

Gym Management System

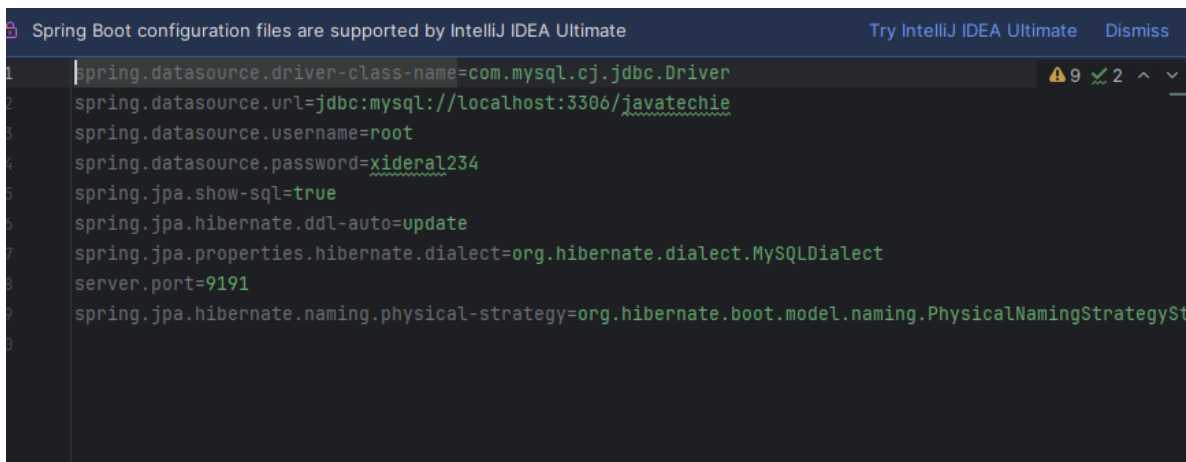
Sistema de gestión de gimnasio desarrollado en Spring Boot

Tecnologías

- Java 1.8
- Spring Boot 2.7.12
- Spring Data JPA
- MySQL
- Lombok
- Maven

Configuración (application.properties)

Ruta: src/main/resources/application.properties



```
Spring Boot configuration files are supported by IntelliJ IDEA Ultimate
Try IntelliJ IDEA Ultimate Dismiss

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/javatechie
spring.datasource.username=root
spring.datasource.password=xideral234
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
server.port=9191
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategySt
```

1 Main Class

Ruta: src/main/java/com/gym/GymManagementApplication.java

```
package com.gym;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

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

2 BaseEntity

Ruta: src/main/java/com/gym/common/BaseEntity.java

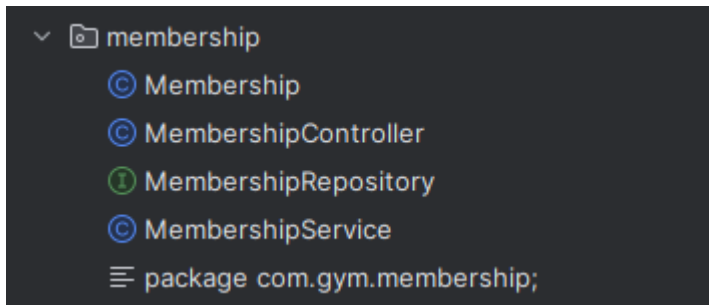
```
package com.gym.common;

import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.MappedSuperclass;
import lombok.Data;

@MappedSuperclass
@Data
public class BaseEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
}
```

3 Modulo Membership

Ruta: src/main/java/com/gym/membership/



4 Manejo de Módulos y Eventos

El proyecto está dividido en módulos independientes:

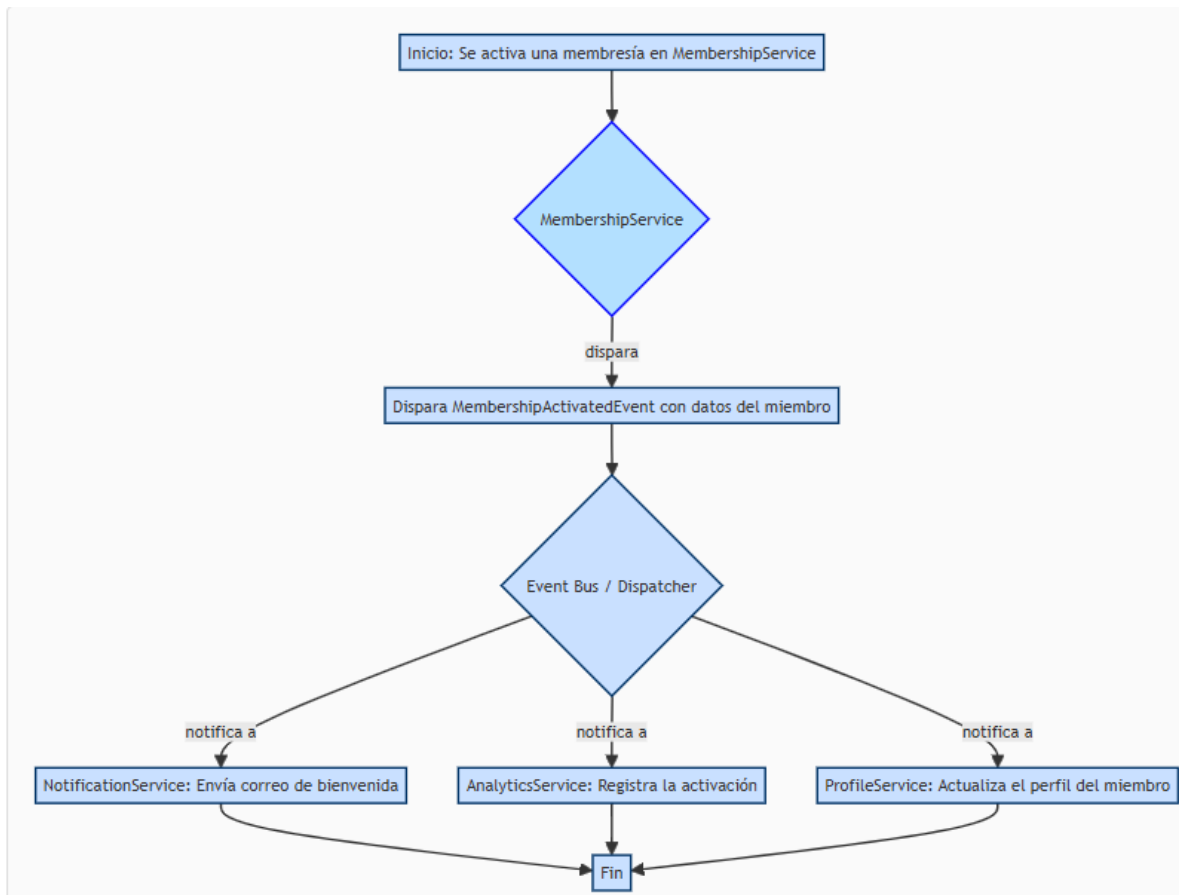
- ✓ Membership,
- ✓ Classes,
- ✓ Equipment,
- ✓ Trainers,
- ✓ Billing
- ✓ Access.

Cada módulo tiene su propia: - Entidad (Entity) - Repositorio (Repository)
- Servicio (Service) - Controlador (Controller)

Comunicación entre módulos con eventos

- **ApplicationEventPublisher:** Cada servicio puede disparar eventos usando `publisher.publishEvent(event)`.
- **Eventos definidos:**
 - ✓ `MembershipActivatedEvent`
 - ✓ `ClassBookedEvent`
 - ✓ `EquipmentMaintenanceEvent`
 - ✓ `AccessGrantedEvent`
- **Listeners:** Otros servicios o componentes pueden escuchar estos eventos mediante clases anotadas con `@EventListener`.
- Esto permite que un módulo notifique a otros módulos sobre acciones importantes (p.ej., una membresía activada o clase reservada) sin acoplar los módulos directamente.

Ejemplo de flujo de evento



1. Se activa una membresía en MembershipService.
2. Se dispara MembershipActivatedEvent con los datos del miembro.
3. Otros módulos o componentes registrados como listeners reciben el evento y pueden reaccionar (p.ej., enviar notificación o actualizar otra entidad).

Esta arquitectura desacoplada mejora la escalabilidad y mantenibilidad del sistema.

5 Otros Módulos

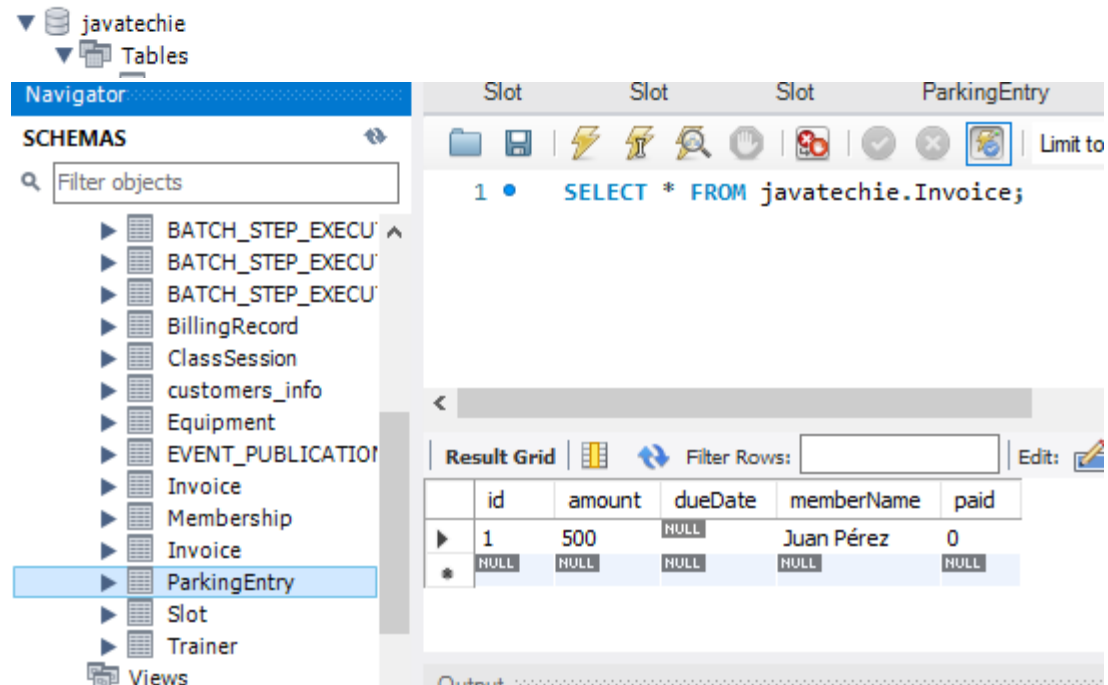
Los módulos Classes, Equipment, Trainers, Billing y Access siguen la misma estructura: Entidad, Repositorio, Servicio y Controlador.

6 Eventos

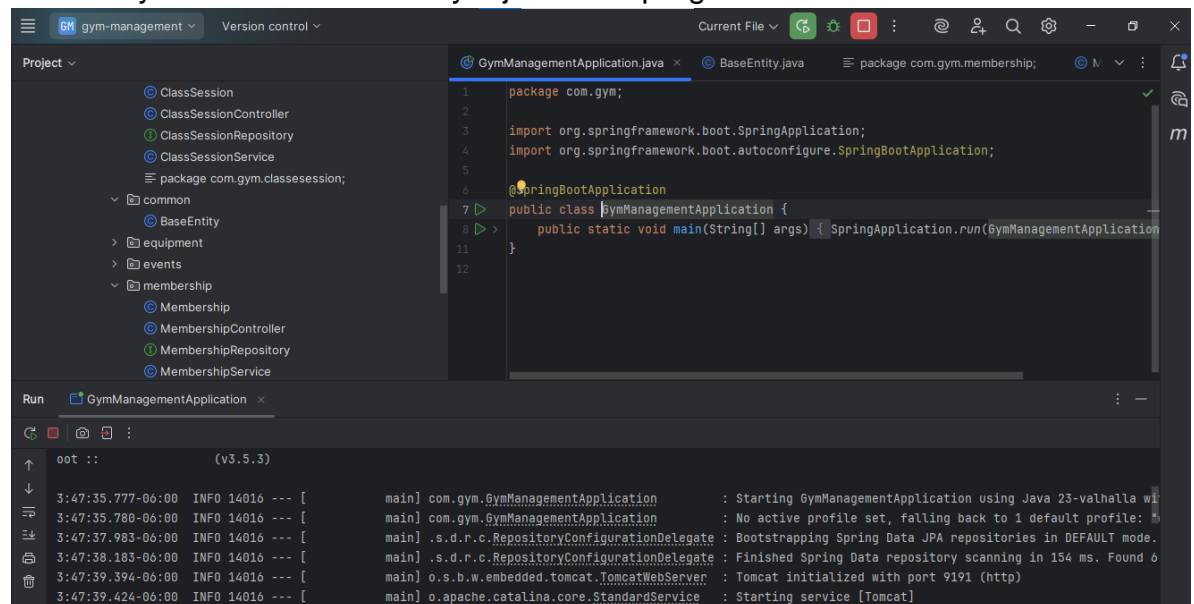
- MembershipActivatedEvent
- ClassBookedEvent
- EquipmentMaintenanceEvent
- AccessGrantedEvent

7 Ejecucion

1. Crear base de datos: CREATE DATABASE javatechie;



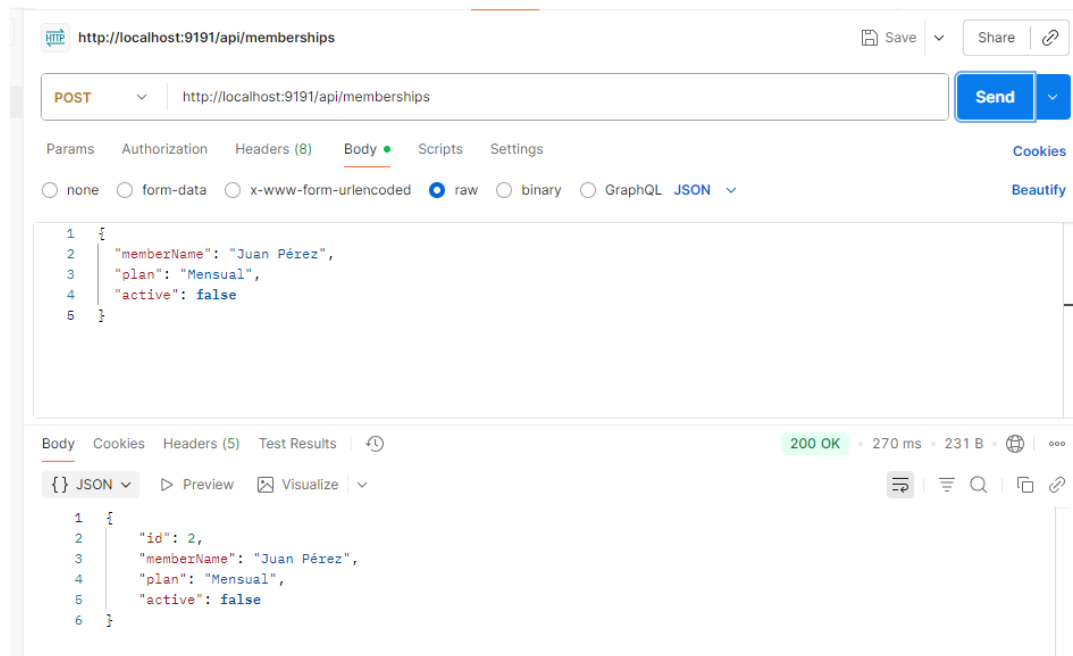
2. Abrir Proyecto en IntelliJ IDEA y ejecutar el programa



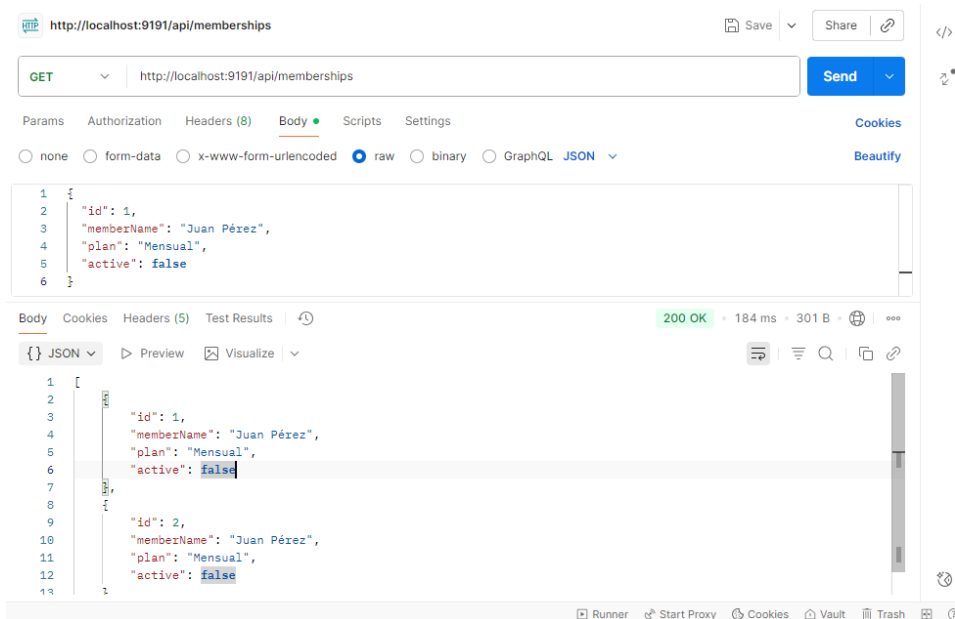
Prueba de Endpoints con Postman.

Membresías

- Crear nueva membresía



Listar membresías



Classes

Crear clase:

- ✓ POST <http://localhost:9191/api/classes>
- ✓ Content-Type: application/json

The screenshot shows a REST client interface with a POST request to `http://localhost:9191/api/classes`. The request body is a JSON object: `{ "className": "Yoga", "schedule": "2025-10-01 10:00" }`. The response is a 200 OK status with a response time of 38 ms and a body size of 245 B. The response body is a JSON object: `{ "id": 2, "name": null, "instructor": null, "schedule": "2025-10-01 10:00", "capacity": 8 }`.

```
POST http://localhost:9191/api/classes
```

Params Authorization Headers (8) Body Scripts Settings Cookies Beautify

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

```
1 {
2   |   "className": "Yoga",
3   |   "schedule": "2025-10-01 10:00"
4   | }
5   |
6   |
```

Body Cookies Headers (5) Test Results 200 OK 38 ms 245 B

{ } JSON Preview Visualize

```
1 {
2   |   "id": 2,
3   |   "name": null,
4   |   "instructor": null,
5   |   "schedule": "2025-10-01 10:00",
6   |   "capacity": 8
7   | }
```

Reservar clase:

- POST <http://localhost:9191/api/classes/1/book?member=Juan Pérez>

The screenshot shows a REST client interface with a POST request to `http://localhost:9191/api/classes/1/book?member=Juan Pérez`. The request body is empty. The response is a 200 OK status with a response time of 36 ms and a body size of 245 B. The response body is a JSON object: `{ "id": 1, "name": null, "instructor": null, "schedule": "2025-10-01 10:00", "capacity": 8 }`.

```
POST http://localhost:9191/api/classes/1/book?member=Juan Pérez
```

Params Authorization Headers (7) Body Scripts Settings Cookies Beautify

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

```
1 Ctrl+Alt+P to Ask AI
```

Body Cookies Headers (5) Test Results 200 OK 36 ms 245 B

{ } JSON Preview Visualize

```
1 {
2   |   "id": 1,
3   |   "name": null,
4   |   "instructor": null,
5   |   "schedule": "2025-10-01 10:00",
6   |   "capacity": 8
7   | }
```

Equipos

- Registrar equipo:
- POST `http://localhost:9191/api/equipment`

The screenshot shows a REST client interface with the URL `http://localhost:9191/api/equipment`. The request method is **POST**. The request body is a JSON object: `{ "name": "Cinta de correr", "status": "Disponible" }`. The response status is **200 OK** with a response time of 35 ms and a body size of 239 B. The response body is a JSON object: `{ "id": 2, "name": "Cinta de correr", "serialNumber": null, "status": "Disponible" }`.

```
1 {
2   "name": "Cinta de correr",
3   "status": "Disponible"
4 }
5
6
```

```
1 {
2   "id": 2,
3   "name": "Cinta de correr",
4   "serialNumber": null,
5   "status": "Disponible"
6 }
```

Reportar mantenimiento:

- POST `http://localhost:9191/api/equipment/1/maintenance?issue=Motor averiado`

The screenshot shows a REST client interface with the URL `http://localhost:9191/api/equipment/1/maintenance?issue=Motor averiado`. The request method is **POST**. The request body is a JSON object: `{ "name": "Cinta de correr", "status": "Disponible" }`. The response status is **200 OK** with a response time of 32 ms and a body size of 240 B. The response body is a JSON object: `{ "id": 1, "name": "Cinta de correr", "serialNumber": null, "status": "MAINTENANCE" }`.

```
1 {
2   "name": "Cinta de correr",
3   "status": "Disponible"
4 }
5
6
```

```
1 {
2   "id": 1,
3   "name": "Cinta de correr",
4   "serialNumber": null,
5   "status": "MAINTENANCE"
6 }
```


Accesos

- Conceder acceso:
- POST `http://localhost:9191/api/access/grant?member=Juan Pérez&door=Puerta Principal`

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:9191/api/access/grant?member=Juan Pérez&door=Puerta Principal`
- Method:** POST
- Body (raw):**

```
1 {
2   "name": "Cinta de correr",
3   "status": "Disponible"
4 }
5
6
```
- Response (JSON):**

```
1 {
2   "id": 2,
3   "memberName": "Juan Pérez",
4   "timestamp": "2025-09-28T14:08:52.6359343",
5   "door": "Puerta Principal"
6 }
```
- Status:** 200 OK, 44 ms, 267 B

Facturación

- Crear factura:
- POST `http://localhost:9191/api/billing`

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:9191/api/billing`
- Method:** POST
- Body (raw):**

```
1 {
2   "memberName": "Juan Pérez",
3   "amount": 500
4 }
5
6
```
- Response (JSON):**

```
1 {
2   "id": 3,
3   "memberName": "Juan Pérez",
4   "dueDate": null,
5   "amount": 500.0,
6   "paid": false
7 }
```
- Status:** 200 OK, 29 ms, 242 B

Pagar factura:

POST http://localhost:9191/api/billing/1/pay

The screenshot shows a REST client interface with the following details:

- URL:** http://localhost:9191/api/billing/1/pay
- Method:** POST
- Body (raw):**

```
1 {  
2   "memberName": "Juan Pérez",  
3   "amount": 500  
4 }  
5  
6
```
- Response:** 200 OK, 45 ms, 241 B. The response body is shown in JSON format:

```
1 {  
2   "id": 1,  
3   "memberName": "Juan Pérez",  
4   "dueDate": null,  
5   "amount": 500.0,  
6   "paid": true  
7 }
```