc mision (1:N)**  
Un NPC puede otorgar múltiples misiones.
- **grupos → personajes (1:N)**  
Un grupo puede estar conformado por varios personajes.

## Tecnologías utilizadas y configuración del entorno

### Tecnologías utilizadas

El proyecto está desarrollado utilizando **Spring Boot** como framework principal, en conjunto con otras tecnologías para garantizar una arquitectura robusta y segura:

- **Spring Boot:** Facilita la creación y gestión de la API REST.
- **Spring Boot Starter Web:** Proporciona las dependencias necesarias para desarrollar servicios web.
- **Spring Boot Starter Data JPA:** Permite la integración con bases de datos utilizando JPA e Hibernate.
- **Spring Boot Starter Security:** Implementa mecanismos de autenticación y autorización.
- **Spring Boot Starter Thymeleaf:** Permite la generación de vistas dinámicas.
- **MySQL:** Sistema de gestión de bases de datos utilizado para almacenar la información del juego.
- **Lombok:** Reduce la escritura de código repetitivo en las entidades y servicios.
- **JWT (JSON Web Token):** Se utiliza para la autenticación y gestión de sesiones de usuario.
- **BCrypt:** Manejo de contraseñas de forma segura mediante hashing.

- **Spring Doc Open API:** Facilita la generación automática de documentación para la API.

## **Configuración del entorno**

Para el desarrollo de este proyecto, se utiliza el siguiente entorno:

- **IDE:** IntelliJ IDEA
- **Gestor de dependencias:** Maven
- **Servidor de aplicación:** Embebido en Spring Boot (Tomcat)
- **Base de datos:** MySQL
- **Control de versiones:** Git/GitHub

## PROPERTIES

```
<dependencies>
  <!-- Spring Boot Starters -->
  <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>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>

  <!-- MySQL Connector -->
  <dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
  </dependency>

  <!-- Lombok -->
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <scope>provided</scope>
  </dependency>

  <!-- JWT para autenticación -->
  <dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.12.3</version>
  </dependency>

  <!-- BCrypt para manejo de contraseñas -->
  <dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-crypto</artifactId>
  </dependency>

  <!-- Documentación OpenAPI -->
  <dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.2.0</version>
  </dependency>
</dependencies>
```

## Configuración de la base de datos

En el archivo `application.properties` o `application.yml`, se configura la conexión con la base de datos MySQL:

```
spring.datasource.url=jdbc:mysql://localhost:3306/api_rpg_bd
spring.datasource.username=root
spring.datasource.password=abc123.
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

jwt.secret = bNikTaC8AqJ0+OGQJsxL0XA7DFxc/cjkAPDU7wCPeRQ=

upload.path=uploads/images/
```



## . Controladores (Controllers)

El **Controlador** es la capa que se encarga de manejar las solicitudes HTTP (GET, POST, PUT, DELETE, etc.) provenientes del cliente (como un navegador o una aplicación móvil). Los controladores gestionan las rutas de la API y delegan la lógica de negocio a los servicios.

- **Responsabilidad principal:** Recibir las solicitudes HTTP y devolver las respuestas HTTP correspondientes (generalmente en formato JSON o XML).
- **Interacción:** El controlador usa el servicio para ejecutar la lógica de negocio y luego devuelve la respuesta apropiada.

### Características:

- **Anotaciones HTTP:** En Spring Boot, los controladores se definen con las anotaciones `@RestController` o `@Controller`. Cada método de controlador está anotado con las anotaciones `@GetMapping`, `@PostMapping`, `@PutMapping`, etc.
- **Manejo de excepciones:** Los controladores pueden manejar errores, como los códigos de error 404 o 500, y devolver respuestas adecuadas al cliente.

```

@RestController no usages ▲ Laureano De Sousa +1
@RequestMapping("/api/mapa")
@Tag(name = "Mapa", description = "API para gestionar mapas")
public class MapaController {

    @Autowired 6 usages
    private MapaService mapaService;

    @GetMapping("/") no usages ▲ Laureano De Sousa
    @Operation(summary = "Obtener todos los mapas o filtrar por tipo de mapa")
    public List<Mapa> obtenerListaMapas(@RequestParam(required = false) Long id) {
        if (id == null) {
            return mapaService.getAll();
        } else {
            TipoMapa tipoMapa = new TipoMapa();
            tipoMapa.setTipoMapaId(id);
            return mapaService.getByTipoMapa(tipoMapa);
        }
    }

    @GetMapping("/{id}") no usages ▲ Laureano De Sousa
    @Operation(summary = "Obtener un mapa por ID")
    public Mapa obtenerMapa(@PathVariable Long id) { return mapaService.getByID(id); }

    @PutMapping("/{id}") no usages ▲ Laureano De Sousa
    @Operation(summary = "Actualizar un mapa por ID")
    public ResponseEntity<Object> actualizarMapa(@PathVariable Long id, @RequestBody Mapa mapaActualizar) {
        if (mapaActualizar.getMapa_id().equals(id)) {
            return ResponseEntity.ok(mapaService.setItem(mapaActualizar));
        } else {
            return ResponseEntity.badRequest().body("El ID proporcionado no coincide con el ID del mapa.");
        }
    }

    @PostMapping no usages ▲ Laureano De Sousa
    @Operation(summary = "Crear un nuevo mapa")
    public Mapa guardarMapa(@RequestBody Mapa mapaGuardar) { return mapaService.setItem(mapaGuardar); }

    @DeleteMapping("/{id}") ▲ Laureano De Sousa
    @Operation(summary = "Eliminar un mapa por ID")
    public void borrarMapa(@PathVariable Long id) { mapaService.deleteByID(id); }
}

```

## Servicios (Services)

El **Servicio** es la capa encargada de contener la lógica de negocio de la aplicación. Los servicios interactúan con los repositorios para acceder a los datos y luego realizan las operaciones necesarias sobre esos datos antes de devolverlos al controlador.

- **Responsabilidad principal:** Contener la lógica de negocio, procesar datos y coordinar operaciones.
- **Interacción:** Los servicios llaman a los repositorios para realizar operaciones CRUD (crear, leer, actualizar, eliminar) en la base de datos y procesan los datos antes de devolverlos al controlador.

### Características:

- Los servicios deben ser **stateless**, es decir, no deben mantener estado entre las llamadas.
- En Spring, los servicios se anotan con **@Service** para indicar que son componentes de lógica de negocio.

- Pueden incluir validaciones, procesamiento de datos o coordinación de otras operaciones.

LA MAYOR PARTE DE PROYECTO USA SERVICIOS DE CRUD BASE

```
> import ...

@Service 8 usages ▲ Laureano +1
public class GrupoService {

    @Autowired 5 usages
    private GrupoRepository grupoRepository;

>     public List<Grupo> getAll() { return grupoRepository.findAll(); }
>     public Grupo getByID(Long id) { return grupoRepository.findById(id).orElse( other: null); }
>     public Grupo setItem(Grupo o) { return grupoRepository.save(o); }
>     public void deleteByID(Long id) { grupoRepository.deleteById(id); }

    public List<Grupo> getBytipoGrupo(TipoGrupo tu) { 1 usage ▲ Laureano De Sousa
        return grupoRepository.getBytipoGrupo(tu);
    }
}
```

## Repositorios (Repositories)

El **Repositorio** es la capa que maneja la interacción directa con la base de datos. Su tarea es recuperar, almacenar, actualizar y eliminar los datos. Utiliza tecnologías como **JPA (Java Persistence API)**, **Hibernate** o **Spring Data JPA** para realizar estas operaciones.

- **Responsabilidad principal:** Manejar las operaciones CRUD directamente sobre la base de datos.
- **Interacción:** Los repositorios reciben las solicitudes de los servicios y ejecutan las consultas a la base de datos.

### Características:

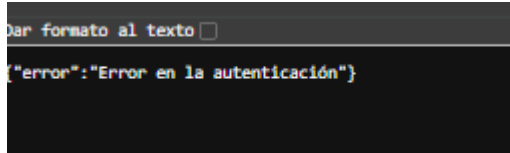
- En Spring, los repositorios se anotan con `@Repository`, aunque con **Spring Data JPA** no es necesario agregar esta anotación explícitamente, ya que se proporciona automáticamente.
- Los repositorios extienden interfaces como `JpaRepository`
- Los métodos comunes como `findById`, `save`, `deleteById`, etc.

LA MAYOR PARTE DE PROYECTO USA REPOSITARIOS VACÍOS

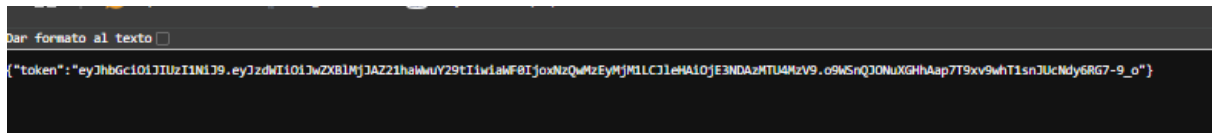
```
8
9  @Repository 4 usages ⓘ Laureano De Sousa *
10 public interface UsuarioRepository extends JpaRepository<Usuario, Long> {
11     List<Usuario> getByTipoUsuario(Long tu); 1 usage ⓘ Laureano De Sousa
12     Usuario findByCorreo(String email); 1 usage ⓘ Laureano De Sousa
13 }
14
```

# JWT y Spring Security

1. **Autenticación inicial:** Cuando un usuario se autentica con su nombre de usuario y contraseña, el servidor valida las credenciales.



2. **Generación de JWT:** Si las credenciales son válidas, el servidor genera un JWT firmado y lo envía de vuelta al cliente.



3. **Acceso a recursos protegidos:** El cliente envía el JWT en las solicitudes HTTP posteriores (generalmente en el encabezado `Authorization`).
4. **Validación del JWT:** El servidor valida el JWT en cada solicitud entrante. Si es válido y no ha expirado, se permite el acceso a los recursos protegidos.

```
public class JwtSecretKeyGenerator {

    public static void main(String[] args) throws Exception {
        KeyGenerator keyGen = KeyGenerator.getInstance("HmacSHA256");
        keyGen.init(256);
        SecretKey secretKey = keyGen.generateKey();
        String base64Key =
Base64.getEncoder().encodeToString(secretKey.getEncoded());
        System.out.println("Clave secreta en Base64: " + base64Key);
    }
}
```

Con el codigo proporcionando generamos la key que usaremos en nuestro proyecto en el properties, esta sera la semilla de cifrados.

```

        @PostMapping("/login")
        public ResponseEntity<?> login(@RequestParam String correo, @RequestParam String
contraseña, Model model) {
            try {
                System.out.println("Intentando login para: " + correo);

                // Intentar autenticar al usuario
                Authentication authentication = authenticationManager.authenticate(
                    new UsernamePasswordAuthenticationToken(correo, contraseña)
                );
                System.out.println("Autenticación exitosa");

                // Si la autenticación es exitosa, obtener los detalles del usuario
                UserDetails userDetails = userDetailsService.loadUserByUsername(correo);

                // Generar el token JWT
                String token = jwtService.generateToken(userDetails.getUsername());

                // Devolver el token en la respuesta
                return ResponseEntity.ok().body(Map.of("token", token));

            } catch (Exception e) {
                System.out.println("Error en login: " + e.getMessage());
                model.addAttribute("error", "Error en la autenticación: " +
e.getMessage());
                return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(Map.of("error", "Error en la
autenticación"));
            }
        }
    }

//fetch('/login', {
//    method: 'POST',
//    body: JSON.stringify({ correo: 'usuario@example.com', contraseña:
'contraseña' }),
//    headers: { 'Content-Type': 'application/json' }
//})
//.then(response => response.json())
//.then(data => {
//    localStorage.setItem('token', data.token); // Almacenar el token
//    window.location.href = '/admin'; // Redirigir al panel de control
//});

```

Fichero encargado de gestionar los token de salida y ejemplo de código para que lo valide el cliente.

```

@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .csrf(AbstractHttpConfigurer::disable)
        .cors(Customizer.withDefaults())
        .headers(headers -> headers
            .frameOptions(frame -> frame.disable()))
        .sessionManagement(session ->
            session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .authorizeHttpRequests(auth -> auth
            // Swagger UI
            .requestMatchers("/v3/api-docs/**", "url/**",
                "/swagger-ui/**",
                "/swagger-ui.html",
                "/swagger-resources/**",
                "/webjars/**").permitAll()
            .requestMatchers("/auth/**").permitAll() Para Pruebas.
            // .requestMatchers("/api/**").permitAll()
            // .requestMatchers("admin/**").permitAll()
            // .requestMatchers("/api/**").hasAnyAuthority("1", "2", "3") Uso de numeros para las jerarquias.
            // .requestMatchers("/logs/**").hasAnyAuthority("2", "3")
            // .requestMatchers("/admin/**").hasAuthority("3")
            .anyRequest().authenticated()
        )
        .authenticationProvider(authenticationProvider())
        .addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);

    return http.build();
}

@Bean
public CorsConfigurationSource corsConfigurationSource() {
    CorsConfiguration configuration = new CorsConfiguration();
    configuration.setAllowedOrigins(Arrays.asList(""));
    configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "OPTIONS"));
    configuration.setAllowedHeaders(Arrays.asList(""));
    configuration.setExposedHeaders(Arrays.asList("Authorization"));

    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration("/**", configuration);
    return source;
}
}

```

Aquí se asignan los diferentes permisos ,en mi caso para no usar el token, lo tengo que permitir todo.

# Desarrollo del Backend con Thymeleaf (Explicación de Vistas y Controladores)

## Introducción a Thymeleaf

Thymeleaf es un motor de plantillas para Java que permite generar vistas dinámicas en aplicaciones web. Se integra fácilmente con **Spring Boot**, permitiendo el uso de datos dinámicos en las vistas HTML sin necesidad de usar tecnologías más complejas como JSP.

## Estructura del Backend

El backend de la aplicación se basa en un modelo **MVC (Modelo-Vista-Controlador)** donde:

- **Modelo:** Representa los datos del juego (entidades JPA).
- **Vista:** Se generan con **Thymeleaf** y se encuentran en `src/main/resources/templates/`.
- **Controlador:** Se encarga de gestionar las solicitudes y enviar datos a las vistas.

---

## Configuración de Thymeleaf en Spring Boot

Spring Boot incluye Thymeleaf por defecto, por lo que solo es necesario agregar la dependencia en el `pom.xml`:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Funcionan igual que cualquier página html pero con posibilidad de aplicar bucles y condiciones dinámicas.

SIEMPRE TENEMOS LAS MISMAS ESTRUCTURAS EN LOS FORMULARIOS

DENTRO DE UN HTML ESTÁNDAR CON POSIBILIDAD DE E



```

<!-- Contenido principal -->

<div class="container mt-4">

    <!-- Mensaje de Error si ocurre algún problema -->

    <div th:if="{error}" class="alert alert-danger">

        <p th:text="{error}"></p>

    </div>

    <h2 class="text-center mb-4">Gestión de Tipos de NPC</h2>

    <div class="form-container">

        <h3 th:text="{tipoNPC.tipo_npc_id != null ? 'Editar Tipo de NPC' : 'Crear Nuevo Tipo de NPC'}"></h3>

        <form th:action="@{/admin/npc/tipoNPC/save}" th:object="{tipoNPC}"
method="post">

            <input type="hidden" th:field="{tipo_npc_id}" />

            <div class="form-group mb-3">

                <label for="nombre" class="form-label">Nombre</label>

                <input type="text" id="nombre" th:field="{nombre}"
class="form-control" placeholder="Nombre del tipo de NPC" required />

            </div>

            <div class="form-group mb-3">

                <label for="descripcion" class="form-label">Descripción</label>

                <textarea id="descripcion" th:field="{descripcion}"
class="form-control" placeholder="Descripción del tipo de NPC" required></textarea>

            </div>

            <button type="submit" class="btn btn-primary">

                <span th:text="{tipoNPC.tipo_npc_id != null ? 'Actualizar' :
'Crear'}"></span>

            </button>

        </form>

    </div>

```

SIEMPRE TENEMOS LAS MISMAS ESTRUCTURAS EN LAS TABLAS

```
<!-- Listado de Tipos de NPC -->

<div th:unless="{tipoNPC.tipo_npc_id != null}" class="table-container">

    <h3 class="mb-4">Listado de Tipos de NPC</h3>

    <table class="table table-bordered">

        <thead class="thead-dark">

            <tr>

                <th>ID de Tipo NPC</th>

                <th>Nombre</th>

                <th>Descripción</th>

                <th>Acciones</th>

            </tr>

        </thead>

        <tbody>

            <tr th:each="tipoNPC : {tiposNPC}">

                <td th:text="{tipoNPC.tipo_npc_id}"></td>

                <td th:text="{tipoNPC.nombre}"></td>

                <td th:text="{tipoNPC.descripcion}"></td>

                <td>

                    <a href="#"
th:href="@{/admin/npc/tipoNPC/edit/{id} (id={tipoNPC.tipo_npc_id})}" class="btn
btn-warning btn-sm">Editar</a>

                    <a href="#"
th:href="@{/admin/npc/tipoNPC/delete/{id} (id={tipoNPC.tipo_npc_id})}" class="btn
btn-danger btn-sm">Eliminar</a>

                </td>

            </tr>

        </tbody>

    </table>

</div>
```

CONTROLADORES DE FRON:

```
@Controller
```

```
@RequestMapping("/admin/estado/estados")
```

```
public class EfectoWebController {
```

```
    private final String rutaHTML = "/admin/estado/estados";
```

```
    @Autowired
```

```
    private EfectoEstadoService service;
```

```
    // Listar todos los estados y mostrar el formulario
```

```
    @GetMapping
```

```
    public String listar(Model model) {
```

```
        try {
```

```
            List<EfectoEstado> estados = service.getAll();
```

```
            model.addAttribute("estados", estados);
```

```
            model.addAttribute("estado", new EfectoEstado());
```

```
            return rutaHTML;
```

```
        } catch (Exception e) {
```

```
            model.addAttribute("error", "Error al cargar los estados: " + e.getMessage());
```

```
            return rutaHTML;
```

```
        }
```

```
    }
```

```
    // Guardar o actualizar
```

```
    @PostMapping("/save")
```

```
    public String guardar(@ModelAttribute("estado") EfectoEstado estado, Model model) {
```

```
        try {
```

```
            service.setItem(estado);
```

```
            return "redirect:" + rutaHTML;
```

```
        } catch (Exception e) {
```

```
            model.addAttribute("error", "Error al guardar el estado: " + e.getMessage());
```

```
            return rutaHTML;
```

```

        }

    }

    // Eliminar

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

    public String eliminar(@PathVariable("id") Long id, Model model) {

        try {

            service.deleteByID(id);

            return "redirect:" + rutaHTML;

        } catch (Exception e) {

            model.addAttribute("error", "Error al eliminar el estado: " + e.getMessage());

            return rutaHTML;

        }

    }

    // Cargar formulario de edición

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

    public String editar(@PathVariable("id") Long id, Model model) {

        try {

            EfectoEstado estado = service.getByID(id);

            model.addAttribute("estado", estado);

            return rutaHTML;

        } catch (Exception e) {

            model.addAttribute("error", "Error al cargar el estado para editar: " +
e.getMessage());

            return rutaHTML;

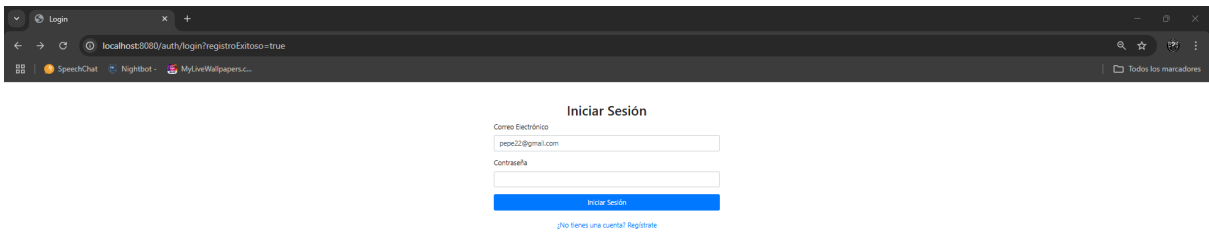
        }

    }

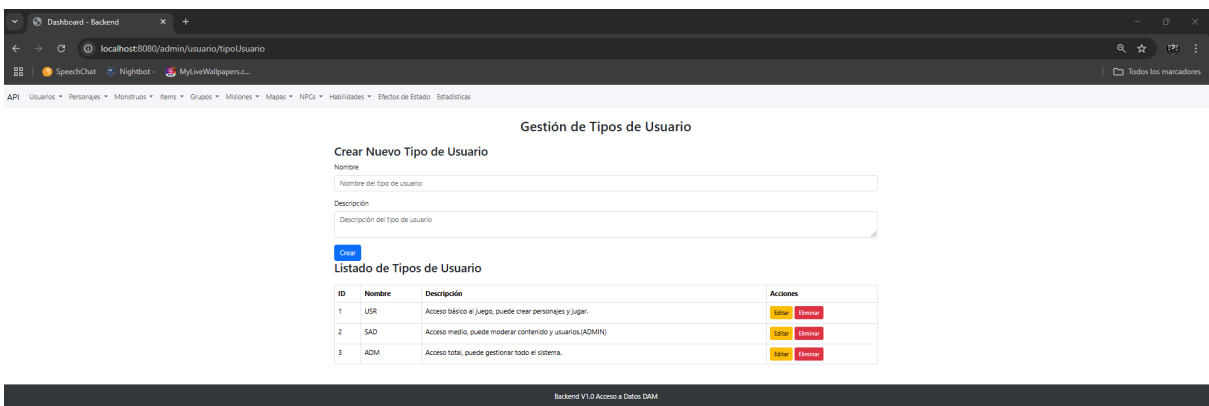
}

```

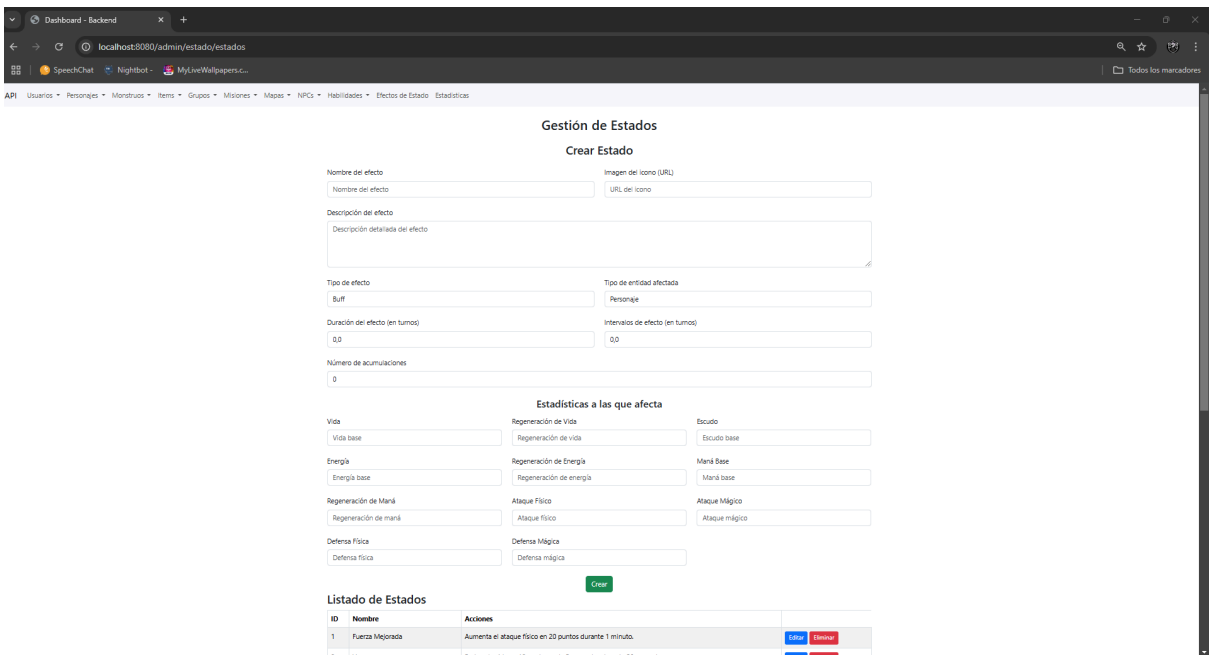
LOGIN DE USUARIOS



UNA DE MUCHAS VISTAS GENERICAS ,TODAS LA MISMA ESTRUCTURA, CAMBIAN LOS INPUTS.



CUENTAS CON 37 VISTAS DIFERENTES QUE GESTIONAN CADA UNA UNA TABLA.



REALIZAREMOS CRUD BÁSICOS PARA CADA TABLA Y DE PASO SON TEST.NO USO EL TOKEN POR QUE ES UN COÑAZO.

Usuario Controller <small>Operaciones CRUD para la gestión de usuarios</small>			^
GET	/api/usuario/{id}	Obtener un usuario por ID	▼
PUT	/api/usuario/{id}	Actualizar un usuario	▼
DELETE	/api/usuario/{id}	Eliminar un usuario	▼
GET	/api/usuario	Obtener lista de usuarios	▼
POST	/api/usuario	Crear un nuevo usuario	▼

LA GESTIÓN DE USUARIO NO ES UN PROBLEMA ,SOLO DEFINIREMOS LA ESTRUCTURA Y LA INTRODUCIMOS. COMO SE PUEDE VER PASA POR UN DTO

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/usuario/1' \
  -H 'accept: */*'
```

Request URL

http://localhost:8080/api/usuario/1

Server response

Code

Details

200

Response body

```
{
  "id": 1,
  "imagen": "perfil.jpg",
  "nombrePublico": "Aragorn23",
  "listaPersonajes": [],
  "nombrePrivado": "*****y",
  "correo": "****@example.com",
  "contrasena": "*****",
  "conexion": "2023-10-01T10:00:00.000+00:00",
  "fecha_creacion": "2023-10-01T10:00:00.000+00:00",
  "estado": true,
  "tipoUsuario": {
    "tipo_usuario_id": 1,
    "nombre": "USR",
    "descripcion": "Acceso básico al juego, puede crear personajes y jugar."
  }
}
```

Response headers

```
cache-control: no-cache,no-store,max-age=0,must-revalidate
content-type: application/json
expires: 0
pragma: no-cache
```

Responses

```
curl -X 'GET' \

'http://localhost:8080/api/usuario/1' \

-H 'accept: */*'
```

EN CASO DE METER INFORMACIÓN MAL POR EJEMPLO, CLAVES REPETIDAS EXISTENTES, O USAR ENUMS EN MINÚSCULAS REGRESARÁ UN MENSAJE PERSONALIZADO.

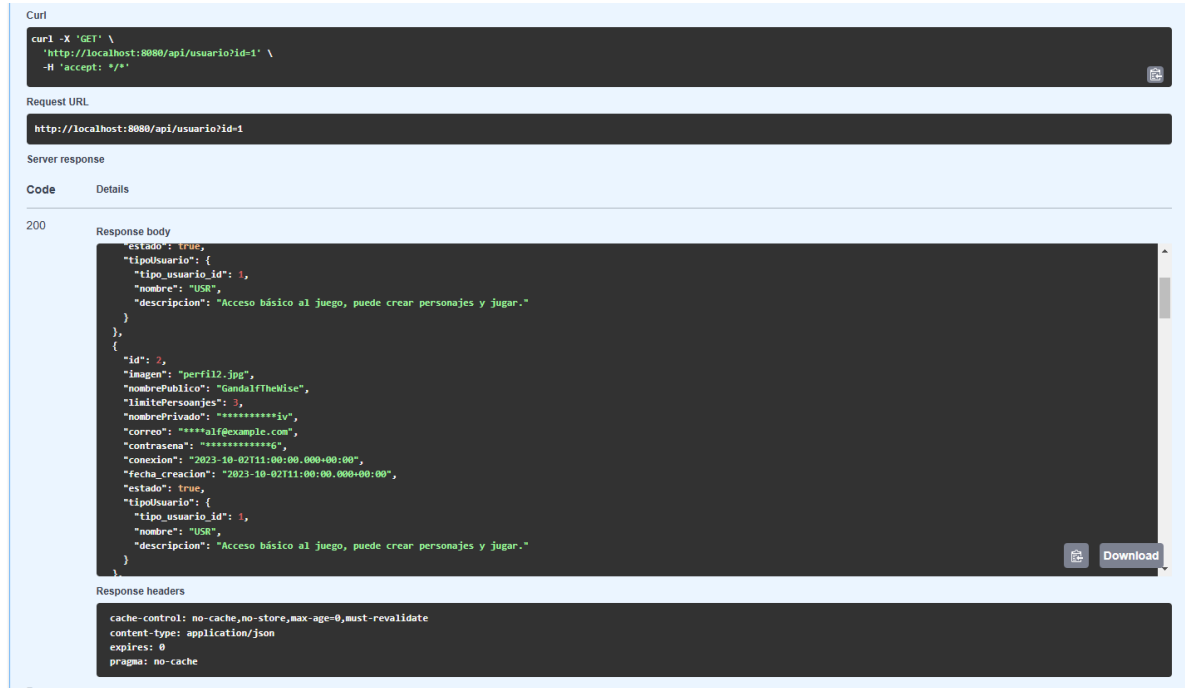
Code	Details
400	Error: response status is 400
Undocumented	<p>Response body</p> <pre>No es posible actualizar revise campos.</pre> <p>Response headers</p> <pre>cache-control: no-cache,no-store,max-age=0,must-revalidate content-length: 39 content-type: text/plain;charset=UTF-8 expires: 0 pragma: no-cache</pre>
Responses	

```
curl -X 'PUT' \
'http://localhost:8080/api/usuario/1' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{
  "usuario_id": 1,
  "imagen_perfil": "img12.jpg",
  "nombre_usuario_pub": "Usuario1",
  "limite_personajes": 3,
  "nombre_usuario_priv": "user1",
  "correo": "user1@example.com",
  "contraseña": "password1",
  "ultima_conexion": "2025-02-23T11:06:25.143Z",
  "fecha_creacion": "2025-02-23T11:06:25.143Z",
  "estado_cuenta": true,
  "tipoUsuario": 1
}'
```

TEN EN CUENTA, MISMO ID Y CAMPOS VÁLIDOS, ES DECIR, NO SE REPITA NOMBRE PRIV, CORREO O EL TIPO DE USUARIO ESTÉ FUERA DE RANGO. PARA BORRAR UN USUARIO. PRIMO DEBES BORRAR TODAS LAS TABLAS UNIDAS A ÉL. ES DECIR SUS PERSONAJES. ES MÁS FÁCIL DESACTIVAR LA CUENTA.

Curl	
<pre>curl -X 'DELETE' \ 'http://localhost:8080/api/usuario/8' \ -H 'accept: */*'</pre>	
Request URL	
<pre>http://localhost:8080/api/usuario/8</pre>	
Server response	
Code	Details
400	Error: response status is 400
Undocumented	<p>Response body</p> <pre>No es posible borrar si tiene campos enlazados.</pre> <p>Response headers</p> <pre>cache-control: no-cache,no-store,max-age=0,must-revalidate content-length: 47 content-type: text/plain;charset=UTF-8 expires: 0 pragma: no-cache</pre>
Responses	

EL MÉTODO GET FILTRO NOS PERMITE VER TODOS O FILTRAR POR TIPO DE USUARIO.



```
curl -X 'GET' \
  'http://localhost:8080/api/usuario?id=1' \
  -H 'accept: */*'

Request URL
http://localhost:8080/api/usuario?id=1

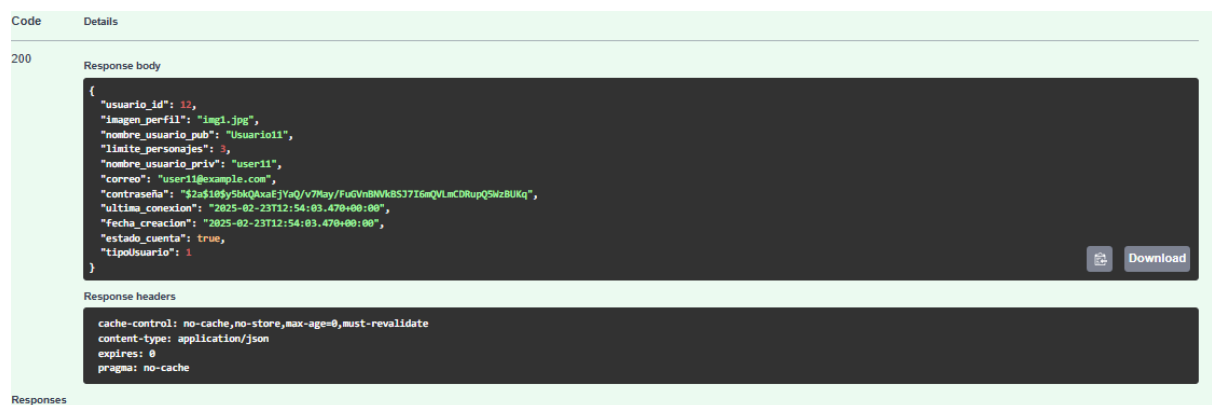
Server response

Code    Details
200

Response body
{
  "estado": true,
  "tipoUsuario": {
    "tipo_usuario_id": 1,
    "nombre": "USR",
    "descripcion": "Acceso básico al juego, puede crear personajes y jugar."
  },
  {
    "id": 2,
    "imagen": "perfil2.jpg",
    "nombrePublico": "GandalfTheWise",
    "limitePersonajes": 3,
    "nombrePrivado": "*****ji",
    "correo": "****ali@example.com",
    "contraseña": "*****+g",
    "conexion": "2023-10-02T11:00:00.000+00:00",
    "fecha_creacion": "2023-10-02T11:00:00.000+00:00",
    "estado": true,
    "tipoUsuario": {
      "tipo_usuario_id": 1,
      "nombre": "USR",
      "descripcion": "Acceso básico al juego, puede crear personajes y jugar."
    }
  }
}

Response headers
cache-control: no-cache,no-store,max-age=0,must-revalidate
content-type: application/json
expires: 0
pragma: no-cache
```

CUANDO INSERTAS UN USUARIO SE ENCRYPTA SU CLAVE.



```
Code    Details
200

Response body
{
  "usuario_id": 12,
  "imagen_perfil": "img1.jpg",
  "nombre_usuario_pub": "Usuario11",
  "limite_personajes": 3,
  "nombre_usuario_priv": "user11",
  "correo": "user11@example.com",
  "contraseña": "$2a$10$y5bkQaxaEjYaq/v7May/FuGVnBNWk8S7T16mQVmcDRupQ5kzBUKq",
  "ultima_conexion": "2025-02-23T12:54:03.470+00:00",
  "fecha_creacion": "2025-02-23T12:54:03.470+00:00",
  "estado_cuenta": true,
  "tipoUsuario": 1
}

Response headers
cache-control: no-cache,no-store,max-age=0,must-revalidate
content-type: application/json
expires: 0
pragma: no-cache
```

ESTRUCTURA BASE DE UN USUARIO SOLO TENER CUIDADO CON LOS CAMPOS, LAS FECHAS SE GENERAN AUTOMÁTICAMENTE.

```
{
  "usuario_id": 1, /* NO INTRODUCIR O ACTUALIZA
  "imagen_perfil": "img1.jpg",
  "nombre_usuario_pub": "Usuario1",
  "limite_personajes": 3,
  "nombre_usuario_priv": "user1", /* NO REPETIR
  "correo": "user1@example.com", /* NO REPETIR
  "contraseña": "password1",
  "ultima_conexion": "2025-02-23T12:49:01.119Z",
  "fecha_creacion": "2025-02-23T12:49:01.119Z",
  "estado_cuenta": true,
  "tipoUsuario": 1 /* VALORES DE 1 a 3
}
```



VAMOS CON OTRO SIMPLE CLASE/TIPO PERSONAJE.

ClasePersonaje API para gestionar clases de personajes	
GET	/api/personaje/clase/{id} Obtener una clase de personaje por ID
PUT	/api/personaje/clase/{id} Actualizar una clase de personaje por ID
DELETE	/api/personaje/clase/{id} Eliminar una clase de personaje por ID
POST	/api/personaje/clase Crear una nueva clase de personaje
GET	/api/personaje/clase/ Obtener todas las clases de personajes

ESTE EL SU GETER POR ID

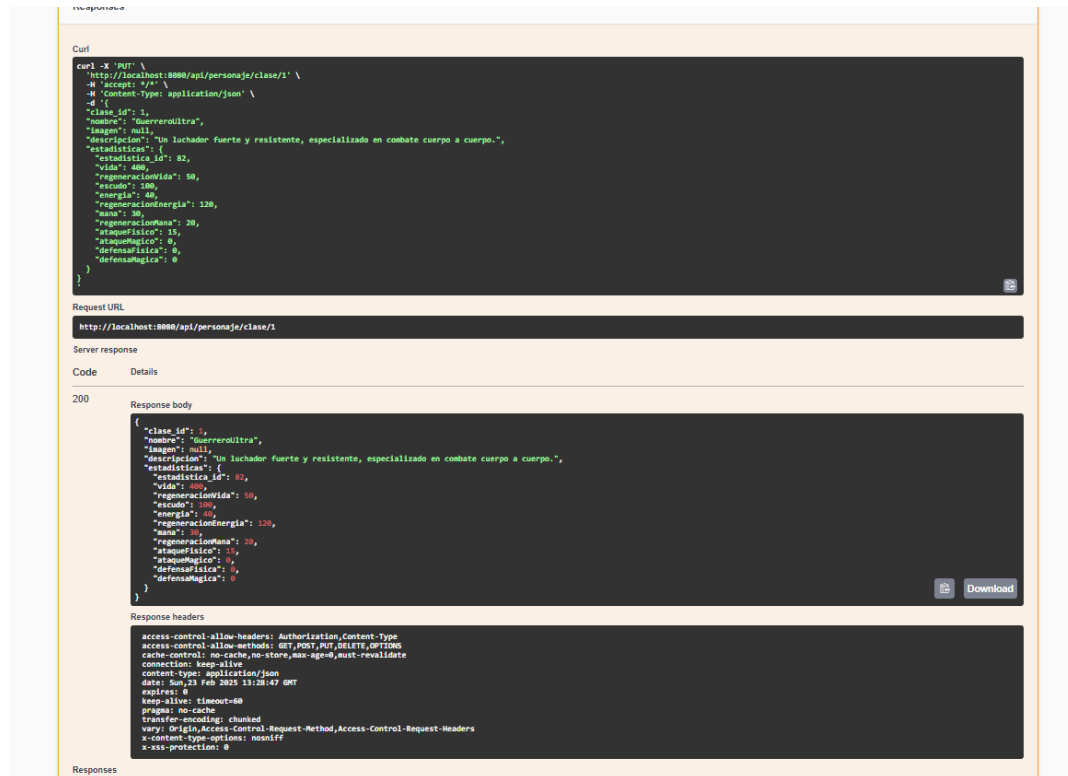
AQUI EMPIEZA LO DIVERTIDO. CUANDO GUARDAMOS UN PERSONAJE, CLASE , MONSTRUO O HABILIDAD DEBEMOS PASAR LOS ATRIBUTOS ASOCIADOS PARA QUE NO SEAN NULOS



```
curl -X 'GET' \
'http://localhost:8080/api/personaje/clase/1' \
-H 'accept: */*'

{
  "class_id": 1,
  "nombre": "Guerrero",
  "imagen": null,
  "descripcion": "Un luchador fuerte y resistente, especializado en combate cuerpo a cuerpo.",
  "estadisticas": {
    "estadistica_id": 82,
    "vida": 400,
    "regeneracionVida": 50,
    "escudo": 100,
    "energia": 40,
    "regeneracionEnergia": 120,
    "mana": 30,
    "regeneracionMana": 20,
    "ataqueFisico": 15,
    "ataqueMagico": 0,
    "defensaFisica": 0,
    "defensaMagica": 0
  }
}
```

POR LO TANTO PARA ACTUALIZAR , DEBEMOS PASAR EL OBJETO ENTERO.

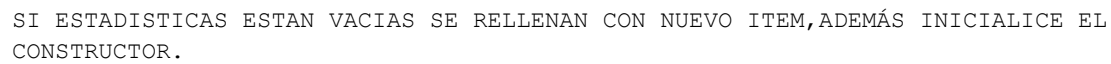


```
curl -X 'PUT' \
  'http://localhost:8080/api/personaje/clase/1' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "clase_id": 1,
    "nombre": "GuerreroUltra",
    "imagen": null,
    "descripcion": "Un luchador fuerte y resistente, especializado en combate cuerpo a cuerpo.",
    "estadisticas": {
      "estadistica_id": 82,
      "vida": 400,
      "regeneracionVida": 50,
      "escudo": 100,
      "energia": 40,
      "regeneracionEnergia": 120,
      "mana": 30,
      "regeneracionMana": 20,
      "ataqueFisico": 15,
      "ataqueMagico": 0,
      "defensaFisica": 0,
      "defensaMagica": 0
    }
  }'
```

```

"nombre": "Guerrero32",
"imagen": null,
"descripcion": "Un luchador fuerte y resistente, especializado en combate cuerpo a cuerpo.",
"estadisticas": {
  "fuerza": 85,
  "velocidad": 70,
  "defensa": 90,
  "agilidad": 60,
  "resistencia": 80,
  "habilidades": [
    "Golpe de Puño",
    "Patada",
    "Defensa",
    "Esquivas",
    "Resistencia"
  ]
}
}

```



POR EJEMPLO EN CREAR PERSONAJE TENGO ESTE VALIDADO..

Code	Description	Links
200	<p>Personaje actualizado correctamente.</p> <p>Media type</p> <p>Controls Accept header.</p> <p>Example Value   Schema</p> <pre>{   "id": 0,   "imagen": "string",   "nombre": "string",   "creacion": "2025-02-23T14:20:40.169Z",   "nivel": 0,   "xp_acumulada": 0,   "almas": 0,   "estadisticas": {     "estadistica_id": 1,     "vida": 100,     "regeneracionVida": 5,     "escudo": 50,     "energia": 100,     "regeneracionEnergia": 10,     "mana": 100,     "regeneracionMana": 5,     "ataqueFisico": 20,     "ataqueMagico": 10,     "defensaFisica": 10,     "defensaMagica": 10   },   "logros": {     "logro_id": 1,     "email": 10,     "miniboss": 0,     "boss": 1,     "muertes_totales": 0,     "total_dano_infligido": 500   } }</pre>	No links
400	<p>El ID proporcionado no coincide con el ID del personaje.</p>	No links

```

"imagen": "FEF44E",
"nombre": "trya44n",
"creacion": "2025-02-23T17:04:02.000+00:00",
"nivel": 43544,
"xp_acumulada": 435,
"almas": 345,
"usuario":{
  "usuario_id" : 1
},
"capacidad_inventario": 20345,
"inventario": null
}

```

The screenshot shows the Postman REST client interface. The left sidebar displays a collection of API endpoints under the name 'Personaje'. The main panel shows a POST request to the endpoint `http://localhost:8080/api/personaje/`. The request body is a JSON object with the following structure:

```

{
  "imagen": "FEF44E",
  "nombre": "trya44n",
  "creacion": "2025-02-23T17:04:02.000+00:00",
  "nivel": 43544,
  "xp_acumulada": 435,
  "almas": 345,
  "usuario": {
    "usuario_id": 1
  },
  "capacidad_inventario": 20345,
  "inventario": null
}

```

The response is a 200 OK status with a response time of 16 ms and a body size of 841 B. The response body is a JSON object with the same structure as the request body, but with some values updated, such as `"id": 14` and `"estadistica_id": 110`.

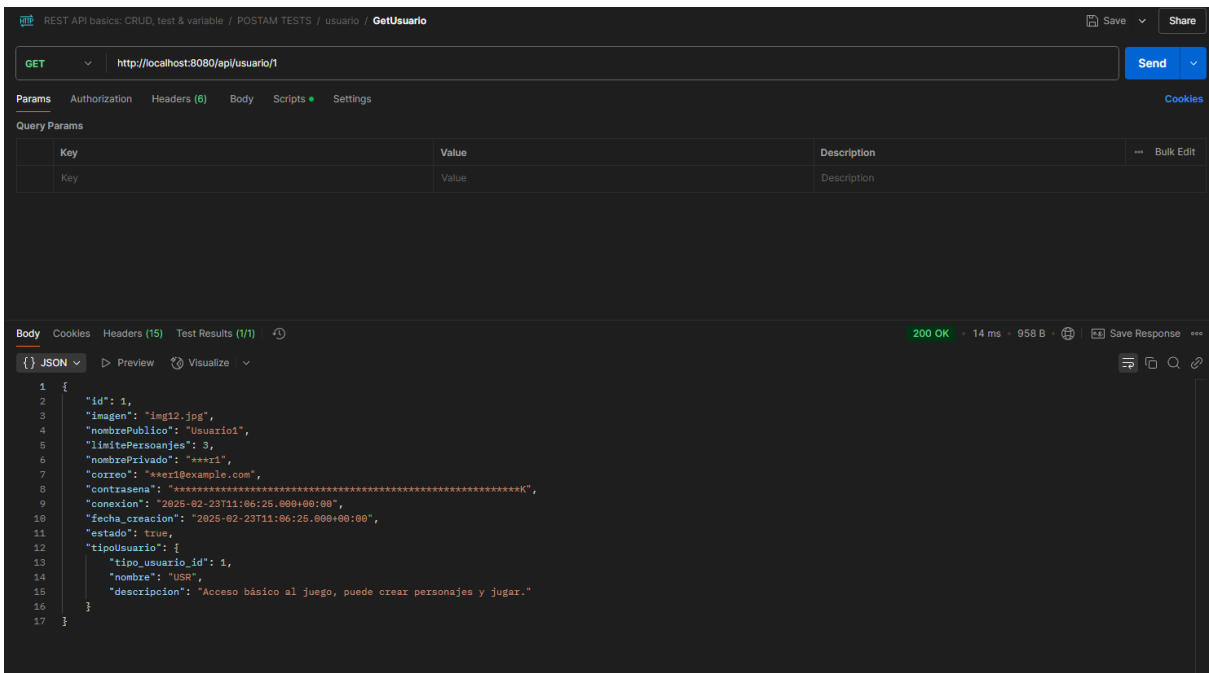
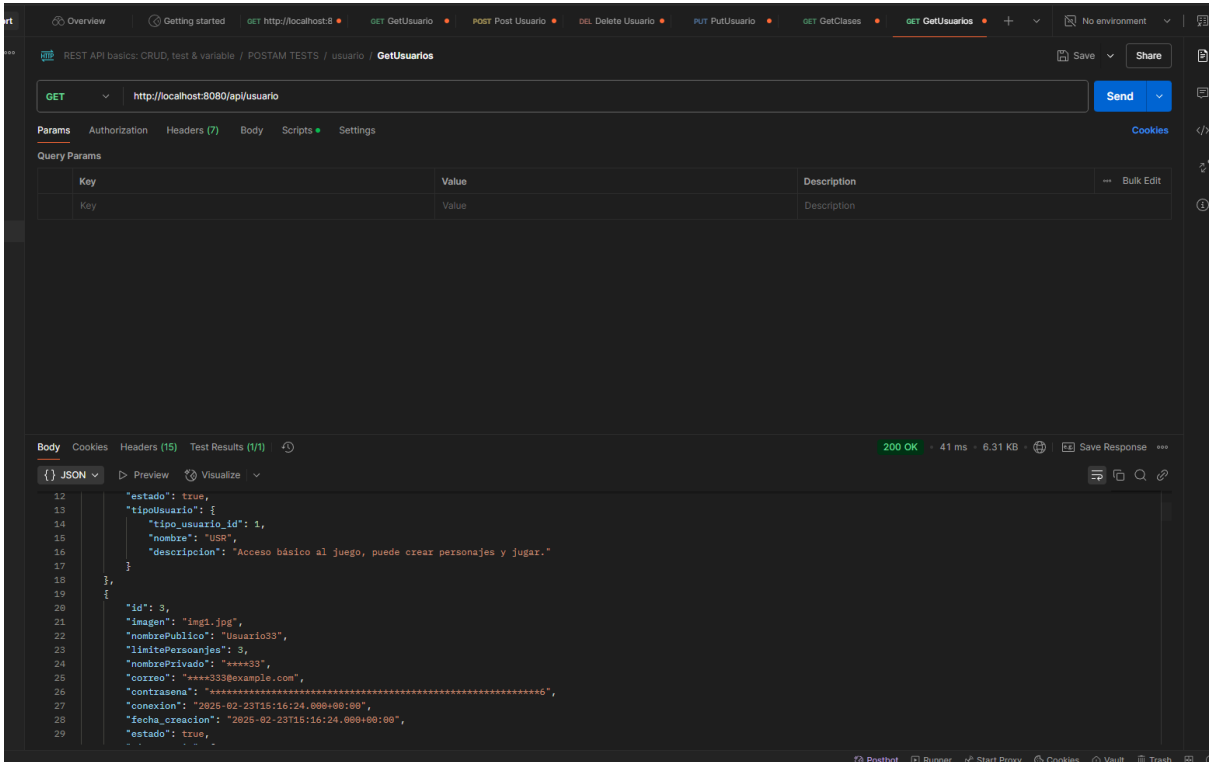
# Conclusiones finales.

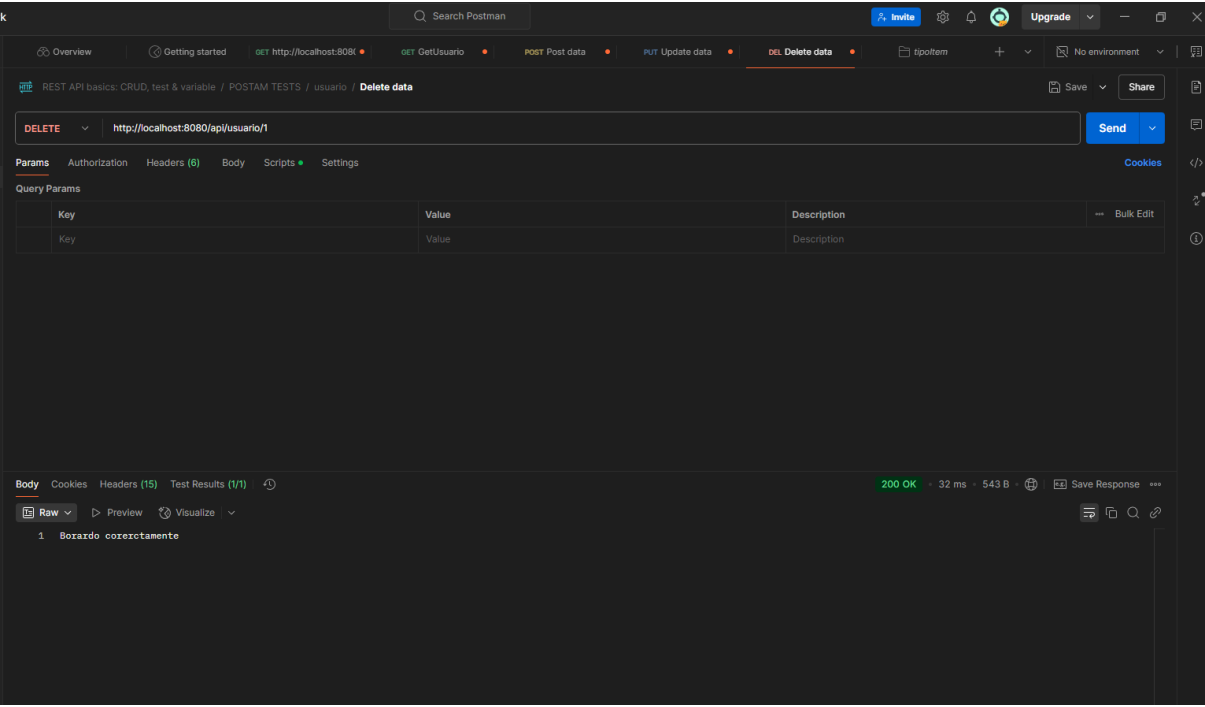
CONCLUSIONES ,SE NECESITA MAS TIEMPO Y EXPERIENCIA PARA LA SEGURIDAD EN API.  
Y FUE DIVERTIDO ACER ESTA.

PRUEBAS DE TODOS LOS ENDPOINTS E INTERFAZ.  
INFORMACIÓN ADICIONAL EN FICHERO HTML Y YAMAL, NO ES GRAN COSA, PERO ES ALGO.  
adjunto imagenes de informacionadicional.

EXPORTAR LIBRERÍAS DE POSTMAN O LO QUE SEA

COSAS DE POSTMAN POR SI ACASO, PERO MEJOR DIGO.. EN LA CARPETA HAY UN HTML , CON AL ABRES EL YAML Y TIENES TODOS LOS ENDPOINTS Y TEST .





Postman interface showing a POST request to `http://localhost:8080/api/usuario`. The request body is a JSON object:

```
1 {
2   "imagen_perfil": "img1123211.jpg",
3   "nombre_usuario_pub": "Usuario1451233211",
4   "limite_personajes": 3,
5   "nombre_usuario_priv": "user123354211",
6   "correo": "user1124543321@example.com",
7   "contraseña": "password",
8   "ultima_conexion": "2025-02-23T15:01:06.599Z",
9   "fecha_creacion": "2025-02-23T15:01:06.599Z",
10  "estado_cuenta": true,
11  "tipoUsuario": 1
12 }
13 }
```

The response is a 200 OK status, indicating success. The response body is a JSON object:

```
1 {
2   "usuario_id": 19,
3   "imagen_perfil": "img1123211.jpg",
4   "nombre_usuario_pub": "Usuario1451233211",
5   "limite_personajes": 3,
6   "nombre_usuario_priv": "user123354211",
7   "correo": "user1124543321@example.com",
8   "contraseña": "$2a$10$2YlssodoEXCppAb9HFd5xuFUIQx6X6.qxTd14Mx6LMBaZx2IFKvG",
9   "ultima_conexion": "2025-02-23T15:01:06.599+00:00",
10  "fecha_creacion": "2025-02-23T15:01:06.599+00:00",
11  "estado_cuenta": true,
12  "tipoUsuario": 1
13 }
```

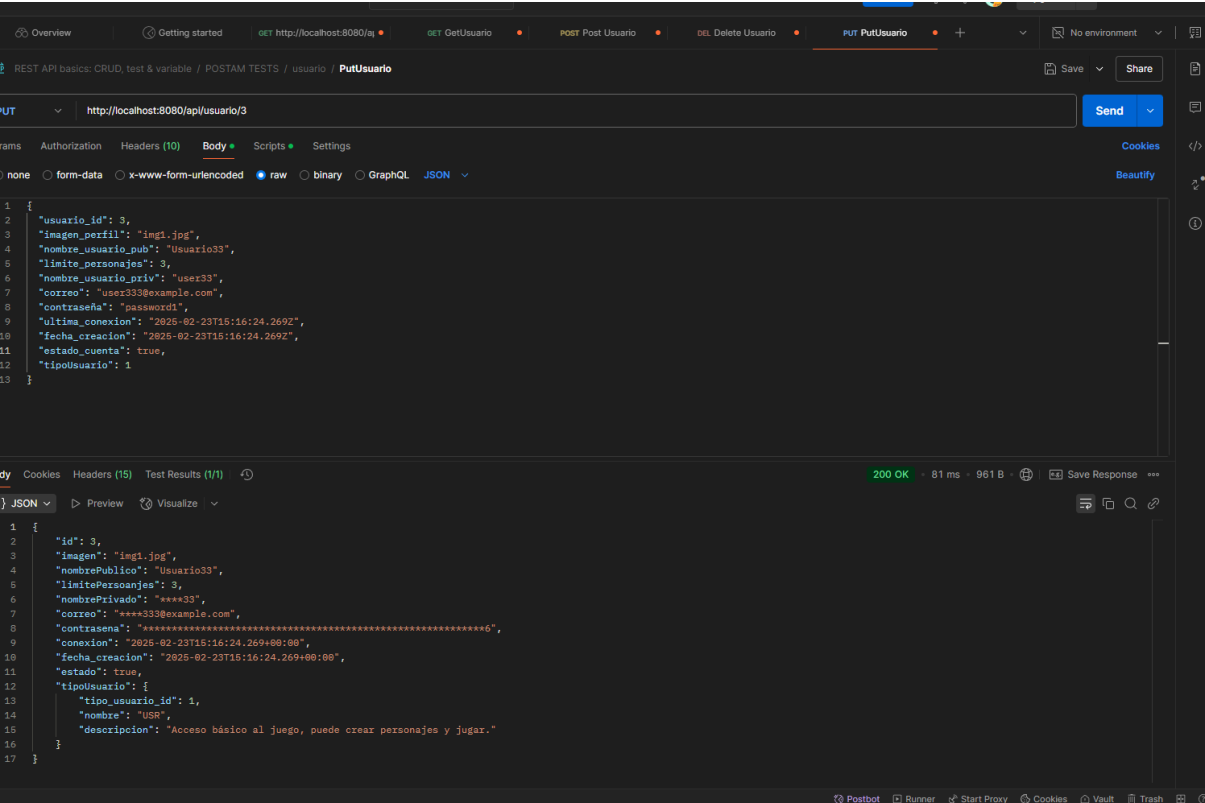
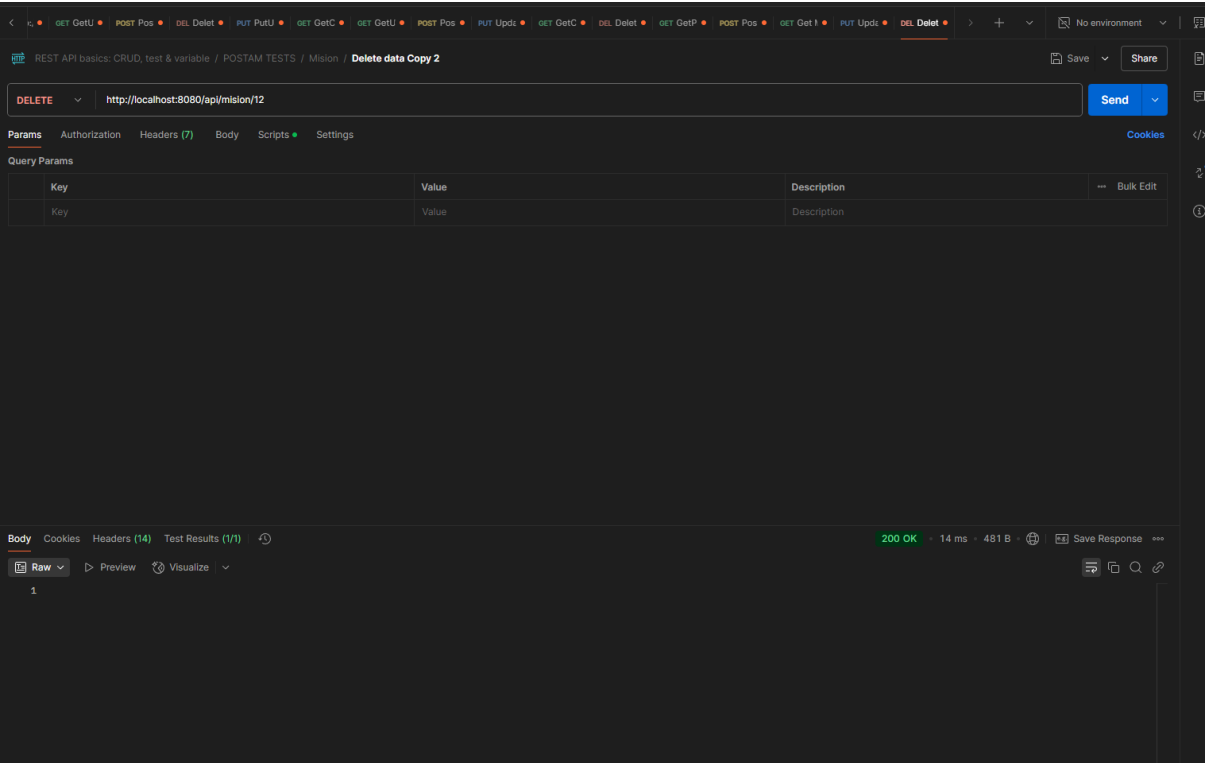
Postman interface showing a PUT request to `http://localhost:8080/api/mision/1`. The request body is a JSON object:

```
1 {
2   "mision_id": 1,
3   "nombre": "Cazar Goblin",
4   "descripcion": "Derrota a 5 goblins",
5   "nivel_minimo": 1,
6   "recompensa_almas": 60,
7   "recompensa_experiencia": 100,
8   "tiempo_limite": 30
9 }
```

The response is a 200 OK status, indicating success. The response body is a JSON object:

```
1 {
2   "nombre": "Cazar Goblin",
3   "descripcion": "Derrota a 5 Goblins",
4   "nivel": 1,
5   "almas": 60,
6   "experiencia": 100,
7   "tiempo": 30,
8   "recompensas": [],
9   "id": 1
10 }
```





Overview Getting started GET http://localhost:8080/ POST Post Usuario DEL Delete Usuario PUT PutUsuario GET GetClases No environment

REST API basics: CRUD, test & variable / POSTAM TESTS / ClasePersonaje / GetClases Save Share

Method GET URL http://localhost:8080/api/personaje/clase/ Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Key Value Description Bulk Edit

200 OK 45 ms 5.07 KB Save Response

JSON Preview Visualize

```
1 {
2   "ataqueMagico": 0,
3   "defensaFisica": 0,
4   "defensaMagica": 0
5 }
6 {
7   "clase_id": 3,
8   "nombre": "Arquero",
9   "imagen": null,
10  "descripcion": "Un tirador experto que ataca desde la distancia con precisión letal.",
11  "estadisticas": {
12    "estadistica_id": 84,
13    "vida": 2500,
14    "regeneracionVida": 500,
15    "escudo": 500,
16    "energia": 300,
17    "regeneracionEnergia": 500,
18    "mana": 400,
```

Overview Getting startx GET http://localhost:8080/ POST Post Usuario DEL Delete Usuari PUT PutUsuario GET GetClases GET GetUsuarios POST Post data Co No environment

REST API basics: CRUD, test & variable / POSTAM TESTS / ClasePersonaje / Post data Copy Save Share

Method POST URL http://localhost:8080/api/personaje/clase Send

Params Authorization Headers (10) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "nombre": "Mago1233",
3   "imagen": "mago.jpg",
4   "descripcion": "Especialista en ehchizos y ataques a larga distabcia"
5 }
```

Body Cookies Headers (15) Test Results (1/1) Save Response

JSON Preview Visualize

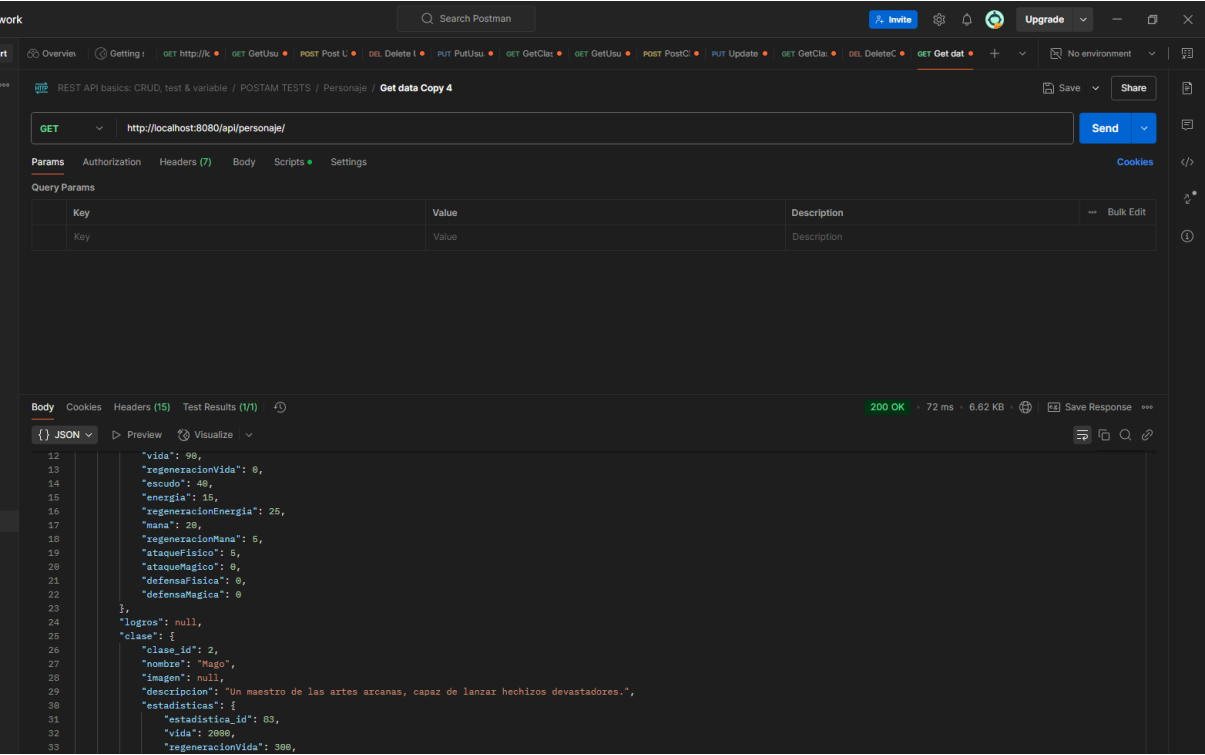
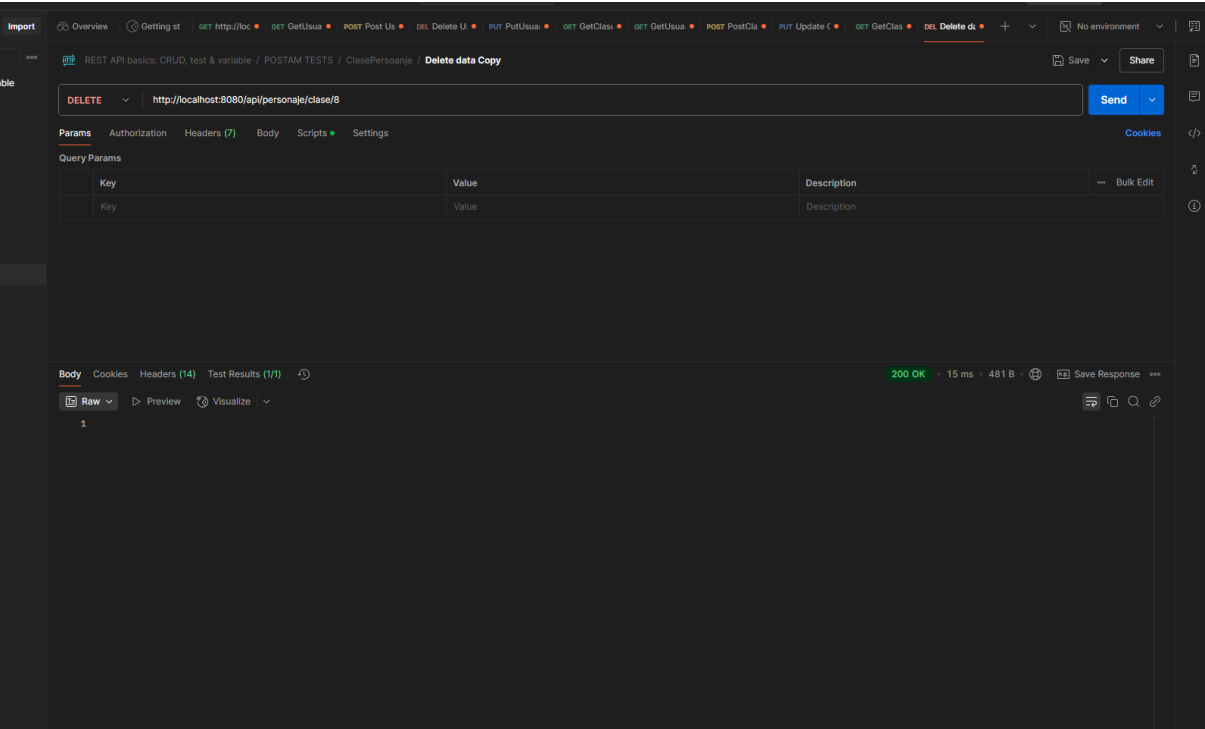
```
1 {
2   "clase_id": 16,
3   "nombre": "Mago1233",
4   "imagen": "mago.jpg",
5   "descripcion": "Especialista en ehchizos y ataques a larga distabcia",
6   "estadisticas": {
7     "estadistica_id": 110,
8     "vida": 0,
9     "regeneracionVida": 0,
10    "escudo": 0,
11    "energia": 0,
12    "regeneracionEnergia": 0,
13    "mana": 0,
```

Postman interface showing a GET request to `http://localhost:8080/api/personaje/clase/1`. The response is a JSON object with the following structure:

```
1 {
2   "clase_id": 1,
3   "nombre": "GuerreroUltra",
4   "imagen": null,
5   "descripcion": "Un luchador fuerte y resistente, especializado en combate cuerpo a cuerpo.",
6   "estadisticas": {
7     "estadistica_id": 82,
8     "vida": 400,
9     "regeneracionVida": 50,
10    "escudo": 100,
11    "energia": 40,
12    "regeneracionEnergia": 120,
13    "mana": 30,
14    "regeneracionMana": 20,
15    "ataqueFisico": 15,
16    "ataqueMagico": 0,
17    "defensaFisica": 0,
18    "defensaMagica": 0
19  }
20 }
```

Postman interface showing a PUT request to `http://localhost:8080/api/personaje/clase/1`. The request body is a JSON object with the following structure:

```
1 {
2   "clase_id": 1,
3   "nombre": "GuerreroUltraFuerte",
4   "imagen": null,
5   "descripcion": "Un luchador fuerte y resistente, especializado en combate cuerpo a cuerpo.",
6   "estadisticas": {
7     "estadistica_id": 82,
8     "vida": 400,
9     "regeneracionVida": 504,
10    "escudo": 100,
11    "energia": 40,
12    "regeneracionEnergia": 1420,
13    "mana": 30,
14    "regeneracionMana": 240,
15    "ataqueFisico": 145,
16    "ataqueMagico": 40,
17    "defensaFisica": 30,
18    "defensaMagica": 30
19  }
20 }
```



REST API basics: CRUD, test & variable / POSTAM TESTS / Mision / **Get data Copy 3**

GET  Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (15) Test Results (1/1) 200 OK 12 ms 1.06 KB Save Response

**JSON** Preview Visualize

```
1 {
2   "nombre": "Cosecha en peligro",
3   "descripcion": "Ayuda al granjero Juan a proteger sus cultivos de las plagas.",
4   "nivel": 1,
5   "almas": 50,
6   "experiencia": 100,
7   "tiempo": 60,
8   "recompensas": [
9     {
10      "item": {
11        "item_id": 21,
12        "nombre": "Mierro",
13        "imagen": "img1.png"
14      }
15    }
16  ]
17 }
```

REST API basics: CRUD, test & variable / POSTAM TESTS / Mision / **Post data Copy 2**

POST  Send

Params Authorization Headers (10) **Body** Scripts Settings Cookies

none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** Beautify

```
1 {
2   "mision_id": 1,
3   "nombre": "Cazar alyo Goblin",
4   "descripcion": "Derrota a 5 Goblins",
5   "nivel_minimo": 1,
6   "recompensa_almas": 50,
7   "recompensa_experiencia": 100,
8   "tiempo_limite": 30
9 }
```

Body Cookies Headers (15) Test Results (1/1) 200 OK 81 ms 663 B Save Response

**JSON** Preview Visualize

```
1 {
2   "nombre": "Cazar alyo Goblin",
3   "descripcion": "Derrota a 5 Goblins",
4   "nivel": 1,
5   "almas": 50,
6   "experiencia": 100,
7   "tiempo": 30,
8   "recompensas": [],
9   "id": 1
10 }
```

REST API basics: CRUD, test & variable / POSTAM TESTS / EstadisticasGenrales / Get data Copy 3

GET http://localhost:8080/ap/estadisticas/ Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

body Cookies Headers (15) Test Results (1/1) 200 OK · 336 ms · 21.37 KB Save Response

JSON Preview Visualize

```
29 {
30   "estadistica_id": 5,
31   "vida": 0,
32   "regeneracionVida": 0,
33   "escudo": 25,
34   "energia": 0,
35   "regeneracionEnergia": 35,
36   "mana": 0,
37   "regeneracionMana": 0,
38   "ataqueFisico": 0,
39   "ataqueMagico": 0,
40   "defensaFisica": 0,
41   "defensaMagica": 0
42 }
```

Search Postman

REST API basics: CRUD, test & variable / POSTAM TESTS / EstadisticasGenrales / Post data Copy 3

POST http://localhost:8080/ap/estadisticas/ Send

Params Authorization Headers (10) Body Scripts Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "vida": 100,
3   "regeneracionVida": 5,
4   "escudo": 50,
5   "energia": 100,
6   "regeneracionEnergia": 10,
7   "mana": 100,
8   "regeneracionMana": 5,
9   "ataqueFisico": 20,
10  "ataqueMagico": 15,
11  "defensaFisica": 10,
12  "defensaMagica": 10
13 }
14
```

body Cookies Headers (15) Test Results (1/1) 201 Created · 30 ms · 738 B Save Response

JSON Preview Visualize

```
1 {
2   "estadistica_id": 115,
3   "vida": 100,
4   "regeneracionVida": 5,
5   "escudo": 50,
6   "energia": 100,
7   "regeneracionEnergia": 10,
8   "mana": 100,
9   "regeneracionMana": 5,
10  "ataqueFisico": 20,
11  "ataqueMagico": 15,
12  "defensaFisica": 10,
13  "defensaMagica": 10
14 }
```

Search Postman

REST API basics: CRUD, test & variable / POSTAM TESTS / EstadísticasGenrales / Update data Copy 3

PUT <http://localhost:8080/api/estadisticas/115> Send

Params Authorization Headers (10) **Body** Scripts Settings Cookies </> Beautify

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON

```
1 {
2   "estadistica_id": 115,
3   "vida": 150,
4   "regeneracionVida": 5,
5   "escudo": 50,
6   "energia": 100,
7   "regeneracionEnergia": 10,
8   "mana": 100,
9   "regeneracionMana": 5,
10  "ataqueFisico": 20,
11  "ataqueMagico": 15,
12  "defensaFisica": 10,
13  "defensaMagica": 10
14 }
```

Body Cookies Headers (14) Test Results (0/1) 400 Bad Request · 8 ms · 565 B Save Response

Raw Preview Visualize

1 El ID proporcionado no coincide con el ID de las estadísticas.

REST API basics: CRUD, test & variable / POSTAM TESTS / EstadísticasGenrales / Update data Copy 3

PUT <http://localhost:8080/api/estadisticas/115> Send

Params Authorization Headers (10) **Body** Scripts Settings Cookies </> Beautify

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON

```
1 {
2   "estadistica_id": 115,
3   "vida": 150,
4   "regeneracionVida": 5,
5   "escudo": 50,
6   "energia": 100,
7   "regeneracionEnergia": 10,
8   "mana": 100,
9   "regeneracionMana": 5,
10  "ataqueFisico": 20,
11  "ataqueMagico": 15,
12  "defensaFisica": 10,
13  "defensaMagica": 10
14 }
```

Body Cookies Headers (15) Test Results (1/1) 200 OK · 15 ms · 733 B Save Response

JSON Preview Visualize

```
1 {
2   "estadistica_id": 115,
3   "vida": 150,
4   "regeneracionVida": 5,
5   "escudo": 50,
6   "energia": 100,
7   "regeneracionEnergia": 10,
8   "mana": 100,
9   "regeneracionMana": 5,
10  "ataqueFisico": 20,
11  "ataqueMagico": 15,
12  "defensaFisica": 10,
13  "defensaMagica": 10
14 }
```

REST API basics: CRUD, test & variable / POSTAM TESTS / EstadísticasGenrales / Delete data Copy 3

DELETE http://localhost:8080/ap/estadisticas/115

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (15) Test Results (1/1)

200 OK · 14 ms · 556 B · Save Response

Raw Preview Visualize

1 Estadística eliminada con éxito.

Search Postman

POSTAM TESTS / Efectos / Get data Copy 4

GET http://localhost:8080/ap/efectos/1

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (15) Test Results (1/1)

200 OK · 13 ms · 994 B · Save Response

JSON Preview Visualize

```
{
  "vida": 0,
  "regeneracionVida": 0,
  "escudo": 0,
  "energia": 0,
  "regeneracionEnergia": 20,
  "mana": 0,
  "regeneracionMana": 0,
  "ataqueFisico": 0,
  "ataqueMagico": 0,
  "defensaFisica": 0,
  "defensaMagica": 0
}
```





```

//nombre privado y unico de cada usuario
@NotNull
@Size(max = 100)
@Column(name = "nombre_usuario_priv", nullable = false, unique = true, length = 100)
@Schema(description = "Nombre privado del usuario (para login)", example = "user1")
private String nombre_usuario_priv;

//Correo de cada usuario
@NotNull
>Email
@Size(max = 100)
@Column(name = "correo", nullable = false, unique = true, length = 100)
@Schema(description = "Correo electrónico del usuario", example = "user1@example.com")
private String correo;

//Contraseña de cada usuario
@NotNull
@Column(name = "contraseña", nullable = false, length = 255)
@Schema(description = "Contraseña del usuario", example = "password1")
private String contraseña;

//Ultima conexion
@NotNull
@DateTimeFormat
@Column(name = "ultima_conexion")
@Schema(description = "Fecha y hora de la última conexión del usuario", example = "2023-10-01T12:00:00")
private Date ultima_conexion;

//Fecha en la que se crea el usuario
@NotNull
@DateTimeFormat
@Column(name = "fecha_creacion", nullable = false)
@Schema(description = "Fecha y hora de creación del usuario", example = "2023-10-01T12:00:00")
private Date fecha_creacion = new Date();

//Estado de la cuenta sin ser borrada, activa o inactiva
@NotNull
@Column(name = "estado_cuenta", nullable = false)
@Schema(description = "Estado de la cuenta del usuario (activa/inactiva)", example = "true")
private boolean estado_cuenta;

```

work

Search Postman

Invite Upgrade

REST API basics: CRUD, test & variable / POSTAM TESTS / Efectos / Post Efecto

POST http://localhost:8080/api/efectos

Send

Params Authorization Headers (10) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
3  "imagen_icono": "potenciacion.png",
4  "nombre": "Fuerza",
5  "descripcion": "Aumenta el ataque en 5 puntos",
6  "tipo": "BUFF",
7  "tipo_afectado": "PERSONAJE",
8  "duracion_efecto": 5,
9  "intervalos_efecto": 1,
10 "acumulaciones": 0,
11 "estadisticas": {
12   "estadistica_id": 1,
13   "vida": 100,
14   "regeneracionVida": 5,
15   "escudo": 50,
16   "energia": 100,
17   "regeneracionEnergia": 10,
18   "mana": 100,
19   "regeneracionMana": 5,
20   "ataqueFisico": 20,
21   "ataqueMagico": 15,
22   "defensaFisica": 10,
23   "defensaMagica": 10
24 }
25 }
```

Body Cookies Headers (15) Test Results (1/1)

JSON Preview Visualize

```
4  "nombre": "Fuerza",
5  "descripcion": "Aumenta el ataque en 5 puntos",
6  "tipo": "BUFF",
7  "tipo_afectado": "PERSONAJE",
8  "duracion_efecto": 5.0,
9  "intervalos_efecto": 1.0,
10 "acumulaciones": 0,
11 "estadisticas": {
12   "estadistica_id": 1,
13   "vida": 100,
14   "regeneracionVida": 5,
15   "escudo": 50,
16   "energia": 100,
17   "regeneracionEnergia": 10,
18   "mana": 100,
19   "regeneracionMana": 5,
20   "ataqueFisico": 20,
21   "ataqueMagico": 15,
22   "defensaFisica": 10,
23   "defensaMagica": 10
24 }
```

200 OK · 63 ms · 966 B

Save Response

Postman Runner Start Proxy Cookies Vault Trash

REST API basics: CRUD, test & variable / POSTMAN TESTS / Efectos / Update Efecto

Method: PUT URL: http://localhost:8080/ap/efectos/1

Body: raw

```
1 {
2   "efecto_id": 1,
3   "imagen_icono": "potenciacion.png",
4   "nombre": "FUERZA",
5   "descripcion": "Aumenta el ataque en 5 puntos",
6   "tipo": "BUFF",
7   "tipo_afectado": "PERSONAJE",
8   "duracion_efecto": 5,
9   "intervalos_efecto": 1,
10  "acumulaciones": 0,
11  "estadisticas": {
12    "estadistica_id": 1,
13    "vida": 100,
14    "regeneracionVida": 5,
15    "escudo": 50,
16    "energia": 100,
17    "regeneracionEnergia": 10,
18  }
```

200 OK · 18 ms · 966 B

JSON

```
1 {
2   "efecto_id": 1,
3   "imagen_icono": "potenciacion.png",
4   "nombre": "FUERZA",
5   "descripcion": "Aumenta el ataque en 5 puntos",
6   "tipo": "BUFF",
7   "tipo_afectado": "PERSONAJE",
8   "duracion_efecto": 5.0,
9   "intervalos_efecto": 1.0,
10  "acumulaciones": 0,
11  "estadisticas": {
12    "estadistica_id": 1,
13    "vida": 100,
14    "regeneracionVida": 5,
15    "escudo": 50,
16    "energia": 100,
17    "regeneracionEnergia": 10,
18    "mana": 100,
19    "regeneracionMana": 5,
```

REST API basics: CRUD, test & variable / POSTMAN TESTS / Efectos / Delete Efecto

Method: DELETE URL: http://localhost:8080/ap/efectos/20

200 OK · 18 ms · 560 B

Body: Raw

1 Efecto de estado eliminado con éxito.

```

/**
 * Obtener todos los efectos de estado.
 *
 * @return Lista de todos los efectos de estado.
 */
@GetMapping("/") no usages 1 Laureano De Sousa
@Operation(summary = "Obtener todos los efectos de estado")
public ResponseEntity<?> obtenerUsuario() {
    try {
        return ResponseEntity.ok(efectoEstadoService.getAll());
    } catch (Exception e) {
        return ResponseEntity.status(500).body("Error al obtener los efectos de estado: " + e.getMessage());
    }
}

/**
 * Obtener un efecto de estado por su ID.
 *
 * @param id el ID del efecto de estado a obtener.
 * @return ResponseEntity con el efecto de estado o un mensaje de error si no se encuentra.
 */
@GetMapping("/{id}") 1 Laureano De Sousa
@Operation(summary = "Obtener un efecto de estado por ID")
public ResponseEntity<?> obtener(@PathVariable Long id) {
    try {
        EfectoEstado efectoEstado = efectoEstadoService.getByID(id);
        if (efectoEstado == null) {
            return ResponseEntity.status(404).body("Efecto de estado no encontrado con ID: " + id);
        }
        return ResponseEntity.ok(efectoEstado);
    } catch (Exception e) {
        return ResponseEntity.status(500).body("Error al obtener el efecto de estado: " + e.getMessage());
    }
}

```

```

// Convertor de Usuario a UsuarioDTO
public UsuarioDTO convertorUsuarioDTO(Usuario usuario) { 3 usages 1 Laureano De Sousa +1
    UsuarioDTO usuarioDTO = new UsuarioDTO();
    usuarioDTO.setId(usuario.getUsuario_id());
    usuarioDTO.setImagen(usuario.getImagen_perfil());
    usuarioDTO.setNombrePublico(usuario.getNombre_usuario_pub());
    usuarioDTO.setNombrePrivado(CensorController.ocultarNumero(usuario.getNombre_usuario_priv(), mostrar: 2));
    usuarioDTO.setCorreo(CensorController.ocultarEmail(usuario.getCorreo(), mostrar: 3));
    usuarioDTO.setContraseña(CensorController.ocultarNumero(usuario.getContraseña(), mostrar: 1));
    usuarioDTO.setLimitePersonajes(usuario.getLimite_personajes());
    usuarioDTO.setConexion(usuario.getUltima_conexion());
    usuarioDTO.setTipoUsuario( tipoUsuarioService.getByID(usuario.getTipoUsuario() ));
    usuarioDTO.setFecha_creacion(usuario.getFecha_creacion());
    usuarioDTO.setEstado(usuario.isEstado_cuenta());
    return usuarioDTO;
}

```

```

private List<Habilidad> hl; 3 usages
private List<EfectoEstado> el; 3 usages

// listar todas las habilidades y efectos
@GetMapping @Laureano De Sousa *
public String listar(Model model) {
    try {
        try{ el = efectoEstadoService.getAll(); }catch (Exception e){}
        try{ hl = habilidadService.getAll(); }catch (Exception e){}
        List<HabilidadEfecto> habilidades = service.getAll();
        model.addAttribute( attributeName: "habilidadesEfectos", habilidades);
        model.addAttribute( attributeName: "habilidadEfecto", new HabilidadEfecto());
        model.addAttribute( attributeName: "habilidadLista", hl);
        model.addAttribute( attributeName: "efectoLista", el);
        return "admin/habilidadesEfectos";
    } catch (Exception e) {
        model.addAttribute( attributeName: "error", attributeValue: "Error al cargar las habilidades: " + e.getMessage());
        return "admin/habilidadesEfectos";
    }
}

```

```

//nombre privado y unico de cada usuario
@NotNull
@Size(max = 100)
@Column(name = "nombre_usuario_priv", nullable = false, unique = true, length = 100)
@Schema(description = "Nombre privado del usuario (para login)", example = "user1")
private String nombre_usuario_priv;

//Correo de cada usuario
@NotNull
@email
@Size(max = 100)
@Column(name = "correo", nullable = false, unique = true, length = 100)
@Schema(description = "Correo electrónico del usuario", example = "user1@example.com")
private String correo;

//Contraseña de cada usuario
@NotNull
@Column(name = "contraseña", nullable = false, length = 255)
@Schema(description = "Contraseña del usuario", example = "password1")
private String contraseña;

//Ultima conexion
@NotNull
@DateTimeFormat
@Column(name = "ultima_conexion")
@Schema(description = "Fecha y hora de la última conexión del usuario", example = "2023-10-01T12:00:00")
private Date ultima_conexion;

//Fecha en la que se crea el usuario
@NotNull
@DateTimeFormat
@Column(name = "fecha_creacion", nullable = false)
@Schema(description = "Fecha y hora de creación del usuario", example = "2023-10-01T12:00:00")
private Date fecha_creacion = new Date();

//Estado de la cuenta sin ser borrada, activa o inactiva
@NotNull
@Column(name = "estado_cuenta", nullable = false)
@Schema(description = "Estado de la cuenta del usuario (activa/inactiva)", example = "true")
private boolean estado_cuenta;

```

```

// (Correcto)
@NoArgsConstructor 22 usages 1 Laureano De Sousa
@Entity
@Table(name = "personaje_mision")
@Schema(description = "Entidad que representa la relación entre un personaje y una misión")
@Getter
@Setter
public class PersonajeMision {

    //ID de la relacion
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long personaje_mision_id;

    //Personaje relacionado N:1
    @ManyToOne
    @MapsId("personaje_id")
    @JoinColumn(name = "personaje_id", nullable = false)
    @Schema(description = "Personaje asociado a la misión")
    private Personaje personaje;

    //Mision relacionada N:1
    @ManyToOne
    @MapsId("mision_id")
    @JoinColumn(name = "mision_id", nullable = false)
    @Schema(description = "Misión asociada al personaje")
    private Mision mision;

    //Fecha en la que se recibe la mision
    @DateTimeFormat
    @Column(name = "fecha_inicio", nullable = false)
    @Schema(description = "Fecha y hora de inicio de la misión", example = "2023-10-01T12:00:00")
    private Date fecha_inicio;

    //Fecha en la que se termina
    @DateTimeFormat
    @Column(name = "fecha_fin")
    @Schema(description = "Fecha y hora de finalización de la misión", example = "2023-10-02T12:00:00")
    private Date fecha_fin;

    //Estado de la mision
    @Enumerated(EnumType.STRING)
    @Column(name = "estado", nullable = false)
    @Schema(description = "Estado de la misión (en progreso, completada, fallida)", example = "en progreso")
    private EstadoMision estado;
}

```

```

/**
 * Obtener todos los efectos de estado.
 *
 * @return Lista de todos los efectos de estado.
 */
@GetMapping("/") no usages 1 Laureano De Sousa
@Operation(summary = "Obtener todos los efectos de estado")
public ResponseEntity<?> obtenerUsuario() {
    try {
        return ResponseEntity.ok(efectoEstadoService.getAll());
    } catch (Exception e) {
        return ResponseEntity.status(500).body("Error al obtener los efectos de estado: " + e.getMessage());
    }
}

/**
 * Obtener un efecto de estado por su ID.
 *
 * @param id el ID del efecto de estado a obtener.
 * @return ResponseEntity con el efecto de estado o un mensaje de error si no se encuentra.
 */
@GetMapping("/{id}") 1 Laureano De Sousa
@Operation(summary = "Obtener un efecto de estado por ID")
public ResponseEntity<?> obtener(@PathVariable Long id) {
    try {
        EfectoEstado efectoEstado = efectoEstadoService.getByID(id);
        if (efectoEstado == null) {
            return ResponseEntity.status(404).body("Efecto de estado no encontrado con ID: " + id);
        }
        return ResponseEntity.ok(efectoEstado);
    } catch (Exception e) {
        return ResponseEntity.status(500).body("Error al obtener el efecto de estado: " + e.getMessage());
    }
}

```