D D M M Y Y Y Y

MANUAL APPROACH TO PROJECT SETUP

AIM: To achieve setup & testing a project manually using Selenium Automation Test Project.

SOFTWARE REQUIREMENTS:
Selenium JAR
Intellij IDEA
Programming language: Java

PROCEDURE:
Manual setup process
i) Create a simple Java Project in IntellijIDEA
ii) Manually Download Selenium JAR from: https://www.selenium.dev
iii) Create a lib folder & place selenium-server-4.23.1.jar
iv) Manually add the Dependency in IntellijIDEA
File → Project Structure → Libraries → Add SeleniumJAR → Apply → OK
v) Write selenium test in Java (LoginTest.java) to validate login the website & run test manually.

D D M M Y Y Y Y

STEPS IN CREATING THE PROGRAM AND FLOW

i) Importing Required Selenium Libraries
ii) Main method execution
iii) Lauching Chrome Browser
iv) Navigating to Website
v) Entering Username & Password
vi) Clicking Login Button
vii) Closing the Browser

PROGRAM:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class LoginTest{
    public static void main(String[] args) throws
                        InterruptedException
    {
        WebDriver driver = new ChromeDriver();
        driver.get("https://www.saucedemo.com/");
        driver.manage().window().maximize();
        Thread.sleep(2000);
        driver.findElement(By.id("user-name")).send
                            Keys("standard_user");
        Thread.sleep(2000);
        driver.findElement(By.id("password")).send
                            ("secret_sauce");
        Thread.sleep(2000);
        driver.findElement(By.id("login-button")).click
        Thread.sleep(2000);
        driver.quit();
    }
}
```

D D M M Y Y Y Y

Summary of the Automation Process:-

- Opens Google Chrome
- Navigates to SauceDemo Login Page
- Enters Username & Password
- Clicks login button
- closes the browser.

Observations & Analysis:-

- Time consuming:- Downloading & adding the dependencies manually is inefficient
- Error-prone:- Missing JAR files or incorrect configurations manually is inefficient.
- Difficult-to manage:- Dependencies are not automatically updated.
- Not Scalable:- Every team members must manually configure the setup.

Result:- Using Selenium a sample project setup is done & tested where it logs in using username & password for a sample website.

D D M M Y Y Y Y

Experiment-1

Title: Introduction to Maven & Gradle Build Tools

Aim: To understand build automation tools, compare Maven & Gradle, & set up both tools for software development.

Software Requirement: IntelliJ, Maven, Gradle

1. **Overview of Build Automation Tools:**
They simplify the process of compiling, testing, packaging & developing software projects. They manage dependencies, execute tasks, and integrate seamlessly with CI/CD pipelines

Use:
- Ensures consistency across builds.
- Handles dependency management automatically.
- Reduces manual errors & increases efficiency.

Popular Build Automation Tools

- Apache Ant – Script based, manual dependency management.
- Apache Maven – XML based, convention-over-configuration model.
- Gradle – Flexible, fast & supports both Groovy/Kotlin DSL.

D D M M Y Y Y Y

Key Differences b/w Maven & Gradle

Feature	Maven (XML)	Gradle (Groovy / Kotlin)
Build Script	pom.xml	build.gradle / build.gradle.kts
Performance	Sequential, slower	Parallel execution, faster
Flexibility	Convention-based	Highly customizable
Dependency Management	Uses maven repository	Supports multiple repositories
Ease of use	Simple XML structure	Slightly complex but powerful
Caching support	No build caching	Supports the incremental builds.
Best for	Standard Java projects.	Complex, high-performance builds.

D D M M Y Y Y Y

Installation & setup:

Installing Maven:

Step 1: Install Java (JDK 17 Recommended)

Step 2: Download & Install Maven

Download it from: https://maven.apache.org

(or)

In the terminal:

sudo apt install maven

Step 3: Configure Environment Variables.

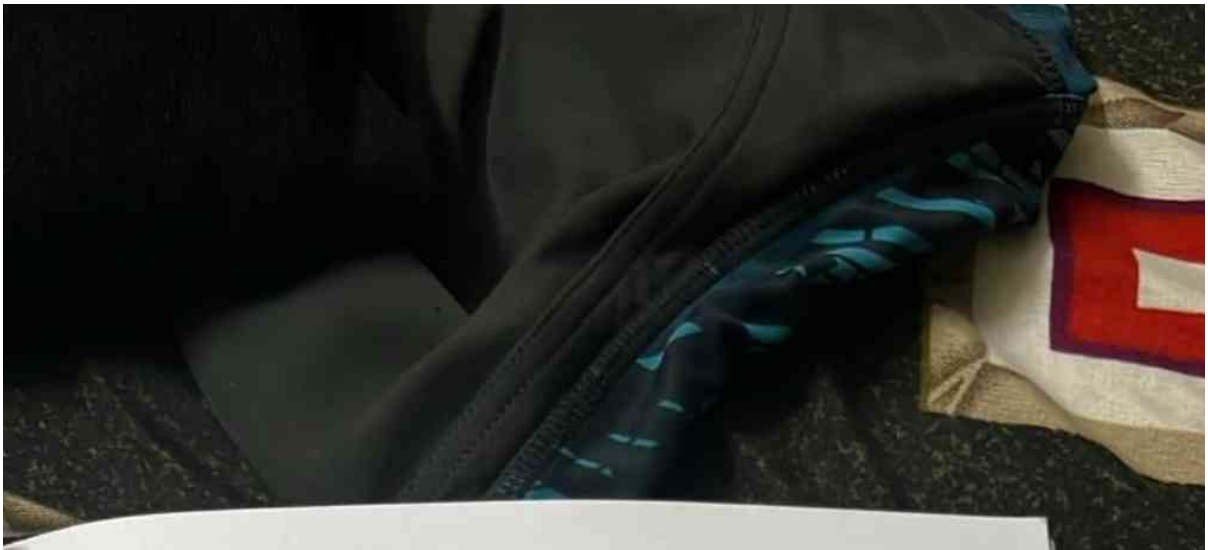
Add following to .bashrc or .zshrc:

export MAVEN_HOME = /opt/maven/apache-maven-<version>

export PATH = $MAVEN_HOME/bin:$PATH

STEP 4: Verify Installation

mvn -version.



D D M M Y Y Y Y

- Installing Gradle:-

STEP 1: Install Java (JDK 17 Recommended)

STEP 2: Download & Install Gradle

From Website https://gradle.org (or)

From terminal: sudo apt install gradle.

STEP 3: Configure Environment Variables

Add the following to .bashrc or .zshrc

export GRADLE_HOME=/opt/gradle/gradle-<version>
export PATH=$GRADLE_HOME/bin:$PATH

STEP 4: Verify installation
gradle -v

Result:
Maven & Gradle were succesfully set-up & used for build automation. Gradle showed better performance, while Maven offered simpler, standarized configuration.

D D M M Y Y Y Y

Experiment-2

Title: Working with Maven Build Automation Tool.

Aim: Creating a Maven Project understanding the POM file, Dependency Management & Plugins.

Software Requirement: IntelliJ, Maven

Procedure:
steps to creak Maven Project in IntelliJ IDEA

1. Install Maven (if not already installed)
2. Create New Maven Project.
   * Open IntelliJ IDEA
   * Go to File > New > Project
   * Select Maven from Project Types.
   * Set project name & location, then click Finish.
   * ~~steh in project~~
3. Set up the pom.xml file:
   * pom.xml file is where you define dependencies, plugins, & other configurations for your maven project.
   * Add dependencies for Selenium & TestNG

D D M M Y Y Y Y

```
<project xmlns="http://maven.apache.org/POM/
4.0.0"

  xmlns:xsi="http://www.w3.org/2007/
  XMLScheme-Instance"
  xsi:schemaLocation="http://maven.apache.org
  /POM/4.0.0 http://maven.
  apache.org/xsd/maven.
  4.0.0 "xsd" >
  <modelVersion> 4.0.0 </modelversion>
  <groupId> com.example </groupId>
  <artifactId> simple-project </artifactId>
  <version> 1.0-SNAPSHOT </version>
  <dependencies>
    <dependency>
      <groupId> org.seleniumhq.selenium
      </groupId>
      <artifactId> selenium-java </artifactId>
      <version> 4.8.0 </version>
    </dependency>
    <dependency>
      <groupId> org.testing </groupId>
      <artifactId> testng </artifact>
      <version> 7.7.0 </version>
      <scope> test </scope>
    </dependency>
  </dependencies>
</project>
```

Add dependencies for selenium & TestNG:
In pom.xml, add Selenium & TestNG dependencies under <dependencies> section.

D D M M Y Y Y Y

5 * Create a simple Website (HTML, CSS & logo).

- In src/main/resource folder, create index.html file, style.css file & place logo.png image.

index.html:

6. Upload the Website to GitHub
- Initialize a git repository in your project folder:
  git init

- Add your files & commit them.
  git add .
  git commit -m "initial commit"

- Create a GitHub repository & push the local project to GitHub:

  git remote add origin "<your-repository-url>"
  git push -u origin master

Deployment:-

1. Modify Maven configuration to copy Static Files to /docs Folder First instead of target directory
   ↳ To, configure Maven Resource Plugin in your pom.xml to copy the files directly into /docs.

2. Build Project:- By running the following command, we build our project & copy resources to /docs.

   mvn clean install

D D M M Y Y Y Y

: After this, index.html, style.css, logo.png sh ould be inside the docs folder in the root of your project.

3. Push Changes to github:-
↳ Now that the files are in the /docs folder, they are ready to be served by GitHub Pages.

```
git add docs/*
git commit -m "deploy site"
git push origin master.
```

4. Enable the GitHub Pages:-
- Go to Github repository.
- Navigate to Settings > Pages (left sidebar)
- Under the Sour section, select main branch & /docs folder as the source.
- Click Save.

5. Access your website: The static website is now hosted on GitHub Pages!

6. Write a simple Selenium Test with TestNG:-
- Create a new Java class WebPageTest.java in src/test/java directory.

7. Run test:-
↳ In IntelliJ, right-click the WebPageTest class & select Run 'WebPage Test'
↳ This test will launch Chrome, open the webpage & validates the title.

D D M M Y Y Y Y

Testing:

Steps to Package the Project as JAR & Run a main class:

1) Add maven-jar-plugin to pom.xml:
   ↳ By doing so, this will tell Maven to Include Main-class in JAR, manifest & specify the main class that should be executed when the JAR is run.

2) Create a Main class:

   In your src/main/java, create a class with a main method. Ex:- MainClass.java.

```
package come.example;

public class MainClass{
        public static void main (String[] args){
            System.out.println("Hello");
    }
}
```

3. Package the project into a JAR.
   ↳
   mvn clean Package.

4. Run the JAR file
   ↳ java -jar target/your-project-name.jar.

Side Lifecycle:- i) Add site Plugin in pom.xml
ii) mvn site
iii) Open the index.html file in target/site & this contains the project documentation

D D M M Y Y Y Y

Experiment - 3

Aim:
PART 1: Working with Gradle: Setting up a Gradle Project, Understanding Build Scripts (Groovy & Kotlin DSL), Dependency management & Task Automation.

Software Required: IntelliJ, Gradle.

Procedure:
STEP 1: Open IntelliJ IDEA & Create a new Project.
1. Click on 'New Project'
2. Select "Gradle"
3. Choose Groovy or Kotlin DSL for the build script.
4. Set the GroupID.
5. Finish.

Build & Run a small Java Application:-

STEP 1: Modify build.gradle (Groovy DSL)

STEP 2: Create a Main.java in src/main/java/com/example.

STEP 3: Build & Run the Project.
In IntelliJ IDEA, open Gradle tool window (View → Tool Windows → Gradle), Click Tasks > application > run.

Or run from terminal:
gradle run.

D D M M Y Y Y Y

STEPS
Hosting a static website on GitHub Pages:-

STEP 1: Create a /docs directory inside the root folder (not in src).
↳ Add all the HTML, CSS & images inside /docs.

STEP 2: Modify build.gradle to copy Website Files. (optional)

STEP 3: commit & push to GitHub.
2. git add.
3. git commit -m "Deploy website using Gradle"

Create a GitHub repository & push the local project to GitHub.

git remote add origin "<your-repository-url>"
git push -u origin master.

STEP 4: Enable GitHub Pages, Go to GitHub Repo → Settings → Pages. Select the /docs folder as the source.

Testing the Website using selenium & TestNG in IntelliJ IDEA.

STEP 1: Add selenium & TestNG Dependencies in build.gradle.

STEP 2: Write a Test Script
(src/test/java/org/test/webtest.java)

D D M M Y Y Y Y

STEP 3: Run the Tests Open the Gradle tool window in IntelliJ.

Click Tasks > Verification > Test

or

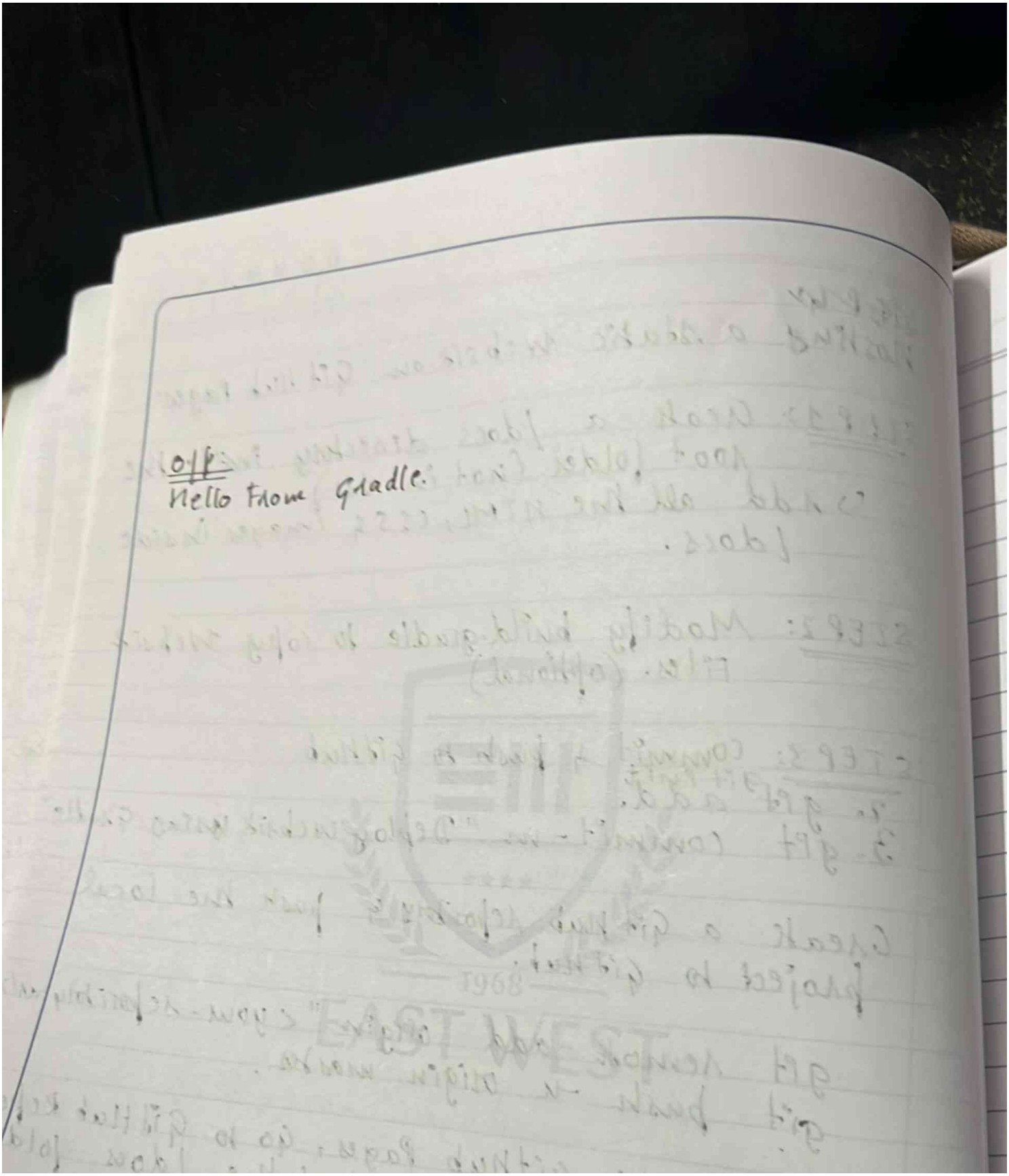
gradle.test

STEP 4:
Packaging a Gradle Project as a JAR

STEP 1: Modify build.gradle for JAR Packaging.

STEP 2: Build & package the JAR.
gradle jar.

STEP 3: Run the JAR
java -jar build/libs/<my-gradle-project>-jar.


Hello From Gradle.

D D M M Y Y Y Y

Gradle Kotlin DSL: Setting up & Building a Kotlin Project in IntelliJ IDEA

1. Setting up a Gradle Project.
   1. Open IntelliJ IDEA
   2. Click on File > New > Project
   3. Select Gradle as build system.
   4. Choose Kotlin as the language.
   5. Select Gradle Kotlin DSL.
   6. Name your Project.

2. Building a simple Kotlin Project.

STEP1: Create a Main.kt file. in src/main/kot

```
package org.example.
func main() {
        println("Hello, Gradle with Kotlin DSL!
}
```

* Building & Running Project
  Build project: ./gradlew build
  Run project: ./gradlew run.

D D M M Y Y Y Y

Experiment-4

Aim: ~~Experiment 4~~
Build & Run a Java Application with Maven, migrate the same Application to Gradle.

Software Required:- IntelliJ, Maven, Gradle.

Procedure:

& STEP 1: Create a Maven Project in IntelliJ IDEA.
i) Open IntelliJ IDEA.
- Lauch IntelliJ IDEA & Click on File→New→Project.

ii) Select Maven
- In the New project window, choose Maven
- Check the Create from archetype & select maven-archetype-quickstart.
- Click Next.

iii) Enter Project Details:
- GroupId: com.example
- ArtifactId: MVNGRDLDEMO
- Click Next & then Find.

iv) Wait for Intellij to Load dependencies

STEP2: Create a Maven Project in IntelliJ IDEA.
i) Open

D D M M Y Y Y Y

STEP2: Updak pom.xml & to Add Build plugin.

STEP3: Build & Run Maven project.

1. Open IntelliJ IDEA Terminal
2. Compile & Package the Project.
   c) Run the following commands to build the project.

```
mvn clean compile
mvn package.
```

3. Locak the JAR File.
   c) After running the above your .jar file will be locaked.
   \target\MVNGRDLDEMO-1.0.SNAPSHOT.jar

4. Run the JAR file.
   java -jar target\MVNGRDLDEMO-1.0.SNAPSHOT.jar

PART2: Migrak Maven Project to Gradle.

STEP1: Initialize Gradle in Your Project.

1. Open Terminal in IntelliJ
   c) Make sure you're in the project directory.
   cd "D:\Idea Projects\MVNGRDLDEMO"

2. Run Gradle Init Command.
   gradle init --type pom

STEP2: Review & Updak build.gradle.

1. Open build.gradle
2. Ensure the following configurations are correct:

D D M M Y Y Y Y

STEP 3: Build & Run the Gradle Project.

i) Clean & Build the Project::

gradle clean build.

ii) Run the Generated JAR file.

java -jar build/libs/MVNGRDLDEMO-1.0-SNAPSHOT.jar

D D M M Y Y Y Y

Experiment-5

Aim: Introduction to Jenkins?: What is Jenkins? Installing Jenkins on Local or Cloud Environment, Configuring Jenkins for First Use.

Jenkins:
↳ it is an open source automation server used for continuous Integration, continuous Deployment, Building pipelines & for Plugin-based Extensibility.

Jenkins automates builds & tests, reduces manual interventions & improves s/w quality.

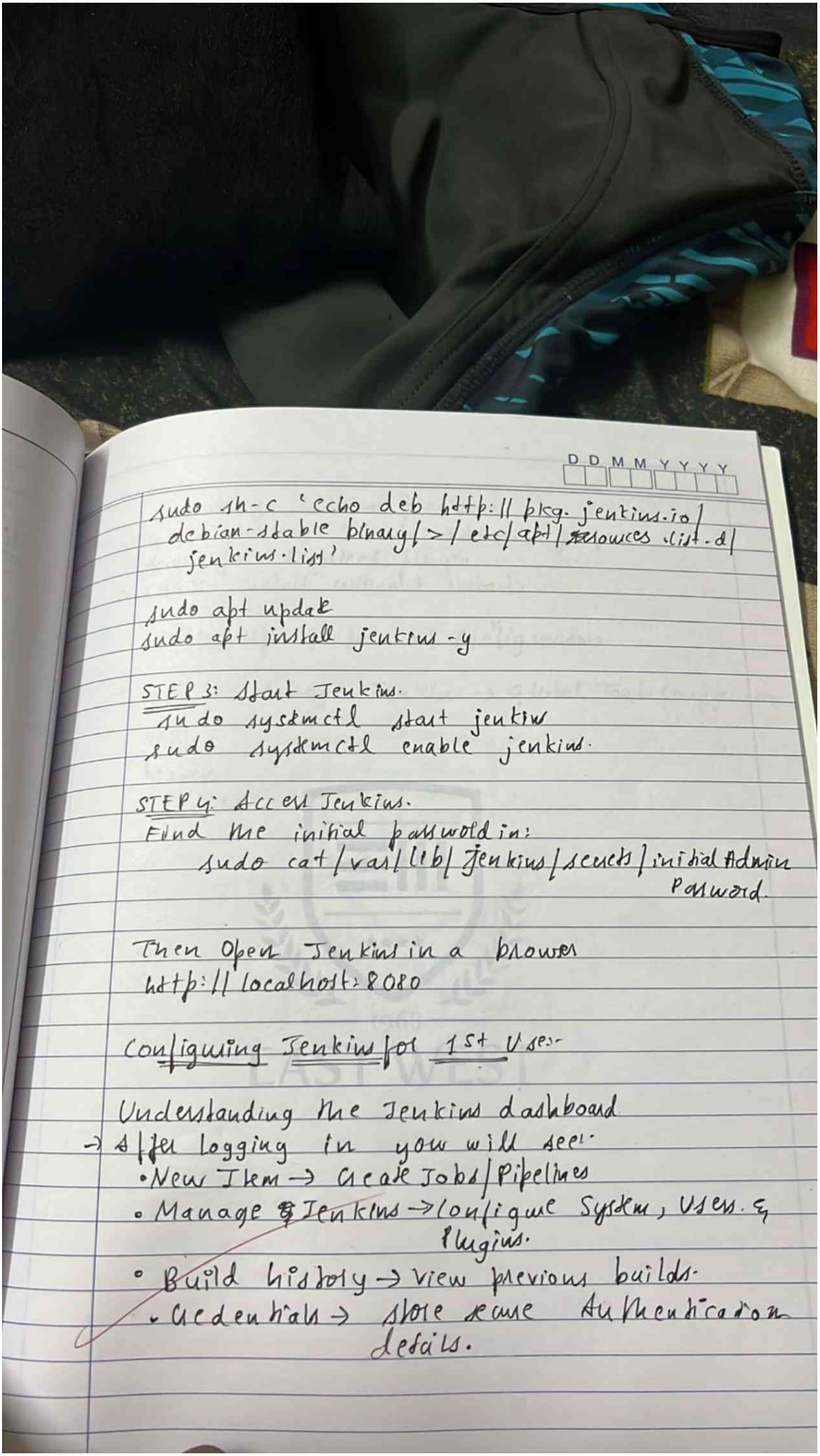
Installing Jenkins:

Prerequisites:
java version 11 or 17 is required for Jenkins.

STEP1: Download Jenkins.
↳ choose Windows Installer (.msi) for an easy set up.

STEP2: Install
~~x~~ Add Jenkins Repository & Install Jenkins.

wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -



D D M M Y Y Y Y

```
sudo sh -c 'echo deb http://pkg.jenkins.io/
  debian-stable binary/ > /etc/apt/sources.list.d/
  jenkins.list'

sudo apt update
sudo apt install jenkins -y
```

STEP 3: Start Jenkins.

```
sudo systemctl start jenkins
sudo systemctl enable jenkins.
```

STEP 4: Access Jenkins.

Find the initial password in:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword.
```

Then Open Jenkins in a browser

http://localhost:8080

Configuring Jenkins for 1st Use:-

Understanding the Jenkins dashboard

→ After logging in you will see:

- New Item → Create Jobs/Pipelines
- Manage Jenkins → Configure System, Users & Plugins.
- Build history → view previous builds.
- Credentials → store secure Authentication details.

D D M M Y Y Y Y

Installing Additional plugins:

→ Go to Manage Jenkins → Manage plugins
→ Search for required plugin
→ Click install without restart.

Setting up Global Tool Configuration.

→ Go to Manage Jenkins → Global Tool Configuration

JDK:
Maven:
Gradle:

Vinutha