



Módulo 3: Back-end

Sesión 2: Conexión con base de datos y consumo de servicios de terceros

Equipo de desarrolladores:

Cristian David Ríos MSc
Daniel Escobar Grisales MSc
Nestor Rafael Calvo MSc

Coordinador del proyecto:

Prof. Dr.-Ing. Juan Rafael Orozco Arroyave





Hola!

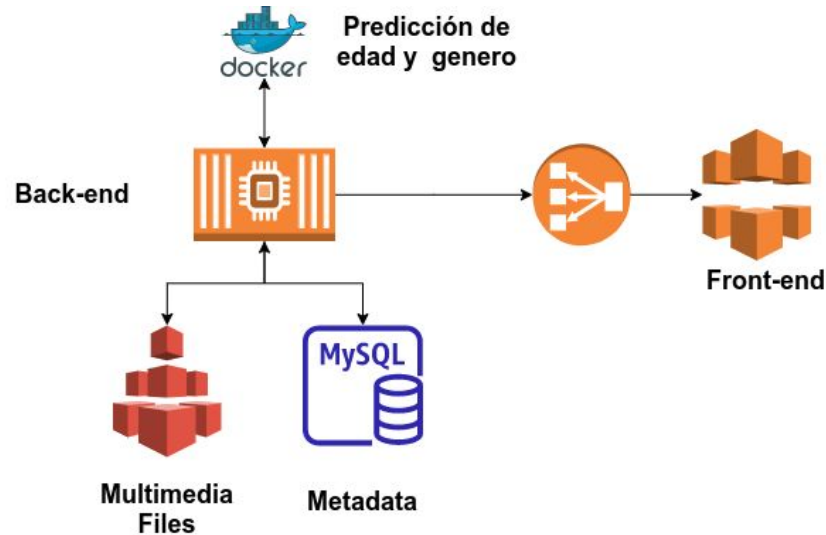
Mi nombre es Daniel Escobar

You can find me at:

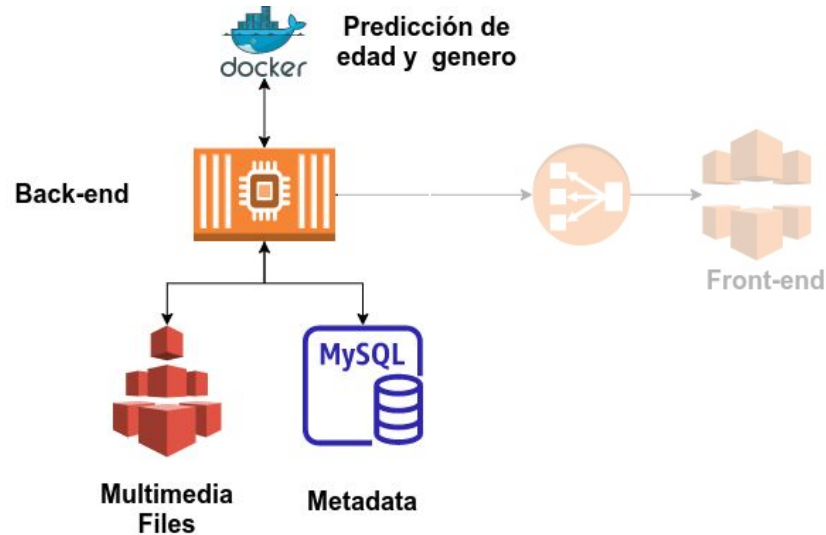
 @dangrisales



Arquitectura y características



Arquitectura y características



Sesión 2: Creación de recursos y consumo de servicios de terceros

Temas

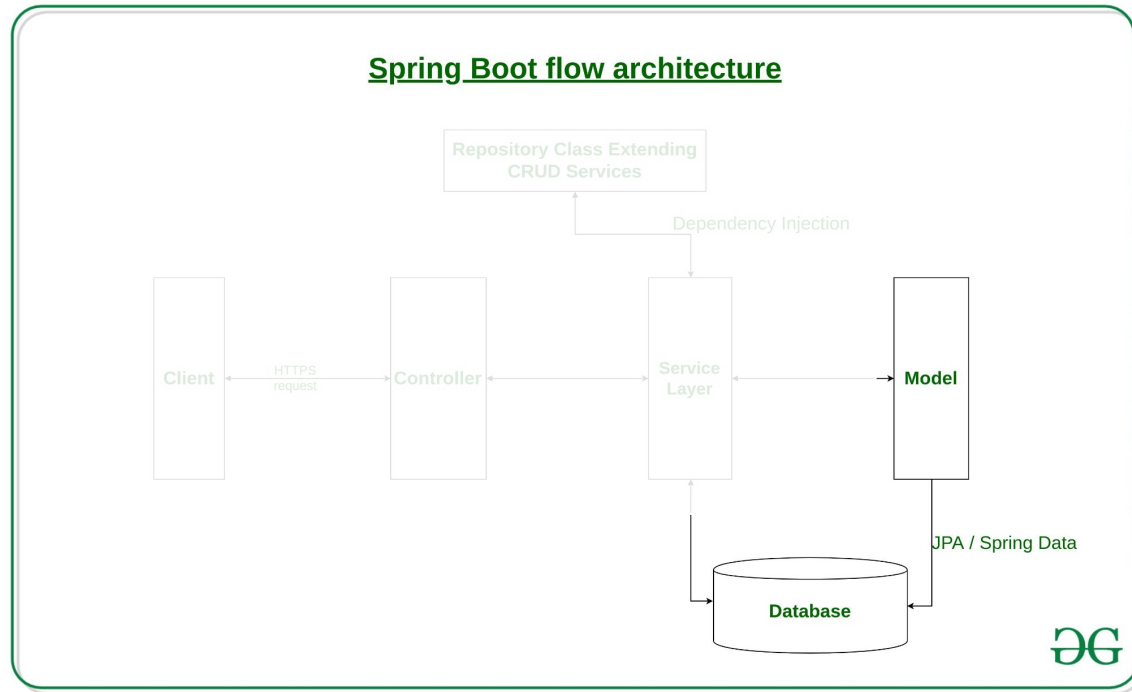
- Conexión con base de datos
 - JPA (Java Persistence API)
 - JPQL (Java Persistence Query Language)
 - Configuración de dependencias
 - Configurar base de datos
- Consumo de servicios terceros
 - Metodología
 - HttpURLConnection
 - Configuración de la petición
 - Creación
 - Agregar parámetros
 - Otras configuraciones

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines, rendered in a light gray color.

1.

Conexión con base de datos

Spring Boot



JPA (Java Persistence API)

Es un framework de Java que ofrece un conjunto de interfaces y APIs para la gestión de datos en bases de datos relacionales.

- ⦿ Permite acceder y persistir los datos entre el objeto/clase Java y la base de datos relacional
 - ⦿ JPA sigue el mapeo objeto-relacional (ORM)
 - ⦿ También proporciona una API de EntityManager en tiempo de ejecución
- Permite usar JPQL (Java Persistence Query Language)

JPQL (Java Persistence Query Language)

```
@Query("SELECT user FROM UserModel user WHERE user.creationDate = :date and user.taskCompleted = 1")  
public List<UserModel> findUserByBioUser(@Param("date") String date);
```

User Model

date	Name	Completed
10/11/2022	Daniel	1
10/11/2022	Juan	1
10/11/2022	Pedro	0
12/11/2022	Damian	1



Resultado

date	Name	Completed
10/11/2022	Daniel	1
10/11/2022	Juan	1

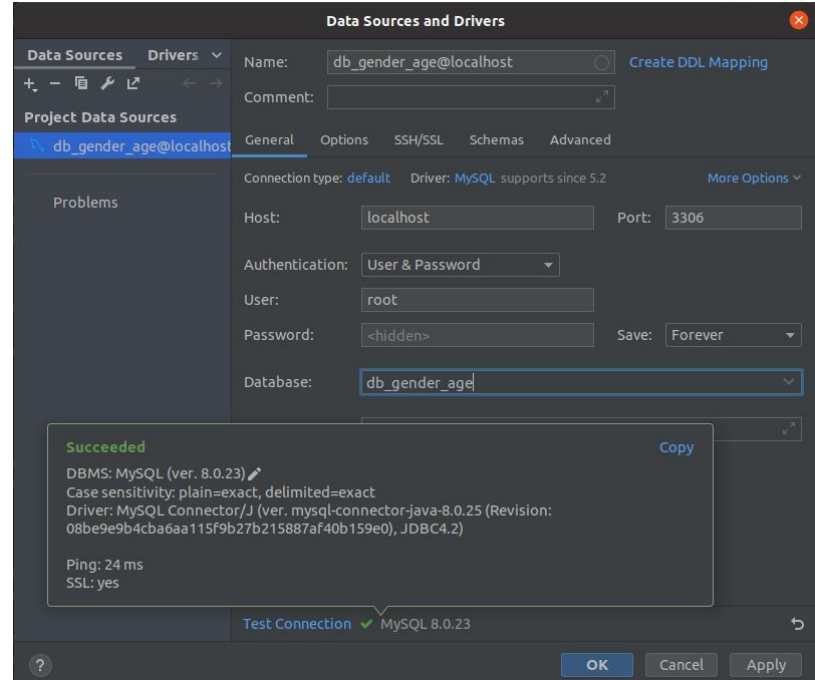
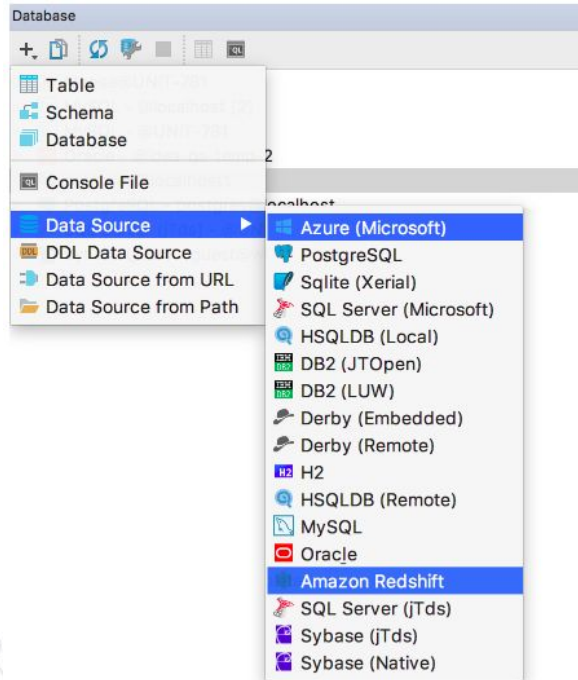
Configuración de dependencias

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

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
</dependency>
```

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```

Configurar base de datos (IntelliJ)



Model y EntityManager

```
@Entity
@Table(name = "user", schema = "db_gender_age")
public class UserVideo {

    @Id
    @Column(name = "id_number", nullable = false)
    private String idNumber;
    2 usages

    @Column(name = "type_id", nullable = false)
    private String typeId = "CC";
    2 usages

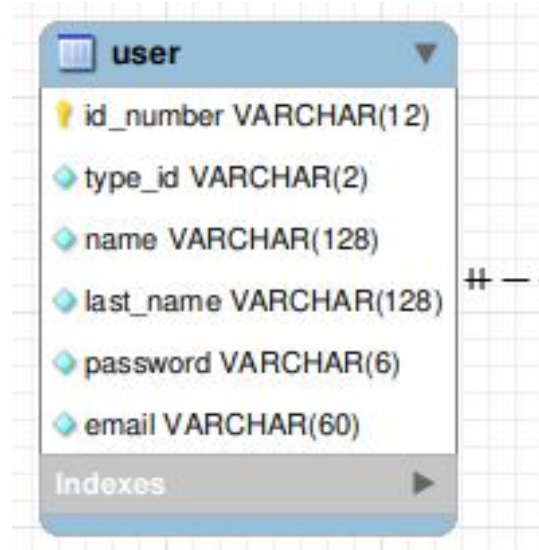
    @Column(name = "name", nullable = false)
    private String name;
    2 usages

    @Column(name = "last_name", nullable = false)
    private String lastName;
    2 usages

    @Column(name = "password", nullable = false)
    private String password;
    2 usages

    @Column(name = "email", nullable = false)
    private String email;
    no usages

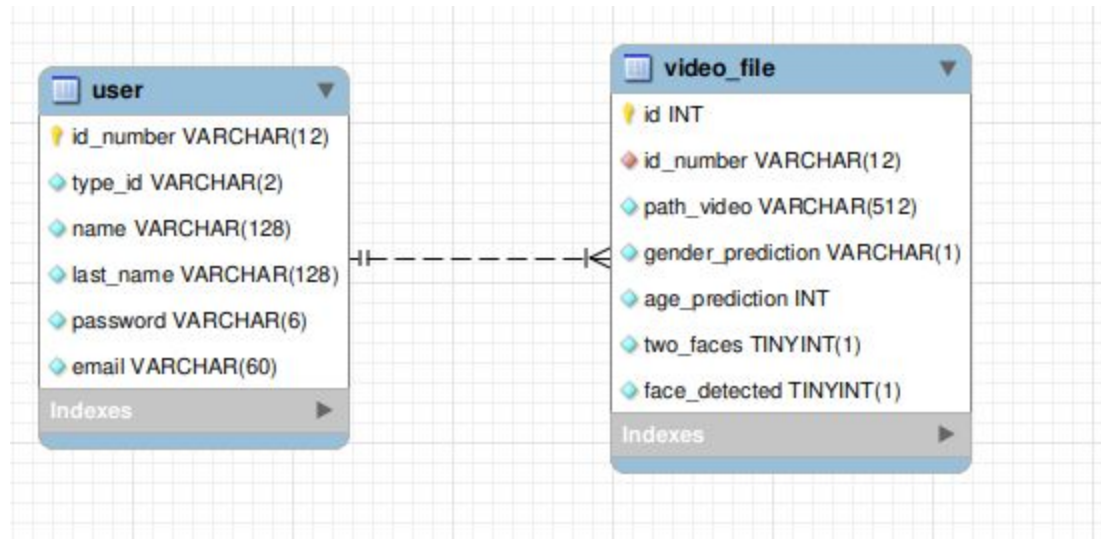
    @OneToMany(mappedBy = "userVideo", fetch = FetchType.LAZY)
    private List<VideoFile> videoFileList;
```



The image shows a screenshot of a database table definition for a table named 'user'. The table has the following columns: 'id_number' (VARCHAR(12)), 'type_id' (VARCHAR(2)), 'name' (VARCHAR(128)), 'last_name' (VARCHAR(128)), 'password' (VARCHAR(6)), and 'email' (VARCHAR(60)). The 'id_number' column is marked as the primary key with a yellow lightning bolt icon. Below the columns, there is a section for 'Indexes' with a right-pointing arrow.

user	
id_number	VARCHAR(12)
type_id	VARCHAR(2)
name	VARCHAR(128)
last_name	VARCHAR(128)
password	VARCHAR(6)
email	VARCHAR(60)
Indexes	

Relaciones uno a muchos



Relaciones uno a muchos

User

```
@OneToMany(mappedBy = "userVideo", fetch = FetchType.LAZY)
private List<VideoFile> videoFileList;
```

Video File

```
@ManyToOne
@Fetch(FetchMode.JOIN)
@JoinColumn(name = "id_number", referencedColumnName = "id_number", nullable = false)
private UserVideo userVideo;
```

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines, with some nodes highlighted in blue.

2.

Manos a la obra

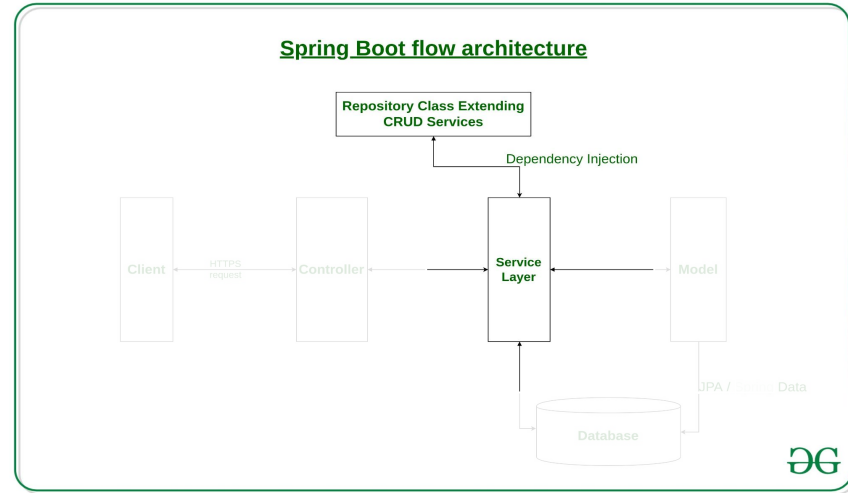
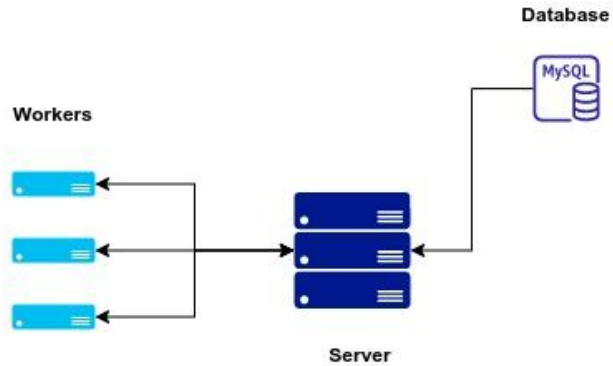
Conexión con la base de datos creada en el módulo 2

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines, with some nodes highlighted in blue and others in grey.

3.

Consumo de servicios de terceros

Metodología



URLConnection class

Permite hacer peticiones básicas HTTP sin usar ninguna librería adicional. Todas las clases que necesitaremos son parte de el paquete java.net

```
<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpclient</artifactId>
  <version>4.5.10</version>
</dependency>
```

```
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.6</version>
</dependency>
```

Creación de una petición

Para una URL dada es necesario configurar un atributo “*requestMethod*” que permite alguno de los siguientes verbos.

```
URL urlExample = new URL( spec: "http://example_url.com");  
URLConnection con = (URLConnection) url.openConnection();  
con.setRequestMethod("GET");
```

Agregar parámetros a la petición

Crear lista de parámetros con *parameter Map*

```
Map<String, String> parameters = new HashMap<>();  
parameters.put("image", path);  
parameters.put("id", token);  
respuesta = requestMethodsBackService.peticionHttpPOST(url, parameters);
```

Crear string en el formato requerido

```
StringBuilder result = new StringBuilder();  
  
for (Map.Entry<String, String> entry : params.entrySet()) {  
    result.append(URLEncoder.encode(entry.getKey(), "UTF-8"));  
    result.append("=");  
    result.append(URLEncoder.encode(entry.getValue(), "UTF-8"));  
    result.append("&");  
}  
return result.toString();
```

Agregar a la petición

```
StringBuilder resultado = new StringBuilder();  
URL url = new URL(destinyURL);  
String postData = convertStringHTTPformat(params);  
  
URLConnection conn = (URLConnection) url.openConnection();  
conn.setRequestMethod("POST");  
conn.setDoOutput(true);  
conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");  
conn.setRequestProperty("Content-Length", Integer.toString(postData.length()));
```

Otras configuraciones de la petición

Agregar configuración a la cabecera de la petición

```
StringBuilder resultado = new StringBuilder();
URL url = new URL(destinyURL);

URLConnection conexion = (URLConnection) url.openConnection();
conexion.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
conexion.setRequestProperty("OAuth-Token", "token de seguridad");
conexion.setRequestProperty("Tipo-Conexion", "Desarrollo");
conexion.setRequestMethod("GET");
```

Obtener respuesta del servicio externo

```
System.out.println("Response Code : " + conexion.getResponseCode());
System.out.println("Response Message : " + conexion.getResponseMessage());

BufferedReader rd = new BufferedReader(new InputStreamReader(conexion.getInputStream()));
String linea;

while ((linea = rd.readLine()) != null) {
    resultado.append(linea);
}

rd.close();

return resultado.toString();
```

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines, with some nodes highlighted in blue.

4.

Manos a la obra

Generar peticiones a los recursos
creados en el módulo 1

Presentación del equipo

Cristian Rios

Daniel Escobar

Nestor Calvo



Gracias!

¿Alguna pregunta o comentario?

Puedes escribirme a:

daniel.esobar@udea.edu.co