DEVOPS		Semester	6
Course Code	**BCSL657D**	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	0:0:2:0	SEE Marks	50
Credits	01	Exam Hours	100
Examination type (SEE)	Practical		

Course objectives:

- To introduce DevOps terminology, definition & concepts
- To understand the different Version control tools like Git, Mercurial
- To understand the concepts of Continuous Integration/ Continuous Testing/ Continuous Deployment)
- To understand Configuration management using Ansible
- Illustrate the benefits and drive the adoption of cloud-based Devops tools to solve real world problems

Sl.NO	Experiments
1	**Introduction to Maven and Gradle:** Overview of Build Automation Tools, Key Differences Between Maven and Gradle, Installation and Setup
2	**Working with Maven:** Creating a Maven Project, Understanding the POM File, Dependency Management and Plugins
3	**Working with Gradle:** Setting Up a Gradle Project, Understanding Build Scripts (Groovy and Kotlin DSL), Dependency Management and Task Automation
4	**Practical Exercise:** Build and Run a Java Application with Maven**,** Migrate the Same Application to Gradle
5	**Introduction to Jenkins:** What is Jenkins?, Installing Jenkins on Local or Cloud Environment, Configuring Jenkins for First Use
6	**Continuous Integration with Jenkins:** Setting Up a CI Pipeline**,** Integrating Jenkins with Maven/Gradle**,** Running Automated Builds and Tests
7	**Configuration Management with Ansible:** Basics of Ansible: Inventory, Playbooks, and Modules, Automating Server Configurations with Playbooks, Hands-On: Writing and Running a Basic Playbook
8	**Practical Exercise:** Set Up a Jenkins CI Pipeline for a Maven Project, Use Ansible to Deploy Artifacts Generated by Jenkins
9	**Introduction to Azure DevOps:** Overview of Azure DevOps Services, Setting Up an Azure DevOps Account and Project
10	**Creating Build Pipelines:** Building a Maven/Gradle Project with Azure Pipelines, Integrating Code Repositories (e.g., GitHub, Azure Repos), Running Unit Tests and Generating Reports
11	**Creating Release Pipelines:** Deploying Applications to Azure App Services, Managing Secrets and Configuration with Azure Key Vault, Hands-On: Continuous Deployment with Azure Pipelines

VTUSYNC.IN

12	**Practical Exercise and Wrap-Up:** Build and Deploy a Complete DevOps Pipeline, Discussion on Best Practices and Q&A

Course outcomes (Course Skill Set):

At the end of the course the student will be able to:

- Demonstrate different actions performed through Version control tools like Git.
- Perform Continuous Integration and Continuous Testing and Continuous Deployment using Jenkins by building and automating test cases using Maven & Gradle.
- Experiment with configuration management using Ansible.
- Demonstrate Cloud-based DevOps tools using Azure DevOps.

VTUSYNC.IN

Introduction to DevOps :

DevOps is a software development methodology that combines software development (Dev) and IT operations (Ops) to streamline the software delivery process. It focuses on automation, collaboration, continuous integration (CI), continuous delivery (CD), and monitoring to ensure faster, more efficient, and reliable software deployment.

In DevOps, teams use build automation tools like Maven and Gradle to automate the compilation, testing, and packaging of applications. These tools help developers manage dependencies, ensure consistent builds, and integrate code changes seamlessly, which are essential for continuous integration and continuous delivery (CI/CD) pipelines.

VTUSYNC.IN

Key Aspects of DevOps:

- **Collaboration and Communication** – DevOps bridges the gap between development and operations teams, fostering a culture of shared responsibility.
- **Automation** – Automating software builds, testing, deployment, and monitoring reduces manual errors and improves efficiency.
- **Continuous Integration & Continuous Delivery (CI/CD)** – Code changes are integrated frequently, tested automatically, and deployed seamlessly.
- **Infrastructure as Code (IaC)** – Infrastructure setup and management are automated using tools like Terraform and Ansible.
- **Monitoring and Feedback** – Continuous monitoring helps in identifying issues early and improving system performance.

DevOps Lifecycle:

The DevOps lifecycle consists of the following stages:

- **Plan** – Define project scope, requirements, and workflows using tools like Jira, Confluence.
- **Develop** – Code development using Git, GitHub, GitLab.
- **Build** – Compile and package code using Maven, Gradle.
- **Test** – Automated testing with tools like JUnit, Selenium.
- **Release** – Prepare for deployment using Jenkins, GitHub Actions.
- **Deploy** – Deploy applications using Ansible, Docker, Kubernetes.
- **Operate** – Manage infrastructure and monitor applications using Terraform, Prometheus.
- **Monitor** – Track system performance and logs using Grafana, ELK Stack.

Benefits of DevOps:

- **Faster Development & Deployment** – Reduces time from development to production.
- **Improved Collaboration** – Developers and IT teams work together efficiently.
- **Better Software Quality** – Automated testing ensures fewer bugs.
- **Scalability & Reliability** – Applications are more stable and scalable.
- **Enhanced Security** – Automated security checks prevent vulnerabilities.

VTUSYNC.IN

1. Introduction to Maven and Gradle: Overview of Build Automation Tools, Key Differences Between Maven and Gradle, Installation and Setup

Introduction to Maven and Gradle

Overview of Build Automation Tools

Build automation tools are essential in software development for managing dependencies, compiling source code, running tests, and packaging applications. They help in streamlining the development process and ensure reproducibility.

Two popular automation tools in the Java ecosystem are Maven and Gradle. Both are great for managing project builds and dependencies, but they have some key differences.

Maven :

Maven is a build automation tool primarily used for Java projects. It uses an XML configuration file called pom.xml (Project Object Model) to define project settings, dependencies, and build steps.

Main Features:

- Predefined project structure and lifecycle phases.
- Automatic dependency management through Maven Central.
- Wide range of plugins for things like testing and deployment.
- Supports complex projects with multiple modules.

VTUSYNC.IN

Gradle :

Gradle is a more modern and versatile build tool that supports multiple programming languages, including Java, Groovy, and Kotlin. It uses a domain-specific language (DSL) for build scripts, written in Groovy or Kotlin.

Main Features:

- Faster builds thanks to task caching and incremental builds.
- Flexible and customizable build scripts.
- Works with Maven repositories for dependency management.
- Excellent support for multi-module and cross-language projects.
- Integrates easily with CI/CD pipelines.

Key Differences Between Maven and Gradle

Aspect	Maven	Gradle
Configuration	XML (`pom.xml`)	Groovy or Kotlin DSL
Performance	Slower	Faster due to caching
Flexibility	Less flexible	Highly customizable
Learning Curve	Easier to pick up	Slightly steeper
Script Size	Verbose	More concise
Dependency Management	Uses Maven Central	Compatible with Maven too
Plugin Support	Large ecosystem	Extensible and versatile

VTUSYNC.IN

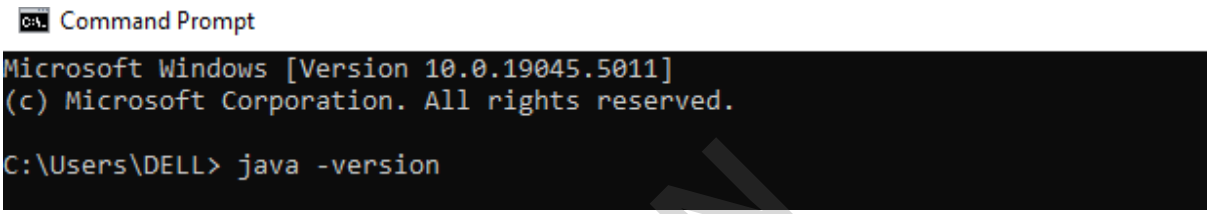
- Maven is great for structured, enterprise-level projects.
- Gradle is ideal for scalable and performance-driven applications.

Installation and Environment Setup

Installing Maven :

1. Install Java (JDK 17 Recommended)

    - Check if Java is installed:



2. **Download Maven**

    - Go to the Maven Download Page and download the latest binary ZIP file.

3. **Extract the ZIP File:**

    - Right-click the downloaded ZIP file and select **Extract All...** or use any extraction tool like WinRAR or 7-Zip.

4. **Move the Folder:**

    - After extraction, move the extracted **Maven folder** (usually named **`apache-maven-x.x.x`**) to a convenient directory like `C:\Program Files\`.

5. Navigate to the `bin` Folder:

    - Open the **Maven folder**, then navigate to the **`bin`** folder inside.
    - Copy the path from the File Explorer address bar (e.g., **`C:\Program Files\apache-maven-x.x.x\bin`**).

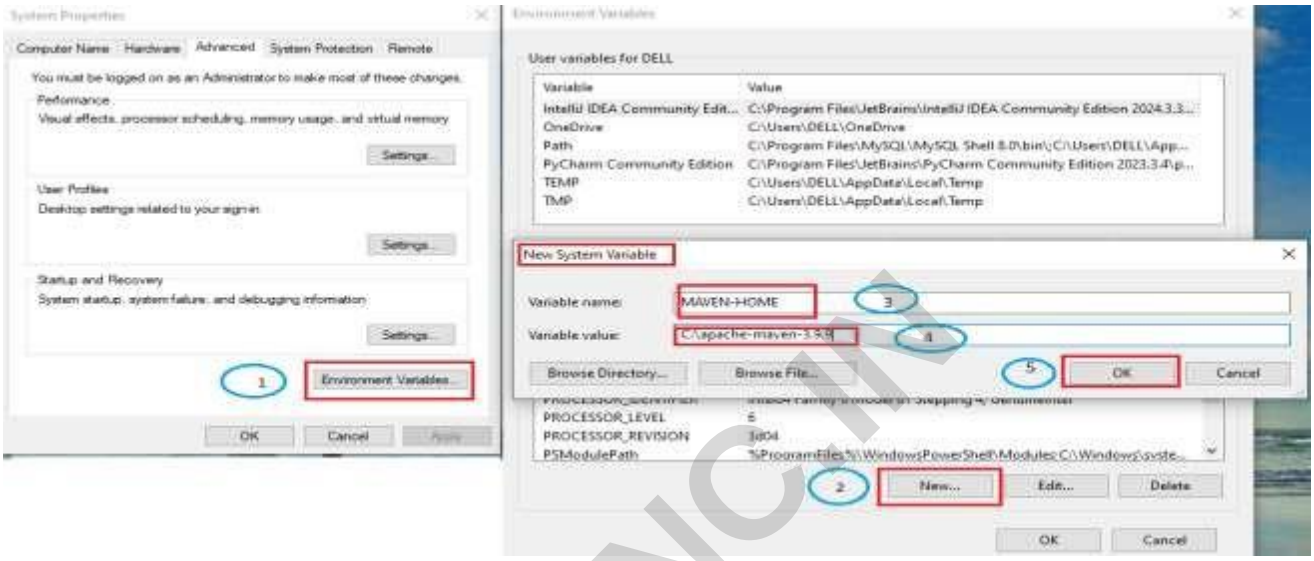
6. **Set Environment Variables:**

    - Open the **Start Menu**, search for **Environment Variables**, and select **Edit the system environment variables**.

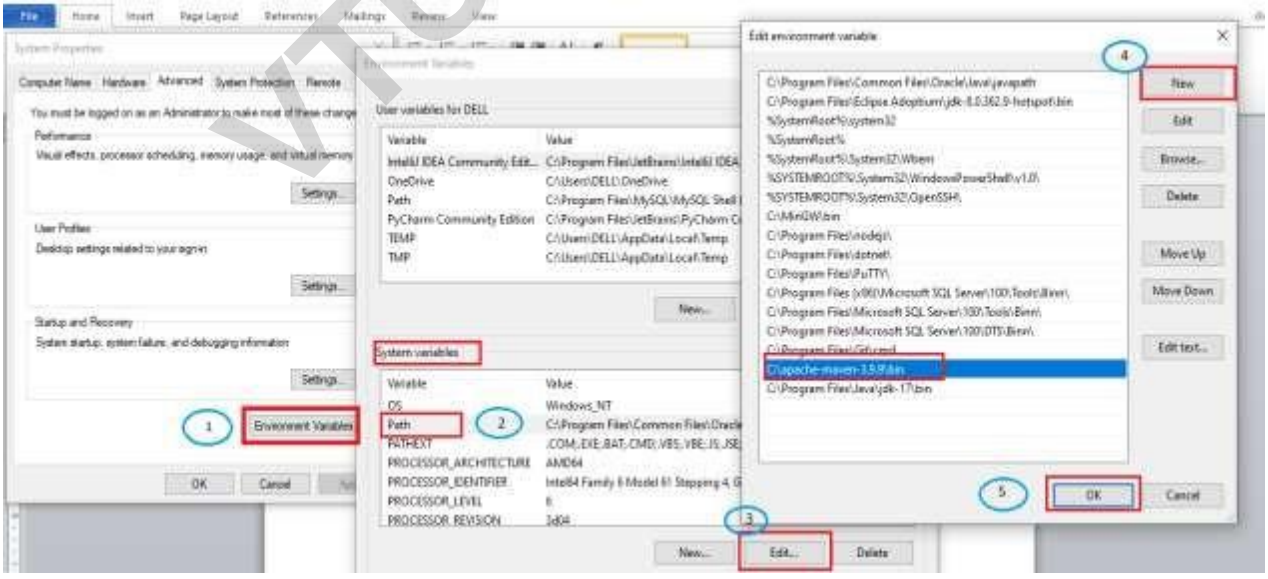
VTUSYNC.IN

- Click **Environment Variables**.
- Under **System Variables**:

(a) click **New** set variable name as MAVEN – HOME , paste the path



(b) Editing "Path" environment variable and adding the maven directory path up until "bin" folder.



VTUSYNCIN

7. **Verify the Installation:**

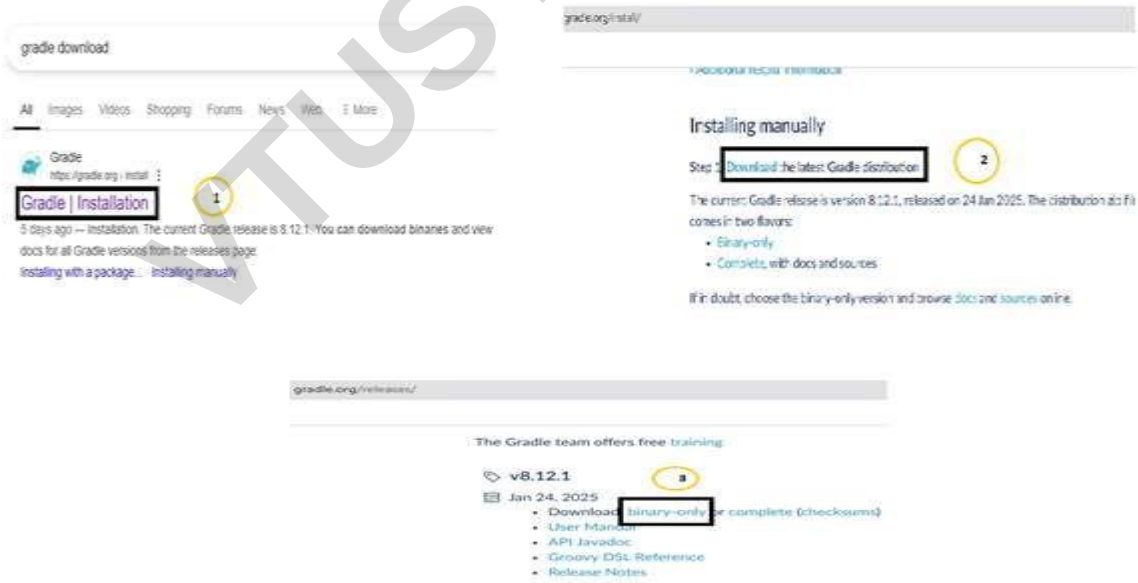
Open Command Prompt and run: mvn –version If Maven is correctly installed, it will display the version number.

VTUSYNC.IN

Installing Gradle

1. Download Gradle :

- Go to the Gradle Download Page and download the latest binary ZIP file.



2. **Extract the ZIP File:**

- Right-click the downloaded ZIP file and select **Extract All…** or use any extraction tool like WinRAR or 7-Zip.

3. **Move the Folder:**

- After extraction, move the extracted **Gradle folder** (usually named **`gradle-x.x.x`**) to a convenient directory like `C:\Program Files\`.

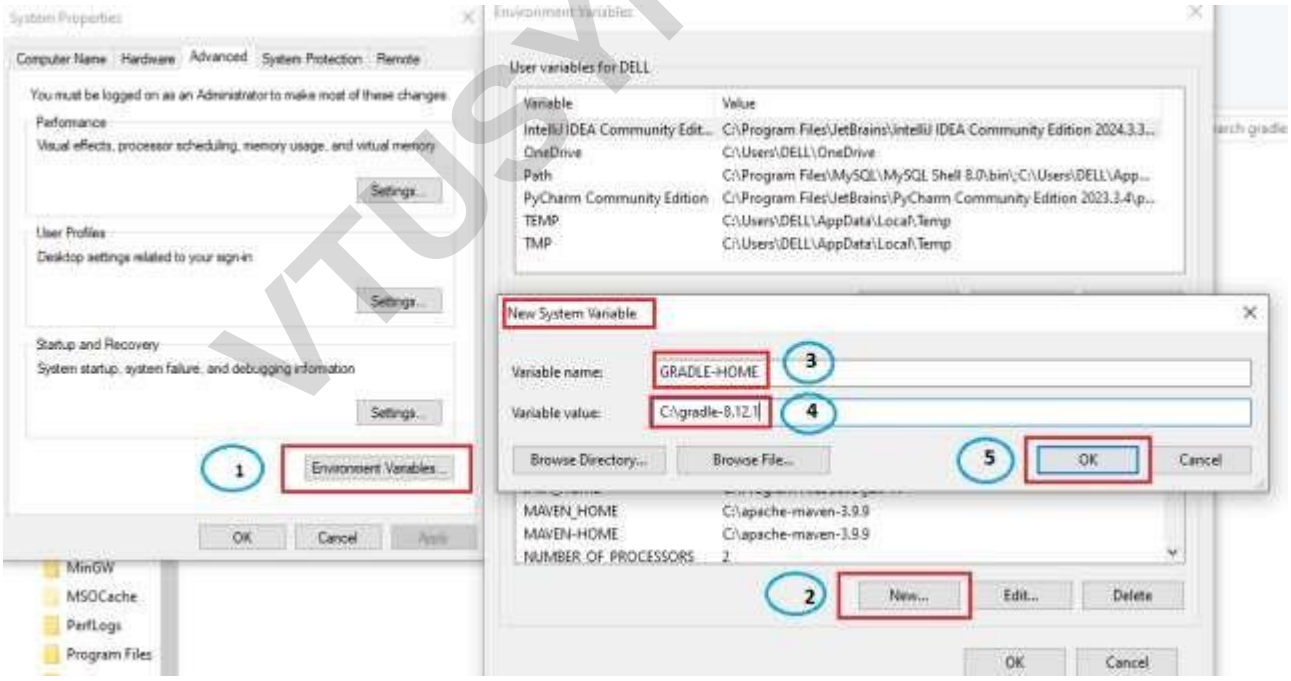
4. **Navigate to the `bin` Folder:**

- Open the **Gradle folder**, then navigate to the **`bin`** folder inside.
- Copy the path from the File Explorer address bar (e.g., **`C:\Program Files\gradle-x.x\bin`**).

5. **Set Environment Variables:**

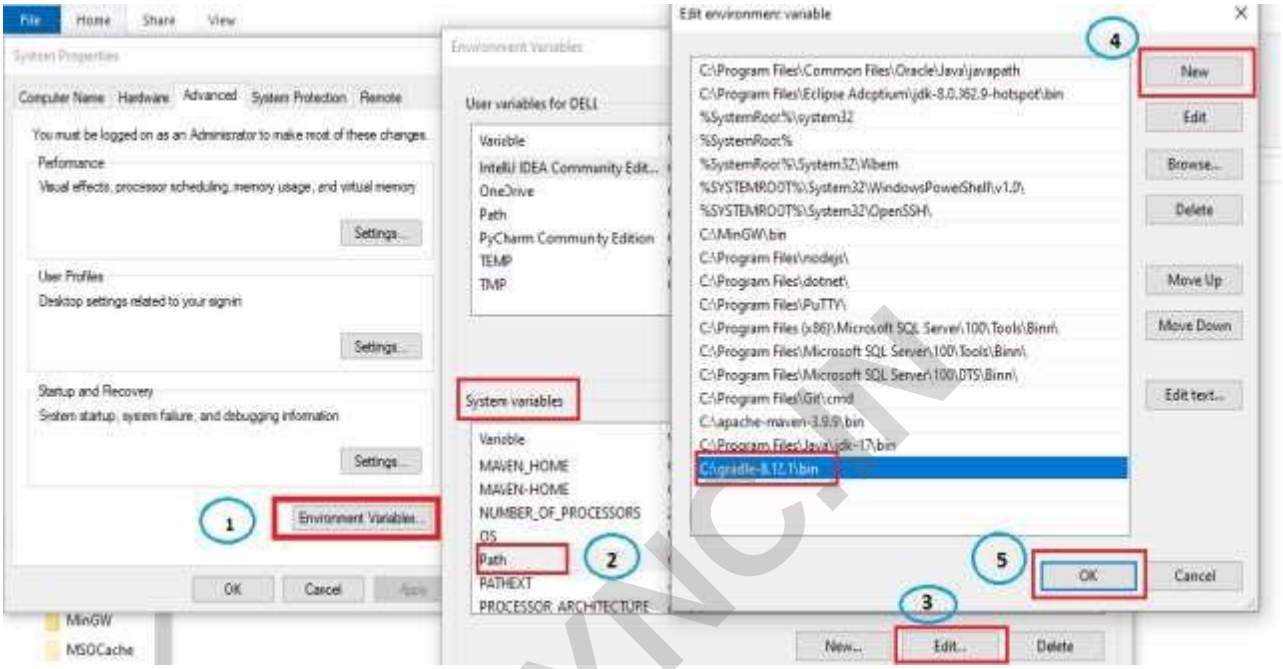
- Open the **Start Menu**, search for **Environment Variables**, and select **Edit the system environment variables**.
- Click **Environment Variables**.
- Under **System Variables**

(a) click **New** set variable name as GRADLE – HOME , paste the path



VTUSYNC.IN

(b) Editing "Path" environment variable and adding the gradle directory path up until "bin" folder.



6. **Verify the Installation:**

- Open a terminal or Command Prompt and run: **`gradle -version`** If it shows the Gradle version, the setup is complete.

```
Command Prompt
Microsoft Windows [Version 10.0.19045.5011]
(c) Microsoft Corporation. All rights reserved.

C:\Users\DELL>gradle -version

Welcome to Gradle 8.12.1!

Here are the highlights of this release:
 - Enhanced error and warning reporting with the Problems API
 - File-system watching support on Alpine Linux
 - Build and test Swift 6 libraries and apps

For more details see https://docs.gradle.org/8.12.1/release-notes.html

------------------------------------------------------------
Gradle 8.12.1
------------------------------------------------------------

Build time:    2025-01-24 12:55:12 UTC
Revision:      0b1ee1ff81d1f4a26574ff4a362ac9180852b140

Kotlin:        2.0.21
Groovy:        3.0.22
Ant:           Apache Ant(TM) version 1.10.15 compiled on August 25 2024
Launcher JVM:  17.0.12 (Oracle Corporation 17.0.12+8-LTS-286)
Daemon JVM:    C:\Program Files\Java\jdk-17 (no JDK specified, using current Java home)
OS:            Windows 10 10.0 amd64

C:\Users\DELL>
```

VTUSYNC.IN

2. Working with Maven: Creating a Maven Project, Understanding the POM File, Dependency Management and Plugins

Overview of the Project

1. Creating a Maven Project

There are a few ways to create a Maven project, such as using the command line, IDEs like IntelliJ IDEA or Eclipse, or generating it via an archetype.

(a) Using Command Line:

- To create a basic Maven project using the command line, you can use the following command

```
E:\Maven-project1>mvn archetype:generate -DgroupId=com.mvit -DartifactId=flipkart-app -DarchetypeArtifactId=maven-archetype-quickstart  -DarchetypeVersion=1.4 -Dinterac
tiveMode=false
```

- **`groupId`:** A unique identifier for the group (usually the domain name).
- **`artifactId`:** A unique name for the project artifact (your project).
- **`archetypeArtifactId`:** The template you want to use for the project.
- **`DinteractiveMode=false`:** Disables prompts during project generation.

This will create a basic Maven project with the required directory structure and pom.xml file.

(b) Using IDEs

Most modern IDEs (like IntelliJ IDEA or Eclipse) provide wizards to generate Maven projects. For example, in IntelliJ IDEA:

1. Go to **File > New Project**.
2. Choose **Maven** from the list of project types.
3. Provide the **`groupId`** and **`artifactId`** for your project.

VTUSYNC.IN

2. Understanding the POM File

The POM (Project Object Model) file is the heart of a Maven project. It is an XML file that contains all the configuration details about the project.

Below is an example of a simple POM file:

VTUSYNC.IN

Key element in `pom.xml`:

- **`<groupId>`:** The group or organization that the project belongs to.
- **`<artifactId>`:** The name of the project or artifact.
- **`<version>`:** The version of the project (often follows a format like `1.0-SNAPSHOT`).
- **`<packaging>`:** Type of artifact, e.g., `jar`, `war`, `pom`, etc.
- **`<dependencies>`:** A list of dependencies the project requires.
- **`<build>`:** Specifies the build settings, such as plugins to use.

3. Dependency Management

Maven uses the <dependencies> tag in the pom.xml to manage external libraries or dependencies that your project needs. When Maven builds the project, it will automatically download these dependencies from a repository (like Maven Central).

Example of adding a dependency:

```
<dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
```

VTUSYNC.IN

- **Transitive Dependencies**
  - Maven automatically resolves transitive dependencies. For example, if you add a library that depends on other libraries, Maven will also download those.

- **Scopes**
  - Dependencies can have different scopes that determine when they are available:
    - **compile** (default): Available in all build phases.
    - **provided**: Available during compilation but not at runtime (e.g., a web server container).
    - **runtime**: Needed only at runtime, not during compilation.
    - **test**: Required only for testing.

4. Using Plugins

Maven plugins are used to perform tasks during the build lifecycle, such as compiling code, running tests, packaging, and deploying. You can specify plugins within the <build> section of your pom.xml.

- **Adding Plugins**
  - You can add a plugin to your `pom.xml` like so:

```
<build>
   <pluginManagement>
     <plugins>
       <plugin>
         <artifactId>maven-clean-plugin</artifactId>
         <version>3.1.0</version>
       </plugin>
       <plugin>
         <artifactId>maven-compiler-plugin</artifactId>
         <version>3.8.0</version>
       </plugin>
       <plugin>
         <artifactId>maven-surefire-plugin</artifactId>
         <version>2.22.1</version>
       </plugin>
     </plugins>
   </pluginManagement>
 </build>
```

VTUSYNC.IN

1. **Common Plugins**
   - **maven-compiler-plugin**: Compiles Java code.
   - **maven-surefire-plugin**: Runs unit tests.
   - **maven-jar-plugin**: Packages the project as a JAR file.
   - **maven-clean-plugin**: Cleans up the `target/` directory.

2. **Plugin Goals** Each plugin consists of goals, which are specific tasks to be executed. For example:
   - **`mvn clean install`:** This will clean the target directory and then install the package in the local repository.
   - **`mvn compile`:** This will compile the source code.
   - **`mvn test`:** This will run unit tests.

Working with Maven Project

Note: Always create separate folder to do any program.

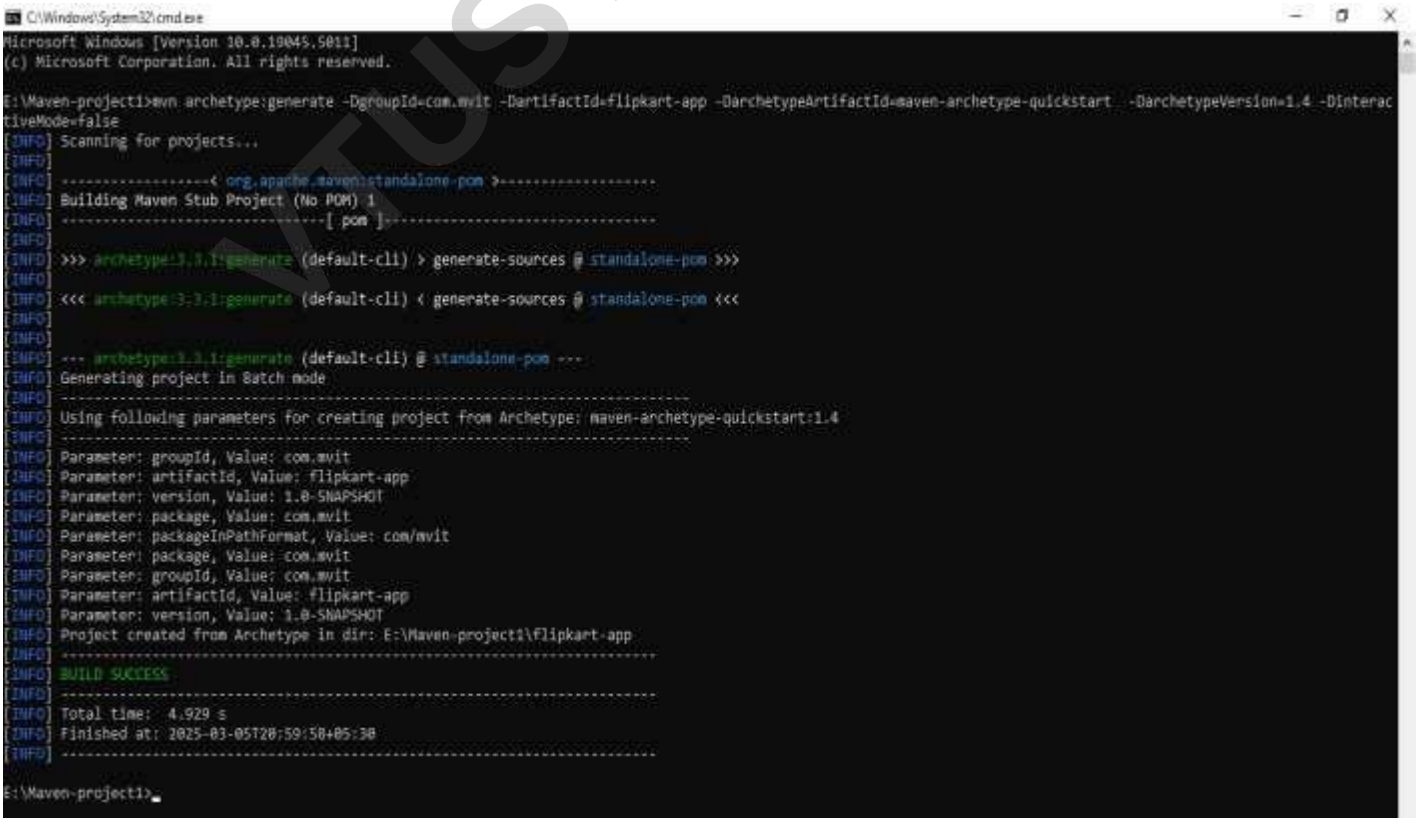
- Create a folder in a local drive D or E and then open a command prompt
- After then follow the below steps to working with Maven project.

Step 1: Creating a Basic Maven Java Application Project

- You can create a Maven project using the mvn command or through an IDE (such as IntelliJ IDEA or Eclipse). Below is the essential pom.xml configuration and corresponding Java code required for the project.
- To create a Maven project using the command line, use the following command:

```
E:\Maven-project1>mvn archetype:generate -DgroupId=com.mvit -DartifactId=flipkart-app -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -Dintera
ctiveMode=false
```

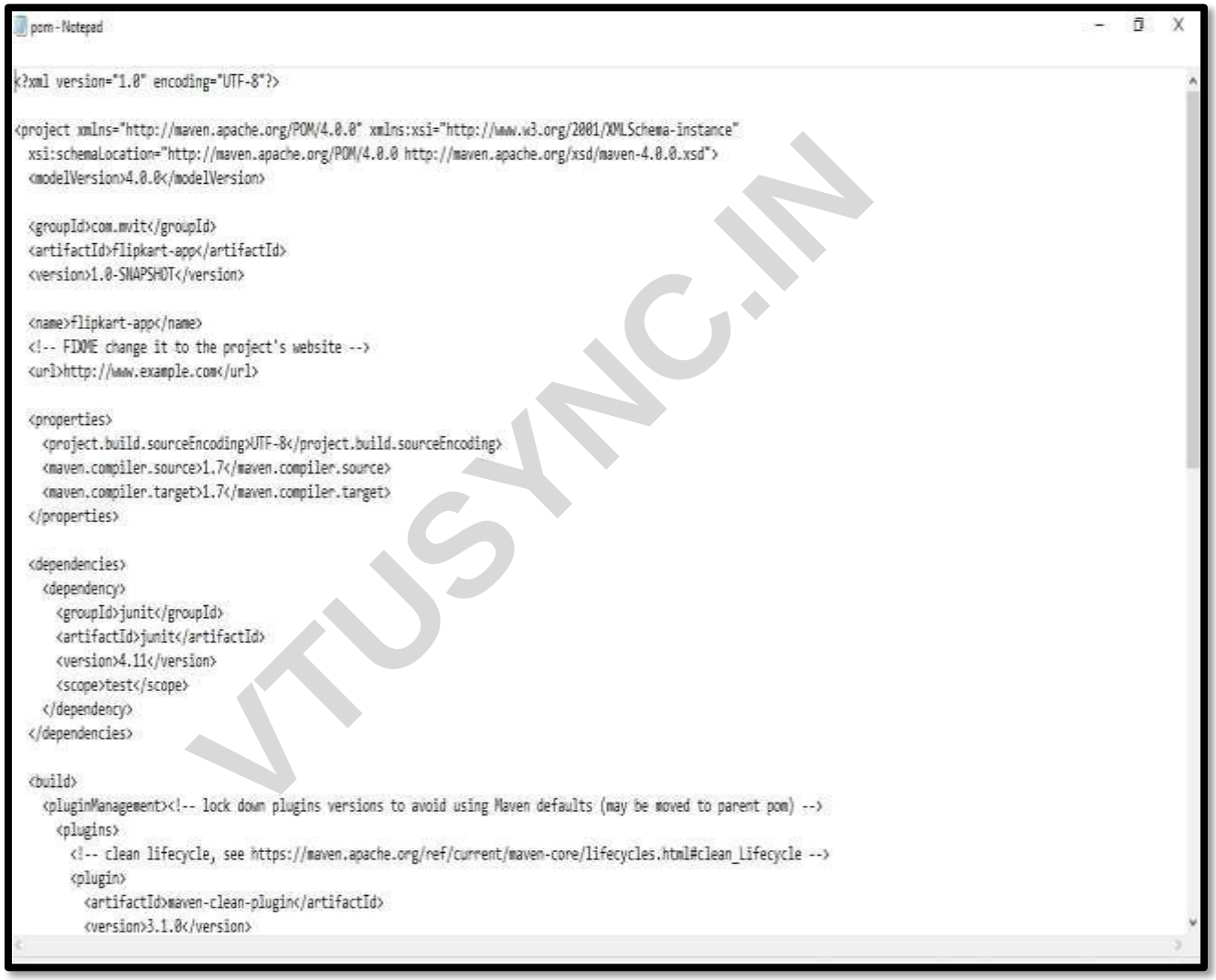
This command generates a basic maven java application with a predefined structure using the Quickstart archetype.



VTUSYNC.IN

Step 2: Open The pom.xml File

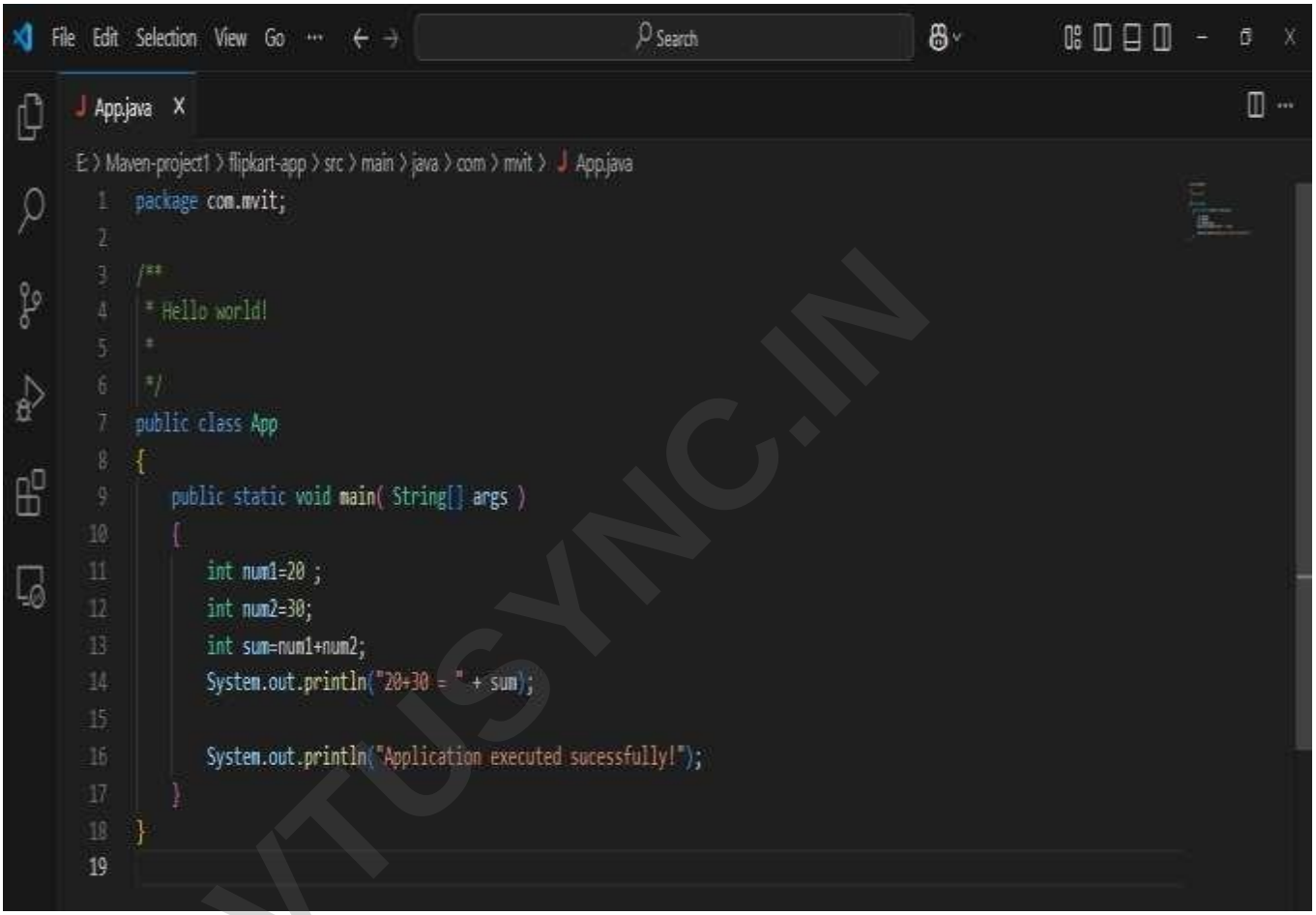
- You can manually navigate the **project folder** named call flipkart-app and open the file pom.xml
- In case if you not getting project folder then type command in your cmd.
  - **cd flipkart-app** – is use to navigate the project folder.
  - **notepad pom.xml** – is use to open pom file in notepad.



VTUSYNC.IN

Step 3: Open Java Code (App.java) File

- Open a file **`App.java`** inside the **`src/main/java/com/mvit/`** directory.
- After opening the **App.java** write the simple java program



VTUSYNC.IN

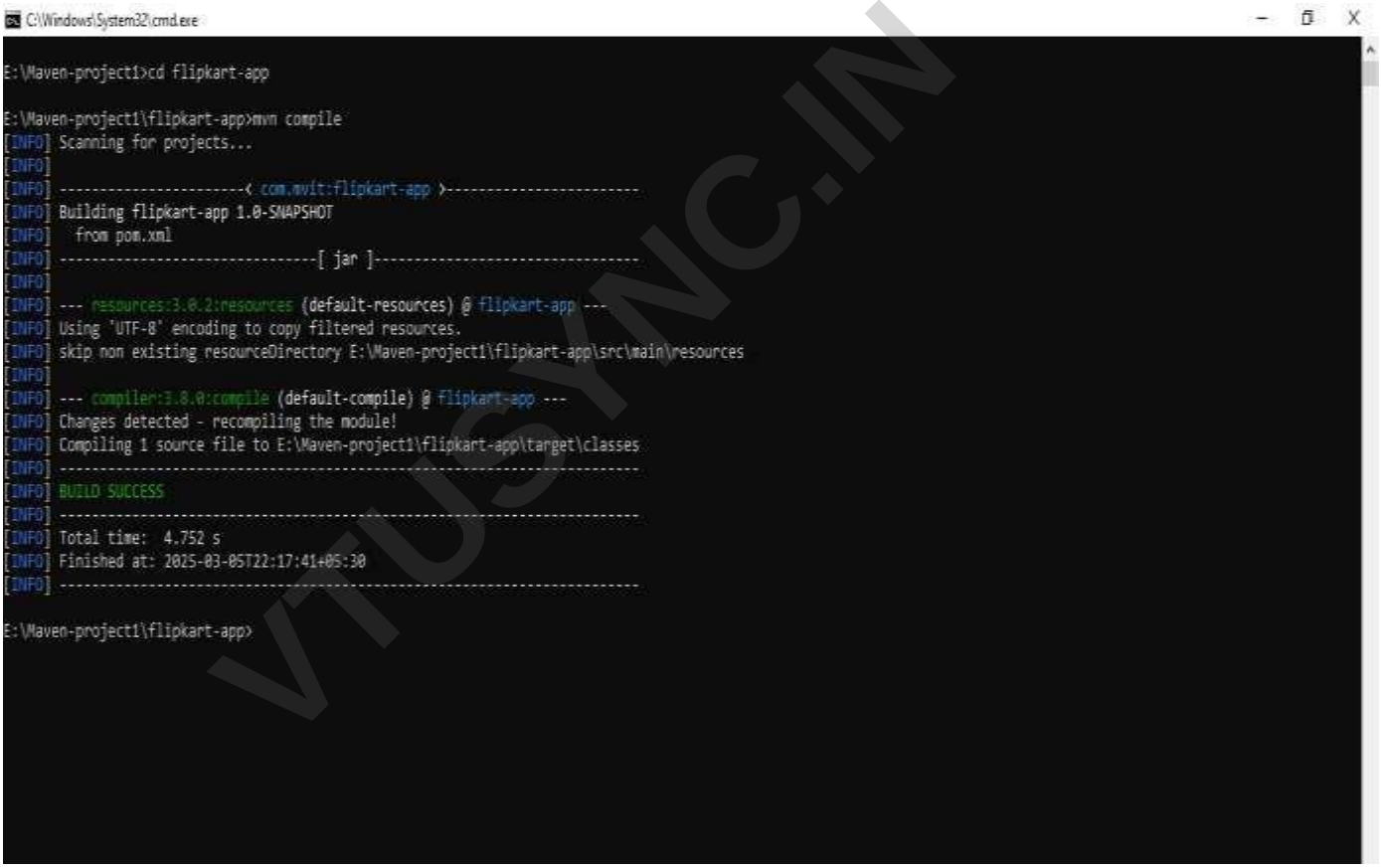
Note: before building the project make sure you are in the project folder if not navigate the project folder type command in your command prompt **cd flipkart-app**

Step 4: Building the Project

To build and run this project, follow these steps:

1. **Compile the Project**

```
C:\Windows\System32\cmd.exe

E:\Maven-project1>cd flipkart-app

E:\Maven-project1\flipkart-app> mvn compile
```

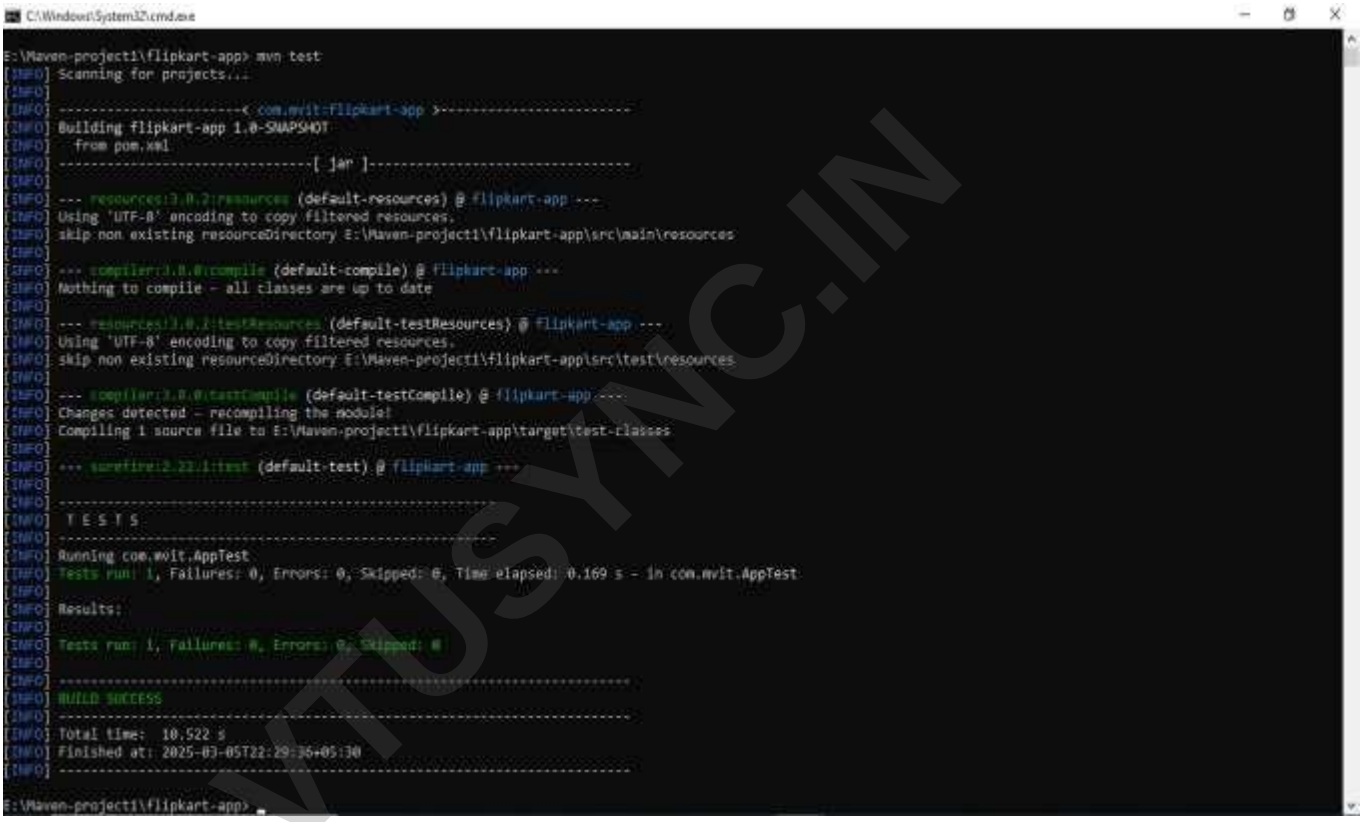


VTUSYNC.IN

2. Run the Unit Tests

C:\Windows\System32\cmd.exe

```
E:\Maven-project1\flipkart-app> mvn test
```

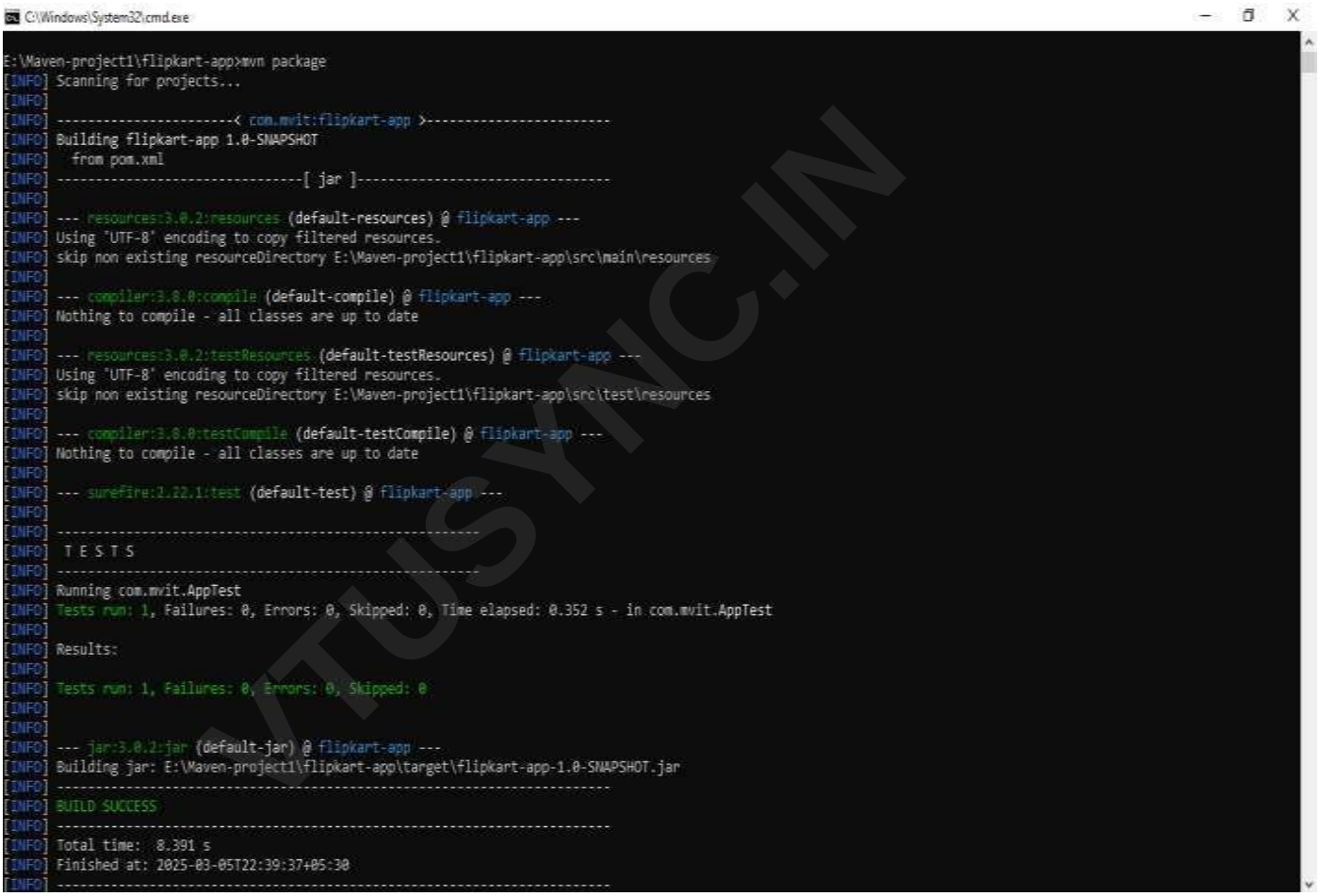


VTUSYNC.IN

3. Package the project into a JAR

C:\Windows\System32\cmd.exe

```
E:\Maven-project1\flipkart-app> mvn package
```



VTUSYNC.IN

4. Run the application (using JAR)

The above command is used to **run a Java application** from the command line. Here's a breakdown of each part:

- `java`: This is the Java runtime command used to run Java applications.
- `-cp`: This stands for **classpath**, and it specifies the location of the classes and resources that the JVM needs to run the application. In this case, it's pointing to the JAR file where your compiled classes are stored.
- `target/flipkart-app-1.0-SNAPSHOT.jar`: This is the **JAR file** (Java ARchive) that contains the compiled Java classes and resources. It's located in the `target` directory, which Maven creates after you run `mvn package`.
- `com.mvit.App`: This is the **main class** that contains the `main()` method. When you run this command, Java looks for the `main()` method inside the `App` class located in the `com.example` package and executes it.

VTUSYNC.IN

Creating a Basic Maven Web-Based Java Application Project

To create a Maven-based Java web application, the essential requirements include a JDK, Maven, a web server like Tomcat, an IDE, a web browser, and necessary dependencies. Setting up a proper environment ensures smooth project development and deployment.

1. j**ava Development Kit (JDK)**

- Java is the programming language used for web applications.
- JDK provides tools like the Java compiler and runtime environment.
- Install JDK 17 and set up the `JAVA_HOME` environment variable.

2. **Apache Maven**

- Maven helps automate project setup, dependency management, and builds.
- It ensures that the required libraries are downloaded and organized.
- Install Maven and verify its setup using `mvn -version`.

3. **Web Server (Apache Tomcat)**

- Tomcat is used to run Java web applications.
- It acts as a servlet container and processes web requests.
- Install Tomcat and start it to deploy and test the application.

4. Integrated Development Environment (IDE)

- An IDE makes coding easier with features like debugging and auto-completion.
- Examples: **Eclipse, IntelliJ IDEA, NetBeans**.
- Choose an IDE that supports Maven projects.

5. Web Browser

- A browser is required to access and test the web application.
- Examples: **Google Chrome, Mozilla Firefox, Microsoft Edge**.

6. Maven Dependencies

- Dependencies are external libraries required for the web application.
- Important dependencies:
  - **Servlet API** (to handle web requests).
  - **JSP API** (for dynamic web pages).
  - **JSTL** (to simplify JSP coding).
- These dependencies are added in the `pom.xml` file.

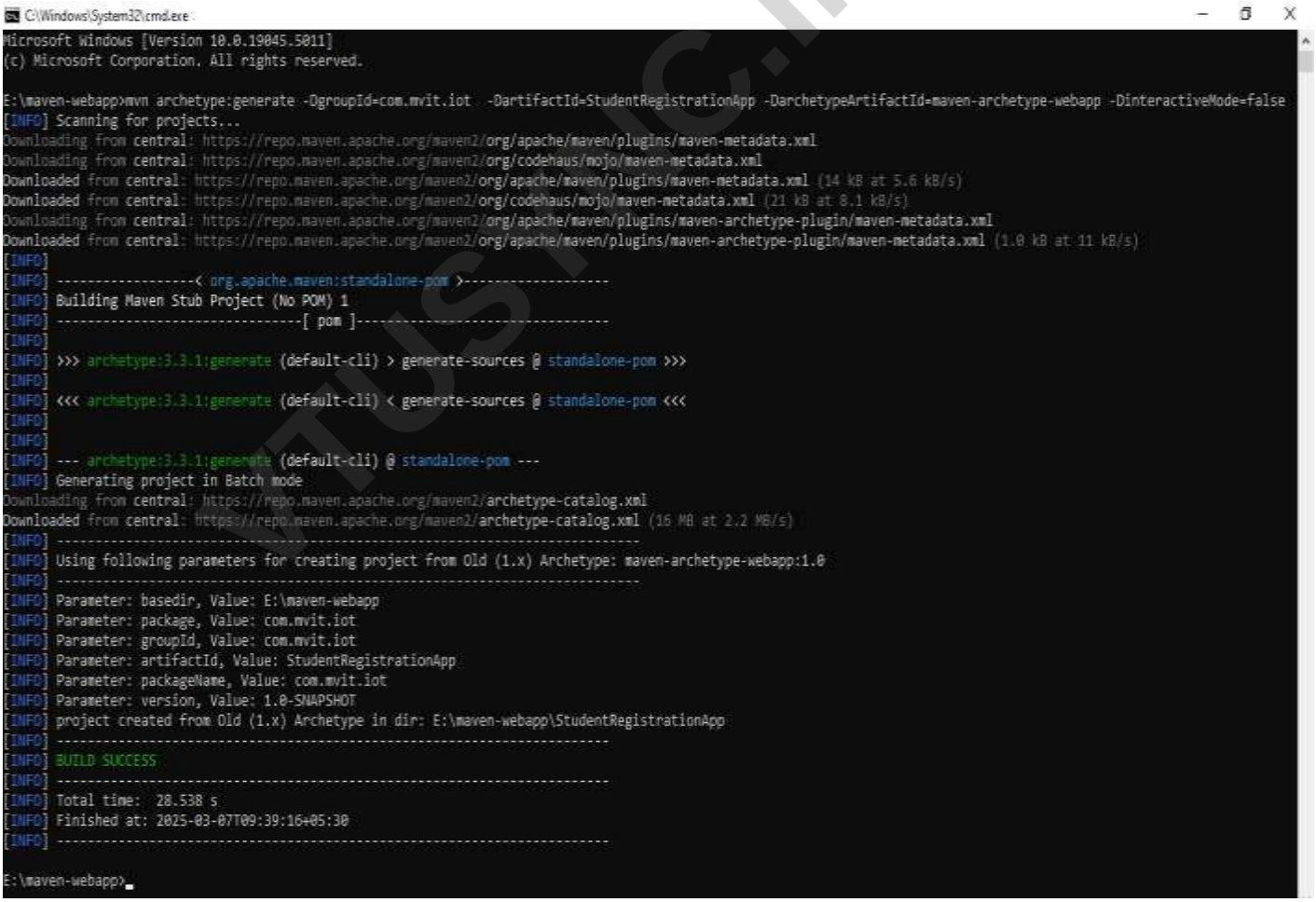
VTUSYNC.IN

Create a Maven Web Application

Step 1: Generate Maven Project

Use the Maven archetype to create a web project**:**

This will create a basic web application



TUSHIC.IN

Project Structure

The generated project will have the following structure:

StudentRegistrationApp

```
├── src
│   ├── main
│   │   ├── java
│   │   ├── resources
│   │   ├── webapp
│   │   │   ├── WEB-INF
│   │   │   │   ├── web.xml
│   │   │   ├── index.jsp
├── pom.xml
```

- **`src/main/java`** → Contains Java source files (Servlets, Controllers, etc.).
- **`src/main/webapp`** → Contains JSP files, HTML, CSS, and JS.
- **`WEB-INF`** → Holds configuration files like `web.xml`.
- **`pom.xml`** → Defines Maven dependencies and plugins.

VTUSYNC.IN

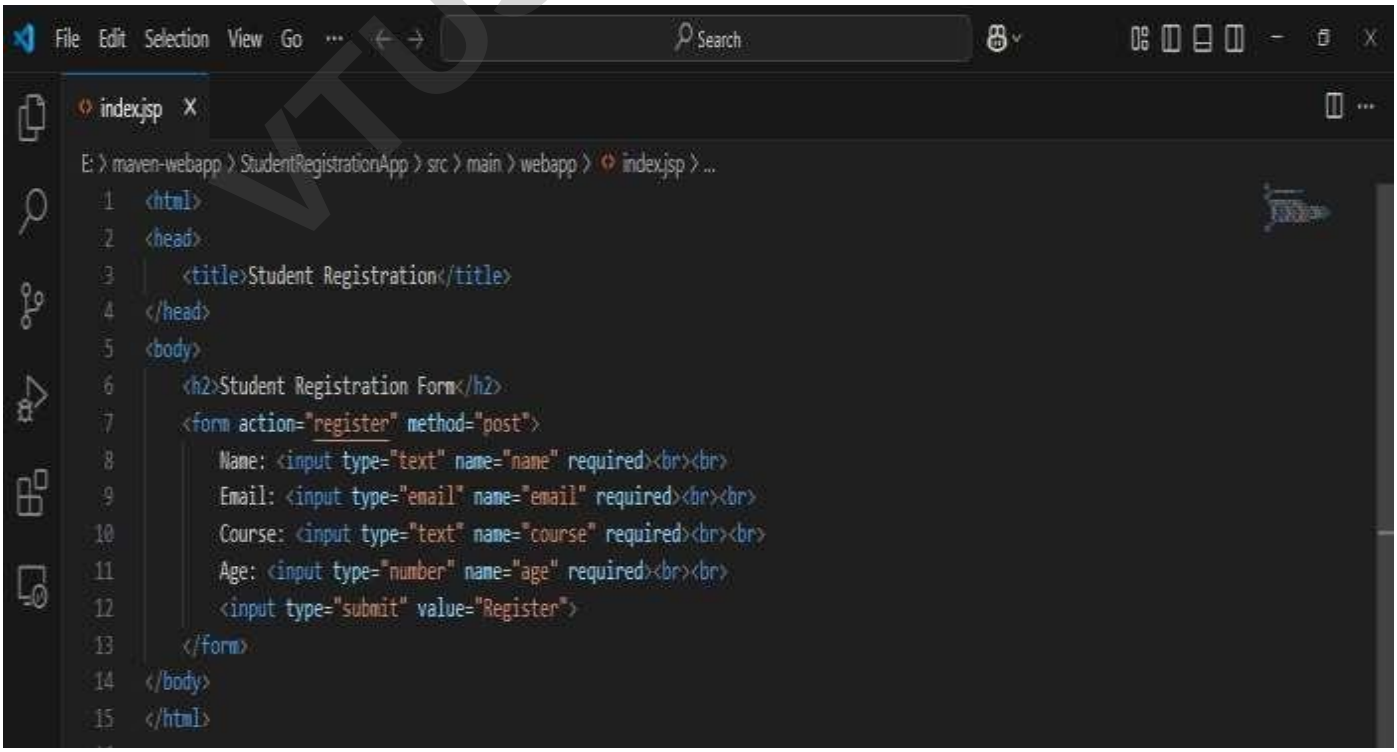
Step 2 : Configure pom.xml

Add Tomcat Plugin for Running the Application

```
    <plugins>
        <plugin>
            <groupId>org.apache.tomcat.maven</groupId>
            <artifactId>tomcat7-maven-plugin</artifactId>
            <version>2.2</version>
            <configuration>
                <path>/</path>
                <port>8080</port>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>
```

Step 3: Create index.jsp (Student Registration Form)

Create `index.jsp` inside `src/main/webapp/`:



VTUSYNC.IN

Step 4: Build and Run the Application

4.1 compile the project

C:\Windows\System32\cmd.exe

```
Microsoft Windows [Version 10.0.19045.5011]
(c) Microsoft Corporation. All rights reserved.

E:\maven-webapp\StudentRegistrationApp> mvn compile
```

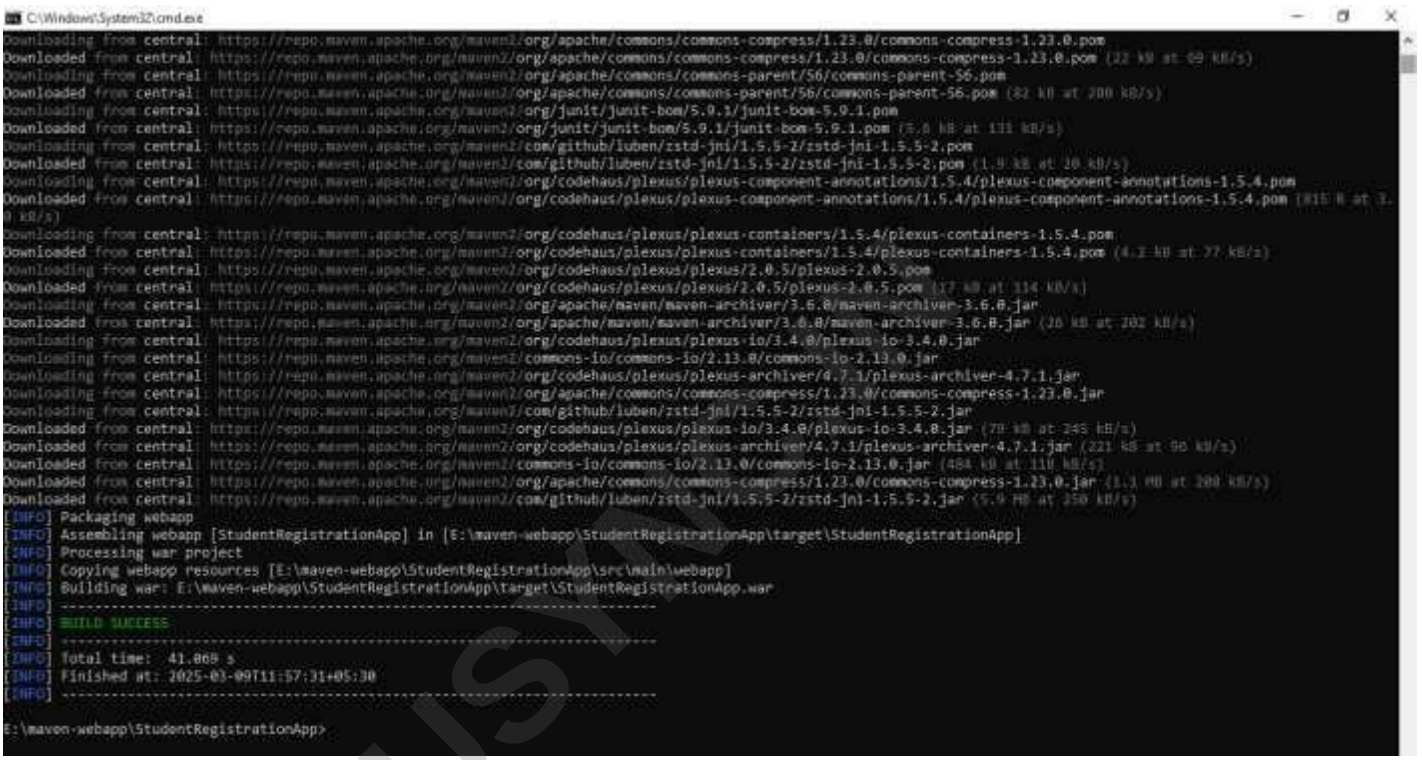
C:\Windows\System32\cmd.exe

```
Microsoft Windows [Version 10.0.19045.5011]
(c) Microsoft Corporation. All rights reserved.

E:\maven-webapp\StudentRegistrationApp> mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------< com.mvit.iot:StudentRegistrationApp >----------------
[INFO] Building StudentRegistrationApp Maven Webapp 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ war ]---------------------------------
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ StudentRegistrationApp ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource from src\main\resources to target\classes
[INFO]
[INFO] --- compiler:3.13.0:compile (default-compile) @ StudentRegistrationApp ---
[INFO] No sources to compile
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  5.409 s
[INFO] Finished at: 2025-03-09T11:52:35+05:30
[INFO] ------------------------------------------------------------------------

E:\maven-webapp\StudentRegistrationApp>
```

VTUSYNC.IN

4.2 package the project

```
E:\maven-webapp\StudentRegistrationApp>  mvn package
[INFO] Scanning for projects
```



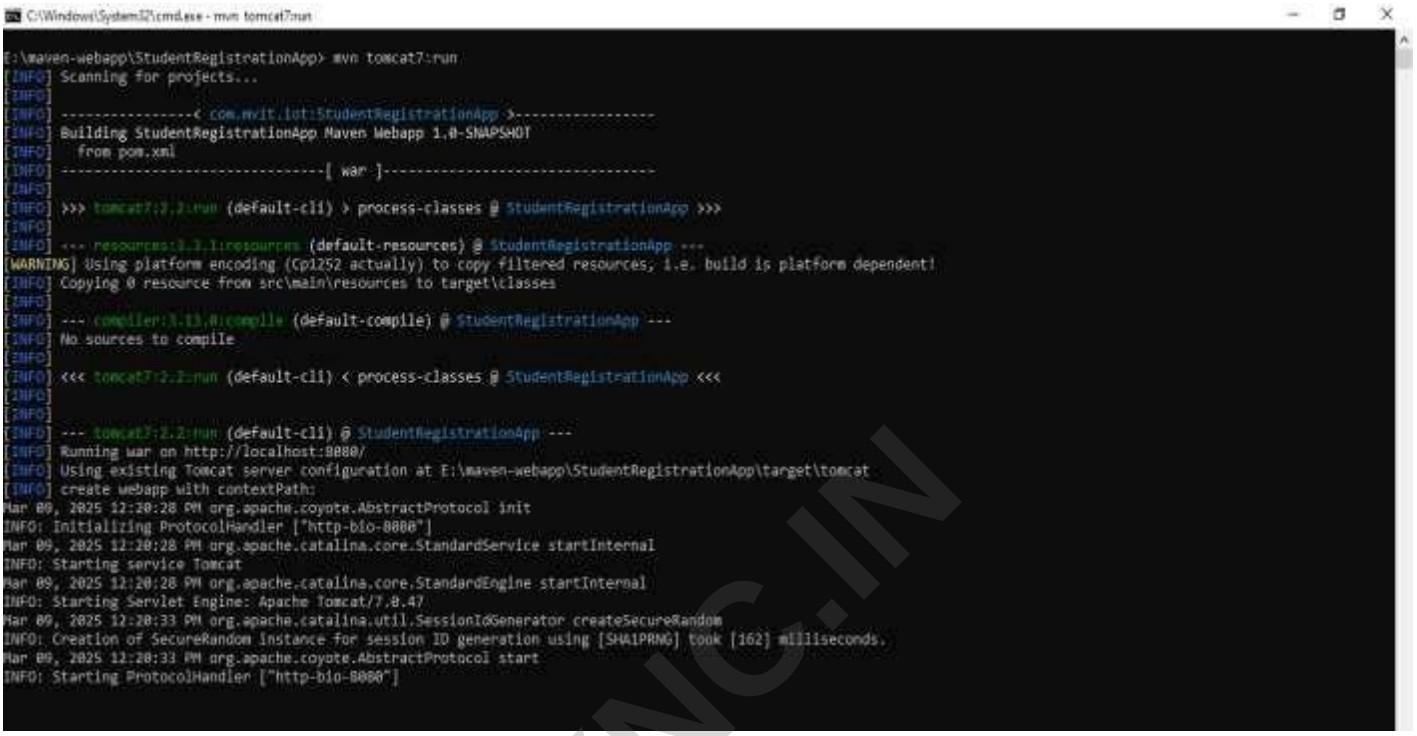
This will generate a WAR file in the target/ directory.

4.3 Deploy and Run on Embedded Tomcat

Run the following command:

C:\Windows\System32\cmd.exe - mvn tomcat7:run

```
E:\maven-webapp\StudentRegistrationApp> mvn tomcat7:run
```



VTUSYNC.IN

Step 5 : <u>Open a browser and visit</u>:

- **http://localhost:8080/index.jsp** → Student Registration Form

3. Working with Gradle: Setting Up a Gradle Project, Understanding Build Scripts (Groovy and Kotlin DSL), Dependency Management and Task Automation

Gradle Project Overview

1: Setting Up a Gradle Project

- **To create a new Gradle project using the command line:**

```
E:\Gradle-project1> gradle init
```

This command creates a new Java application project with a sample **build.gradle** file.

2. Understanding Build Scripts

Gradle uses a DSL (Domain-Specific Language) to define the build scripts. Gradle supports two DSLs:

- **Groovy DSL** (default)
- **Kotlin DSL** (alternative)

Groovy DSL: This is the default language used for Gradle build scripts (**`build.gradle`**).

Example of a simple **`build.gradle`** file (Groovy DSL):

```
plugins {

    id 'java'
}

repositories {
    mavenCentral()
}

dependencies {

        Implementation 'org.springframework.boot:spring-boot-starter-web:2.5.4'
}
```

VTUSYNC.IN

Kotlin DSL: Gradle also supports Kotlin for its build scripts (build.gradle.kts).

Example of a simple build.gradle.kts file (Kotlin DSL):

```
plugins {

    id java
}

repositories {
    mavenCentral()
}

dependencies {

        Implementation ("org.springframework.boot:spring-boot-starter-web:2.5.4")
}
```

Difference between Groovy and Kotlin DSL:

- **Syntax**: Groovy uses a more concise, dynamic syntax, while Kotlin offers a more structured, statically-typed approach.
- **Error handling**: Kotlin provides better error detection at compile time due to its static nature.

3: Dependency Management

Dependencies in Gradle are defined inside the dependencies block

- Open `build.gradle` or `build.gradle.kts`.
- Add a dependency (e.g., **JUnit** for testing)

```
dependencies {

        testImplementation 'junit:junit:4.13.2'
}
```

VTUSYNC.IN

4: Task Automation

Gradle tasks automate various tasks in your project lifecycle, like compiling code, running tests, and creating builds .

1. **Using predefined tasks**: Gradle provides many predefined tasks for common activities, such as:
   - **`build`** – compiles the project, runs tests, and creates the build output.
   - **`test`** – runs tests.
   - **`clean`** – deletes the build output.
2. Example of running the `build` task:

```
> gradle build
```

3. **Creating custom tasks**: You can define your own tasks to automate specific actions. For example, creating a custom task to print a message.

   - **Example Groovy DSL:**

```
task printMessage {
    doLast {
        println 'This is a custom task automation'
    }
}
```

   - **Example Kotlin DSL:**

```
tasks.register("printMessage") {
    doLast {
        println("This is a custom task automation")
    }
}
```

4. Running Gradle Tasks

- To run the `build` task: **`gradle build`**
- To run a custom task: **`gradle printMessage`**

VTUSYNC.IN

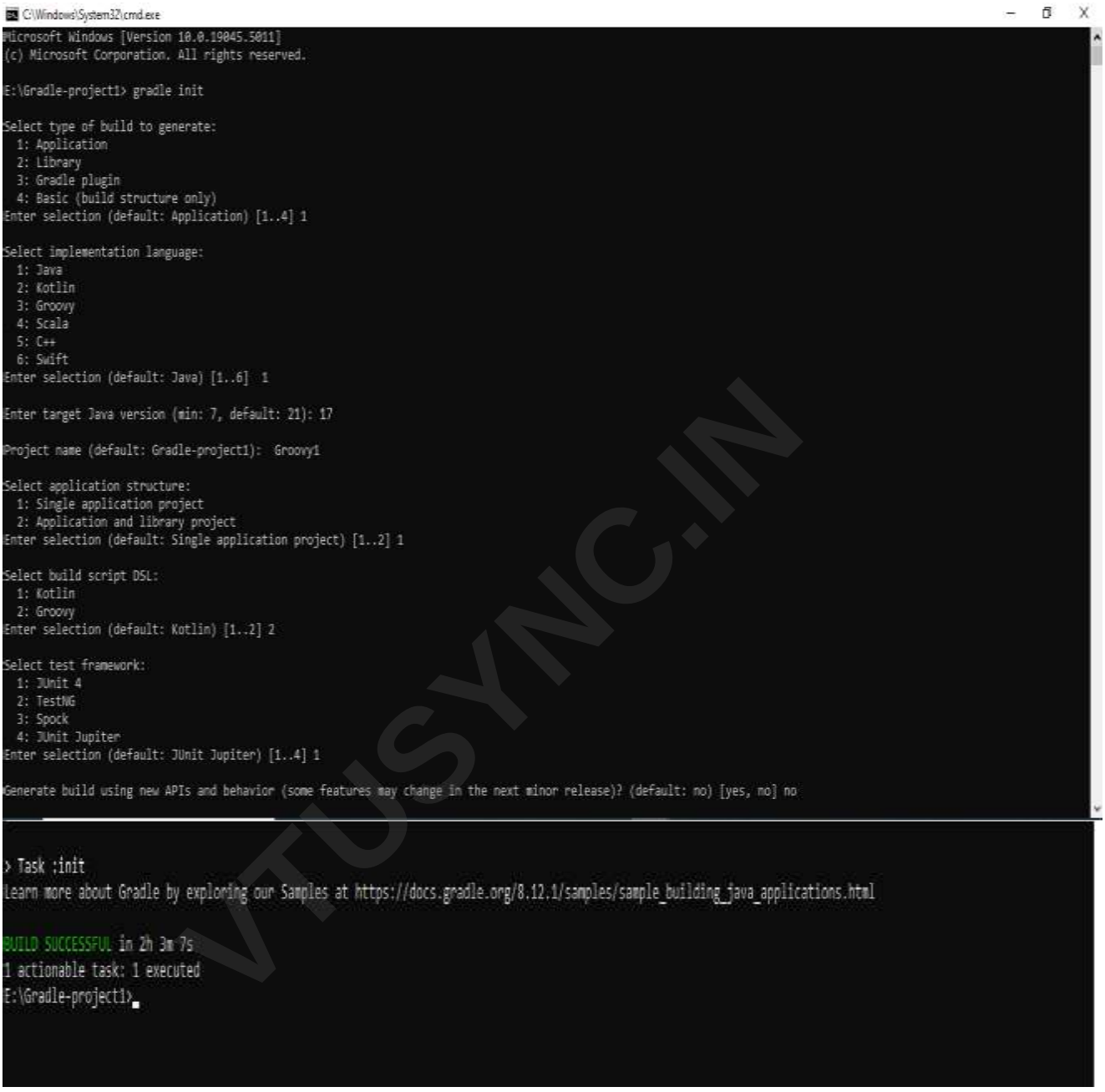
Working with Gradle Project (Groovy DSL):

Step 1: Create a new Project

```
E:\Gradle-project1> gradle init
```

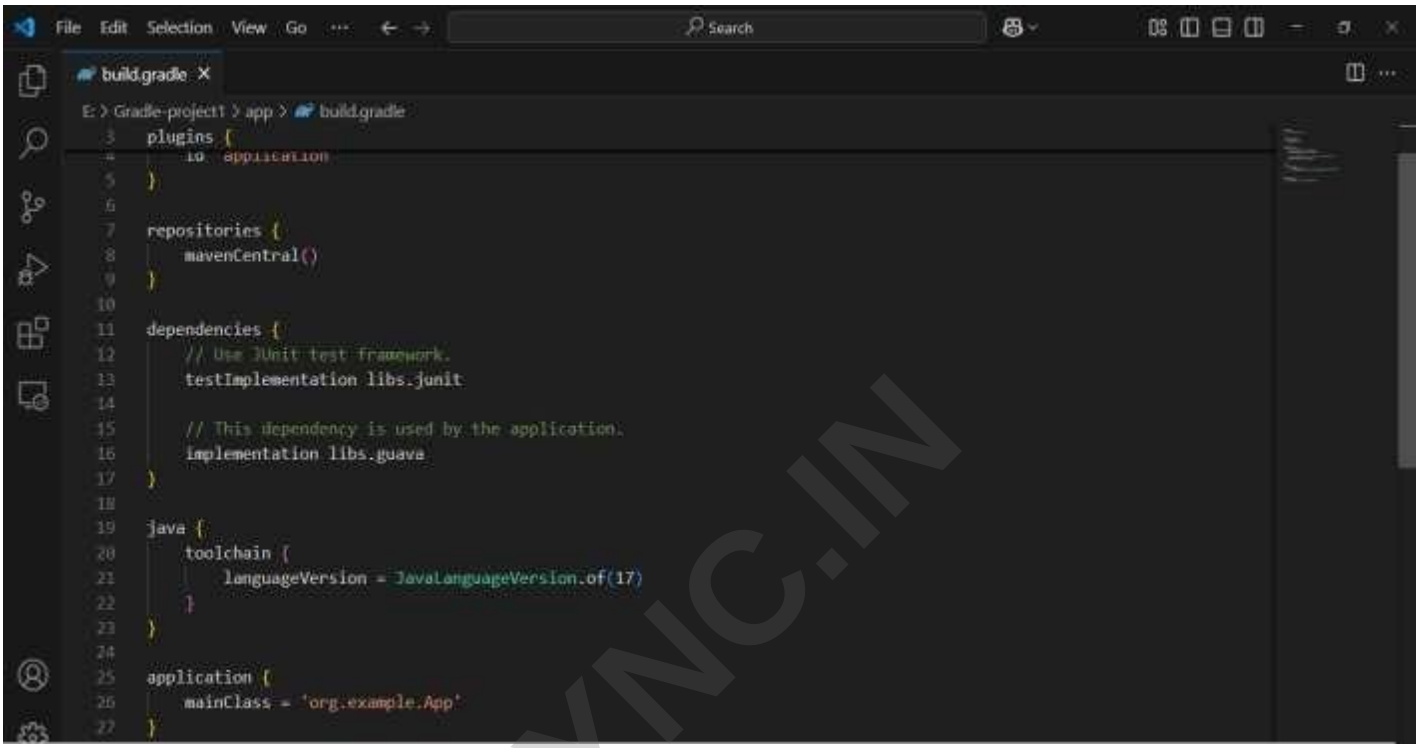
- while creating project it will ask necessary requirement:
  - **Enter target Java version (min: 7, default: 21):** 17
  - **Project name (default: program3-groovy):** groovyProject
  - **Select application structure:**
    - 1: Single application project
    - 2: Application and library project
      - **Enter selection (default: Single application project) [1..2]** 1
  - **Select build script DSL:**
    - 1: Kotlin
    - 2: Groovy
      - **Enter selection (default: Kotlin) [1..2]** 2
  - **Select test framework:**
    - 1: JUnit 4
    - 2: TestNG
    - 3: Spock
    - 4: JUnit Jupiter
      - **Enter selection (default: JUnit Jupiter) [1..4]** 1
  - **Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]**
    - no

VTUSYNC.IN


C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.5011]
(c) Microsoft Corporation. All rights reserved.
E:\Gradle-project1> gradle init
Select type of build to generate:
1: Application
2: Library
3: Gradle plugin
4: Basic (build structure only)
Enter selection (default: Application) [1..4] 1
Select implementation language:
1: Java
2: Kotlin
3: Groovy
4: Scala
5: C++
6: Swift
Enter selection (default: Java) [1..6] 1
Enter target Java version (min: 7, default: 21): 17
Project name (default: Gradle-project1): Groovy1
Select application structure:
1: Single application project
2: Application and library project
Enter selection (default: Single application project) [1..2] 1
Select build script DSL:
1: Kotlin
2: Groovy
Enter selection (default: Kotlin) [1..2] 2
Select test framework:
1: JUnit 4
2: TestNG
3: Spock
4: JUnit Jupiter
Enter selection (default: JUnit Jupiter) [1..4] 1
Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no] no
> Task :init
Learn more about Gradle by exploring our Samples at https://docs.gradle.org/8.12.1/samples/sample_building_java_applications.html
BUILD SUCCESSFUL in 2h 3m 7s
1 actionable task: 1 executed
E:\Gradle-project1>

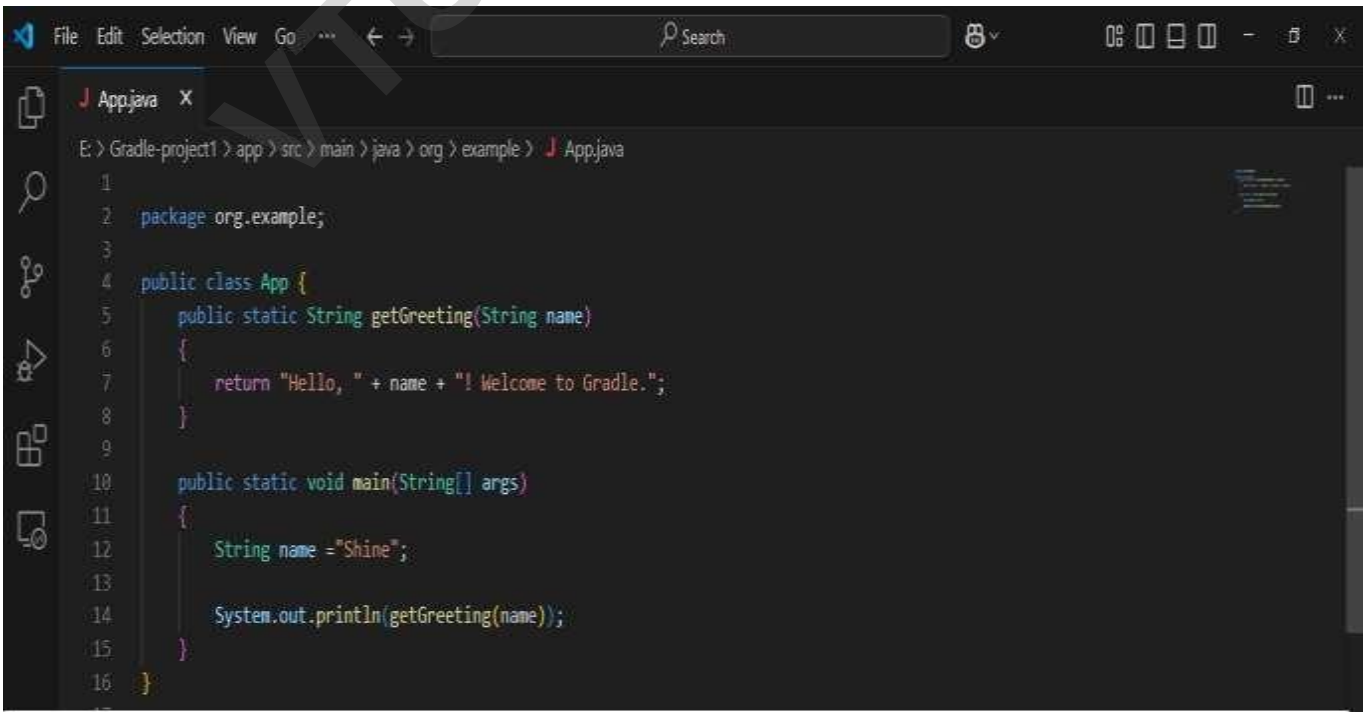
VTUSYNC.IN

Step 2: build.gradle (Groovy DSL)



Step 3 : Manually navigate the folder path like **src/main/java/org/example/**

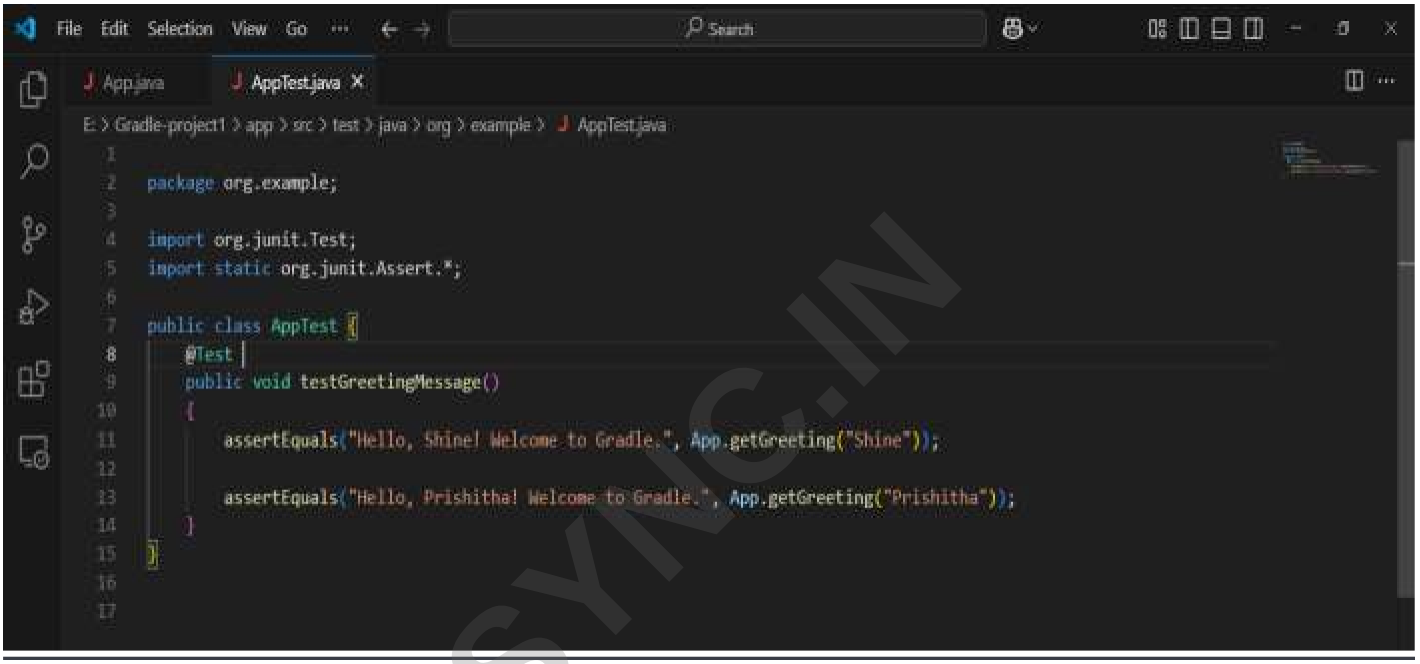
Write a simple program



VTUSYNC.IN

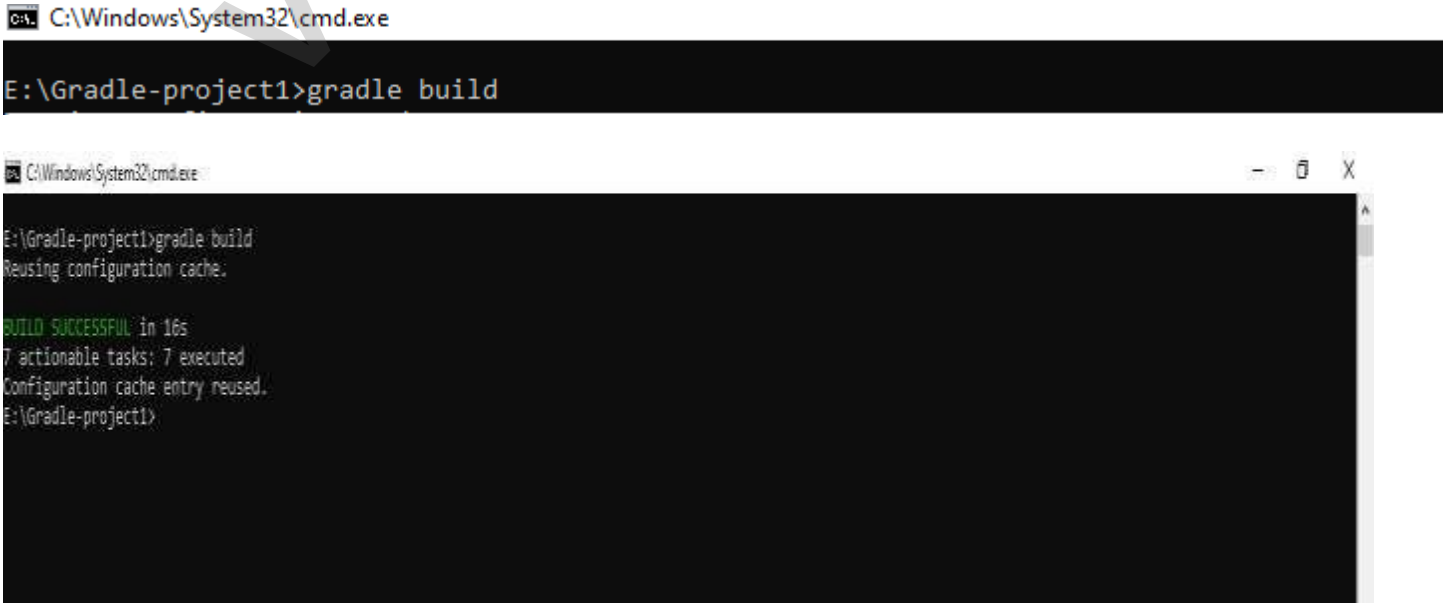
Step 4: AppTest.java (JUnitTest) (update the below code)

- Manually navigate the folder path like **src/test/java/org/example/**
- After then open that file and copy the below code and paste it, save it.



Step 5 : Run Gradle Commands

- To **build** the project:



VTUSYNC.IN

- To **run** the project:

C:\Windows\System32\cmd.exe

```
E:\Gradle-project1> gradle run
```

C:\Windows\System32\cmd.exe

```
E:\Gradle-project1> gradle run
Reusing configuration cache.

> Task :app:run
Hello, Shine! Welcome to Gradle.

BUILD SUCCESSFUL in 6s
2 actionable tasks: 2 executed
Configuration cache entry reused.
E:\Gradle-project1>
```

VTUSYNC.IN

- To **test** the project

C:\Windows\System32\cmd.exe

```
E:\Gradle-project1> gradle test
```

C:\Windows\System32\cmd.exe

```
E:\Gradle-project1> gradle test
Calculating task graph as no cached configuration is available for tasks: test

BUILD SUCCESSFUL in 13s
3 actionable tasks: 2 executed, 1 up-to-date
Configuration cache entry stored.
E:\Gradle-project1>
```

4. Practical Exercise: Build and Run a Java Application with Maven, Migrate the Same Application to Gradle

Step 1: Creating a Maven Project

You can create a **Maven project** using the `mvn` command (or through your **IDE**, as mentioned earlier).

- **Using Command Line:**
  - To create a basic Maven project using the command line, you can use the following command

```
E:\Maven to Gradle>mvn archetype:generate -DgroupId=com.example -DartifactId=maven-project -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

VTUSYNC.IN

Step 2: Open The `pom.xml` File

- You can manually navigate the **project folder** named call **maven-example** and open the file pom.xml
- In case if you not getting project folder then type command in your cmd.
  - **cd maven-example** – is use to navigate the project folder.
  - **notepad pom.xml** – is use to open pom file in notepad.

Step 3: Open Java Code (App.java) File

- Open a file **`App.java`** inside the **`src/main/java/com/example/`** directory.
- After opening the **App.java** write a simple program

VTUSYNC.IN

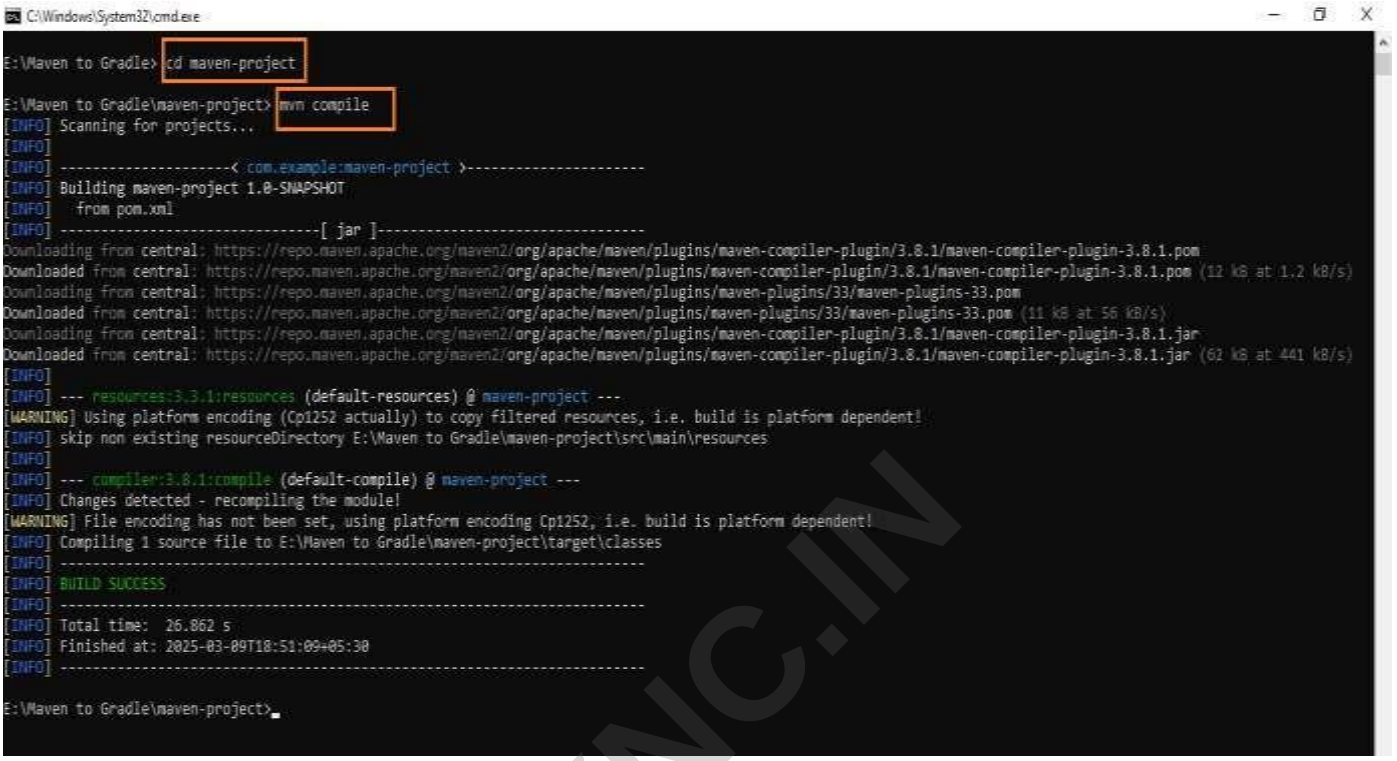
Note: before building the project make sure you are in the project folder if not navigate the project folder type command in your command prompt **cd maven-example**.

Step 4: Run the Project

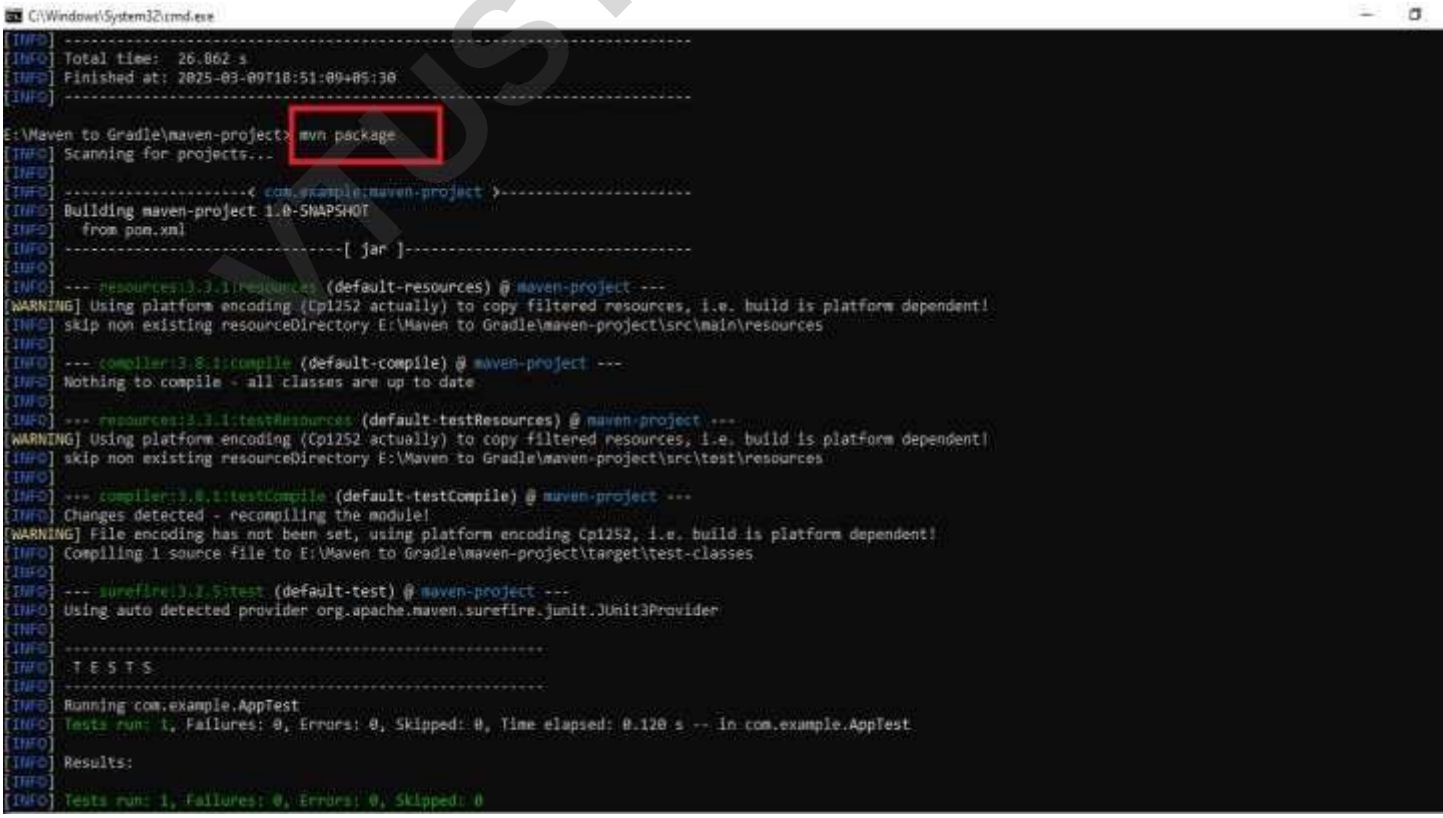
To build and run this project, follow these steps:

- Open the terminal in the project directory and run the following command to build the project.

```
mvn compile
```

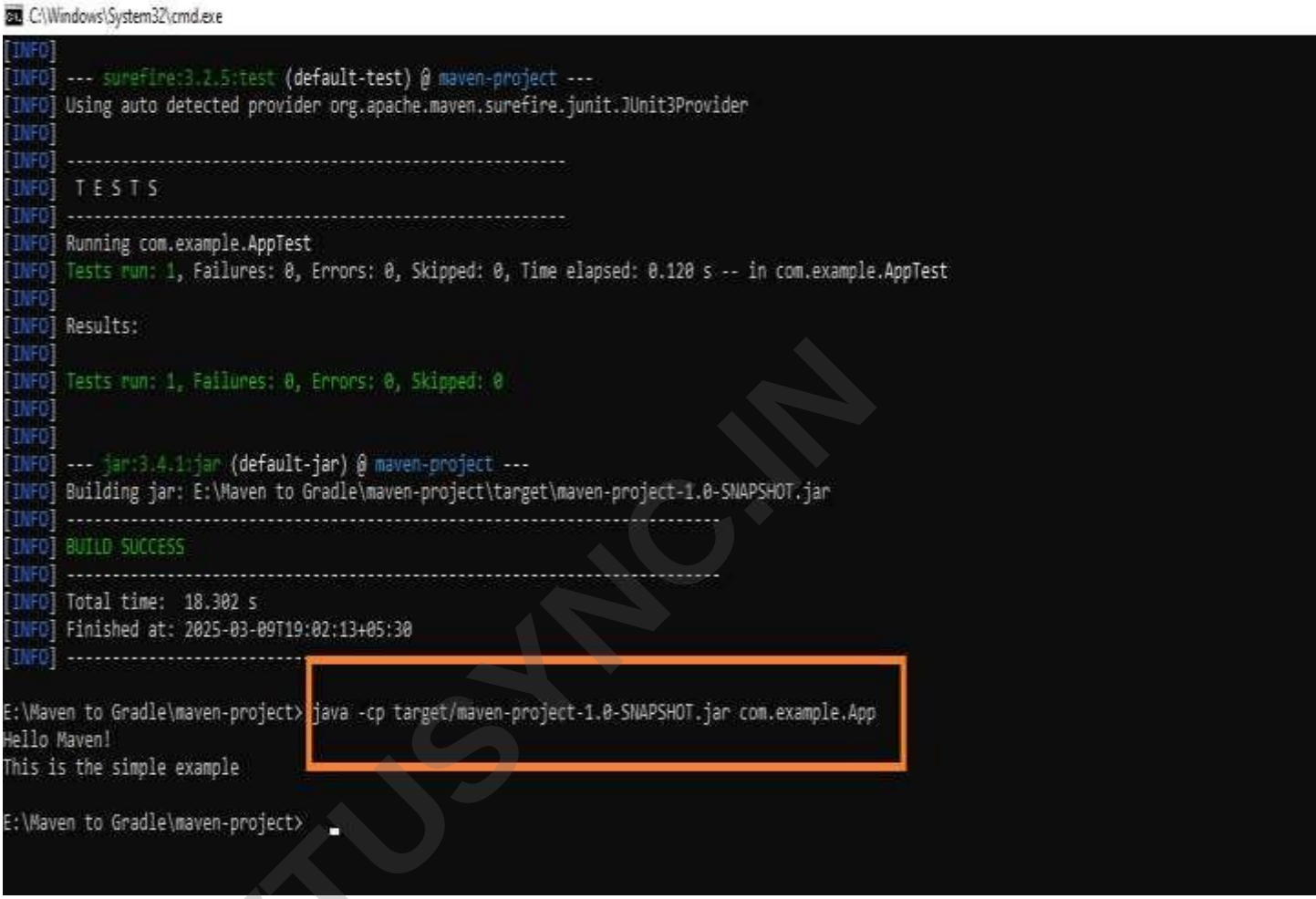


- **package the project**



VTUSYNC.IN

- Run the program with below command:



VTUSYNC.IN

Step 5: Migrate the Maven Project to Gradle

1. **Initialize Gradle**

```
E:\Maven to Gradle\maven-project> gradle init
```

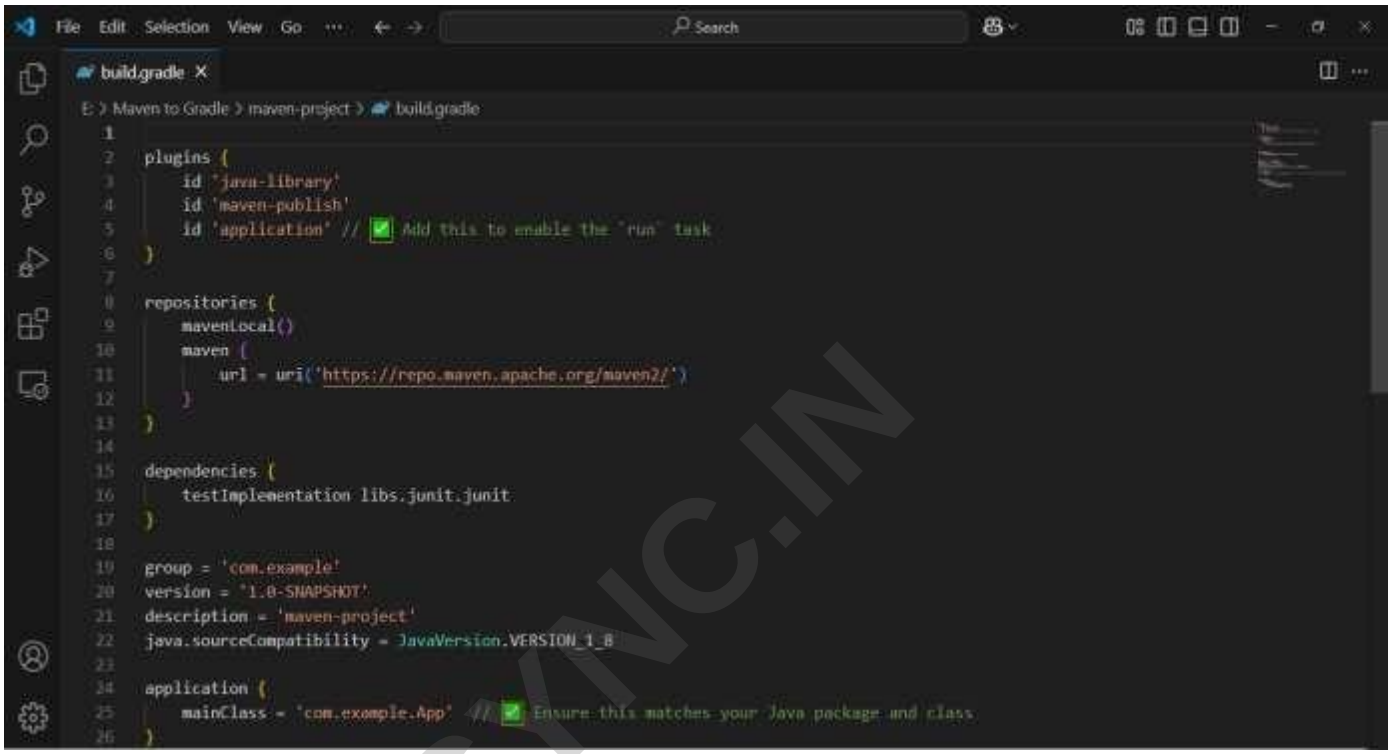
- It will ask **Found a Maven build. Generate a Gradle build from this? (default: yes) [yes, no]**
  - Type **Yes**
- **Select build script DSL:**
  - 1: Kotlin
  - 2: Groovy
  - Enter selection (default: Kotlin) [1..2]
    - Type **2**
- **Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]**
  - Type **No**

VTUSYNC.IN

C:\Windows\System32\cmd.exe

```
E:\Maven to Gradle\maven-project> gradle init

Found a Maven build. Generate a Gradle build from this? (default: yes) [yes, no]    yes

Select build script DSL:
  1: Kotlin
  2: Groovy
Enter selection (default: Kotlin) [1..2]  2

Generate build using new APIs and behavior (some features may change in the next minor rel


> Task :init
Maven to Gradle conversion is an incubating feature.
For more information, please refer to https://docs.gradle.org/8.12.1/userguide/migrating_f

BUILD SUCCESSFUL in 4m 55s
1 actionable task: 1 executed
E:\Maven to Gradle\maven-project>
```

2. **Navigate the project folder and open `build.gradle` file then the do the necessary changes and save it.**



Step 6: Run the Gradle Project

- **Build the Project**: run the below command to build the project.

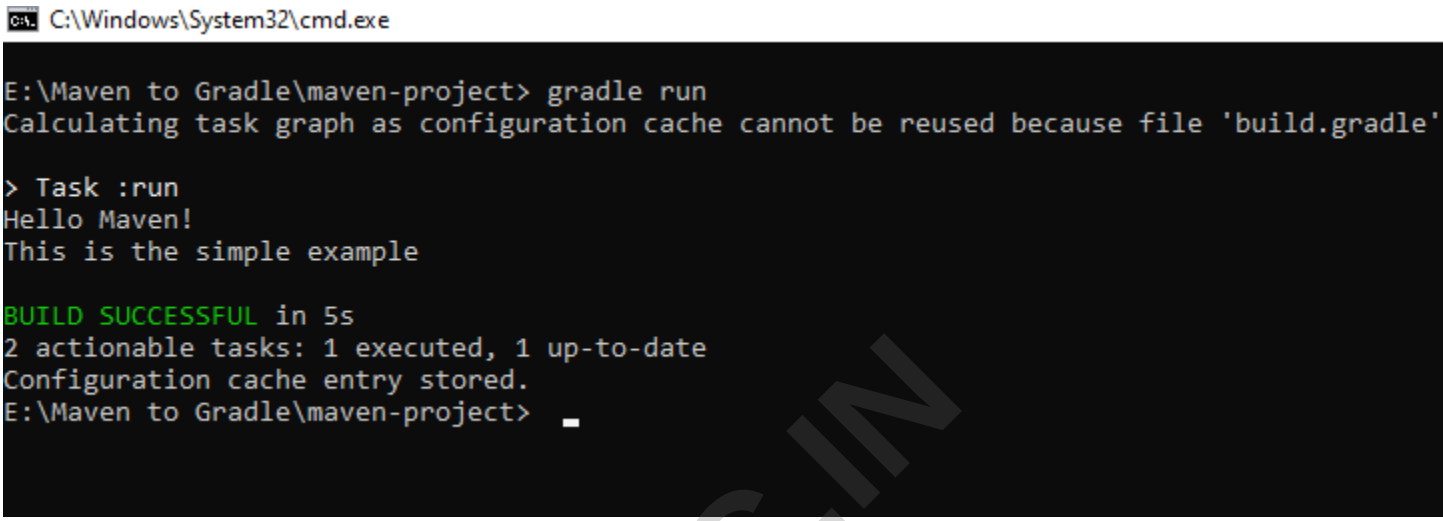
```
>gradle build
```

C:\Windows\System32\cmd.exe

```
E:\Maven to Gradle\maven-project>gradle build
Calculating task graph as configuration cache cannot be reused because file 'build.gradle'

BUILD SUCCESSFUL in 19s
7 actionable tasks: 3 executed, 4 up-to-date
Configuration cache entry stored.
E:\Maven to Gradle\maven-project>
```

VTUSYNC.IN

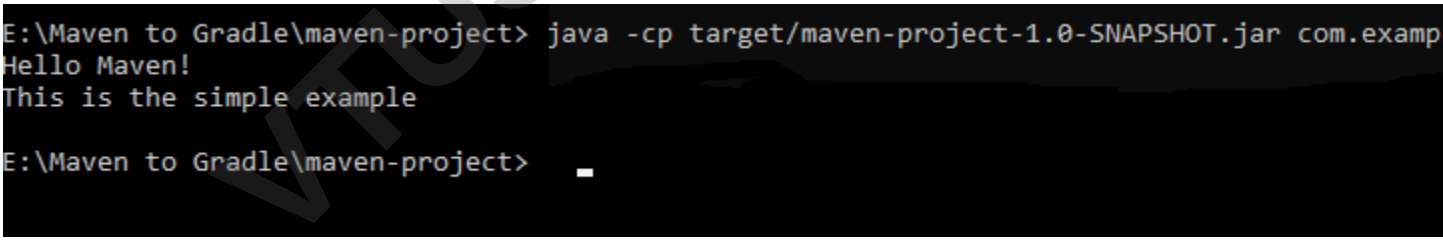
- **Run the Application**: Once the build is successful, run the application using below command



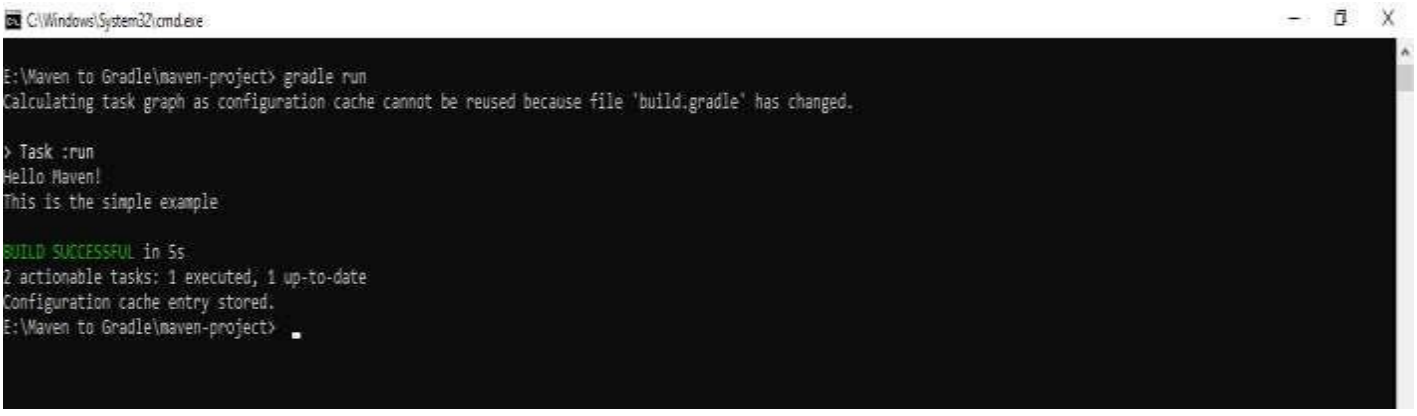
Step 7: Verify the Migration

Compare the Output: Make sure that both the **Maven** and **Gradle** builds produce the same output

- **Maven Output**:



- **Gradle Output**:



VTUSYNC.IN

VTUSYNC.IN