Universidad de Castilla-La Mancha
CAMPUS DE EXCELENCIA INTERNACIONAL

Facultad de Ciencias Sociales
Talavera de la Reina. UCLM
CAMPUS DE EXCELENCIA INTERNACIONAL

# Gestión de la configuración

*NextGen Software*

*Sara Gutiérrez Caja*

*Ángela Jiao Peng*

*Ingrid Niveiro Ben*

*Houda El Ouahabi Khirat*

# Índice

# 1. Introducción

## 1.1 Descripción del proyecto

El proyecto se trata de una aplicación web diseñada para gestionar servicios de reparto a domicilio, especialmente enfocada a la conexión entre restaurantes, repartidores y clientes. Busca optimizar la experiencia de pedido y entrega.

La aplicación web esta desarrollada en 'Spring Boot' como framework, 'Apache Derby' como base de datos y 'Maven' para la gestión de dependencias y construcción del proyecto. El entorno de desarrollo se configura en 'IntelliJ IDEA'. Además. El repositorio del proyecto está en 'GitHub', permitiendo un control de versiones y trabajo colaborativo. Por otra parte, se utiliza la herramienta SonarCloud para el análisis continuo de calidad del código, permitiendo identificar errores y vulnerabilidades en la etapa del desarrollo del proyecto.

### 1.2.1 Objetivos del proyecto

- **Gestión de pedidos.** Permitir a los clientes realizar pedidos.
- **Gestión de repartidores.** Asignar y coordinar entregas a sus repartidores.
- **Persistencia de datos.** Almacenar la información como la de los usuarios, restaurantes, pedidos y repartidores en una base de datos.
- **Escalabilidad y modularidad.** Diseñar una aplicación web que sea fácil de mantener y desplegar.

## 1.2 Objetivos de la gestión de configuración

La gestión de configuración en el proyecto tiene como finalidad garantizar un control adecuado sobre todos los elementos del desarrollo del software. Esto posibilita mantener la integridad y consistencia del proyecto a lo largo de su ciclo de vida. Los objetivos principales son:

1. **Centralización y control de versiones.**
   Garantizar que todos los artefactos mantengan una organización y sean accesibles desde un repositorio único, en este caso es 'Github'. Esto permite implementar un control de versiones utilizando Git, permitiendo gestionar cambios realizados en el código y otros componentes del proyecto.
2. **Automatización de la configuración.**
   El uso de herramientas como 'Maven' para gestionar dependencias y construir el proyecto. Se utiliza archivos como pom.xml para centralizar configuraciones.
3. **Estabilidad y trazabilidad.**
   Se debe de garantizar que cada versión del software sea estable y funcional, evitando errores derivados de configuraciones o conflictos de dependencias. Por otra parte, proveer una trazabilidad completa.
4. **Reproducibilidad.**
   Hay que asegurar que cualquier desarrollador pueda clonar el repositorio desde GitHub, configurar su entorno rápidamente y ejecutar el proyecto sin problemas.
5. **Control de calidad del código.**
   Garantizar que el software mantenga los estándares más altos de caliad en tér,inos de mantenibilidad y fiabilidad. La herramienta 'SonarCloud', nos permite identificar códigos duplicados, malas prácticas de desarrollo y vulnerabilidades. Al integrarse esta herramienta, 'SonarCloud' nos ayuda a prevernir la acumulación de deuda técnica.

# 2. Herramientas y entorno

## 2.1 Maven como herramienta de construcción

Maven es una herramienta de gestión de proyectos y construcción utilizado en el desarrollo Java. En nuestro proyecto, Maven desarrolla un papel fundamental para gestionar las dependencias, construir el proyecto y estandarizar el entorno.

Esta herramienta permite gestionar las librerías necesarias para el proyecto, esto lo hace a través de un archivo de configuración (pom.xml). Esto simplifica la inclusión de herramientas como Spiring Boot, Apache Derby y otras librerías. De esta forma, se descarga de forma automática desde los repositorios remotos.

En nuestro proyecto, el archivo pom.xml incluye dependencias esenciales como:

- Spring Boot Starter Web y Thymeleaf.
- Spring Boot Strarter Data JPA.
- Apache Derby.

## 2.2 Base de datos Apache Derby

Apache Derby es una base de datos embebida la cual es utilizada en nuestro proyecto para almacenar y gestionar datos relacionados con usuarios, pedidos, restaurantes, repartidores y demás entidades en el sistema.

### 2.2.1 Configuración de Apache Derby

La integración de la base de datos en el proyecto se realiza mediante dependencias declaradas en el archivo pom.xml. Las dependencias esenciales son:

```
<dependency>
    <groupId>org.apache.derby</groupId>
    <artifactId>derby</artifactId>
    <version>10.16.1.1</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.apache.derby</groupId>
    <artifactId>derbytools</artifactId>
    <scope>runtime</scope>
</dependency>
```

*Ilustración 1: pom.xml*

Por otro lado, para establecer la conexión a la base de datos se debe de realizar a través del archivo de configuración de application.properties. En este archivo se especifican las propiedades necesarias para la conexión a la base de datos.

```
spring.application.name=RepartoDomicilio

spring.datasource.url=jdbc:derby:database;create=true
spring.datasource.username=derbyuser
spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=update


spring.jackson.serialization.indent-output=true
```

*Ilustración 2: application.properties*

### 2.2.1 Utilización de Apache Derby en el proyecto

Se utiliza Spring Data JPA para mapear las clases Java del proyecto (por ejemplo, Cliente) a tablas en la base de datos.

Se utilizan anotaciones como @Entity, @Id, @GeneratedValue y @Column facilitan la creación automática de tablas y sus relaciones.

Ejemplo de la clase Cliente mapeada:

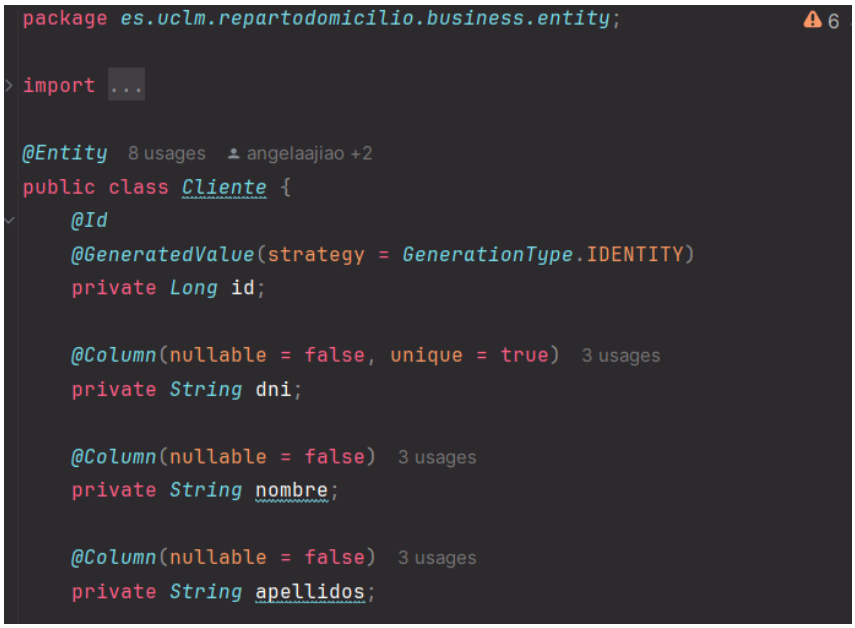


*Ilustración 3: Clase Cliente*

Posibles problemas a la hora de la incorporación de la base de datos:

- **Error a la hora de ejecutar el proyecto con maven:** para solucionar ese problema, la base de datos no debe de estar conectada. Por ende, la base de dates siempre debe de estar desconectada para lanzar el proyecto en ejecución.
- **Error de conexión a la de conectar la base de datos local con el apache Derby:** Hay que tomar en cuenta los Drivers y la ruta de directorio de la database.

## 2.3 Control de versiones con Git y Github

El control de versiones es fundamental para gantizar la organización y la estabilidad del código fuente. En este desarrollo de software, la herramientas Git y GitHub son herramientas principales para gestionar el control de versiones y el trabajo entre los desarrolladores.

Github y Git nos permite registrar y rastrear los cambios realizados en el código, además de proporcionarnos una plataforma en la nube para centralizar el repositorio.

### 2.3.1 Organización del repositorio

El repositorio está organizado de forma estructurada para facilitar la gestión del código y otros recursos.

1. **Carpetas y archivos principales:**

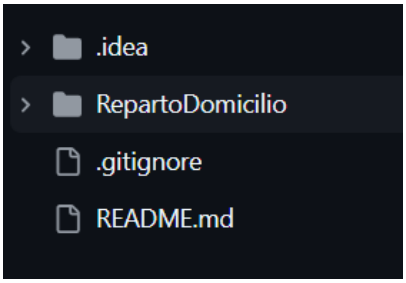


*Ilustración 4: Repositorio GitHub*

- **.idea:** Es una carpeta generada automáticamente por nuestro entorno IntelliJ IDEA, contine configuraciones específicas del entorno de desarrollo.
- **.gitignore:** Archivo que define qué archivos y carpetas no deben de ser rastreadas ni subidos al repositorio. Aquí se incluye configuraciones y archivos locales como derby.log.
- **RepartoDomicilio:** Carpeta que contine el núcleo del proyecto. Aquí se encuentra subcarpetas y otros elementos relacionados con el desarrollo y ejecución del proyecto.

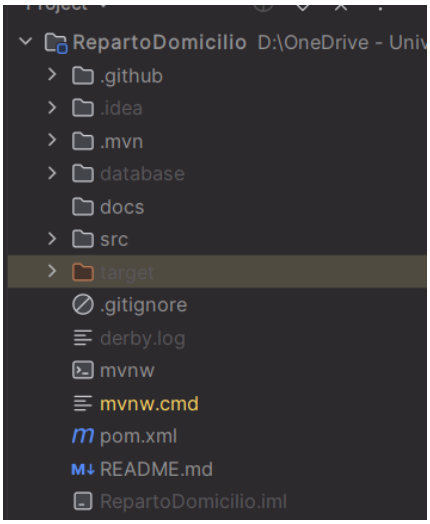


*Ilustración 5: Estructura de proyecto*

- **Pom.xml:** Archivo de configuración del proyecto.
- **/src/main/resources:** Archivos estáticos (imágenes) y archivos html. Por otro lado, también se encuentra el recurso 'application.properties' en el que se define la conexión a la base de datos.
- **/src/main/java:** Código fuente de la aplicación separado por carpetas; controller, entity, persistence, cumpliendo el principio de encapsulación.

### 2.3.2 Estrategia de ramificación

El uso de ramas en Github nos permite trabajar de forma simultánea en diferentes funcionalidades, características, correcciones de errores sin afectar la estabilidad del código principal.

Ramas principales:

- **Main:** Rama de producción del proyecto que contiene la versión estable y funcional del software. En esta rama se especificaciones de proyecto.
- **Hotfix:** Se destina a resolver errores críticos y urgentes que estén en la rama de producción. Tras la corrección de los errores, se mergea tanto en main como en develop.
- **Develop:** En esta rama se integra y prueba todo el trabajo antes de ser mergeado a la rama 'main'.
- **Release:** Rama para preparar una nueva versión del software antes de pasarla a producción.
- **Ramas de funcionalidades:** Se crean según el desarrollo de un funcionalidad especifica del proyecto. Cada rama sigue la convención de: dominio/funcionalidad.

### 2.3.3 Clonación del proyecto

Previamente debemos de tener instalado 'git' en nuestro sistema.

```
C:\Users\sarit>git -v
git version 2.46.2.windows.1
```

Por otro lado, debemos de tener Java Development Kit (JDK), Maven instalado y configurado en nuestras variables de entorno.

```
C:\Users\sarit>java -version
java version "19.0.1" 2022-10-18
Java(TM) SE Runtime Environment (build 19.0.1+10-21)
Java HotSpot(TM) 64-Bit Server VM (build 19.0.1+10-21, mixed mode, sharing)
```

```
C:\Users\sarit>mvn -version
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
Maven home: D:\workspace\apache-maven-3.9.9-bin\apache-maven-3.9.9
Java version: 19.0.1, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-19
Default locale: es_ES, platform encoding: UTF-8
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

Accedemos al repositorio GitHub y copiamos la URL del repositorio.

https://github.com/angelaajiao/NextGenSoftware.git

En la terminal, incorporamos:

Git clone <url-del-repositorio>

## 2.4 Implementación de SonarCloud

En el proyecto se utiliza SonarCloud como herramienta clave para el análisis de calidad del código e identificación de errores. Esta implementación se realiza de manera automatizada y está integrada dentro del flujo de desarrollo.

### 2.4.1 Configuración en el archivo pom.xml

La integración de SonarCloud se realiza a través del archivo pom.xml donde se especifican las siguientes propiedades.

```
<properties>
    <java.version>17</java.version>
    <sonar.organization>angelaajiao</sonar.organization>
    <sonar.host.url>https://sonarcloud.io</sonar.host.url>
</properties>
```

*Ilustración 6: SonarCloud pom.xml*

Esta propiedad va permitir que Maven pueda comunicarse con los servidores de SonarCloud para enviar los resultados del análisis.

### 2.4.2 Ejecución manual del análisis

Además de la ejecución automatizada, también es posible realizar análisis manuales a través de los siguientes comandos:

- **mvn sonar:sonar.** Realiza el análisis de SonarCloud y envía los resultados.
- **mvn verify sonar:sonar.** Ejecuta todas las pruebas definidas en el proyecto.

# 3. Ejecución del proyecto

Para ejecutar el proyecto en el entorno IntelliJ IDEA, debemos como hemos mecionado con anterioridad, la herramienta Maven instalada en nuesto sistama e incluirla en nuestras variables de entorno, tras ello se ejecuta el siguiente comando:

**mvn spring-boot:run**

Tras ello, desde nuestro navegador introducimos:

**localhost:8080**

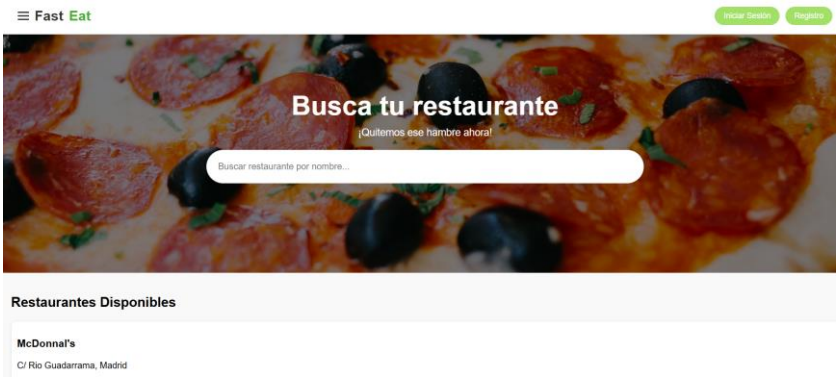


*Ilustración 7: Inicio*