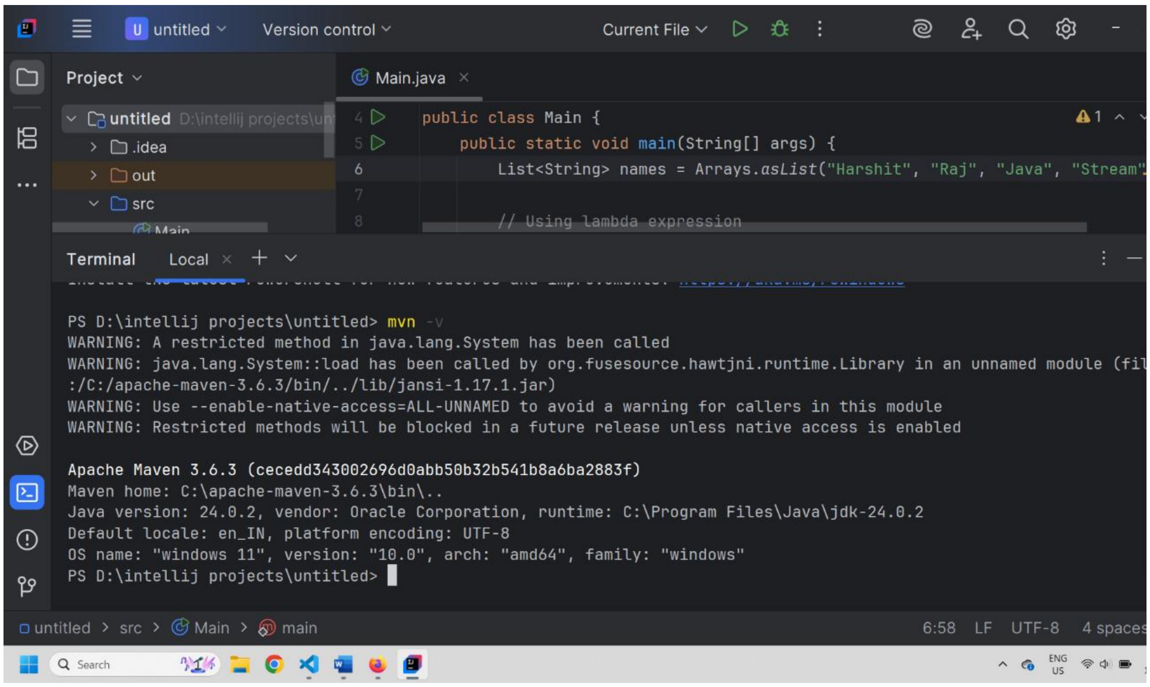
Week 2: Assignment 1 - Maven, Spring and Spring Boot

1. **Install maven 3.6 or above. Execute mvn -v in the local terminal/command prompt and share the screenshot.**

Ans.)



```java
public class Main {
    public static void main(String[] args) {
        List<String> names = Arrays.asList("Harshit", "Raj", "Java", "Stream"

        // Using lambda expression
```

```
PS D:\intellij projects\untitled> mvn -v
WARNING: A restricted method in java.lang.System has been called
WARNING: java.lang.System::load has been called by org.fusesource.hawtjni.runtime.Library in an unnamed module (fil
:/C:/apache-maven-3.6.3/bin/../lib/jansi-1.17.1.jar)
WARNING: Use --enable-native-access=ALL-UNNAMED to avoid a warning for callers in this module
WARNING: Restricted methods will be blocked in a future release unless native access is enabled

Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)
Maven home: C:\apache-maven-3.6.3\bin\..
Java version: 24.0.2, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-24.0.2
Default locale: en_IN, platform encoding: UTF-8
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"
PS D:\intellij projects\untitled>
```

2. **What is the difference between maven central repository and local repository?**

Ans.) The Maven Central Repository is a large online library of all the commonly used Java libraries, plugins, and dependencies. It is hosted on the internet at https://repo.maven.apache.org/maven2. When we build a project, Maven automatically downloads any required dependencies from the Central Repository if they are not available on your computer.

The Maven Local Repository is a folder on your own computer where Maven stores all the downloaded dependencies from the Central Repository. The default location of the local repository is .m2/repository in your user directory. Before downloading from the internet, Maven first checks this local repository. If the dependency is already there, it uses it directly without downloading again.

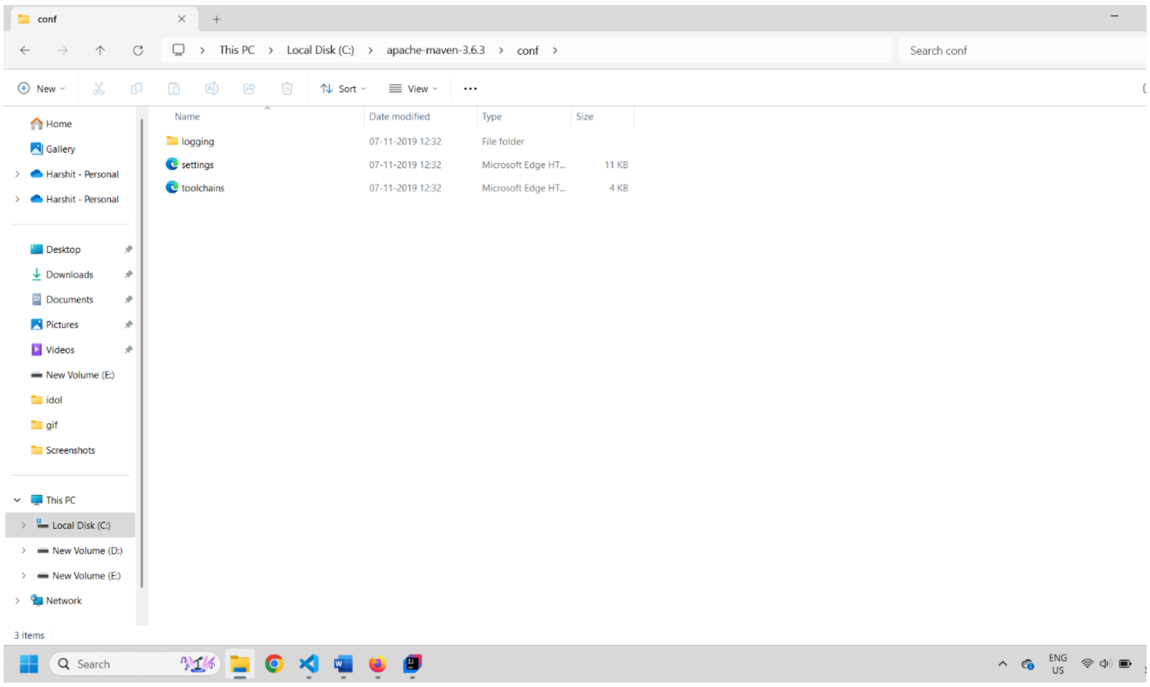
3. **Maven commands**

a. **To build the maven project - mvn clean install**

b. **To run the maven tests - mvn test**

4. Please locate the maven settings.xml file and local maven repository in your machine and share the screenshot

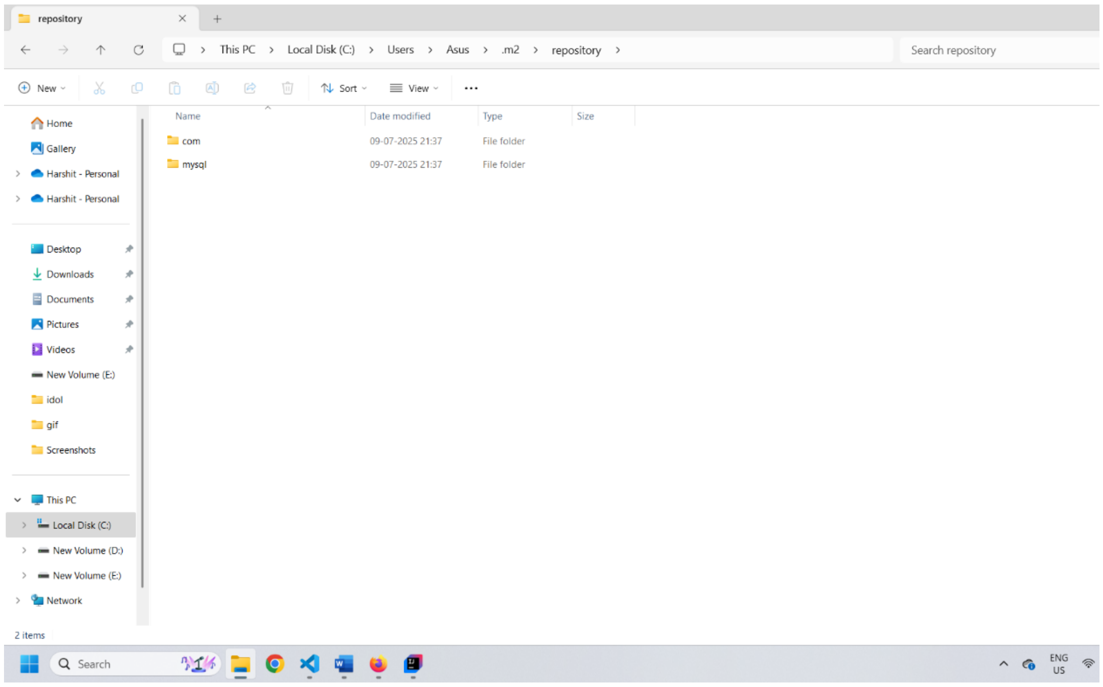
Ans.) settings.xml file location


This PC › Local Disk (C:) › apache-maven-3.6.3 › conf ›

Name	Date modified	Type	Size
logging	07-11-2019 12:32	File folder	
settings	07-11-2019 12:32	Microsoft Edge HT...	11 KB
toolchains	07-11-2019 12:32	Microsoft Edge HT...	4 KB

3 items

local maven repository location


This PC › Local Disk (C:) › Users › Asus › .m2 › repository ›

Name	Date modified	Type	Size
com	09-07-2025 21:37	File folder	
mysql	09-07-2025 21:37	File folder	

2 items

5. **The basic principle behind Dependency Injection (DI) is that the objects define their dependencies. What are the different ways in which an object can define its dependency?**

Ans.) Dependency Injection is a design pattern where the dependencies (objects a class needs) are provided from outside rather than the class creating them itself. This makes the code more flexible, testable, and easier to maintain.

There are three main types of Dependency Injection:

Constructor Injection - Dependencies are provided through the class constructor.

Example:

```
class Car {
   private Engine engine;
   Car(Engine engine) { // dependency injected via constructor
      this.engine = engine;
   }
}
```

Setter Injection - Dependencies are provided using public setter methods after the object is created.

Example:

```
class Car {
   private Engine engine;
   public void setEngine(Engine engine) { // dependency injected via setter
      this.engine = engine;
   }
}
```

Interface Injection - The dependency provides an injector method that will inject the dependency into any client passed to it.

Example:

```
interface EngineSetter {
   void setEngine(Engine engine);
}
```

```
class Car implements EngineSetter {
    private Engine engine;
    public void setEngine(Engine engine) {
        this.engine = engine;
    }
}
```

6. **What is the difference between the @Autowired and @Inject annotation?**

Ans.)

@Autowired	@Inject
Comes from Spring Framework (org.springframework.beans.factory.annotation.Autowired).	Comes from Javax (JSR-330) (javax.inject.Inject).
Spring-specific annotation with extra features.	Standard Java annotation (works in multiple frameworks).
By default, it **requires** the dependency (can set required=false for optional).	No required attribute; optional handled with @Nullable or Optional.
Supports @Qualifier for selecting specific beans.	Does not support @Qualifier directly; uses standard annotations.
Used only in Spring applications.	Can be used in any JSR-330 supported framework (Spring, Java EE, etc.).

7. **Explain the use of @Respository, @Component, @Service and @Controller annotations with an example for each.**

Ans.)

1. @Component

- **What it is:**
  A generic stereotype annotation for any Spring-managed component.
- **Use case:**
  Marks a class as a Spring bean (general-purpose).
- **Example:**

```
import org.springframework.stereotype.Component;

@Component
```

```
public class MyBean {
    public void display() {
        System.out.println("Hello from @Component bean!");
    }
}
```

2. @Repository - Specialized version of @Component for **DAO (Data Access Object)** classes.

- **Use case:** Marks a class that interacts with the database. It also provides automatic exception translation for persistence-related exceptions.
- **Example:**

```
import org.springframework.stereotype.Repository;
@Repository
public class UserRepository {
    public void saveUser() {
        System.out.println("User saved to database!");
    }
}
```

3. @Service - Specialized version of @Component for **service layer** classes.

- **Use case:** Marks a class that holds business logic.
- **Example:**

```
import org.springframework.stereotype.Service;
@Service
public class UserService {
    public void processUser() {
        System.out.println("Processing user in service layer!");
    }
}
```

4. @Controller - Specialized version of @Component for **web controllers** in Spring MVC.

- **Use case:** Marks a class as a controller that handles HTTP requests.
- **Example:**

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
@Controller
public class HomeController {
  @GetMapping("/")
  public String home() {
    return "home"; // returns view name
  }
}
```

8. **Fix the code and explain why?**

The following code tries to inject a property from application.properties, but the appName field is always null. Identify and fix the issue.

```
@Component
public class AppNamePrinter {
        @Value("app.name")
    private String appName;


        public void printAppName() {
        System.out.println("Application Name: " + appName);
        }
}
```

Ans.)

The issue is with this line: @Value("app.name")

The, "app.name" is treated as a **hardcoded String**, not as a property key from application.properties.

That's why appName is always null.

Fix : Use **${}** to tell Spring to fetch the value from application.properties:

```
@Component
public class AppNamePrinter {
    @Value("${app.name}") // Fetches property value correctly
    private String appName;

    public void printAppName() {
        System.out.println("Application Name: " + appName);
    }
}
```

9. What does the @SpringBootApplication annotation do?

Ans.) The SpringBootApplication annotation is a meta-annotation in Spring Boot that combines Configuration, EnableAutoConfiguration, and ComponentScan. It simplifies application setup by configuring Spring Beans, enabling auto-configuration, and scanning for components in the specified package.

Uses :

- It simplifies Spring Boot configuration.
- Instead of writing all three annotations separately, you just use @SpringBootApplication.
- @SpringBootApplication makes it easy to start a Spring Boot application by enabling **configuration**, **auto-configuration**, and **component scanning** in a single step.

10. What is the maven command to start the SpringBootApplication?

Ans.) mvn spring-boot:run

11. Implement EmployeeCRUD using Spring and JDBC with the below Employee class. In the branch feature-spring, create a folder Employee-Spring. Push the solution to the branch and share the link.

```
class Employee{

        private int id;

        private String name;

        private String department;

}
```

Ans.) GitHub link - https://github.com/Harshit-Raj-14/PayPal-RG-Assignment-Harshit-Raj/tree/main/Week%202/Assignment%201/Maven%2C%20Spring%20and%20Spring%20Boot/Employee-Spring

12. Implement EmployeeCRUD using SpringBoot and Spring Data JPA with the below Employee class. In the branch feature-spring, create a folder Employee-SpringBoot-JPA. Push the solution to the branch and share the link.

```
class Employee{

        private int id;

        private String name;

        private String department;

}
```

Ans.) GitHub link – https://github.com/Harshit-Raj-14/PayPal-RG-Assignment-Harshit-Raj/tree/main/Week%202/Assignment%201/Maven%2C%20Spring%20and%20Spring%20Boot/Employee-SpringBoot-JPA

**13.	Follow the demo in the pre-work link https://www.youtube.com/watch?v=hr2XTbKSdAQ&t=18s and create a Spring Batch application that processes customer data. In the branch feature-spring, create a folder Customer-SpringBatch. Push the solution to the branch and share the link.**

Ans.) GitHub link - https://github.com/Harshit-Raj-14/PayPal-RG-Assignment-Harshit-Raj/tree/main/Week%202/Assignment%201/Maven%2C%20Spring%20and%20Spring%20Boot/Customer-SpringBatch