**PROYECTO GRADLE**

| **Información general** | |
| --- | --- |
| **Duración estimada en minutos:** | 90 |
| **Docente:** | Carlos Andrés Florez Villarraga |
| **Guía no.** | 03 |

| **Información de la Guía** |
| --- |

**OBJETIVO**

Entender qué es y para qué sirve un proyecto Gradle en el desarrollo de aplicaciones con Java.

**CONCEPTOS BÁSICOS**

Archivos DSL y archivos de propiedades. MongoDB y creación de proyectos con Java.

**CONTEXTUALIZACIÓN TEÓRICA**

Una herramienta de gestión y construcción de proyectos automatiza el proceso de desarrollo de software, incluyendo la compilación del código, la ejecución de pruebas, el empaquetado de artefactos, y el despliegue. También gestiona las dependencias necesarias, permite la personalización del proceso de construcción, y asegura que las construcciones sean repetibles y consistentes. En Java las herramientas que más se usan para esto son: Maven y Gradle.

**Gradle**

Gradle es un sistema de automatización de construcción de software que se basa en conceptos de Apache Ant y Apache Maven, pero introduce un lenguaje específico del dominio (DSL) basado en Kotlin (o Groovy), en lugar del XML utilizado por Apache Maven para la configuración del proyecto.

En Gradle, la configuración del proyecto, incluidas las dependencias externas, plugins y propiedades para la compilación, se define en archivos llamados build.gradle.kts (uno por módulo). Además, se generan dos archivos adicionales: settings.gradle.kts, que permite definir los módulos del proyecto, y gradlew, que facilita la ejecución del proyecto directamente desde el archivo. Si utilizamos un IDE como IntelliJ IDEA, este IDE ejecutará automáticamente el archivo gradlew por nosotros.

La estructura mínima de un archivo build.gradle.kts es:

| plugins **{**  *java*  **}**  *group* = "org.example"  *version* = "1.0-SNAPSHOT"  *description* = "Breve descripción de la aplicación"  *repositories* **{**  mavenCentral()  **}**  *dependencies* **{**  *testImplementation*("org.junit.jupiter:junit-jupiter-api:5.8.1")  *testRuntimeOnly*("org.junit.jupiter:junit-jupiter-engine:5.8.1")  **}**  *tasks*.*test* **{**  useJUnitPlatform()  **}** |
| --- |

Se puede destacar la sección de plugins, donde se incluye el plugin de Java. A continuación, se especifica la información del proyecto, como la descripción, la versión y el grupo. También se define el repositorio para consultar las dependencias necesarias, se enumeran las dependencias que se utilizarán y se configura la gestión de pruebas unitarias del proyecto.

**Maven**

Otro sistema para la construcción de proyectos es Maven. Un proyecto de Maven puede tener dependencias externas (o locales), plugins y librerías, definidas en un archivo de configuración que se llama pom.xml. Este archivo tiene la siguiente estructura:

| <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <packaging>pom</packaging>  <groupId>com.example</groupId>  <artifactId>demo-test</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>demo-test</name>  <description>Demo project for Spring Boot</description>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>com.google.code.gson</groupId>  <artifactId>gson</artifactId>  </dependency>  <dependency>  <groupId>javax</groupId>  <artifactId>javaee-api</artifactId>  <version>8.0</version>  <scope>provided</scope>  </dependency>  </dependencies>  </project> |
| --- |

Para más información consulte: <https://maven.apache.org/guides/>

Para conocer más sobre gradle pueden visitar: <https://docs.gradle.org/current/userguide/userguide.html>

**PRECAUCIONES Y RECOMENDACIONES**

Recuerde verificar que tiene instalado el JDK de Java (preferiblemente la versión 21). Asegúrese de que tiene el servidor de MongoDB (mongod) en ejecución.

**ARTEFACTOS**

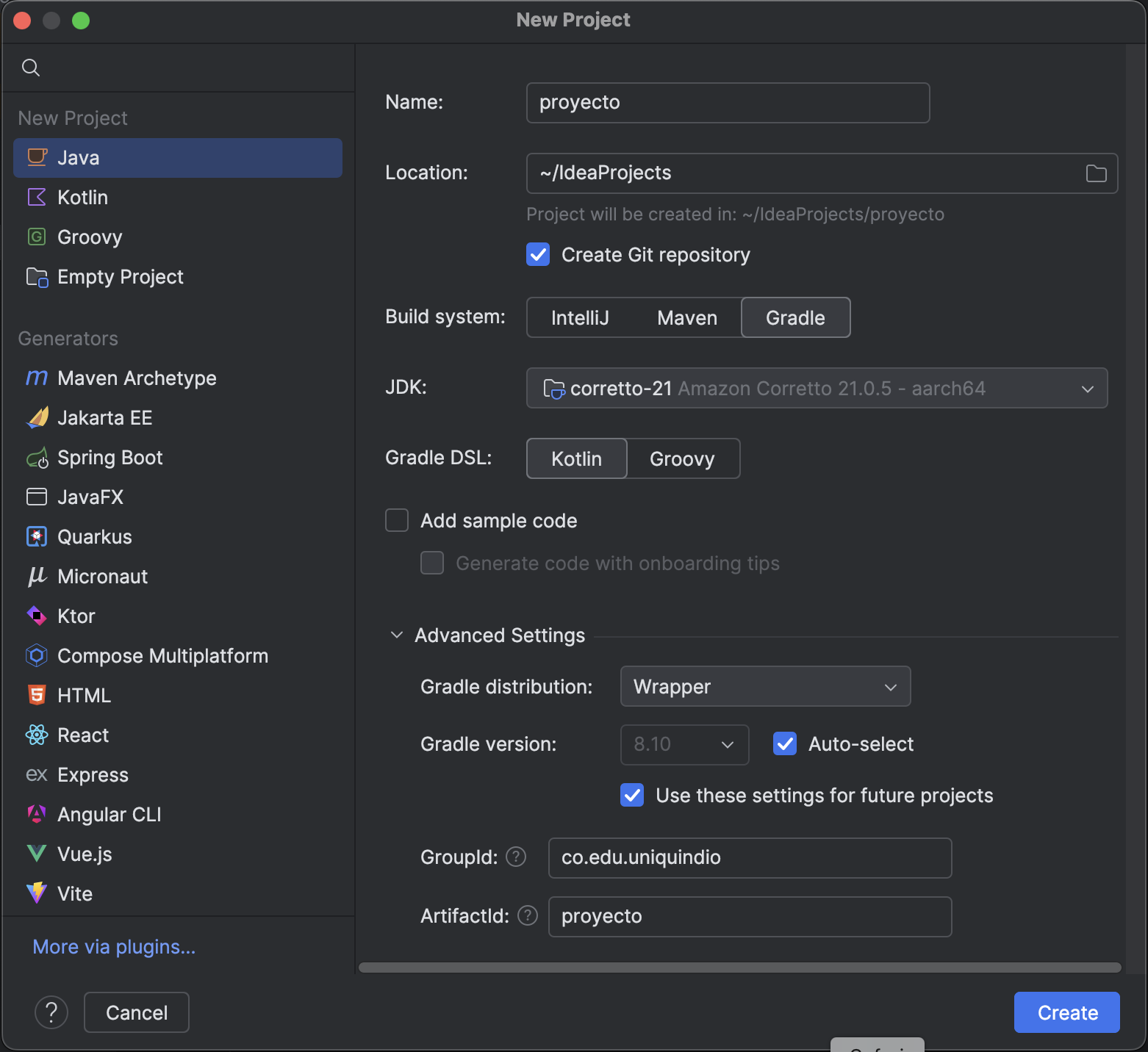
Se requiere tener instalado IntelliJ IDEA en su versión Ultimate, así como MongoDB Community Server, MongoDB Compass y el controlador de versiones GIT.

**EVALUACIÓN O RESULTADO**

Se espera que el estudiante cree un proyecto con las configuraciones básicas en IntelliJ IDEA usando Gradle.

| **Procedimiento** |
| --- |

1. Primero, vamos a crear un proyecto para el backend utilizando Gradle. Para esto vamos a File - New - Project. Seleccionamos la opción Gradle y elegimos el JDK 21 (se puede descargar directamente desde el IDE).



Escribimos el nombre del proyecto, activamos “Create Git Repository”, en sistema de construcción elegimos Gradle, en Gradle DSL elegimos Kotlin, y luego damos click en Create.

1. Modifique el archivo build.gradle.kts para que tenga una configuración más adecuada para nuestro proyecto, así:

| plugins **{**  *java*  id("org.springframework.boot") *version* "3.4.1"  id("io.spring.dependency-management") *version* "1.1.7"  **}**  *group* = "co.edu.uniquindio"  *version* = "0.0.1-SNAPSHOT"  *description* = "Breve descripción de la aplicación"  *java* **{**  toolchain **{**  *languageVersion* = JavaLanguageVersion.of(21)  **}**  **}**  *configurations* **{**  *compileOnly* **{**  extendsFrom(*configurations*.*annotationProcessor*.get())  **}**  **}**  *repositories* **{**  mavenCentral()  **}**  *dependencies* **{**  *implementation*("org.springframework.boot:spring-boot-starter-web")  *implementation*("org.springframework.boot:spring-boot-starter-validation")  *compileOnly*("org.projectlombok:lombok")  *annotationProcessor*("org.projectlombok:lombok")  *testImplementation*("org.springframework.boot:spring-boot-starter-test")  *testRuntimeOnly*("org.junit.platform:junit-platform-launcher")  **}**  *tasks*.*withType*<Test> **{**  useJUnitPlatform()  **}** |
| --- |

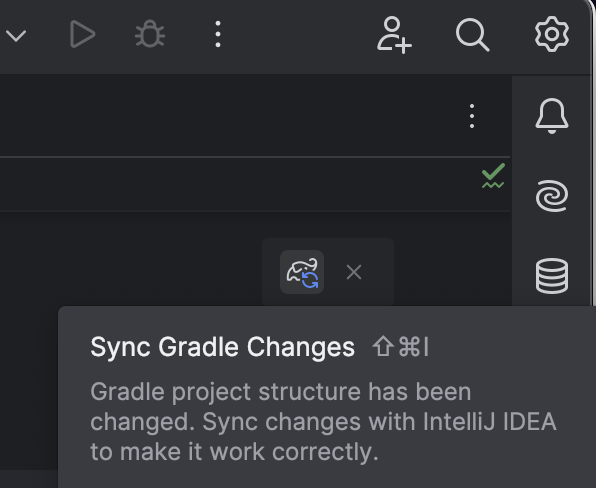
En la primera sección, se definen los plugins que se utilizarán en el proyecto. En este caso, incluimos el plugin de Spring Boot, el gestor de dependencias de Spring y, como era de esperar, el plugin de Java.

A continuación, se especifica la información del proyecto, incluyendo el grupo, la versión, la descripción y otros datos relevantes.

Luego, se enumeran las dependencias que se utilizarán en el proyecto.

Finalmente, se configura la plataforma para las pruebas unitarias; en este caso, se utilizará JUnit.

1. Actualice Gradle para que compile todo lo que acabamos de definir en el archivo build.gradle.kts, para ello de click en el botón flotante que aparece en la esquina superior derecha del archivo (el que tiene un icono de elefante).



Este paso es importante, si no lo hacemos, entonces el proyecto no se compilará y por lo tanto no se descargan las dependencias y librerías necesarias para que nuestro proyecto funcione correctamente.

**NOTA:** Recuerde hacer este paso siempre que se modifique algún archivo build.gradle.kts.

1. Ahora, agregue un nuevo archivo en la carpeta main/resources, dicho archivo se debe llamar application.properties. Para esto, haga clic derecho en dicha carpeta y vaya a New - File.

El archivo application.properties es fundamental en un proyecto de Spring. En este archivo se define la configuración de una aplicación, como por ejemplo, la conexión a la base de datos, el uso de modo debug, puerto de ejecución, hacer un logging más detallado, entre otros. Para conocer más sobre las propiedades que pueden ir en este archivo, podemos visitar:

<https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html>

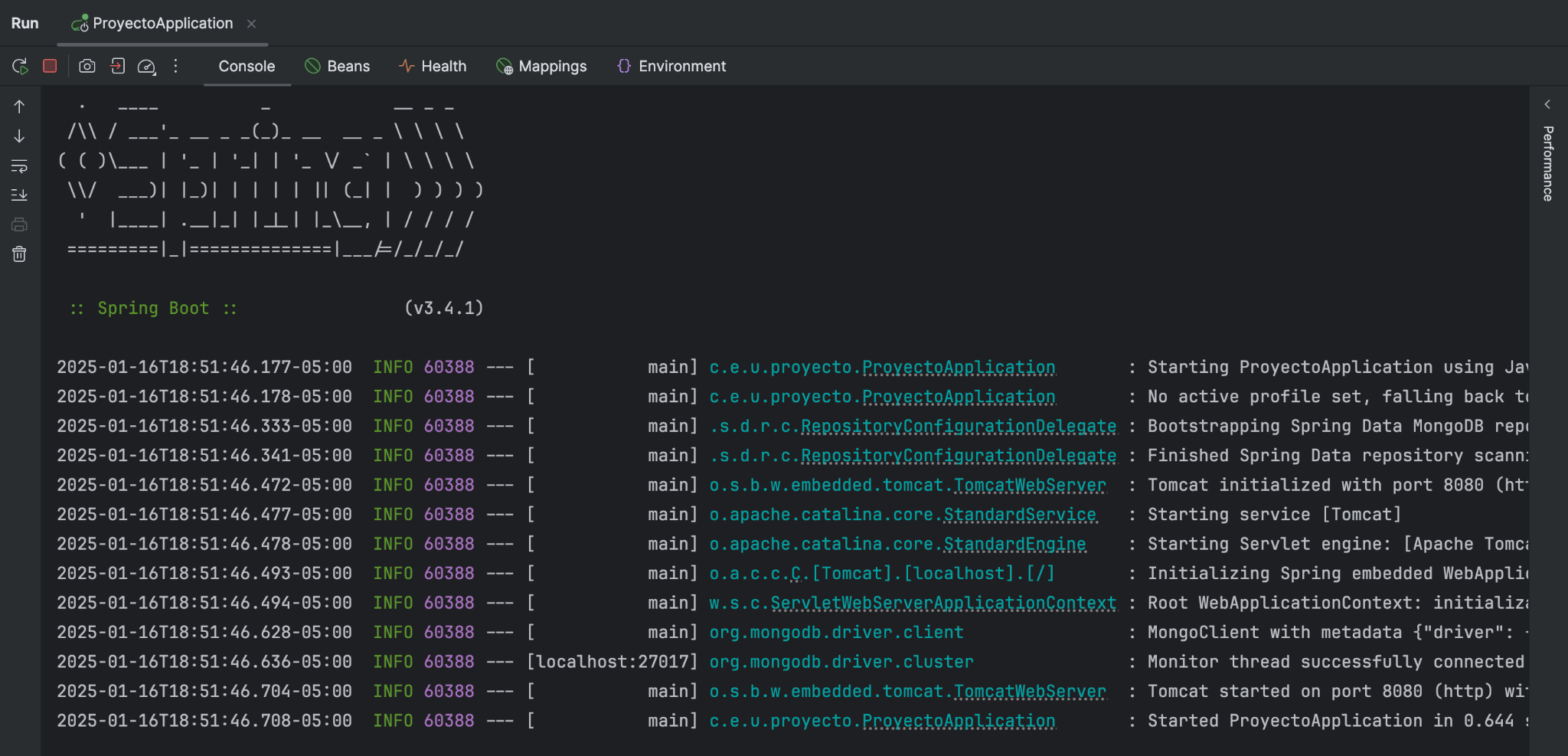
Por ahora, este archivo puede estar vacío.

1. Ahora, cree un paquete en la carpeta main/java que se llame así: co.edu.uniquindio.proyecto. Para esto simplemente haga clic derecho sobre la carpeta main/java y luego en New - Package. Una vez creado el paquete, cree allí una clase nueva que se llame ProyectoApplication.
2. Agregue las siguientes líneas de código a la clase ProyectoApplication para que quede así:

| package co.edu.uniquindio.proyecto;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class ProyectoApplication {    public static void main(String[] args) {  SpringApplication.*run*(ProyectoApplication.class, args);  }  } |
| --- |

Esta clase será la clase principal del proyecto Spring. Acá estamos configurando nuestro proyecto para que haga uso de Spring Boot. Cuando queramos desplegar el proyecto, simplemente ejecutamos el main de esta clase.

1. Ahora, despliegue el proyecto y verifique que todo ha funcionado correctamente. Para esto, ejecute el método main de la clase ProyectoApplication. Una vez hecho esto, se debe mostrar en la consola el proceso de ejecución donde irán apareciendo mensajes sobre el despliegue del proyecto, conexión a la base de datos, errores, advertencias, etc. En la consola le debe salir algo así:



Este es un despliegue normal, sin errores. Aunque nuestro proyecto aún no hace nada, ya tenemos una configuración básica que funciona y que está lista para todo lo que sigue. Es normal que la aplicación quede ejecutándose de forma indefinida.

1. Ya tenemos un proyecto Java creado con Spring boot y Gradle.
2. Hay archivos que no deberían guardarse en el repositorio, como los archivos .class y otros archivos de configuración del IDE. Estos archivos se pueden excluir. Para esto, cree un archivo con el nombre .gitignore en la raíz del proyecto. Y escriba lo siguiente dentro del archivo:

| /.idea/  \*.class  /.gradle/  /\*/bin/  /\*/build/  /build/ |
| --- |

**NOTA**: si ya existe el archivo .gitignore asegúrese que se están ignorando los archivos y carpetas descritas previamente.

1. Añada los archivos a su repositorio de Git y haga el commit inicial de su proyecto.
2. Agregue el repositorio remoto. Haga push de su repositorio local al repositorio remoto de GitHub.

| **Para la próxima clase** |
| --- |

Leer sobre los siguientes temas: Documentos en Spring Boot, DTO.