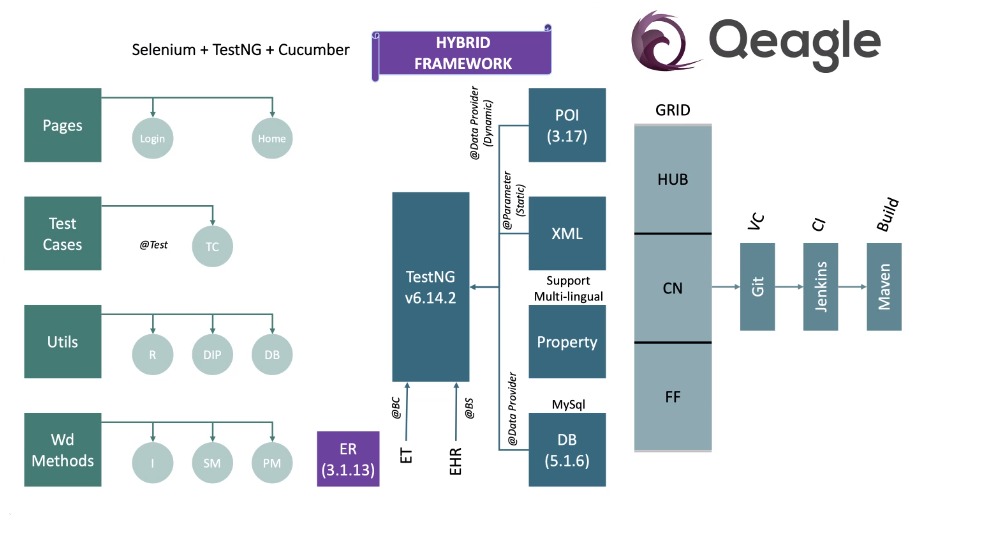
**FrameWork Introduction:**

* Framework is designed based on the design pattern POM(Page Object Model) and it integrates both the API and Browser automation Testing.
* To interact with the webElements of Web application, selenium with Java is used and the test run is done using TestNg framework.
* External data like Excel files are handled with Apachi POI which helps to access multiple data required by the project.
* BDD approach also been combined with the framework with the aid of Cucumber tool.
* Html reports are been generated for each step of the Testcases using Extent Report. All these integration is achieved through Maven build tool.

**FrameWork Architecture:**



**Components of FrameWork:**

**First Level of Components**

**Pages:** Pages all the methods which shows the navigation of the application**.** Each page will have class file where the corresponding methods required for the page is implemented. **Testcases:** The class file where the specific test case starts its execution using TestNg runner class.

**Utils:** The component helps to hold the external data files like db, excel files and also the reporter class to capture the status of the project. It also holds the reporter configuration for the whole project

**WebDriver Methods:** The component is composed of all the interfaces and wrapper methods which are required to interact with the webelements.

**Second Level of components**

**TestNg components:** Components is used as the execution platform for the overall project. This includes the other frameworks for the integration of

\*Dynamic data which are maintained in Excel format are handled with ApachiPOI

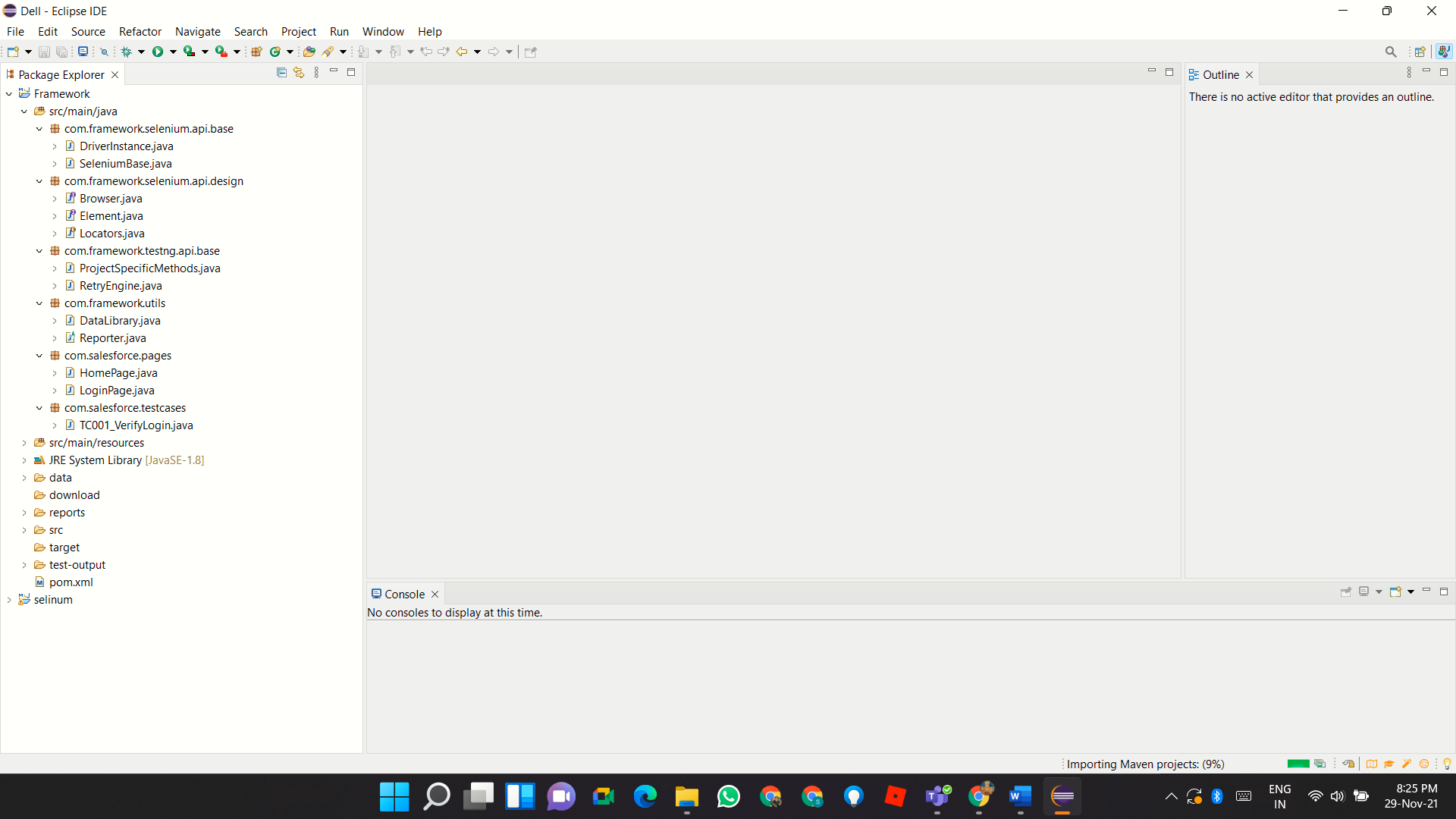
\* Multilingual data and other common data with XML are maintained using Property file.

\* The component also holds connection for accessing the database(MySQL).

**Third Level of Components:**

Third level of components communicates with the CI/CD tools like Jenkins and uses version control tool as GIT. Grid module supports the Cross-browser testing.

The project is build based on **Maven build Tool** for easy maintenance and provides flexibility for the automation process.

Let’s get into how the implementation of framework is done in the project.

***Selenium Design:***

Selenium Design package holds 3 interfaces separately for Browser interaction, Webelement Interaction and also the locators to communicate with the Browser elements.

**Browser Interface:**

Browser interface class created with all methods which required communicating with the browser. The browser commands include all precondition and post condition methods.

***Element Interface***

The Element interface class is designed with all the actions related to the web applications.

**Locators class:**

The locator class is designed as enumerator type so that variables of this class will be constant and used throughout the project. The string variables used here are the locators

***ID***, ***XPATH***, ***CLASS\_NAME***, ***NAME***, ***CSS***, ***LINK\_TEXT***, ***PARTIAL\_LINKTEXT***, ***TAGNAME***

***SeleniumBase package:***

WebDriver Methods implementation is parted with three classes:

* DriverInstance
* Selenium Base
* ProjectSpecificMethods

**DriverInstance:**

**\***Here in this project, we have declared the Remote Webdriver as a wrapper class inside the ThreadLocal class which is a private final and static class so that the invoked driver is made unchanged in the subsequent calls.

\*It acts as the super most class for the framework

\*In simple words, the single unique driver instance is used by all the testcases and this helps to avoid the ambiguity when executing the testcases parallelly.

\*Class is designed based on Encapsulation concept of OOPs and the getter and setter methods are used to get the driver and also to set the required driver value and browser options. The class also be called when there is a need of pausing the poll time for the commands to execute and those are also declared under Private static final ThreadLocal fields.

\*The class declaration is done as   
 **private** **static** **final** ThreadLocal<RemoteWebDriver> ***remoteWebdriver*** = **new** ThreadLocal<RemoteWebDriver>();

**SeleniumBase:**

\*Selenium base class is a wrapper class which implements all the methods from Browser and Element Interface class.

\* The class contains all the selenium and AUI Commands.

\*The class is designed with reports generation for the verification on the implied commands and also the exceptions are captured under Try Catch block.

\*Retry Listener also been placed in this component to run the failed Testcases

**ProjectSpecificMethods:**

The base class that holds the common methods which are specific to the project. For example, in this project, the common methods like to start and teardown the application are kept in the PSM. In simple ways to say, the pre and post conditions required for the project is implemented here.

**Internal Working of framework:**

DriverInstance extends

Reporter class extends

SeleniumBase extends

ProjectSpecificMethods

extends

Pages Testcases

* The project starts its execution from the TestNg runner class file.
* First step will start with the setting up of report platform whereas the each and every steps are captured in the flow for the reporting the status of the application.
* The next step will be execution of testcase which set the values like in which browser , the project has to be executed and then it is invoke the reporter class to seizes the flow of implementation of methods.
* The third step, the dataprovider method is invoked to pass the data as input in the required methods and then the testcase methods are invoked from the ProjectSpecificMethods and it continues with the implementation of all the page object methods.
* With the implementation of all the methods, the status of the application is generated using Extent report method. The below steps shows the hierarchy of execution

**Hierarchy of Execution:**

**@BeforeSuite --> Reporter (**startReport**)**

**@BeforeTest --> TestCase class (**setValues**)**

**@BeforeClass --> Reporter (report)**

**@DataProvider --> ProjectSpecificMethods @BeforeMethod --> ProjectSpecificMethods @Test --> TestCase class @AfterMethod --> ProjectSpecificMethods**

**@AfterClass --> No Method**

**@AfterTest --> No Method**

**@AfterSuite --> Reporter**

**Mandatory Changes to be made to use this FrameWork in a project**

Step: 1 Changing the URL of the automating application and other login credentials in the ProjectSpecificMethods

Step: 2 Make changes in the report step wherever needed in the Page object classes.

Step: 3 Attach your required excel files in your project and update your workbook location in the data library class file in the utils package. Then set the values to your testcase classes.

Step: 4 If the project has multilingual data, then add the required property file in your src/main/resources package.

**Appendix:**

TC -Testcase

R -Reporter

DIP -DataProvider

DB -Database

Wd Methods -Webdriver Methods

I -Interface

SM -SeleniumBase Methods

PM -ProjectSpecific Methods

POI -Poor Obfuscation Implementation

XML -Extended Markup Language

VC -Version Control

CI -Continuous Integration

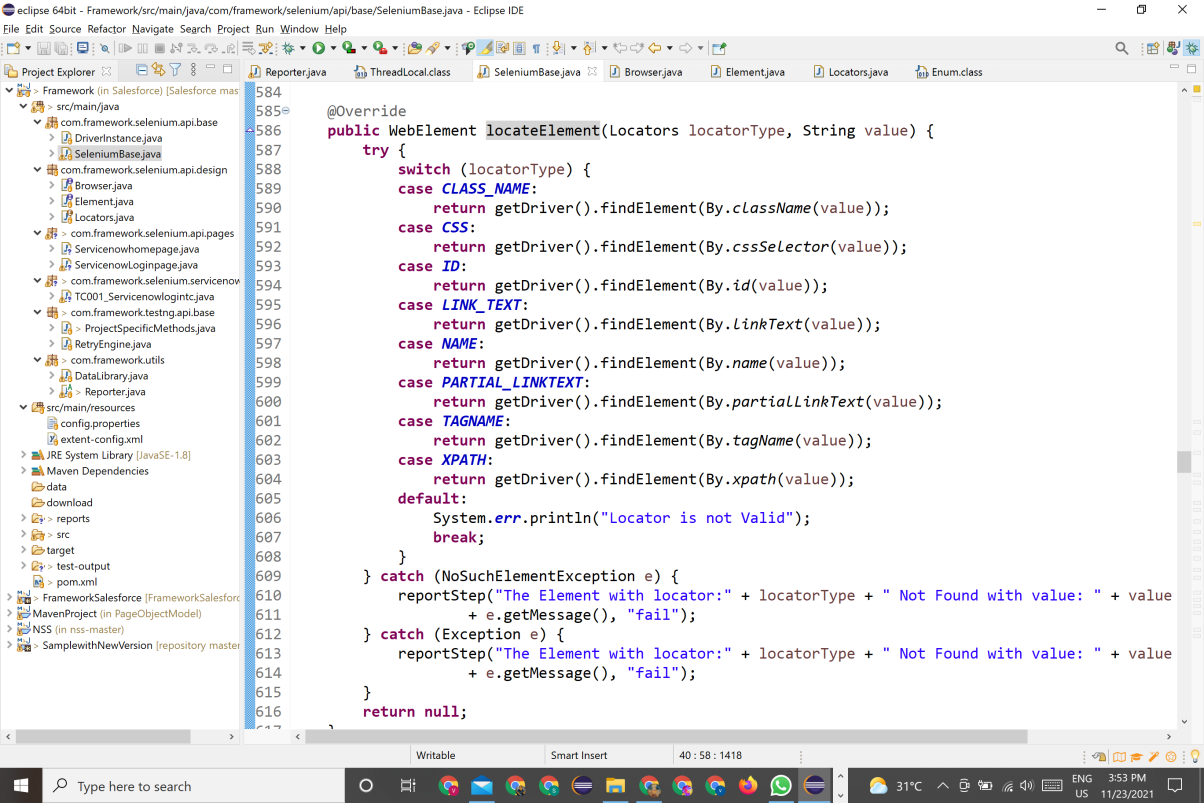
ER -Extent Report

ET -Extent Test

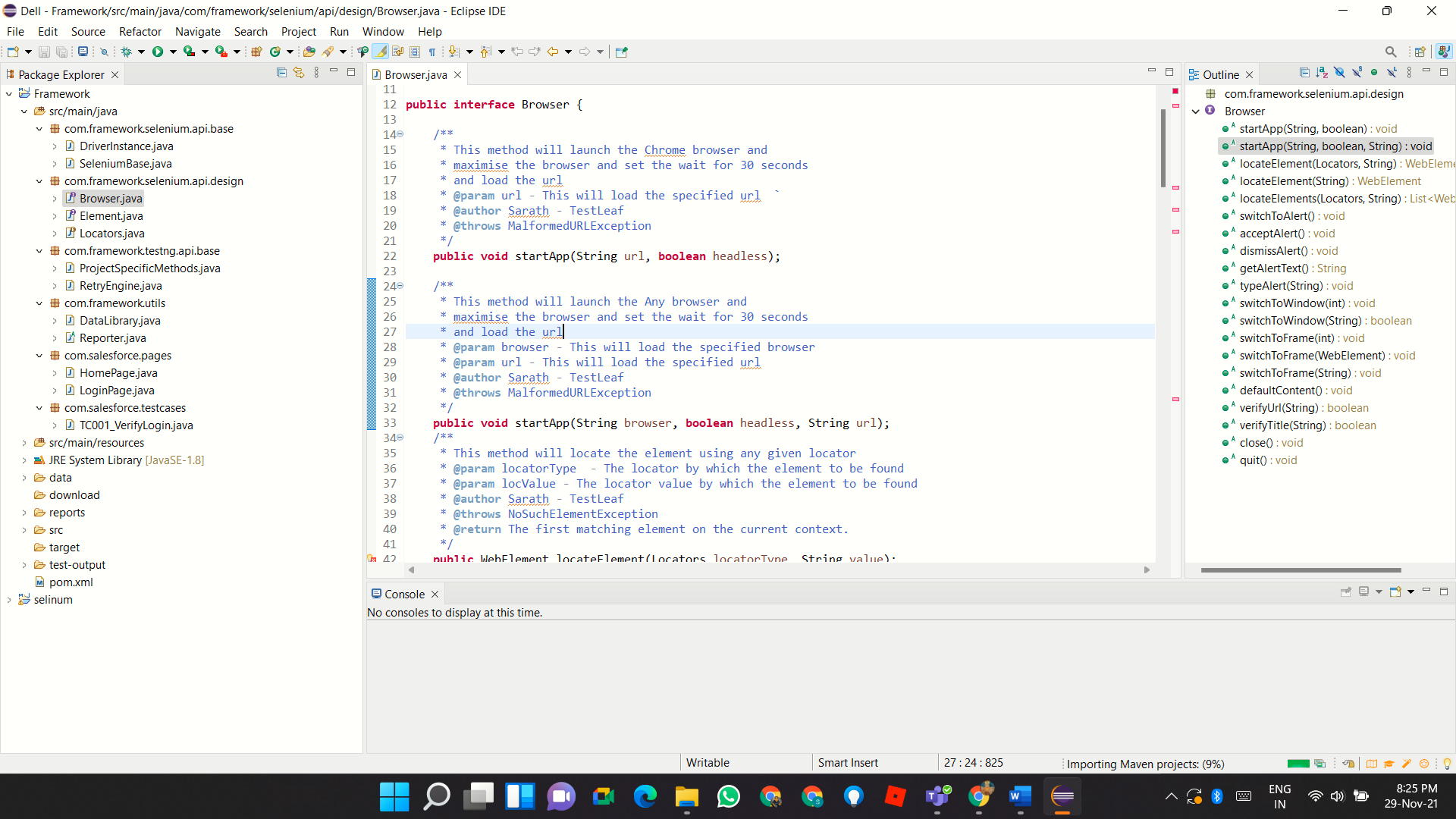
EHT -ExtentHTMLReport

CN -Configure Nodes

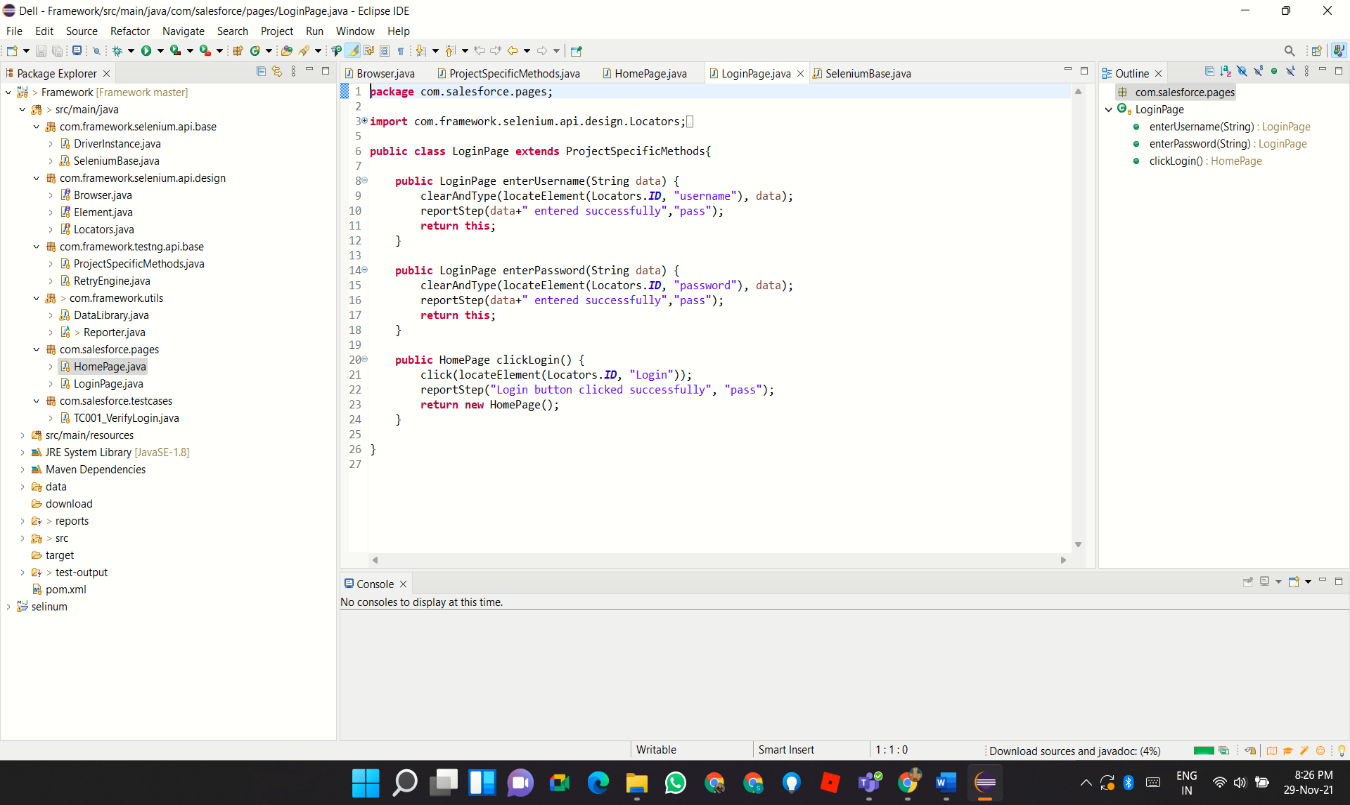
FF

**Snapshots of Some methods:**

To locate the WebElement using locators



Browser Interface Methods



Implementation of Webdriver Methods