**Clase 6.1.1**

* Levantar MySQL
* Crear un endpoint de prueba con una entity básica
* Implementar un POST que agregue una row de la entity creada
* Agregar un GET que obtenga todos los registros de la tabla

**Clase 6.1.2**

* Partir de la API de la clase pasada
* Agregar endpoints para hacer CRUD, con la correspondiente operación sobre MySQL

**Clase 6.1.3**

* Sobre los endpoints existentes agregar relaciones ManyToOne
* Tener en cuenta el body de los requests que estamos mandando

**Clase 6.2.1**

1. **Demo en Vivo**

* Hacer con Postman un POST a un endpoint, para así obtener un token
  + Armar una API de prueba
* Usar ese token para pegarle a otro endpoint
* Hacer pruebas con un token diferente

1. **Implementar un login básico 🤪**

* Un endpoint POST que recibe en el body la combinación username/password
* Usando TypeORM, crear una tabla “usuario” que tenga como columnas usuario y contraseña
* La API debe chequear que esa combinación coincida lo que haya en la base de datos
* En caso de que no exista, la API debe devolver un mensaje de error
* En caso de que exista, devolver un texto fijo
  + Este texto fijo lo vamos a usar como *token*

1. **Implementar un endpoint básico**

* La funcionalidad básica: que retorne un JSON con un “hola mundo”
* El requisito es que el “hola mundo” se debe devolver **solamente** si en el body del request le pasamos el token que obtuvimos en el otro endpoint

1. **Lo que vimos fue un manejo muy básico de usuarios**

* Para todos los usuarios estamos devolviendo el mismo token
* Estaría bueno poder manejar diferentes tipos de usuarios → ejemplo: user, admin, etc.
* El contenido del token lo podemos ver directo desde Postman → problemas de seguridad
  + Necesitaríamos que el token esté encriptado
* Existe un mecanismo de tokens muy utilizado actualmente → JSON Web Token (JWT)

**Clase 6.2.2**

1. **Demo en Vivo**

* A partir de la API de prueba
  + Pegarle al POST para obtener un token
  + Acceder al endpoint GET con el token recibido
* Tener cuidado que la API de ejemplo genera tokens que expiran después de cierto tiempo
  + En ese caso, generarlo de vuelta
  + Los tokens siempre tienen que tener un tiempo de expiración por cuestiones de seguridad

1. **Ejercicios**

* Partir de la API armada en la clase pasada, y agregar JWT en función al tutorial de NestJS

<https://docs.nestjs.com/techniques/authentication>

* En la documentación se arma una API teniendo los usuarios hardcodeados en un arreglo interno
* La idea sería que esa información la tomemos desde la base de datos