https://platzi.com/cursos/java-spring/

https://platzi.com/cursos/postgresql/

**SOFTWARE UTILIZADO**

INTELLIJ IDEA

https://www.jetbrains.com/idea/download/

OPENJDK 11

https://adoptium.net/es/

POSTGRESQL (Port 5432)

https://www.enterprisedb.com/downloads/postgres-postgresql-downloads

POSTMAN

https://www.postman.com/downloads/

Despliegue: con Tomcat (por defecto), Jetty o Undertow.

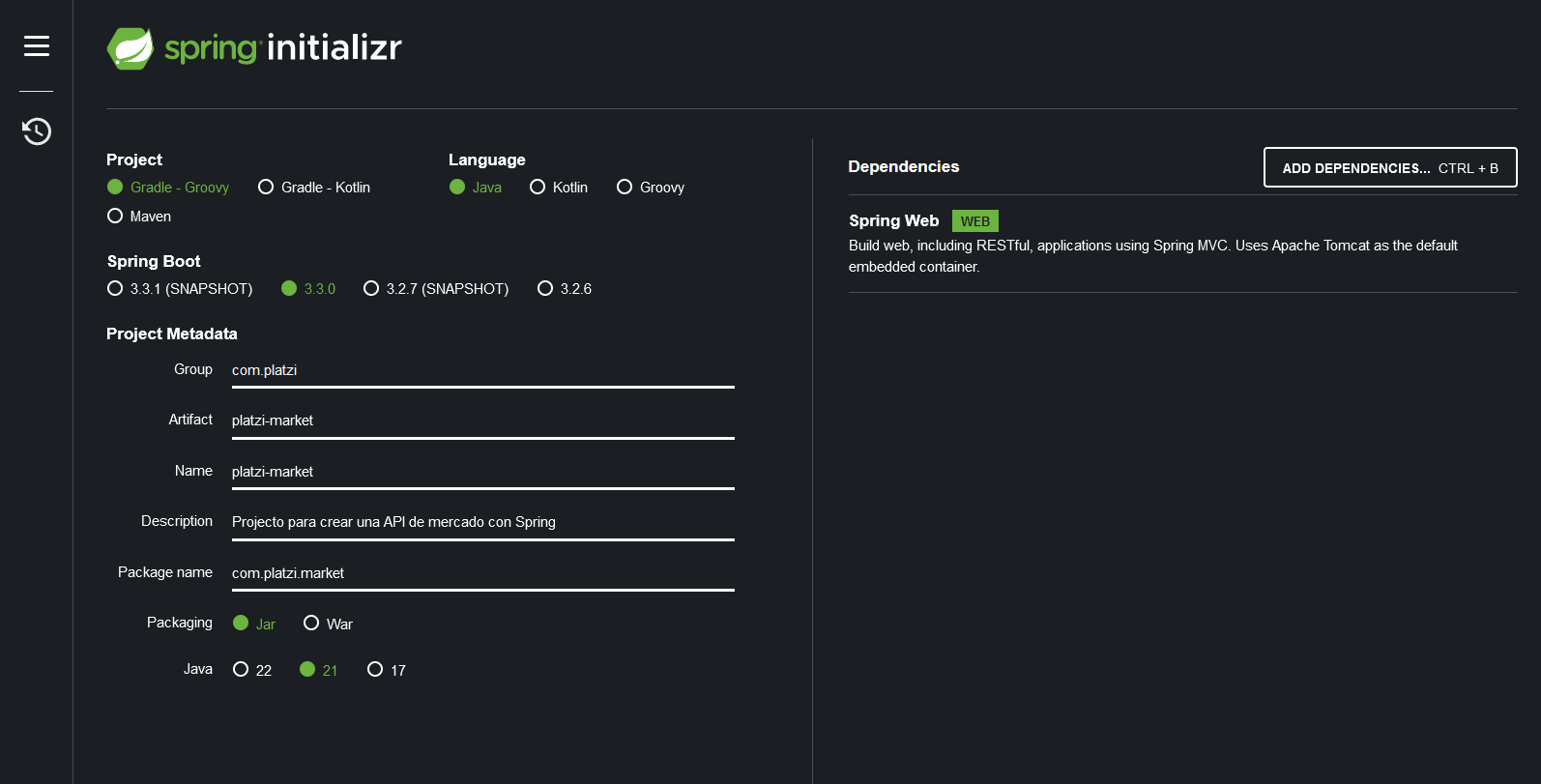
**SPRING**

Framework: gestionado con gradle

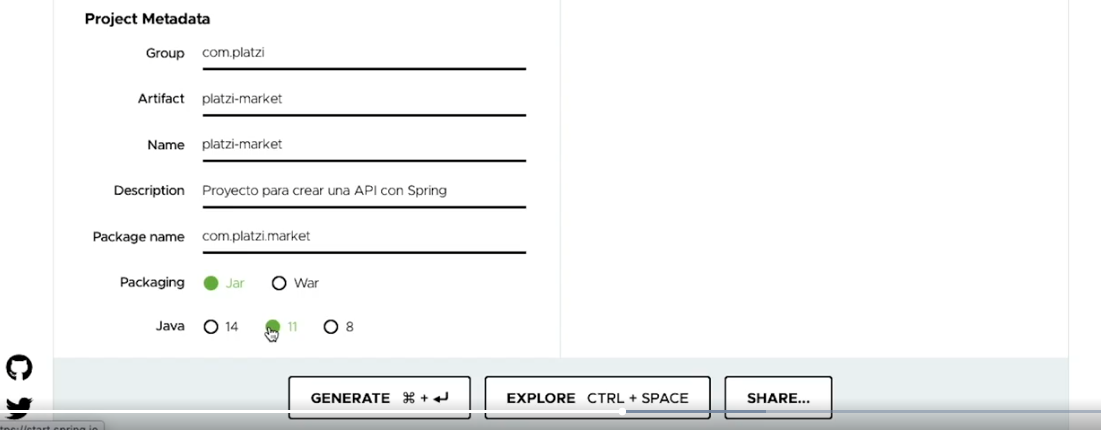
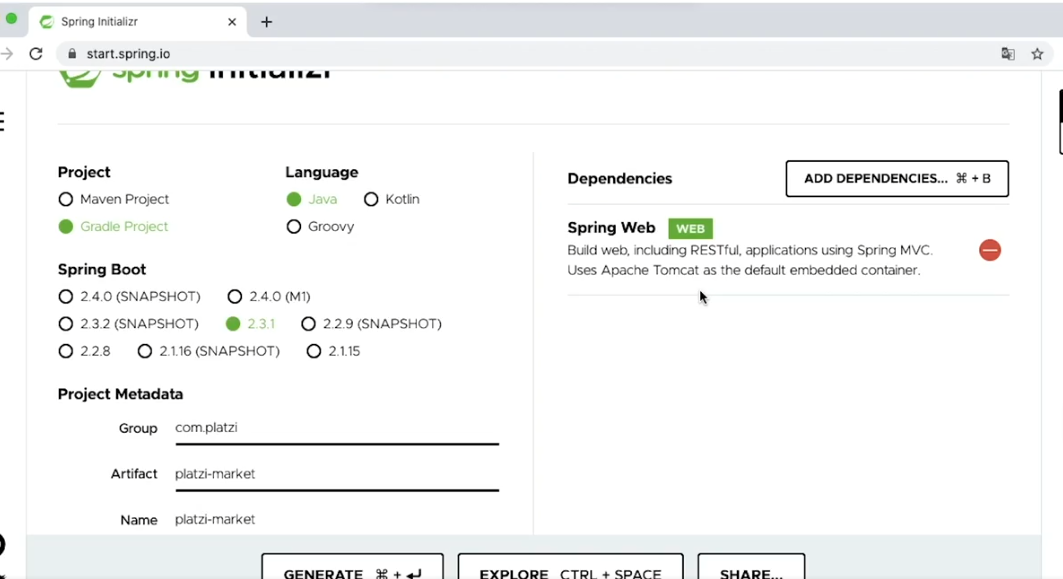
SPRING INITIALIZR: <https://start.spring.io/>

https://start.spring.io/#!type=gradle-project&language=java&platformVersion=3.3.0&packaging=jar&jvmVersion=21&groupId=com.platzi&artifactId=platzi-market&name=platzi-market&description=Projecto%20para%20crear%20una%20API%20de%20mercado%20con%20Spring&packageName=com.platzi.market&dependencies=web

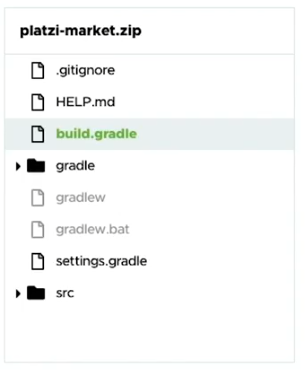
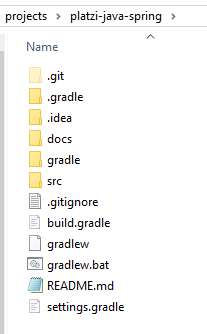
**CONFIGURACION PROYECTO A REALIZAR**



**CONFIGURACION CASO DESARROLLADO EN EL CURSO**



**ESTRUCTURA INICIAL DEL PROYECTO:**

.gitignore: para manejar los archivos que no deben ser gestionador por un gestor de versiones (como git)

Help.md: para describir nuestro proyecto

Build.gradle: escrito en groovy, esta toda la config de nuestra aplicación. Contiene los plugins, grupo, versión, versión de java, de que repo obtenemos las dependencias, las dependencias en sí, y como realizaremos tests de nuestro código.

Gradle (folder), gradlew, gradlew.bat, settings.gradle: contiene toda la info restante para que gradle funcione (nosotros no lo utilizamos, solo interactuamos con gradle a través del archivo build.gradle)

Src (folder): contiene main y test. Main para el proyecto, y test para las pruebas del proyecto. Main contiene java (paquetes de clases java) y resources (contiene recursos varios, y el application.properties que contiene variables de spring, o definidas por nosotros)

Método main: método encargado de ejecutar nuestra aplicación en java. Se le agrega la anotación “@SpringBootApplication” para indicarle a spring que desde aquí se inicia.

Docs: carpeta propia para documentar el desarrollo del proyecto

.git, .gradle, .idea: carpetas de proceso para cada programa. Nosotros no las utilizamos

Build (folder): carpeta donde se compila y desde donde se ejecuta el proyecto

**CONFIGURAR SPRING BOOT**

**Propiedades de la app**

application.properties (o application.yml, o por línea de comando al iniciar el proyecto). Ej modificar puerto donde se ejecuta, el context.path de nuestra app, etc.

Se pueden añadir propiedades propias, y gestionar perfiles según el tipo de despliegue (ej ambiente de desarrollo, y ambiente de producción)

Common app.prop de spring: <https://docs.spring.io/spring-boot/appendix/application-properties/index.html>

**ESTRUCTURA/ ARQUITECTURA DEL PROYECTO**

****

Utilizaremos una por capas orientada al dominio.

**Primer capa: DOMINIO**

* Dtos y objetos de dominio (ej contexto de un supermercado): Hacen parte del contexto del dominio
* Servicios: encargados de puente entre controladores y persistencia/repositorio de bd
* Especificación de repositorios: interfaces que definen reglas de juego o contratos que debe cumplir la persistencia para intervenir entre objetos de dominio y db

**Segunda capa: WEB**

* Controladores de API Rest: cada controlador por el que los usuarios se pueden conectar

**Tercer capa: PERSISTENCIA**

* Repositorios: implementan las especificaciones del dominio
* Entities: clases que mapean y hacen de “tabla” por la info contenida en la db

Flujo normal: Usuario > Controlador API > Servicio > Repositorio

**JPA**

Es una especificación de java (estándar) para un framework ORM. Interactura con tablas de DB en forma de objetos java.

Frameworks ORM mas populares para java: Hibernate, EclipseLink, TopLink y ObjectDB.

Anotaciones: para conectar clases a tablas de nuestra base de datos, para evitar hacerlo nativo con sql, y hacerlo todo con código java.

@Entity: indica a una clase java que representa una tabla de la base de datos

@Table: recibe el nombre de la tabla a la cual está mapeando nuestra clase

@Column: (anotación para atributos de la clase. No obligatoria, solo cuando nombre de columna =/= nombre de tabla)

@Id y @EmbededId: representan clave primaria de nuestra tabla en la clase. Id para CP sencilla, embededId para CP compuesta

@GeneratedValue: genera valores por defecto para las CP en nuestras tablas

@OneToMany y @ManyToOne

SQL:

SELECT \*

FROM products

WHERE id\_category = ?

ORDER BY name ASC;

Query Method (Java):

findByIdCategoryOrderByNameAsc(int idCategory);

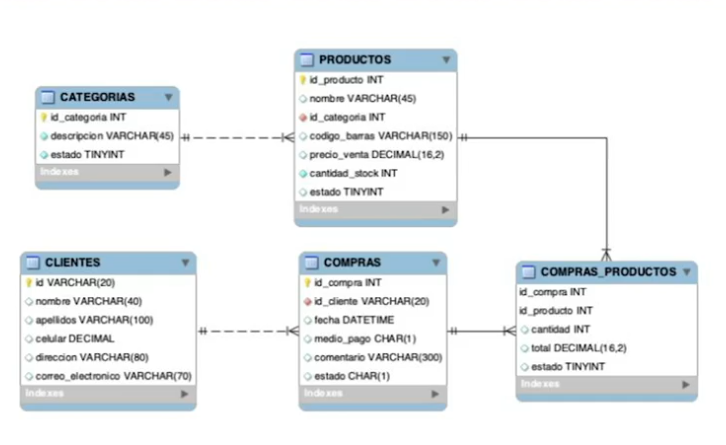
**SPRING DATA**

Proyecto que contiene otros sub proyectos. Nosotros usaremos Spring Data JPA (o Spring Data JDBC).

Se optimizan tareas repetitivas a traves de repos sin código que permiten todo tipo de operaciones en DB sin escribir código por la implementación específica.

Provee un servicio de auditorias para rastrear cuando se insertan, borran o modifican registros en la DB, sin tener que escribir código con las implementaciones específicas.

**MODELO DE DATOS**



Categorias (ej lácteos, aseo, verduras, frutas, etc)

Productos (cada producto individual)

Clientes (info de personas que compran)

Compras (un cliente puede estar en muchas compras)

Compras\_productos (tabla de doble relación con CP compuesta)

**CONECTAR DB**

Se configura en los application.properties respectivos, y agregando dependencias y configuraciones necesarias en nuestro archivo build.gradle

**MAPEAR TABLAS COMO CLASES**

Se crean entity beans que mapean relaciones con las tablas.

**INTERFACE CRUD REPOSITORY**

Tres tipos de repositorios:

* CrudRepository: realizar el crue
* PagingAndSortingRepository: incluye lo de Crud repository ademas de paginación y ordenamiento.
* JPARepository: Ademas de tener CrudRepository y PagingAndSortingRepository nos permite tareas específicas como Flush.

**QUERY METHODS**

Cuando necesitamos hacer consultas mas complejas que las consultas por defecto de Spring Data Repository. Permiten generar consultas mediante el nombre del método, con la opción de retornar objetos Optional<T> para dar mejor soporte a la programación funcional.

* SQL:

SELECT \*   
FROM products   
WHERE id\_category = ?  
ORDER BY name ASC;

* Query Method (Java):

findByIdCategoryOrderByNameAsc(int idCategory);

**DATA MAPPER**

Es la conversion de objetos asociados a tablas de una DB en nuevos objetos desacoplados. Esto evita exponer nuestra base de datos en la API, desacopla la API de una DB puntual, evitar campos innecesarios en la API, y evitar mezclar idiomas en la aplicación.

**INYECCION DE DEPENDENCIAS E INVERSION DE CONTROL CON @AUTOWIRE**

**IMPLEMENTACION SERVICIOS CON @SERVICE**

**IMPLEMENTACIÓN DE REST CON @RESTCONTROLLER**

**CONTROL DE RESPUESTAS HTTP CON @RESPONSEENTITY**