

## Apuntes Springboot

Como tenemos thymeleaf, las páginas se guardan en resources/templates.

A las vistas NO SE PUEDEN ACCEDER DIRECTAMENTE, tenemos que llegar a ellas a través de un controlador que tenga un `RequestMapping` para la ruta que queramos y nos devuelva al archivo `html`.

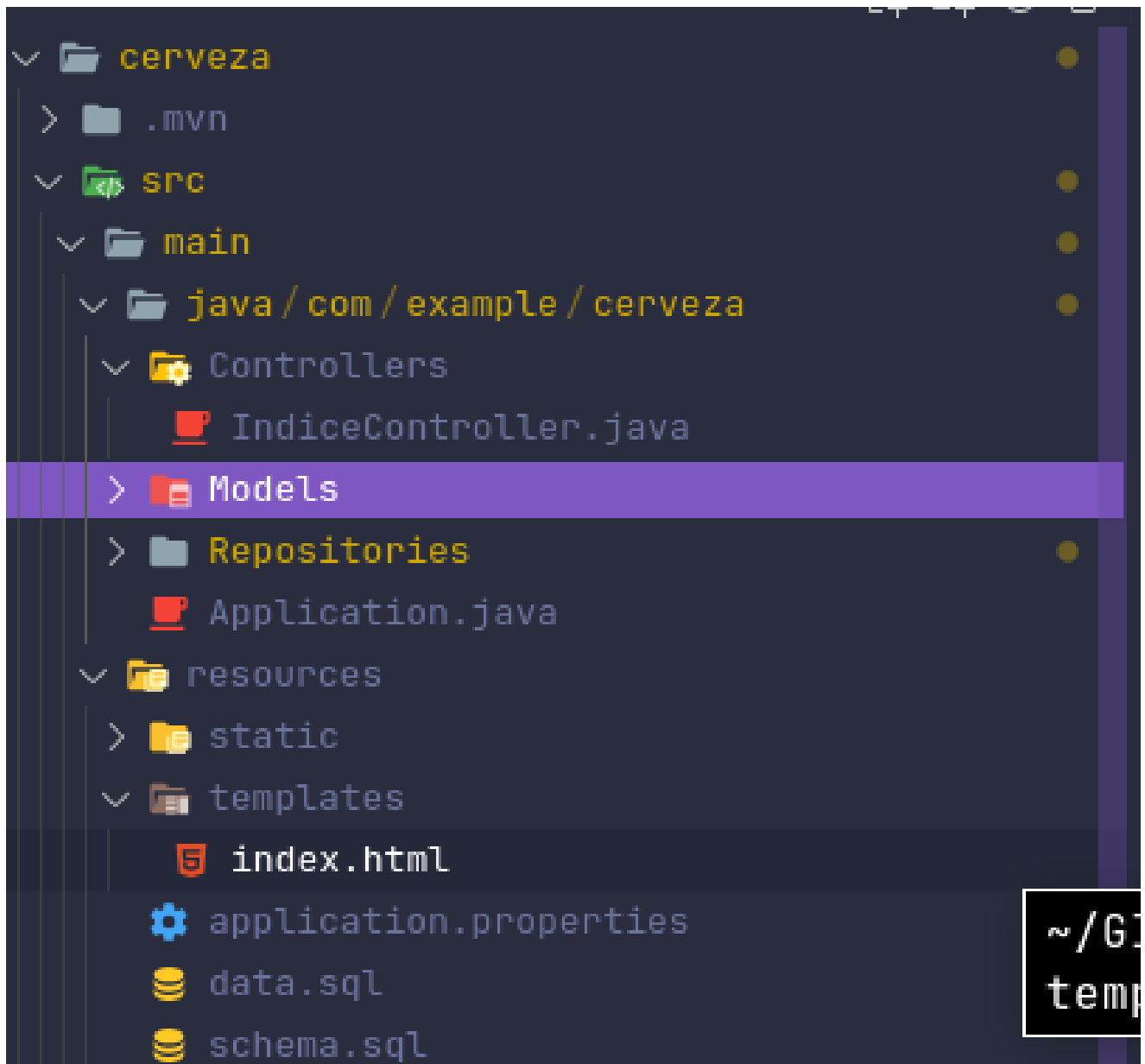
Cada controlador tiene asociado un modelo.

ORDEN DE HACER LAS COSAS:

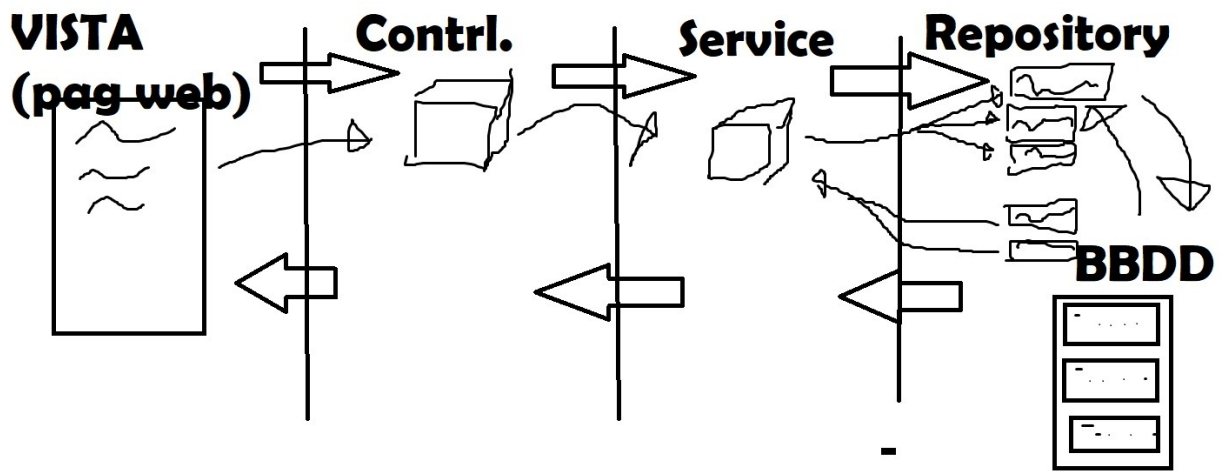
MODELO → Repositorio(`RowMapper`) → Repositorio(Repositorio con los métodos de consulta a la base) → Controlador → VISTAS

podemos usar `lombok` para `getter` y `setter` (poniendo `@Getter` y `@Setter` e importando los metodos: `import lombok.getter` y `lombok.setter`)

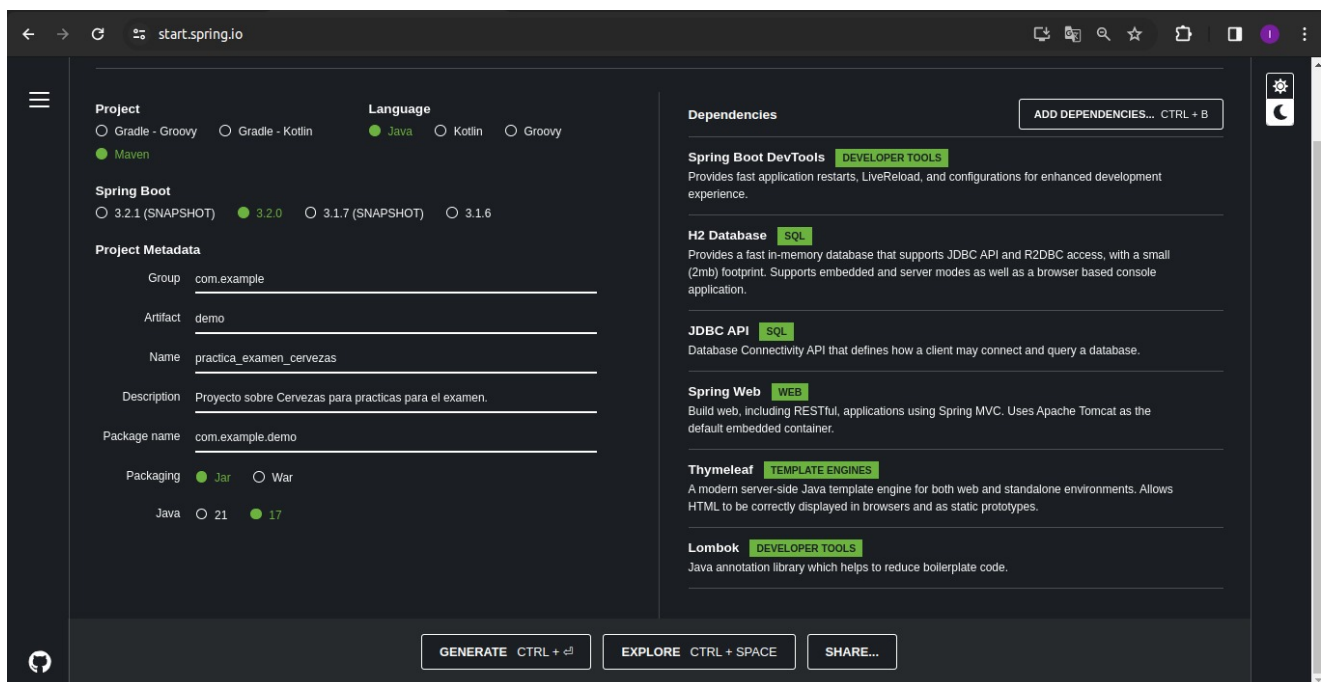
QUE SE HACE EN CADA SITIO:



- Schema.sql: dentro se pone las tablas de la base de datos
- Data.sql: dentro se ponen inserciones de datos a la base de datos.
- Application.properties: dentro se pone todo el tema de h2, etc.
- Templates: dentro van las vistas.html
- Models: dentro se crean las clases/objetos que representan las tablas de la base
- Repositories: dentro de estos se crean 2 tipos de repo:
  - 1 - RowMappers: sirve para que al conectarnos a la base se generen objetos con la info
  - 2- Repositorios normales: sirven para crear métodos de inserción, listado, borrado a la base usando los rowmapper (Consultas SQL)
- Controllers: se conectan con la vista para mostrar datos y los repositorios para obtener/consultar/insertar datos en la base.



## Dependencias:



## Spring Boot DevTools:

Permite visualizar la base de datos y hacer testing de las clases.

Proporciona herramientas como la reinicialización automática de la aplicación al detectar cambios en el código fuente, entre otras características, para agilizar el proceso de desarrollo.

## H2 Database:

- Propósito: Base de datos en memoria para desarrollo y pruebas.
- Funcionalidad: H2 es una base de datos relacional ligera y rápida que se puede integrar fácilmente en aplicaciones Spring Boot, se puede ejecutar en memoria.

## JDBC API:

Permite trabajar con conectores a base de datos de java.

- Propósito: Proporciona una interfaz estándar para la conexión y ejecución de consultas en bases de datos.
- Funcionalidad: JDBC (Java Database Connectivity) es una API que permite a las aplicaciones Java interactuar con bases de datos relacionales. Spring Boot utiliza JDBC para facilitar el acceso a bases de datos y simplificar las operaciones de base de datos.

### Spring Web:

- Propósito: Facilita el desarrollo de aplicaciones web.
- Funcionalidad: Proporciona características para el desarrollo de aplicaciones web, como el manejo de solicitudes y respuestas HTTP, la configuración de controladores, la gestión de sesiones y más. Spring Web facilita la construcción de aplicaciones web robustas y escalables.

### Thymeleaf:

Es como se va a presentar la vista de la aplicación.

- Propósito: Motor de plantillas para la creación de vistas en aplicaciones web.
- Funcionalidad: Thymeleaf hace lo mismo que primefaces, crear vistas de la lógica. Facilita la creación de páginas HTML dinámicas y la presentación de datos desde el servidor.

### Lombok:

- Propósito: Reduce la verbosidad del código Java.
- Funcionalidad: funciona como el create code de IDE's. Utilizando anotaciones, Lombok genera automáticamente este código durante la compilación, lo que hace que el código sea más limpio y conciso. La anotación @Data genera automáticamente los métodos toString, equals, hashCode, y los getters y setters para todos los campos de la clase.

### Opciones del proyecto:

Project

☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ Java ☐ Kotlin ☐ Groovy

☒ Maven

Spring Boot

☐ 3.2.1 (SNAPSHOT) ☒ 3.2.0 ☐ 3.1.7 (SNAPSHOT) ☐ 3.1.6

Project Metadata

Group

com.example

Artifact

cerveza

Name

ismaelbernad

Description

Proyecto de springboot desde 0 para practicar el examen

Package name

com.example.cerveza

Packaging

☒ Jar ☐ War

Java

☐ 21 ☒ 17

### Group (Grupo):

- Propósito: Define el identificador del grupo de tu aplicación.
- Ejemplo: Si estás desarrollando una aplicación para tu empresa llamada "MyCompany", el grupo podría ser algo como com.mycompany.

### Artifact (Artefacto):

- Propósito: Define el identificador único para tu aplicación dentro del grupo.
- Ejemplo: Si tu aplicación se llama "demo", el artefacto podría ser simplemente demo. El artefacto se utiliza para construir el identificador único de tu aplicación, que es a menudo utilizado en la creación de paquetes JAR o WAR.

### Name (Nombre):

- Propósito: Es el nombre completo de tu proyecto.
- Ejemplo: Puedes poner el nombre que desees para tu proyecto. En tu caso, mencionaste "ismaelbernad".

### Description (Descripción):

- Propósito: Proporciona una descripción breve de tu proyecto.

### Package name (Nombre del paquete):

- Propósito: Define la estructura del paquete para tus clases.
- Ejemplo: Utilizando la convención de nomenclatura de paquetes de Java, este campo se utiliza para especificar cómo se estructurarán tus paquetes. En tu caso, com.example.demo sigue la convención común donde com es el dominio de Internet invertido, example es el nombre del grupo de tu aplicación, y demo es el nombre del artefacto.

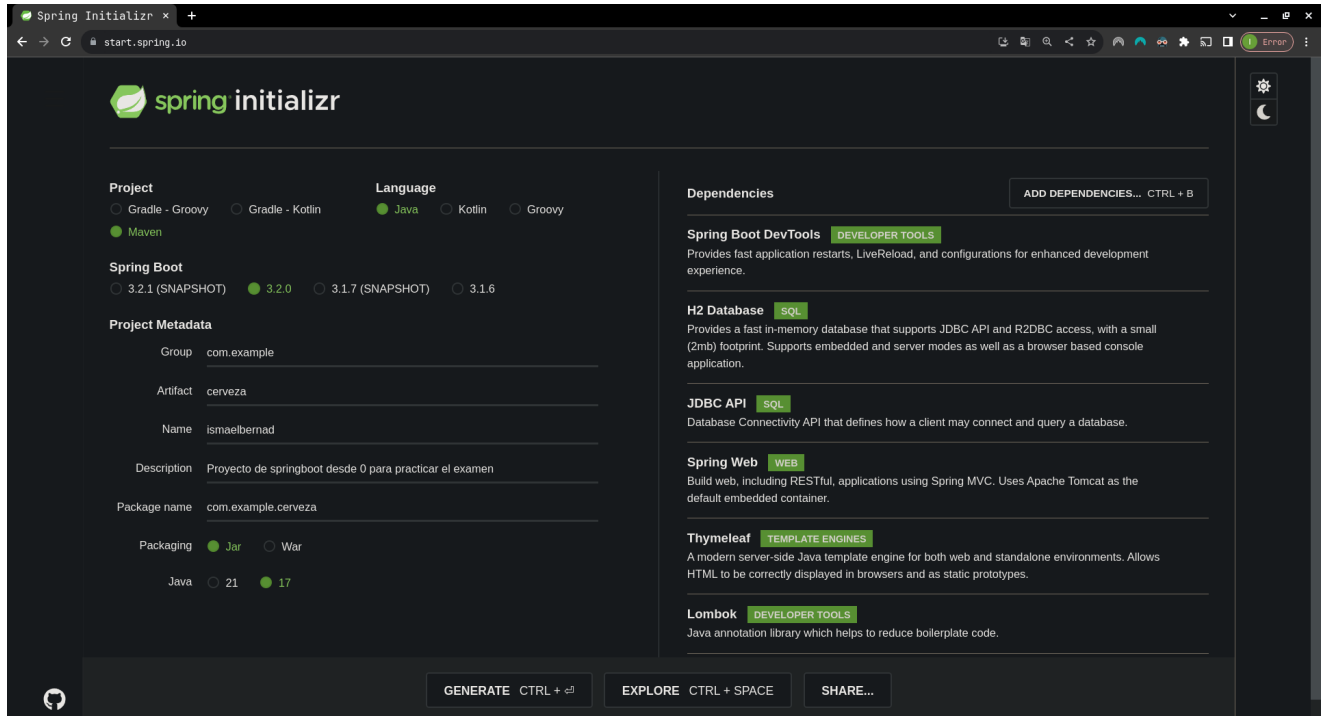
### Packaging (Empaquetado):

- Propósito: Define el tipo de empaquetado que se utilizará para la aplicación (por ejemplo, JAR o WAR).
- Ejemplo: El empaquetado JAR es común para aplicaciones independientes, mientras que el empaquetado WAR se utiliza a menudo para aplicaciones web.

## Crear proyecto de Springboot

Vamos a: <https://start.spring.io/>

Y ponemos las dependencias:



The screenshot shows the Spring Initializr web application interface. The browser address bar displays 'start.spring.io'. The page has a dark theme. On the left, under 'Project', 'Maven' is selected. Under 'Language', 'Java' is selected. Under 'Spring Boot', '3.2.0' is selected. The 'Project Metadata' section contains the following fields: Group (com.example), Artifact (cerveza), Name (ismaelbernad), Description (Proyecto de springboot desde 0 para practicar el examen), Package name (com.example.cerveza), Packaging (Jar), and Java version (17). On the right, the 'Dependencies' section lists several dependencies: Spring Boot DevTools, H2 Database, JDBC API, Spring Web, Thymeleaf, and Lombok. At the bottom, there are three buttons: 'GENERATE', 'EXPLORE', and 'SHARE...'. The browser's developer console shows an 'Error' message.

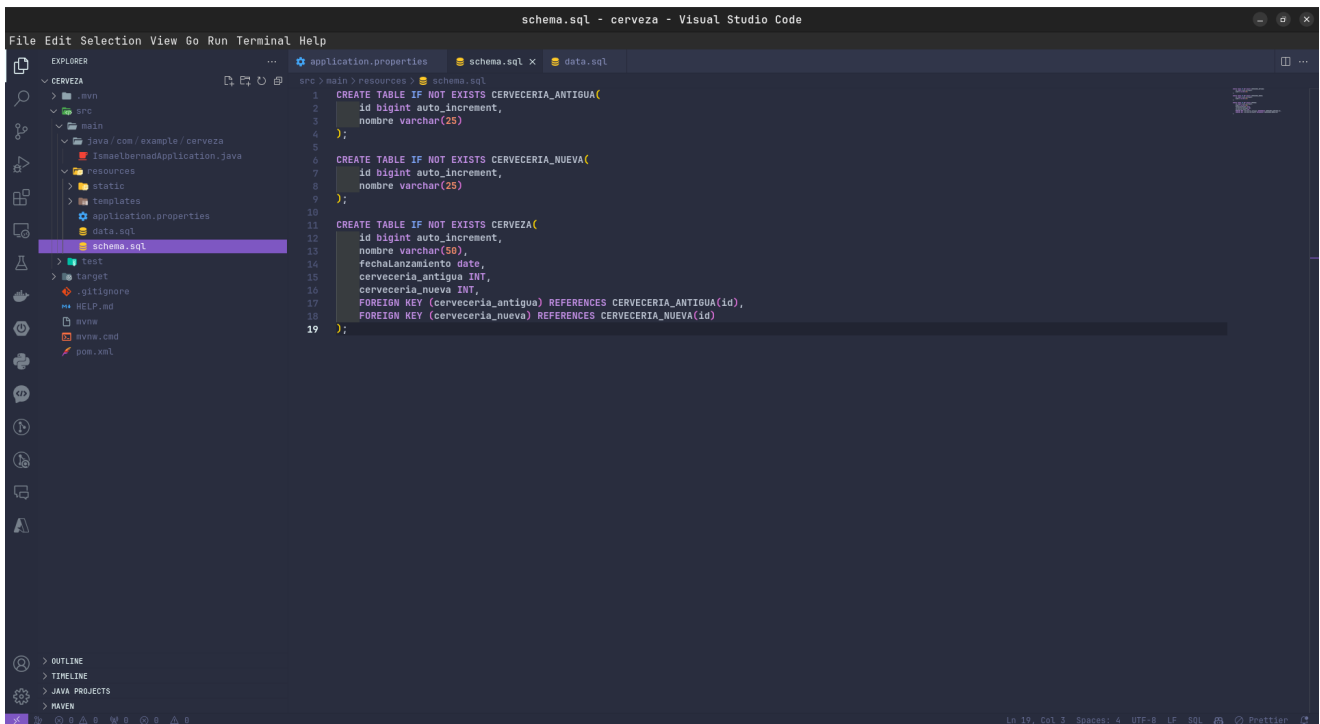
Cambiamos el contenido del application.properties para especificar la creación de la base de datos H2:



The screenshot shows the Visual Studio Code editor with the 'application.properties' file open. The file contains configuration for a Spring Boot application named 'cerveza'. The Explorer sidebar on the left shows the project structure, including 'src/main/resources' and 'application.properties'. The main editor area displays the following properties:

```
src > main > resources > application.properties
1  # Configuración de la url del origen de datos para la base de datos H2 en memoria
2  spring.datasource.url=jdbc:h2:mem:testdb
3  # Si quisieramos cambiar el nombre de la base de datos, lo haríamos de la siguiente manera:
4  #spring.datasource.url=jdbc:h2:mem:nombreBaseDatos
5
6  # Si quisieramos que la base de datos se guardara en un fichero, lo haríamos de la siguiente manera:
7  #spring.datasource.url=jdbc:h2:file:./data/demo
8
9  # Si quisieramos cambiar el puerto de la base de datos, lo haríamos de la siguiente manera:
10 #spring.datasource.url=jdbc:h2:tcp://localhost:9092/mem:nombreBaseDatos
11
12
13 # Especifica la clase del controlador JDBC para H2
14 spring.datasource.driverClassName=org.h2.Driver
15
16
17 # El usuario por defecto de H2 es 'sa'
18 # y la contraseña se deja vacia para facilitar las correcciones a Gorka
19 spring.datasource.username=sa
20 spring.datasource.password=
21
22 # Configuración de la consola web de H2
23 # Activa la consola web de H2
24 spring.h2.console.enabled=true
25 # Especifica la ruta de acceso a la consola web de H2
26 spring.h2.console.path=/h2-console
```

Creamos el esquema de la base de datos con el fichero schema.sql:

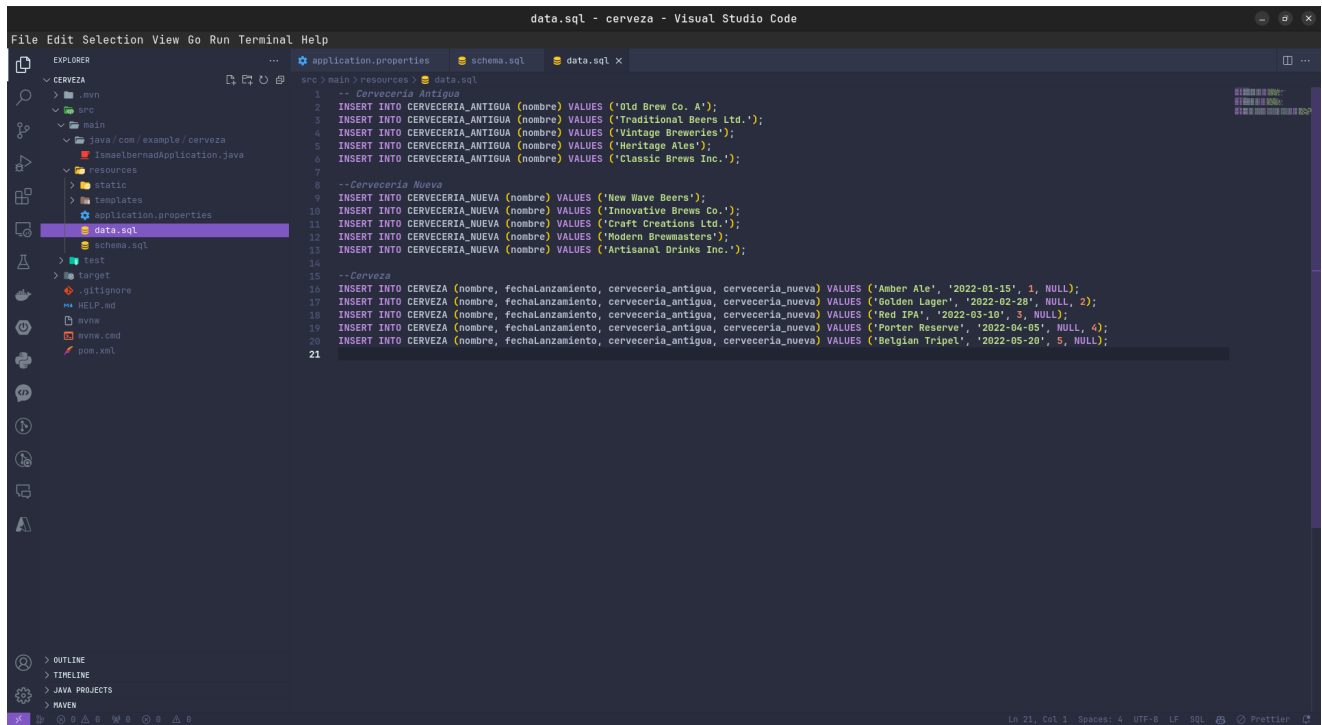


The screenshot shows the Visual Studio Code editor with the 'schema.sql' file open. The file contains SQL statements to create three tables: 'CERVECERIA\_ANTIQUA', 'CERVECERIA\_NUEVA', and 'CERVEZA'. The Explorer sidebar on the left shows the project structure, including 'src/main/resources' and 'schema.sql'. The main editor area displays the following SQL statements:

```
src > main > resources > schema.sql
1  CREATE TABLE IF NOT EXISTS CERVECERIA_ANTIQUA(
2  id bigint auto-increment,
3  nombre varchar(25)
4  );
5
6  CREATE TABLE IF NOT EXISTS CERVECERIA_NUEVA(
7  id bigint auto-increment,
8  nombre varchar(25)
9  );
10
11  CREATE TABLE IF NOT EXISTS CERVEZA(
12  id bigint auto-increment,
13  nombre varchar(50),
14  fechaLanzamiento date,
15  cervezaAntigua INT,
16  cervezaNueva INT,
17  FOREIGN KEY (cerveceria_antigua) REFERENCES CERVECERIA_ANTIQUA(id),
18  FOREIGN KEY (cerveceria_nueva) REFERENCES CERVECERIA_NUEVA(id)
19  );
```

Y añadimos los datos que queremos que se inserten cada vez que se abra la aplicación, ya que la base de datos H2 SOLO ESTÁ EN MEMORIA. SE BORRA

## CADA VEZ QUE SE CIERRA LA APLICACIÓN:

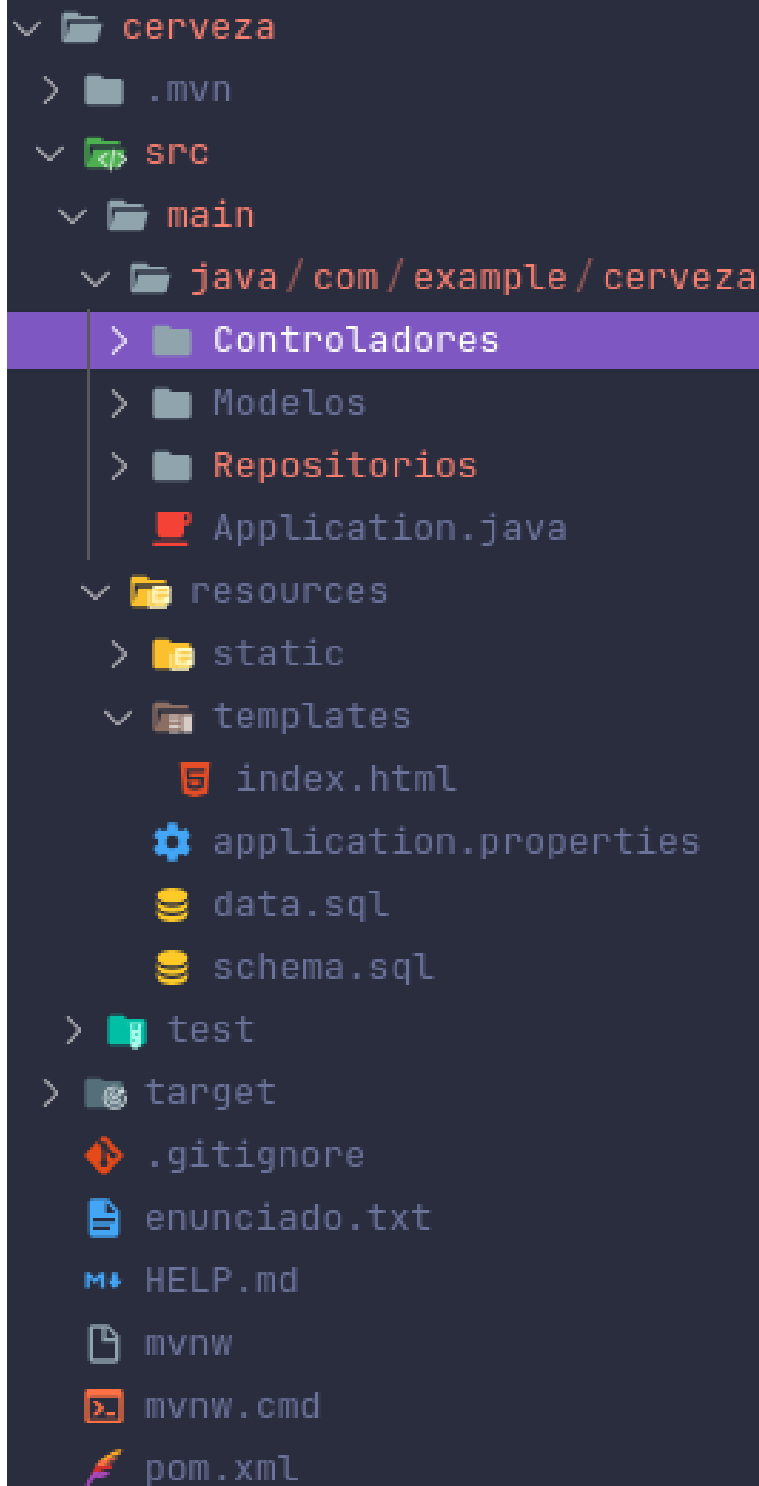


The screenshot shows the Visual Studio Code interface with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like 'resources', 'static', 'templates', and 'application.properties'. The code editor displays a file named 'data.sql' containing SQL insert statements for a beer database. The statements are organized into sections: 'Cerveceria Antigua', 'Cerveceria Nueva', and 'Cerveza'.

```
1 -- Cerveceria Antigua
2 INSERT INTO CERVEceria_ANTIGUA (nombre) VALUES ('Old Brew Co. A');
3 INSERT INTO CERVEceria_ANTIGUA (nombre) VALUES ('Traditional Beers Ltd. ');
4 INSERT INTO CERVEceria_ANTIGUA (nombre) VALUES ('Vintage Breweries ');
5 INSERT INTO CERVEceria_ANTIGUA (nombre) VALUES ('Heritage Ales ');
6 INSERT INTO CERVEceria_ANTIGUA (nombre) VALUES ('Classic Brews Inc. ');
7
8 --Cerveceria Nueva
9 INSERT INTO CERVEceria_NUEVA (nombre) VALUES ('New Wave Beers ');
10 INSERT INTO CERVEceria_NUEVA (nombre) VALUES ('Innovative Brews Co. ');
11 INSERT INTO CERVEceria_NUEVA (nombre) VALUES ('Craft Creations Ltd. ');
12 INSERT INTO CERVEceria_NUEVA (nombre) VALUES ('Modern Brewmasters ');
13 INSERT INTO CERVEceria_NUEVA (nombre) VALUES ('Artisanal Drinks Inc. ');
14
15 --Cerveza
16 INSERT INTO CERVEZA (nombre, fechaLanzamiento, cerveceria_antigua, cerveceria_nueva) VALUES ('Amber Ale', '2022-01-15', 1, NULL);
17 INSERT INTO CERVEZA (nombre, fechaLanzamiento, cerveceria_antigua, cerveceria_nueva) VALUES ('Golden Lager', '2022-02-28', NULL, 2);
18 INSERT INTO CERVEZA (nombre, fechaLanzamiento, cerveceria_antigua, cerveceria_nueva) VALUES ('Red IPA', '2022-03-10', 3, NULL);
19 INSERT INTO CERVEZA (nombre, fechaLanzamiento, cerveceria_antigua, cerveceria_nueva) VALUES ('Porter Reserve', '2022-04-05', NULL, 4);
20 INSERT INTO CERVEZA (nombre, fechaLanzamiento, cerveceria_antigua, cerveceria_nueva) VALUES ('Belgian Tripel', '2022-05-20', 5, NULL);
21
```

Creamos las carpetas Controladores, Modelos y Repositorios en  
java/com/example/examen:





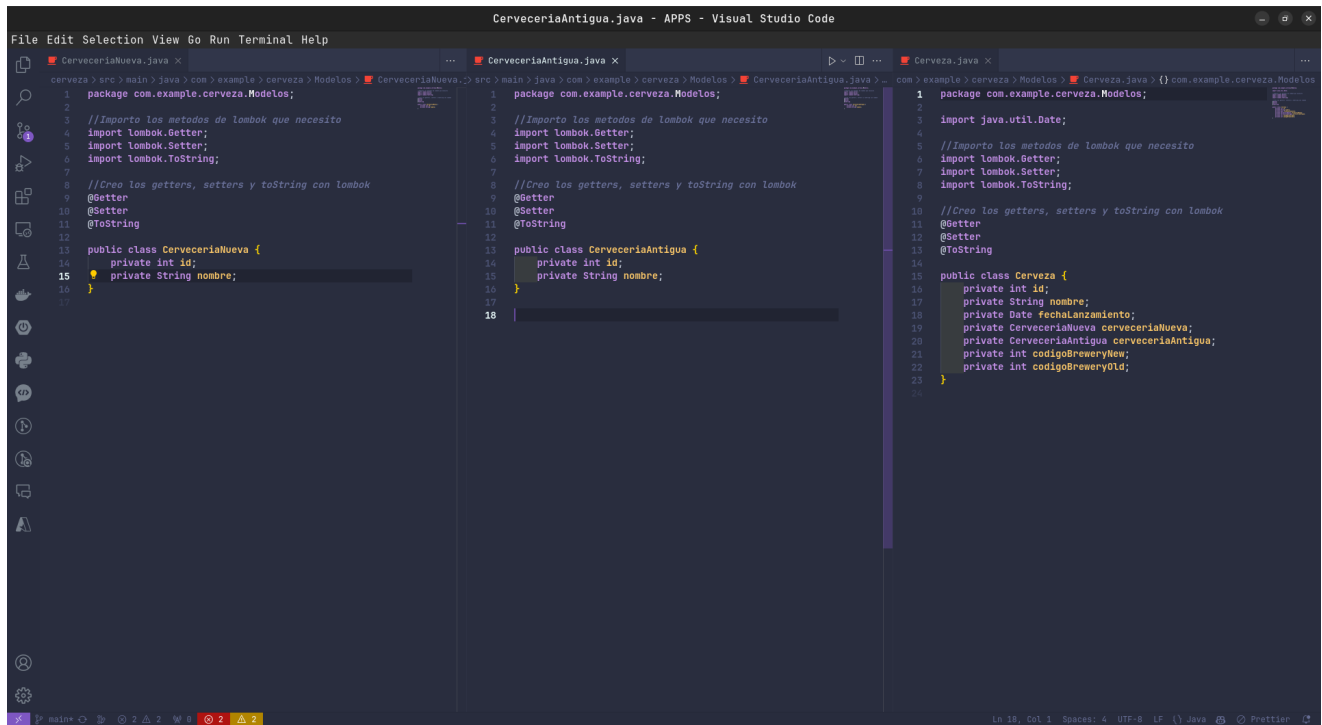
The image shows a file explorer in an IDE with a dark theme. The project structure is as follows:

- cerveza
  - .mvn
  - src
    - main
      - java/com/example/cerveza
        - Controladores** (highlighted)
        - Modelos
        - Repositorios
        - Application.java
    - resources
      - static
      - templates
        - index.html
        - application.properties
        - data.sql
        - schema.sql
    - test
  - target
  - .gitignore
  - enunciado.txt
  - HELP.md
  - mvnw
  - mvnw.cmd
  - pom.xml

Creamos los modelos:

En los modelos ponemos los mismos atributos que tiene la base de datos en la

tabla.



```
File Edit Selection View Go Run Terminal Help

CerveceríaNueva.java X
1 package com.example.cerveza.Modelos;
2
3 //Importo los metodos de Lombok que necesito
4 import lombok.Setter;
5 import lombok.ToString;
6
7 //Crea los getters, setters y toString con Lombok
8 @Setter
9 @ToString
10
11 public class CerveceríaNueva {
12     private int id;
13     private String nombre;
14 }

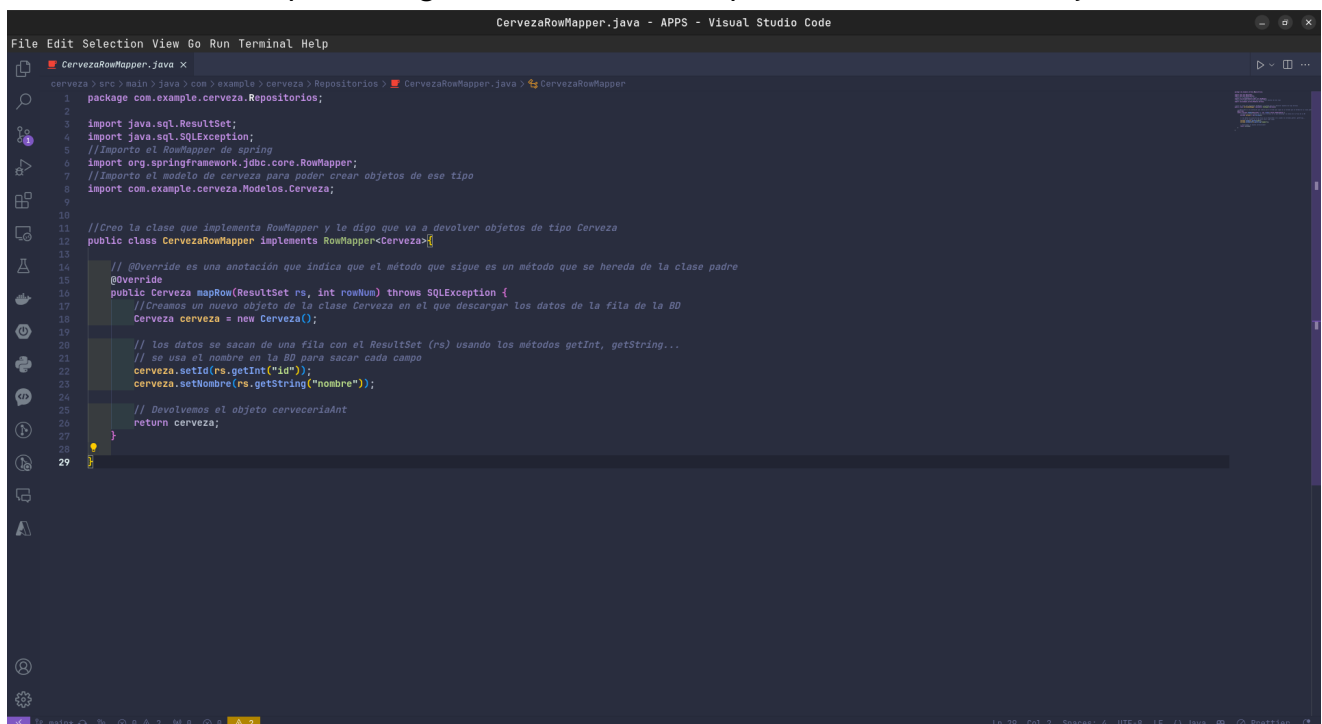
CerveceríaAntigua.java X
1 package com.example.cerveza.Modelos;
2
3 //Importo los metodos de Lombok que necesito
4 import lombok.Getter;
5 import lombok.Setter;
6 import lombok.ToString;
7
8 //Crea los getters, setters y toString con Lombok
9 @Getter
10 @Setter
11 @ToString
12
13 public class CerveceríaAntigua {
14     private int id;
15     private String nombre;
16 }

Cerveza.java X
1 package com.example.cerveza.Modelos;
2
3 import java.util.Date;
4
5 //Importo los metodos de Lombok que necesito
6 import lombok.Getter;
7 import lombok.Setter;
8 import lombok.ToString;
9
10 //Crea los getters, setters y toString con Lombok
11 @Getter
12 @Setter
13 @ToString
14
15 public class Cerveza {
16     private int id;
17     private String nombre;
18     private Date fechaLanzamiento;
19     private CerveceríaNueva cerveceríaNueva;
20     private CerveceríaAntigua cerveceríaAntigua;
21     private int códigoBreweryNew;
22     private int códigoBreweryOld;
23 }
```

Creamos en el repositorio los RowMapper para cada objeto/tabla que tenemos:

(Los RowMapper son métodos que reciben los resultados de una query y los convierte en objetos)

Los necesitamos para luego usarlos en los repositorios de cada objeto.



```
File Edit Selection View Go Run Terminal Help

CervezaRowMapper.java X
1 package com.example.cerveza.Repositorios;
2
3 import java.sql.ResultSet;
4 import java.sql.SQLException;
5 //Importo el RowMapper de Spring
6 import org.springframework.jdbc.core.RowMapper;
7 //Importo el modelo de cerveza para poder crear objetos de ese tipo
8 import com.example.cerveza.Modelos.Cerveza;
9
10 //Crea la clase que implementa RowMapper y le digo que va a devolver objetos de tipo Cerveza
11 public class CervezaRowMapper implements RowMapper<Cerveza> {
12
13     // @Override es una anotación que indica que el método que sigue es un método que se hereda de la clase padre
14     @Override
15     public Cerveza mapRow(ResultSet rs, int rowNum) throws SQLException {
16         //Creamos un nuevo objeto de la clase Cerveza en el que descargar los datos de la fila de la BD
17         Cerveza cerveza = new Cerveza();
18
19         // Los datos se sacan de una fila con el ResultSet (rs) usando los métodos getInt, getString...
20         // se usa el nombre en la BD para sacar cada campo
21         cerveza.setId(rs.getInt("id"));
22         cerveza.setNombre(rs.getString("nombre"));
23
24         // Devolvemos el objeto cerveceríaAnt
25         return cerveza;
26     }
27 }

28
29
```

```
CerveceriaNuevaRowMapper.java - APPS - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER
  APPS
    cerveza
      .mvn
      src
        main
          java/com/example/cerveza
            Controladores
            Modelos
              CerveceriaAntigua.java
              CerveceriaNueva.java
              Cerveza.java
            Repositorios
              CerveceriaAntiguaRowMapper.java
              CerveceriaNuevaRowMapper.java
              CervezaRowMapper.java
              NuevaCerveceriaAntiguaRepositorio.java
              NuevaCerveceriaNuevaRepositorio.java
              NuevaCervezaRepositorio.java
            Application.java
            recursos
              static
              templates
              application.properties
              data.sql
              schema.sql
            test
            target
            .gitignore
            enunciado.txt
            HELP.md
            mvnw
            mvnw.cmd
            pom.xml
            demo_springboot
            EJEMPLO_EXAMEN Abed y gorka
            prueba2
              cerveza.zip
              EJEMPLO_EXAMEN.zip
            OUTLINE
            TIMELINE
            JAVA PROJECTS
            MAVEN

CerveceriaNuevaRowMapper.java x
cerveza > src > main > java > com > example > cerveza > Repositorios > CerveceriaNuevaRowMapper.java > CerveceriaNuevaRowMapper > mapRow(ResultSet, int)
1 package com.example.cerveza.Repositorios;
2
3 import java.sql.ResultSet;
4 import java.sql.SQLException;
5 //Importo el RowMapper de spring
6 import org.springframework.jdbc.core.RowMapper;
7 //Importo el modelo de cerveceria nueva para poder crear objetos de ese tipo
8 import com.example.cerveza.Modelos.CerveceriaNueva;
9
10 //Crea la clase que implementa RowMapper y le digo que va a devolver objetos de tipo CerveceriaAntigua
11 public class CerveceriaNuevaRowMapper implements RowMapper<CerveceriaNueva>{
12
13     // @Override es una anotación que indica que el método que sigue es un método que se hereda de la clase padre
14     @Override
15     public CerveceriaNueva mapRow(ResultSet rs, int rowNum) throws SQLException {
16         //Creamos un nuevo objeto de la clase CerveceriaAntigua en el que descargar los datos de la fila de la BD
17         CerveceriaNueva cerveceriaN = new CerveceriaNueva();
18
19         // los datos se sacan de una fila con el ResultSet (rs) usando los métodos getInt, getString...
20         // se usa el nombre en la BD para sacar cada campo
21         cerveceriaN.setId(rs.getInt("id"));
22         cerveceriaN.setNombre(rs.getString("nombre"));
23
24         // Devolvemos el objeto cerveceriaAnt
25         return cerveceriaN;
26     }
27
28 }
29
```

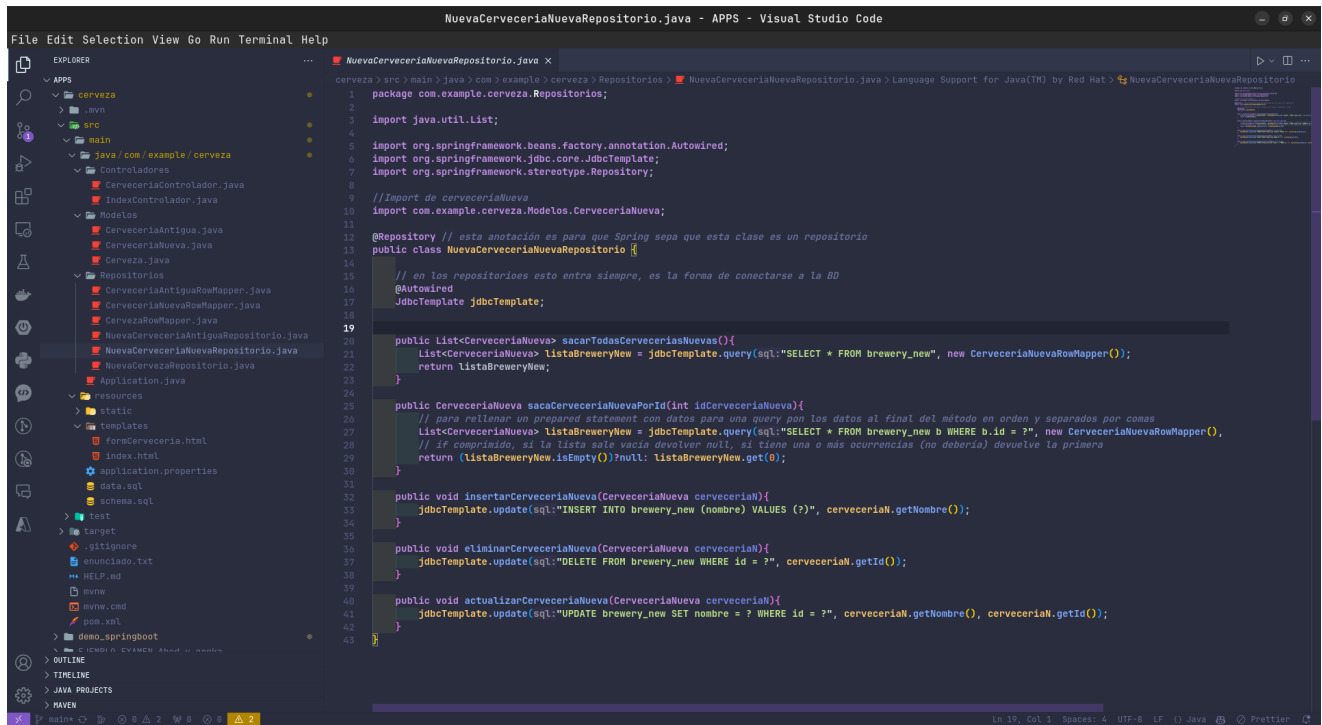
```
CerveceriaAntiguaRowMapper.java - APPS - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER
  APPS
    cerveza
      .mvn
      src
        main
          java/com/example/cerveza
            Controladores
            Modelos
              CerveceriaAntigua.java
              CerveceriaNueva.java
              Cerveza.java
            Repositorios
              CerveceriaAntiguaRowMapper.java
              CerveceriaNuevaRowMapper.java
              CervezaRowMapper.java
              NuevaCerveceriaAntiguaRepositorio.java
              NuevaCerveceriaNuevaRepositorio.java
              NuevaCervezaRepositorio.java
            Application.java
            recursos
              static
              templates
              application.properties
              data.sql
              schema.sql
            test
            target
            .gitignore
            enunciado.txt
            HELP.md
            mvnw
            mvnw.cmd
            pom.xml
            demo_springboot
            EJEMPLO_EXAMEN Abed y gorka
            prueba2
              cerveza.zip
              EJEMPLO_EXAMEN.zip
            OUTLINE
            TIMELINE
            JAVA PROJECTS
            MAVEN

CerveceriaAntiguaRowMapper.java x
cerveza > src > main > java > com > example > cerveza > Repositorios > CerveceriaAntiguaRowMapper.java > ...
1 package com.example.cerveza.Repositorios;
2
3 import java.sql.ResultSet;
4 import java.sql.SQLException;
5 //Importo el RowMapper de spring
6 import org.springframework.jdbc.core.RowMapper;
7 //Importo el modelo de cerveceria antigua para poder crear objetos de ese tipo
8 import com.example.cerveza.Modelos.CerveceriaAntigua;
9
10 //Crea la clase que implementa RowMapper y le digo que va a devolver objetos de tipo CerveceriaAntigua
11 public class CerveceriaAntiguaRowMapper implements RowMapper<CerveceriaAntigua>{
12
13     // @Override es una anotación que indica que el método que sigue es un método que se hereda de la clase padre
14     @Override
15     public CerveceriaAntigua mapRow(ResultSet rs, int rowNum) throws SQLException {
16         //Creamos un nuevo objeto de la clase CerveceriaAntigua en el que descargar los datos de la fila de la BD
17         CerveceriaAntigua cerveceriaAnt = new CerveceriaAntigua();
18
19         // los datos se sacan de una fila con el ResultSet (rs) usando los métodos getInt, getString...
20         // se usa el nombre en la BD para sacar cada campo
21         cerveceriaAnt.setId(rs.getInt("id"));
22         cerveceriaAnt.setNombre(rs.getString("nombre"));
23
24         // Devolvemos el objeto cerveceriaAnt
25         return cerveceriaAnt;
26     }
27
28 }
29
30
```

Creamos los métodos en el repositorio para insertar, buscar, etc de cada tabla:



The screenshot shows the Visual Studio Code editor with the file `NuevaCerveceriaNuevaRepository.java` open. The Explorer sidebar on the left shows the project structure, including the `Repositorios` package. The main editor displays the following Java code:

```
1 package com.example.cerveza.Repositorios;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.jdbc.core.JdbcTemplate;
7 import org.springframework.stereotype.Repository;
8
9 //Import de cerveceriaNueva
10 import com.example.cerveza.Modelos.CerveceriaNueva;
11
12 @Repository // esta anotación es para que Spring sepa que esta clase es un repositorio
13 public class NuevaCerveceriaNuevaRepository {
14
15     // en los repositorios esto entra siempre, es la forma de conectarse a la BD
16     @Autowired
17     JdbcTemplate jdbcTemplate;
18
19     public List<CerveceriaNueva> sacarTodasCerveceriasNuevas(){
20         List<CerveceriaNueva> listaBreweryNew = jdbcTemplate.query(sql:"SELECT * FROM brewery_new", new CerveceriaNuevaRowMapper());
21         return listaBreweryNew;
22     }
23
24     public CerveceriaNueva sacaCerveceriaNuevaPorId(int idCerveceriaNueva){
25         // para rellenar un prepared statement con datos para una query pon los datos al final del método en orden y separados por comas
26         List<CerveceriaNueva> listaBreweryNew = jdbcTemplate.query(sql:"SELECT * FROM brewery_new b WHERE b.id = ?", new CerveceriaNuevaRowMapper(),
27             // si comprimido, si la lista sale vacía devolver null, si tiene una o más ocurrencias (no debería) devuelve la primera
28             return (listaBreweryNew.isEmpty())?null: listaBreweryNew.get(0);
29         );
30     }
31
32     public void insertarCerveceriaNueva(CerveceriaNueva cerveceriaN){
33         jdbcTemplate.update(sql:"INSERT INTO brewery_new (nombre) VALUES (?)", cerveceriaN.getNombre());
34     }
35
36     public void eliminarCerveceriaNueva(CerveceriaNueva cerveceriaN){
37         jdbcTemplate.update(sql:"DELETE FROM brewery_new WHERE id = ?", cerveceriaN.getId());
38     }
39
40     public void actualizarCerveceriaNueva(CerveceriaNueva cerveceriaN){
41         jdbcTemplate.update(sql:"UPDATE brewery_new SET nombre = ? WHERE id = ?", cerveceriaN.getNombre(), cerveceriaN.getId());
42     }
43 }
```