Prezoom Project

Tools and technologies used

Maven

Maven is a build automation tools which is mainly used for Java bases project. Maven can help to build the whole application with just one command instead of manually writing long scripts for building the projects. It is also used for dependency management. Dependencies are nothing but third party libraries which are used in the application. If Maven is not used and any library or JAR is updated, then we have to manually update its JAR file in the project. However, with the help of Maven, it is very easy to update the dependencies i.e. we can just update the version number of the dependency in pom.xml. It dynamically downloads the libraries and plugins from the Maven Central Repository. Maven can also be used with other programming languages such as C#, Scala, and Ruby.

Project Object Model (pom.xml) file is the heart of any maven project. It contains dependencies, plugins, goals to be executed, source directory for code, etc., which is used by Maven to build the application. When we want to build the project using Maven, we supply Maven with the pom.xml file, which needs to be used to execute all the commands.

Maven is useful in the following scenarios: 1. If there are too many dependencies to be handled in the project. 2. When the libraries/JARs being used are updated frequently. 3. Continuous integration, continuous deployment and testing is handled by Maven. 4. Building the application package (JAR, WAR, ZIP etc.) and generating the documentation from source code is very easy using Maven.

JUnit

JUnit is an testing framework for Java applications. We need to create unit test case files in order to do unit testing. Unit test case code can help us ensure that the application logic is working as per our expectations. Testing can of two types as following: 1. **Manual Testing**: It means doing the tests on application manually, without use of any testing script code. It is less reliable, prone to human errors, and time consuming process. 2. **Automation Testing**: It means writing the automated test scripts to test the application logic. It is more reliable and faster as compared manual testing.

JUnit is very much useful due to following reasons: 1. It helps us to identify bugs in our business logic code, which makes our application reliable. 2. Using JUnit will help us save time by not doing the testing manually. 3. JUnit is helpful in test-driven environment. 4. It helps to develop reliable, readable, and bug-free code, which boosts the confidence of developers using the software development.

Following are the some the annotations in Junit testing: 1. **@Test**: It indicates the method is test method. 2. **@Test(timeout=1000)**: It indicates that the test should fail if it is taking more than 1 second (1000 milliseconds) to run. 3. **@BeforeClass**: It indicates that the method will be called before starting any test case. 4. **@Before**: It indicates that the method will be called before each test. 5. **@After**: It indicates that the method will be called after each test. 6. **@AfterClass**: It indicates that the method will be called after completing any test case.

JavaFX

JavaFX is a software platform that provides plenty of media and graphics packages for creating Java desktop application. It can also be used to create rich web applications that can run on wide variety of devices. It helps developers to design, create, deploy, debug, and test client applications that run on variety of of platforms.

Following are some of the features of JavaFX: 1. **FXML**: — It is a XML-based UI markup language created by Oracle for defining and creating UI in Java based applications.

2. **Scene Builder**: It is visual layout toll, which helps the users quickly create UI in JavaFX applications. Users can drag and drop UI components.
3. **Swing Compatibility**: In JavaFX, Swing content can be embedded using Swing Node class. Also, existing Swing applications can be updated with JavaFX features such as rich graphics and embedded web content.
4. **CSS Styling**: We can also use CSS classes to provide styling in our UI.
5. **Rich sets of API's**: JavaFX provide plethora of APIs to develop GUI applications.

Test Driven Development

We have used Test Driven Development (TDD) software practice in our project. It is a "Test-first" approach, which means that it requires unit testing code to be written before implementing the actual business logic code. Therefore, at the beginning all the test cases will be failing, and after writing the code logic, the test cases should pass without changing anything. On the surface, it might seem that writing all the test case is a lot of extra code, and it takes extra time. However, after one has successfully understood, it helps the developer to code faster with unit testing by following TDD rather than without TDD.

Following are some of the advantages of TDD. 1. We receive faster feedback on code logic. 2. Developer has to spend less time on debugging. 3. It reduces the time required to do the rework. 4. It helps to identify whether the last refactoring change broke the code. 5. Error and problems in the code are identifies quickly. 6. It helps to create code which is felexible and easily maintainable. 7. It forces us to write small classes, which has only one focus. Therefore, TDD helps to create SOLID code.

Maven Setup on various OS

Windows

1. Ensure that `JAVA_HOME` in Environment variable is set to JDK installed on the system. For example, `JAVA_HOME` can be `C:\Program Files\Java\jdk-13.0.2`
2. Download Maven from following this URL, and extract it.
3. Add Maven's path to `PATH` variable in Environment Variables. For example, it can be `C:\Program Files\apache-maven-3.6.3\bin`

4. Open new command window and run `mvn --version` to make sure that the installation was successful.

Linux

1. Download Maven using following command: `wget https://www-us.apache.org/dist/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz -P /tmp`
2. Unzip(untar) the downloaded file using following command: `tar xf /tmp/apache-maven-*.tar.gz -C /opt`
3. Install the alternative version for the mvn in your system `sudo update-alternatives --install /usr/bin/mvn mvn /opt/apache-maven-3.6.3/bin/mvn 363`
4. Check if your configuration is ok. You may use your current or the 3.6.3 whenever you wish, running the command below. `sudo update-alternatives --config mvn`
5. Make sure that the installation was successful by using following command. `mvn --version`

MacOS

1. Download latest Maven binary archive from here.
2. Extract the contents of the archive:
3. Move the extracted maven folder to any other location, eg Applications using below command:

`mv Downloads/apache-maven* /Applications/apache-maven-3.6.3`

4. Now open terminal and setup maven class path in environment variable by updating `.bash_profile` file by running following commands:

`open ~/.bash_profile`

5. Now edit the file by adding the following lines:

`export M2_HOME=/Applications/apache-maven-3.6.3` `export PATH=$PATH:$M2_HOME/bin`

6. reload `.bash_profile` by using following command: `source ~/.bash_profile`
7. Make sure that the installation was successful by using following command. `mvn --version`

How to setup and run application

Maven Lifecycle

As we are using Maven for dependency management, there is no need to download the dependencies separately. Maven will take care of it. We can run following maven commands to build the project and run the application.

1. Use the below command to remove the files generated at build-time in project's directory. `mvn clean`
2. Install all the dependencies required by the project using following command: `mvn install`
3. Run the test cases by running following command: `mvn test`
4. Build the application project using following command: `mvn package`
5. Finally, execute the application by running following command: `mvn exec:java`