BDD Automation Framework for Oscar-EMR

By

Harsh Jain
B.Tech., NIT Raipur, 2011

A Project Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

BDD Automation Framework for Oscar-EMR

by

Harsh Jain
B.Tech., NIT Raipur, 2011

Supervisory Committee

Dr. Jens Weber, Department of Computer Science
Supervisor

Dr. Neil Ernst, Department of Computer Science
Departmental Member

Supervisory Committee

Dr. Jens Weber, Department of Computer Science
Supervisor

Dr. Neil Ernst, Department of Computer Science
Departmental Member

Abstract

The purpose of this project is to introduce Behavior Driven Development (BDD) Testing for Oscar EMR Clinical Management System using Cucumber-Selenium Automation Framework.

Report Specification

Audience

The intended audience of this report is the Oscar-EMR Development community and the developers who will further work on BDD Testing Framework for Oscar-EMR system. This report has also been written to partially fulfill the requirement for an industrial master's degree in Computer Science. The UVic supervisory committee is also part of the intended audience and will approve the report before final submission.

Prerequisites

Before reading this report, it will beneficial (but not necessary) for the audience to have functional knowledge of Oscar-EMR and Software Testing methodologies. It will also be helpful to understand Selenium Browser Automation, Cucumber/Gherkin Language and Web development using J2ee Web application. Following are some of the useful resources to get familiar with the prerequisites:

1. Oscar EMR
    - https://oscar-emr.com/about-us/

2. Cucumber/Gherkin language:
    - https://docs.cucumber.io/guides/

3. Selenium Tutorial:
    - https://www.seleniumhq.org/docs/03_webdriver.jsp
    - https://www.guru99.com/selenium-tutorial.html

4. Cucumber-Selenium Framework using Java/Maven
    - http://toolsqa.com/cucumber-automation-framework/

Purpose

This report provides information about requirements and implementation steps taken to introduce Cucumber-Selenium for Oscar EMR Clinical Management System.

Table of Contents

List of Figures

Introduction

As part of my Industrial Project at the University of Victoria, the objective of this report is to introduce Behavior Driven Development (BDD) for Oscar Electronic Medical Record (Oscar EMR) Clinical Management System using Cucumber-Selenium Automation framework.

Oscar EMR Clinical Management System

OSCAR is an open-source Electronic Medical Record (EMR) designed by doctors for doctors, for use in medical offices and by a variety of other frontline healthcare professionals in Canada and other countries [1]. Oscar EMR is primarily used in British Columbia and Ontario provinces and currently holds about 20% of market share in Canada.

Oscar EMR is built using JAVA Programming language, MySQL database and runs using Apache Tomcat Web Container. Several frameworks are used in Oscar web application like Struts, Spring, JPA, Angular and jQuery.

BDD (Behaviour-Driven Development)

BDD (Behaviour-Driven Development) evolved from TDD (Test Driven Development) and facilitates smooth communication between product owners, business analysts, stakeholders and the development team members [2]. It helps in driving the development process by means of automated acceptance tests using Gherkin language [3].

Many times, during the software development process, exact requirements are not properly communicated from business analysts and product managers to the development team members. This leads to rework and delay in project delivery. To resolve this issue, BDD approach is implemented using Gherkin language and Cucumber Feature files. In the BDD approach, Gherkin test steps (When, Then and But) are written by Developers and then approved by Business Analysts, Product Owner and Stakeholders before the development phase begins. This ensures clear requirement specifications and timely delivery of project.

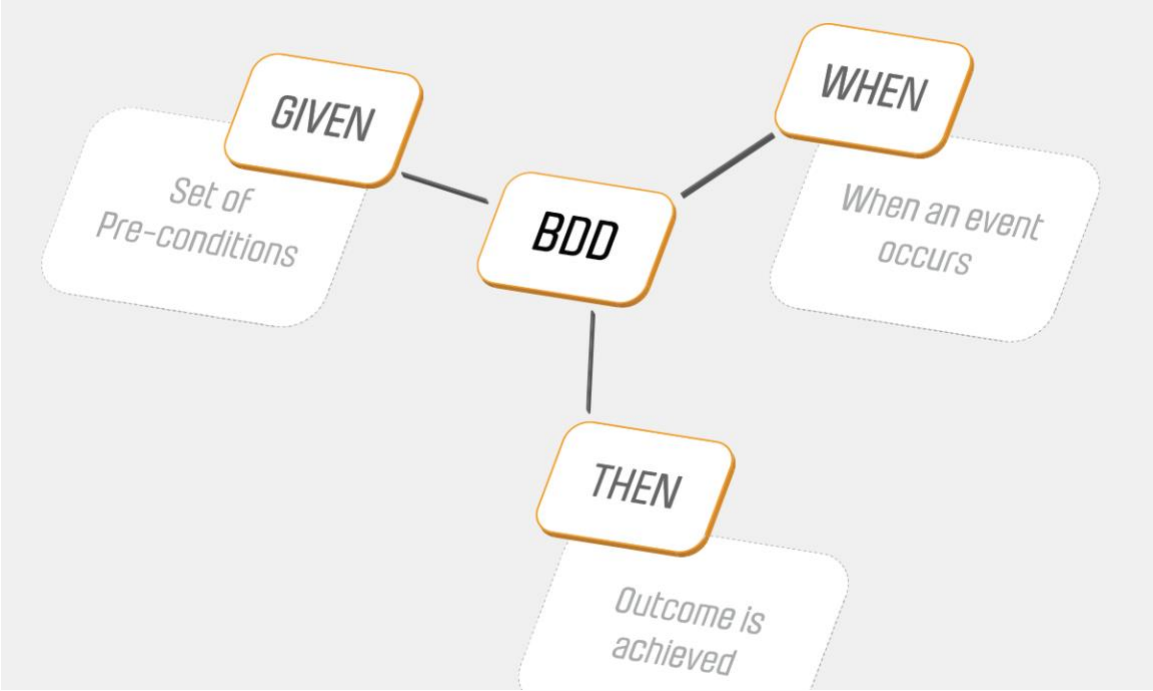


Figure 1 - Behavior Driven Development

(Source: https://site.gitscrum.com/agile-glossary- -driven-development-bdd/)

Test Automation

In test automation, test execution is done automatically by an internal/external software instead of manual user actions [4]. Automation tests are often used during Regression and Smoke testing after a major release deployment and saves time and effort for testers. There are several types of automation testing – User Interface, API Driven Testing and Unit Testing. In User Interface Testing, user interactions with browser are automatically performed by an external utility like WebDriver. For this project, User Interface Testing will be implemented using Selenium Web driver Utility.

Report Overview

The report has been divided into following major sections:

1. **Industrial Context and Requirements**: This section contains details about the choice of frameworks for BDD and Test Automation over other alternatives. It also provides details about the high-level design and package structure used for the project.
2. **Major Components, Packages and Classes**: This section gives a basic overview of package used in the project. It also gives a brief overview of classes and concepts which have been used in the codebase.
3. **Integration of Automation Test Suite in Oscar-EMR repository**: In this section, a sample test case will be picked from Oscar Atlassian Wiki and all the components of

the project will step by step created to introduce BDD Automation for this test case using Cucumber-Selenium framework.

4. **Scaling Test Steps to Cover Entire Consultation Note Screen Functionality and Final Results:** This section will explain how the project has been scaled to include the complete Consultation Note Screen functionality. The scope of the project is to cover OSCAR 15 Consultation Note Screen Test Cases from Oscar Wiki under Consultation module and convert them to Automated Test Scripts using Cucumber Selenium Automation framework. The test cases can be found at https://oscaremr.atlassian.net/wiki/spaces/OS/pages/54001741/OSCAR+15+Consultation+Note+Screen+Test+Cases.
5. **Conclusions and Recommendations**: This section will explain the final results and what can be done in future to improve the speed and efficiency of the project. Major decisions, that were taken during the project keeping in mind the advantages and trade-offs, have also been discussed in this section. This sections also contains details about defects discovered till now during execution of framework and the steps/shortcuts new developers can take to further enhance testing suite and cover additional test cases.

Industrial Context and Requirements

Currently, as part of Oscar-EMR deployment process, Unit and manual tests are run after any major releases in staging/production environment. The Unit Tests are run using JUnit framework to test individual components and manual smoke testing is done after release to make sure Production code is working as expected. Until now, no BDD or Web Browser Automated Testing framework has been implemented for Oscar-EMR.

Tool for BDD

The main objective of the Industrial project is to introduce Automation Testing Framework using BDD (Behaviour Development Development) for Oscar EMR. Cucumber is a tool based on Behavior Driven Development framework and its specification consists of Scenarios and Steps which are written in Gherkin language. Cucumber was chosen over other BDD tools due to following reasons [5] [6]:

1. Cucumber is one of the most advanced and popular BDD frameworks in Java today.
2. Cucumber is easier to use and configure when compared to other BDD tools like JBehave, EasyB and JDave.
3. JDave and EasyB are lightweight tools and not ideal for large scale testing suite.
4. JBehave works fine for large scale projects but it has difficult configuration and syntax compared to Cucumber.
5. Cucumber has many inbuild libraries and features that work well in large and complicated QA Projects.

6. There are several Cucumber plugins both inbuilt as well as by third party that can generate highly detailed reports and graphs.
7. Cucumber has active development community and detailed documentation.
8. Cucumber's integration with Automated Tools like Selenium is easy and has already been implemented by several companies using Agile and BDD approach.

Tool for User Interface Test Automation

For browser automation testing, Selenium tool was chosen for the project . Selenium is a browser automation tool used to automate user action and manual inputs. There are many alternatives to Selenium like Protractor, Jasmine, UFT and TestComplete [7]. Selenium may have some disadvantages like learning curve is bigger and it is time consuming to write Selenium test scripts, but it was still chosen over other tools due to following reasons [7]:

1. Selenium generates robust and less fragile tests compared to other automation tools.
2. Oscar EMR is open source and is built using Java and Maven. Selenium is also open source and works best for applications written in Java.
3. Selenium has many inbuilt libraries to perform complex driver operations and its latest version is up to date with latest developments and trends.
4. Integration of Selenium with Cucumber using Java/Maven is easy and has been already been implemented by several developers and projects.

High Level Design and Package Structure

Following a snapshot of the high-level design structure of the JUnit, Cucumber and Selenium Tests and how they fit together:

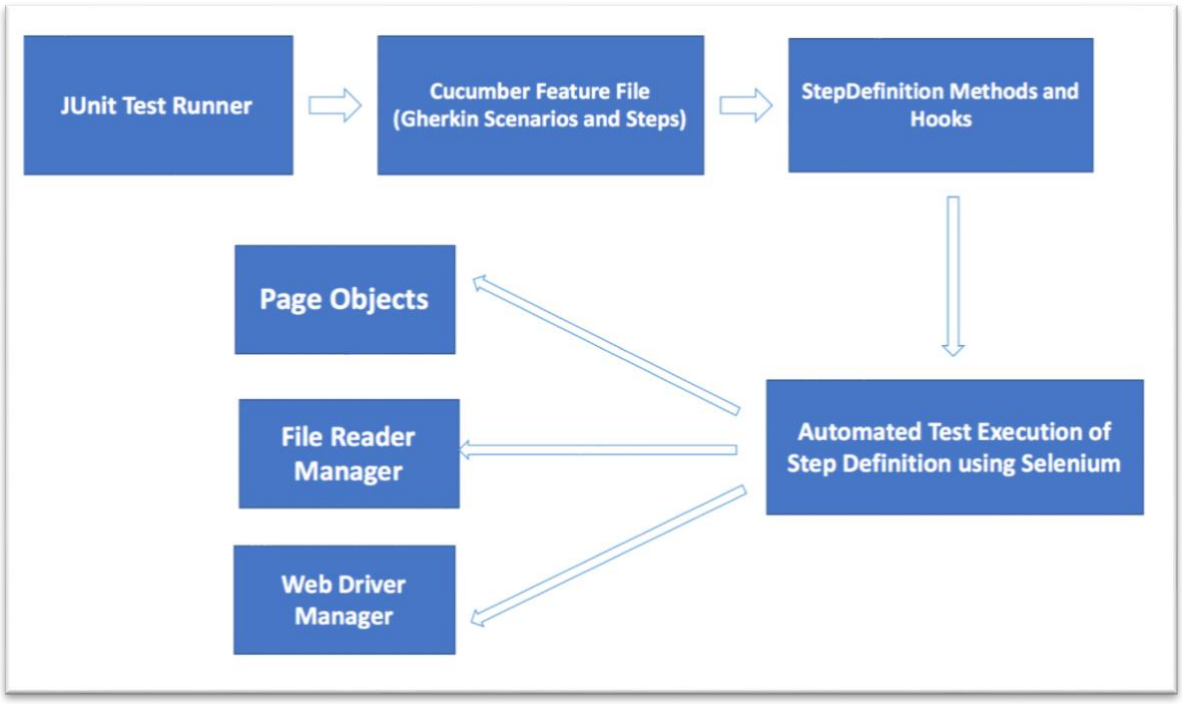


Figure 2 - High Level Design

The project consists of 7 packages added in *src/test/java* directory and a folder *cucumberSeleniumFramework* added in *src/test/resources* directory. Following is a snapshot of the packages and folders added oscar repository:

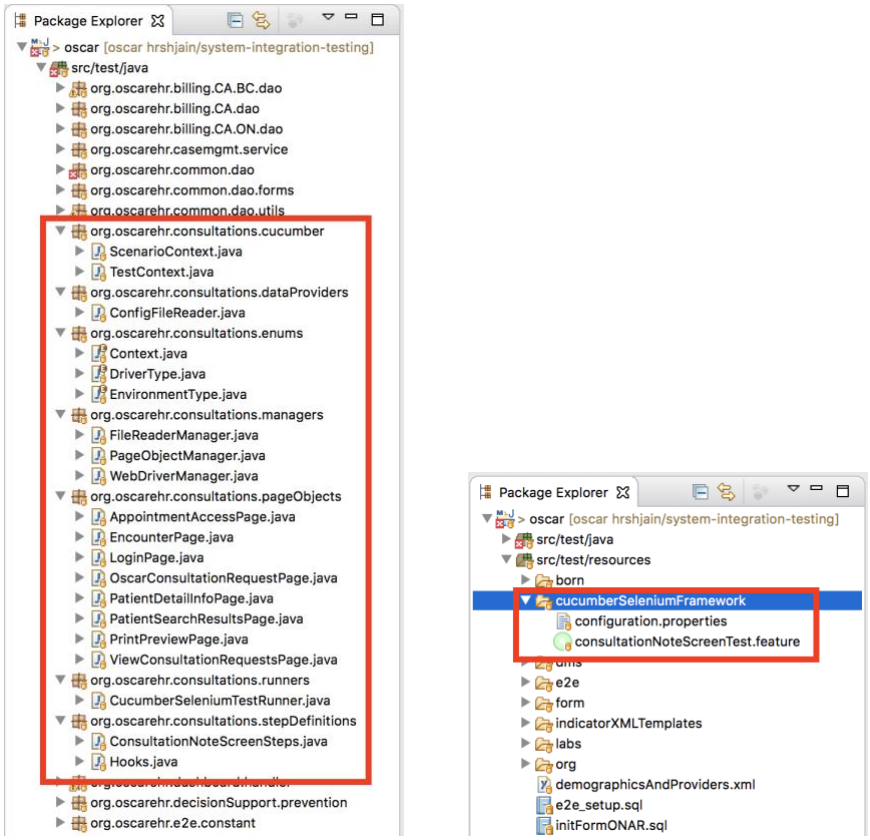


Figure 3 - Package Structure

Following is a brief description of the packages and folder:

org.oscarehr.consultations.runners: This package has the JUnit test runner class which will run the tests using Cucumber feature files.

org.oscarehr.consultations.cucumber: This package has the TextContext and ScenarioText classes. These classes are used to implement Singleton design pattern [8] and to share data between multiple Step Definition files.

org.oscarehr.consultations.stepDefinitions: This package consists of Step Defenition files which contain code corresponding to the Steps written in Cucumber Step Definition file.

org.oscarehr.consultations.pageObjects: This package consists of Page Objects on which the automatic browser actions are being performed. Instead of putting all the automation seleniuim test scripts in a single file, test scripts have been divided into Page Objects based on the driver actions that are being performed in these pages.

org.oscarehr.consultations.dataProviders: This package consists of ConfigFileReader class and is used to ensure data-driven testing for Oscar-EMR.

org.oscarehr.consultations.enums: This package helps in data-driven testing and also for sharing context and data within the project. It is also used to define and provide options for driver and environment during the test run.

org.oscarehr.consultations.managers: This package consist of FileReaderManager, PageObjectManager and WebDriverManager classes. These classes are used to ensure Singleton Design Pattern [8] for PageObjects, ConfigFileReader and WebDriver respectively.

cucumberSeleniumFramework: This folder is located in *src/test/resources* and consists of

a. configuration.properties: Used to store data and configuration parameters used in the project.

consultationNoteScreenTest.feature: Consists of Cucumber Test Steps written in Gherkin language.

Major Components, Packages and Classes

This section will give brief overview of packages, classes and concepts that are being used in the project.

Cucumber Feature File

The Cucumber feature file consists of Scenario and Steps written using Gherkin language [3]. For example, following is a snapshot of Consultation Note Screen Test Cases derived from Oscar Wiki spaces.

1.0 Test Consultation Response/ Request Letterhead Defaults to the Logged in User

Step ID	Description	Precondition	Action	Expected Results	Actual Results	Pass/Fail	Notes
#1	Consultation Response/ Request Letterhead should default to the logged-in user	1. Consultation Response/ Request screen is present 2. User is logged in with correct credentials	Verify that when new Consult Note is started, the default letterhead selection displays for the current Oscar user logged-in	Default letterhead selection should be for the current Oscar user logged-in			
#2	Ability to select other Letterheads selection	Provider records are registered within the OSCAR database	1. Select Letterhead drop down 2. Verify that other Letterheads can be selected using drop down Letterhead field	1. Letterhead is selectable from the drop down list 2. User should be able to select other letterhead name from the drop down			
#3	Ability to save Consultation Response/ Request		Select Save button	1. Save button is available on the floating bar 2. Consultation Request/ Response is saved			
#4	Ability to display Consultation Response/ Request print preview screen		Select Print Preview button	1. Consultation Request/ Response Print preview window pops up 2. Selected Letterhead should populate in FROM section			

Figure 4 - Test Scenario Snapshot

Following is the snapshot of Cucumber Feature File with Scenario and Test Steps converted from above test cases:

```
Feature: Consultation Note Screen functionality

  @1.0TestConsultationResponseRequestLetterhead
  Scenario: 1.0 Test Consultation Response/ Request Letterhead Defaults to the Logged in User
    When New Consult Note is started
    Then Default letterhead selection should be for the current Oscar user logged-in
    When User selects letterhead dropdown
    Then User should be able to select other letterhead name
    When User selects Save button
    Then Save button is available and Consultation Request/ Response is saved
    When User selects Print Preview button
    Then Consultation Request/ Response Print preview window pops up
    And Selected Letterhead should populate in FROM section
```

Figure 5 - Feature File Snapshot

Step Definition

Step Definitions map each Cucumber Gherkin step into runnable programming code to carry out the actions that need to be performed by the step [6]. Each Gherkin step in Cucumber feature file is represented in the form of Step Definition method. The annotations of these methods should match the Gherkin steps. For example, following are the two highlighted Gherkin Steps in Cucumber Feature file:

```
@1.0TestConsultationResponseRequestLetterhead
Scenario: 1.0 Test Consultation Response/ Request Letterhead Defaults to the Logged in User
  When New Consult Note is started
  Then Default letterhead selection should be for the current Oscar user logged-in
  When User selects letterhead dropdown
  Then User should be able to select other letterhead name
  When User selects Save button
  Then Save button is available and Consultation Request/ Response is saved
  When User selects Print Preview button
  Then Consultation Request/ Response Print preview window pops up
  And Selected Letterhead should populate in FROM section
```

Figure 6 - Gherkin Steps

Following are the two methods in Step Definition file that map to the above highlighted Gherkin Steps:

```
@When("^New Consult Note is started$")
public void new_Consult_Note_is_started() {
    //Navigate to Consultations Page
    oscarConsultationRequestPage.navigate_to_consultations_page();

    //Click on New Consultation link
    oscarConsultationRequestPage.start_new_consultation();
}

@Then("^Default letterhead selection should be for the current Oscar user logged-in$")
public void default_letterhead_selection_should_be_for_the_current_Oscar_user_logged_in() {
    // Verify letterhead value
    oscarConsultationRequestPage.verify_default_letterhead_selection();
}
```

Figure 7 - Step Definition Methods

Annotations of these methods should exactly match the text of Gherkin steps. For the method names, it is not required to keep them same as the Gherkin steps but is highly recommended to ensure clean and readable code.

Selenium Tests

These are the browser actions that are performed automatically by the web driver [9]. These browser actions replace manual operations by a tester like starting a new web instance, finding a web element, clicking a web element, entering values in textbox and submitting a button. Following is a sample code snippet that starts Firefox browser instance, enter oscar_url and logs into Oscar EMR:

```
1.  import org.openqa.selenium.By;
2.  import org.openqa.selenium.WebDriver;
3.  import org.openqa.selenium.WebElement;
4.  import org.openqa.selenium.firefox.FirefoxDriver;
5.  import org.openqa.selenium.Keys;
6.  public class MyClass {
7.
8.      public static void main(String[] args) {
9.          // declaration and instantiation of objects/variables
10.         System.setProperty("webdriver.gecko.driver","/Users/hjain/code/geckodriver");
11.         WebDriver driver = new FirefoxDriver();
12.         String baseUrl = " https://localhost:8442/oscar";
13.         // launch Fire fox and direct it to the Base URL
14.         driver.get(baseUrl);
15.         //Get web element for username, password and pin
16.         WebElement username = driver.findElement(By.id("username"));
17.         WebElement password = driver.findElement(By.id("password"));
18.         WebElement pin = driver.findElement(By.id("pin"));
19.
20.         //Enter values for the web element
21.         username.sendKeys("oscardoc");
22.         password.sendKeys("LEADlab!");
23.         pin.sendKeys("1117");
24.
25.         //Click on login button
26.        pin.sendKeys(Keys.RETURN);
27.       //close Fire fox
28.         driver.close();
29.     }
30. }
31.
```

These Selenium tests will be added into Step Definition methods so that the browser actions are performed as part of Cucumber Framework.

Page Objects and Page Object Manager

If more code is added in StepDefinition files, methods in the StepDefinition file can become bloated and the codebase can become longer and difficult to manage. For this

purpose, the code in the Step Definition files (performing Selenium driver operations) is divided into several PageObjects [10].
For example, if the code is accessing 5 pages, then instead of putting all the code in a single StepDefinition method, PageObjects can be created for each page and the code can be divided among these PageObjects. Following is a snapshot of LoginPage PageObject.

LoginPage.java

```
1  package org.oscarehr.consultations.pageObjects;
2
3  import org.openqa.selenium.Keys;
4  import org.openqa.selenium.WebDriver;
5  import org.openqa.selenium.WebElement;
6  import org.openqa.selenium.support.FindBy;
7  import org.openqa.selenium.support.How;
8  import org.openqa.selenium.support.PageFactory;
9  import org.oscarehr.consultations.managers.FileReaderManager;
10
11 public class LoginPage {
12
13     WebDriver driver;
14
15     public LoginPage(WebDriver driver) {
16         this.driver = driver;
17         PageFactory.initElements(driver, this);
18     }
19
20     @FindBy(how = How.NAME, using = "username")
21     static WebElement username;
22
23     @FindBy(how = How.NAME, using = "password")
24     static WebElement password;
25
26     @FindBy(how = How.NAME, using = "pin")
27     static WebElement pin;
28
29     @FindBy(how = How.XPATH, using = "//a[@class='ng-scope ng-binding'][contains(text(),'Schedule')]")
30     static WebElement schedule;
31
32
33     public void login_into_oscar_emr() {
34         // launch Fire fox and direct it to the Base URL
35         String baseUrl = FileReaderManager.getInstance().getConfigReader().getApplicationUrl();
36         driver.get(baseUrl);
37
38         //Enter Username, password and Pin
39         username.sendKeys(FileReaderManager.getInstance().getConfigReader().getOscarUsername());
40         password.sendKeys(FileReaderManager.getInstance().getConfigReader().getOscarPassword());
41         pin.sendKeys(FileReaderManager.getInstance().getConfigReader().getOscarPin());
42         pin.sendKeys(Keys.RETURN);
43
44         //Other selenium methods to wait for page load  until visibility of elements
45         //did not work to resolve the unexpected pop-up after clicking Schedule option
46         try {
47             Thread.sleep(5000);
48         } catch (InterruptedException e) {
49             // TODO Auto-generated catch block
50             e.printStackTrace();
51         }
52     }
```

Figure 8 - Page Object Design

Along with PageObjects, *PageObjectManager.java* class is used to ensure that not more than a single instance of PageObjects is used during the test run and all the PageObjects can be accessed from a single point of access. Following is a snapshot of PageObjectManager.java:

PageObjectManager.java

```java
package org.oscarehr.consultations.managers;

import org.openqa.selenium.WebDriver;
import org.oscarehr.consultations.pageObjects.AppointmentAccessPage;
import org.oscarehr.consultations.pageObjects.EncounterPage;
import org.oscarehr.consultations.pageObjects.LoginPage;
import org.oscarehr.consultations.pageObjects.OscarConsultationRequestPage;
import org.oscarehr.consultations.pageObjects.PatientDetailInfoPage;
import org.oscarehr.consultations.pageObjects.PatientSearchResultsPage;
import org.oscarehr.consultations.pageObjects.PrintPreviewPage;
import org.oscarehr.consultations.pageObjects.ViewConsultationRequestsPage;

public class PageObjectManager {

    private WebDriver driver;
    private OscarConsultationRequestPage oscarConsultationRequestPage;
    private LoginPage loginPage;
    private ViewConsultationRequestsPage viewConsultationRequestsPage;
    private PrintPreviewPage printPreviewPage;
    private AppointmentAccessPage appointmentAccessPage;
    private EncounterPage encounterPage;
    private PatientSearchResultsPage patientSearchResultsPage;
    private PatientDetailInfoPage patientDetailInfoPage;
    public PageObjectManager(WebDriver driver) {
        this.driver = driver;
    }

    public OscarConsultationRequestPage oscarConsultationRequestPage() {
        return (oscarConsultationRequestPage == null) ? oscarConsultationRequestPage = new OscarConsultationRequestPage(driver) : oscarConsultat
    }

    public LoginPage loginPage() {
        return (loginPage == null) ? loginPage = new LoginPage(driver) : loginPage;
    }
```

Figure 9 - Page Object Manager

Hooks

Hooks are used to set up *prerequisites* that are run before a scenario starts and *after-steps* that are run as soon as Scenario is complete [11]. For example, before running any scenario, user needs to be logged into Oscar-EMR and after running a scenario, user needs to be log out and close the browser. @Before annotation is used to run *prerequisite* methods before executing a scenario and @After annotation is used to run *after-step* methods after executing a scenario. Following is a snapshot of Hooks.java class:

Hooks.java

```java
package org.oscarehr.consultations.stepDefinitions;

import org.oscarehr.consultations.cucumber.TestContext;
import org.oscarehr.consultations.pageObjects.AppointmentAccessPage;
import org.oscarehr.consultations.pageObjects.EncounterPage;
import org.oscarehr.consultations.pageObjects.LoginPage;
import org.oscarehr.consultations.pageObjects.PatientDetailInfoPage;
import org.oscarehr.consultations.pageObjects.PatientSearchResultsPage;
import org.oscarehr.consultations.pageObjects.ViewConsultationRequestsPage;

import cucumber.api.java.After;
import cucumber.api.java.Before;

public class Hooks {

    TestContext testContext;
    LoginPage loginPage;
    AppointmentAccessPage appointmentAccessPage;
    PatientSearchResultsPage patientSearchResultsPage;
    EncounterPage encounterPage;
    PatientDetailInfoPage patientDetailInfoPage;
    ViewConsultationRequestsPage viewConsultationRequestsPage;

    public Hooks(TestContext context) {
        testContext = context;
        loginPage = testContext.getPageObjectManager().loginPage();
        appointmentAccessPage = testContext.getPageObjectManager().appointmentAccessPage();
        patientSearchResultsPage = testContext.getPageObjectManager().patientSearchResultsPage();
        encounterPage = testContext.getPageObjectManager().encounterPage();
    }

    @Before(order=0)
    public void user_logs_into_oscar_emr() {
        //Start Browser and enter Enter Login credentials
        loginPage.login_into_oscar_emr();

        // Click on schedule Option
        loginPage.click_on_schedule();
    }
```

Figure 10 - Hooks.java

JUnit Test Runner

JUnit framework is used to execute the Cucumber scenarios and step definition files. *@RunWith(Cucumber.class)* annotation is used to make sure that JUnit runner is run through Cucumber class and *@CucumberOptions* annotation is used to specify the Cucumber feature files that will be included in the test run.

Following is the snapshot of JUnit Test Runner file:

*IntegrationTestRunner.java

```java
package org.oscarehr.integration.consultations.stepDefinitions;

import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
        features = {
                "src/test/resources/integration/ConsultationNoteScreenTest.feature"}
        )

public class IntegrationTestRunner {

}
```

Figure 11 - JUnit Test Runner

A separate JUnit runner has been created for this project within Oscar repository. The runner can be run individually for BDD Tests. In future, it can be integrated with Oscar JUnit runner so that it is part of the Oscar JUnit test suite.

Configuration Property File

Configuration.properties file is used to store test data and parameters in the form of key-value pairs [9]. Using Configuration.properties file, the parameter values can be updated easily and without changing the code.

Following is a snapshot of the Configuration.properties file.

Figure 12 - Configuration.properties file

Config File Reader Class

ConfigFileReader class is used to retrieve parameter values present in the Configuration Property File. This is done using inbuilt packages Buffered Reader and File Reader classes in java. Following is a snapshot of the Config File Reader class.

ConfigFileReader.java

```java
package org.oscarehr.consultations.dataProviders;

import java.io.BufferedReader;

public class ConfigFileReader {

    private Properties properties;
    private final String propertyFilePath= "src/test/resources/integration/Configuration.properties";


    public ConfigFileReader(){
        BufferedReader reader;
        try {
            reader = new BufferedReader(new FileReader(propertyFilePath));
            properties = new Properties();
            try {
                properties.load(reader);
                reader.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
            throw new RuntimeException("Configuration.properties not found at " + propertyFilePath);
        }
    }

    public String getDriverPath(){
        String driverPath = properties.getProperty("driverPath");
        if(driverPath!= null) return driverPath;
        else throw new RuntimeException("driverPath not specified in the Configuration.properties file.");
    }

```

Figure 13 - ConfigFileReader

Each key-value pair in Configuration Property File has its own get method in the ConfigFileReader class. For example, value of oscar_username key is accessed using method getOscarUsername() in ConfigFileReader.

Configuration.properties

```
driverPath=/Users/harshjain/code/geckodriver
oscar_url=https://localhost:8442/oscar
oscar_username=oscardoc
oscar_password=LEADlab!
oscar_pin=1117
oscarRegistrationID=123456
```

Figure 14 - Key, Values in Configuration.properties

ConfigFileReader.java

```java
    public String getOscarUsername() {
        String oscar_username = properties.getProperty("oscar_username");
        if(oscar_username != null) return oscar_username;
        else throw new RuntimeException("url not specified in the Configuration.properties file.");
    }

```

Figure 15 - ConfigFileReader Method to access Parameters

Value of oscar_username parameter is accessed by creating a ConfigFileReader instance in Step Definition file and calling getOscarUsername() method of this instance.

```
//Create ConfigFileReader Object
ConfigFileReader configFileReader = new ConfigFileReader();

//Call getOscarUsername() from the configFileReader object
String oscar_username = configFileReader.getOscarUsername();
```

Figure 16 - Create ConfigFileReader Instance

File Reader Manager Class

To access configuration key-values from several classes, a new ConfigFileReader instance needs to be created every time. This can be avoided by using File Reader Manager. File Reader Manager is used to make sure that there is only one instance of ConfigFileReader class (initiated on first use) used during test run and there is global point of access to it.
Following is the snapshot of FileReaderManager:

*FileReaderManager.java

```
1  package org.oscarehr.consultations.managers;
2
3  import org.oscarehr.consultations.dataProviders.ConfigFileReader;
4
5  public class FileReaderManager {
6
7      private static FileReaderManager fileReaderManager = new FileReaderManager();
8      private static ConfigFileReader configFileReader;
9
10     private FileReaderManager() {
11     }
12
13      public static FileReaderManager getInstance( ) {
14           return fileReaderManager;
15      }
16
17      public ConfigFileReader getConfigReader() {
18          return (configFileReader == null) ? new ConfigFileReader() : configFileReader;
19      }
20
21 }
22
```

Figure 17 - FileReaderManager

For example, ConfigFileReader can be accessed using *FileReaderManager.getInstance().getConfigReader()* method and Oscar Username can be accessed using *FileReaderManager.getInstance().getConfigReader().getOscarUsernamed()* method.

WebDriver Manager

WebDriverManager takes care of operations like locating geckodriver.exe file, initiating and closing browser instance. One more crucial functionality of WebDriverManager is to ensure that a single driver instance is used for one particular Cucumber Scenario

(especially when the driver is accessed by several PageObject classes). Following is snapshot of WebDriveManager:

WebDriverManager.java

```
1  package org.oscarehr.consultations.managers;
2
3  import java.util.concurrent.TimeUnit;
4  import org.openqa.selenium.WebDriver;
5  import org.openqa.selenium.chrome.ChromeDriver;
6  import org.openqa.selenium.firefox.FirefoxDriver;
7  import org.openqa.selenium.ie.InternetExplorerDriver;
8  import org.oscarehr.consultations.enums.DriverType;
9  import org.oscarehr.consultations.enums.EnvironmentType;
10
11 public class WebDriverManager {
12     private WebDriver driver;
13     private static DriverType driverType;
14     private static EnvironmentType environmentType;
15     private static final String CHROME_DRIVER_PROPERTY = "webdriver.chrome.driver";
16     private static final String FIREFOX_DRIVER_PROPERTY = "webdriver.gecko.driver";
17
18     public WebDriverManager() {
19         driverType = FileReaderManager.getInstance().getConfigReader().getBrowser();
20         environmentType = FileReaderManager.getInstance().getConfigReader().getEnvironment();
21     }
22
23     public WebDriver getDriver() {
24         if(driver == null) driver = createDriver();
25         return driver;
26     }
27
28     private WebDriver createDriver() {
29            switch (environmentType) {
30             case LOCAL : driver = createLocalDriver();
31                 break;
32             case REMOTE : driver = createRemoteDriver();
33                 break;
34            }
35            return driver;
36     }
37
38     private WebDriver createRemoteDriver() {
39         throw new RuntimeException("RemoteWebDriver is not yet implemented");
40     }
41
42     private WebDriver createLocalDriver() {
43         switch (driverType) {
44         case FIREFOX :
45             System.setProperty(FIREFOX_DRIVER_PROPERTY, FileReaderManager.getInstance().getConfigReader().getDriverPath());
46             driver = new FirefoxDriver();
47             break;
48         case CHROME :
49             System.setProperty(CHROME_DRIVER_PROPERTY, FileReaderManager.getInstance().getConfigReader().getDriverPath());
```

Figure 18 - WebDriverManager

Test Context and Scenario Context

Test Context uses cucumber-picocontainer library to divide StepDefinition files into multiple classes and also to ensure Singleton Design Pattern for Page Object Manager, and WebDriverManager [12]. The Singleton Design Pattern [8] ensures that single instances of PageObjectManager.class and WebDriverManager.class are used throughout the test run and there is a global point of access to it.

TestContext.java ScenarioContext.java

```
1  package org.oscarehr.consultations.cucumber;
2
3  import org.oscarehr.consultations.managers.PageObjectManager;
5
6  public class TestContext {
7      private WebDriverManager webDriverManager;
8      private PageObjectManager pageObjectManager;
9      public ScenarioContext scenarioContext;
10
11     public TestContext(){
12         webDriverManager = new WebDriverManager();
13         pageObjectManager = new PageObjectManager(webDriverManager.getDriver());
14         scenarioContext = new ScenarioContext();
15     }
16
17     public WebDriverManager getWebDriverManager() {
18         return webDriverManager;
19     }
20
21     public PageObjectManager getPageObjectManager() {
22         return pageObjectManager;
23     }
24
25     public ScenarioContext getScenarioContext() {
26         return scenarioContext;
27     }
28
29 }
30
```

Figure 19 - TestContext.java

ScenarioContext is used to share dynamic data (generated during a test run) among different Stepdefinition files [13].

TestContext.java ScenarioContext.java

```java
package org.oscarehr.consultations.cucumber;

import java.util.HashMap;

public class ScenarioContext {

    private  Map<String, Object> scenarioContext;

        public ScenarioContext(){
            scenarioContext = new HashMap<>();
        }

        public void setContext(Context key, Object value) {
            scenarioContext.put(key.toString(), value);
        }

        public Object getContext(Context key){
            return scenarioContext.get(key.toString());
        }

        public Boolean isContains(Context key){
            return scenarioContext.containsKey(key.toString());
        }

}
```

Figure 20 - ScenarioContext.java

Integration of Automation Test Suite in Oscar-EMR repository

A small test case will be picked from Oscar Atlassian Wiki and all the steps taken to implement Cucumber-Selenium framework for this particular test case will be explained. This will involve several steps like adding libraries to Oscar repository, creating basic Selenium tests, designing Step Definition files, creating Page Objects, adding Manager classes and publishing test Result Reports. This section will help future developers to have a deep understanding of the architecture and concepts so that it is easier for them add Steps for remaining Test Cases.

Following is the High-Level Design for reference:

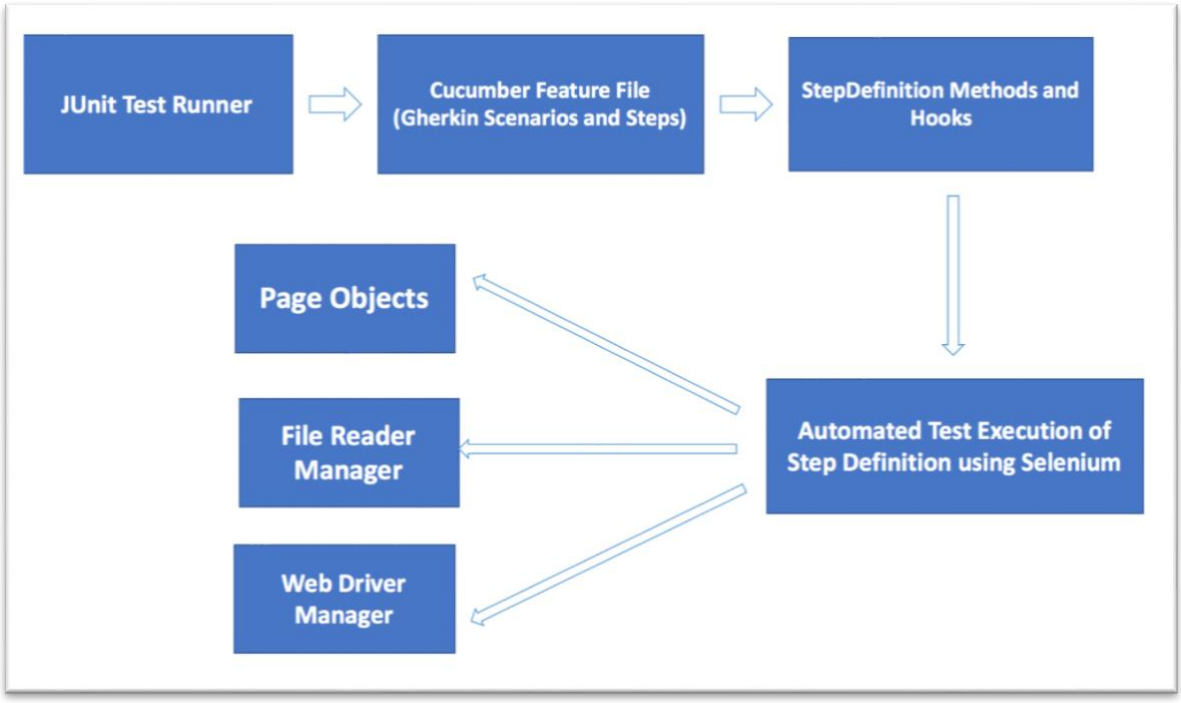


Figure 21 - High Level Design

Setting up Oscar repository from git

The primary up to date oscar repository is located at link https://bitbucket.org/oscaremr/oscar. This repository has been mirrored and forked by UVic Lead Labs Team under scoophealth account at https://github.com/scoophealth/oscar.

The scoophealth repository has been further forked and a new branch *hrshjain/system-integration-testing* has been created for the project. The latest code can be found at link https://github.com/hrshjain/oscar/tree/hrshjain/system-integration-testing and pull request can be reviewed at https://github.com/hrshjain/oscar/pull/1.

If developers are using Eclipse IDE for this project, then it is highly recommended to install Cucumber Eclipse Plugin by selecting Help –> Install New Software and adding link ***http://cucumber.github.com/cucumber-eclipse/update-site*** as location. Also, during the process of adding code, if eclipse displays error of missing selenium/cucumber package and also provides a fix to add packages, developers can go ahead and add those packages in Class files.

Add Selenium and Cucumber Library Dependencies

This is the first crucial step in implementing Cucumber Selenium Automation framework in Oscar EMR repository. Following Selenium and cucumber dependencies should be added to the pom.xml file:

1. selenium-java 3.14.0
2. cucumber-java 1.2.5
3. cucumber-jvm-deps 1.0.5
4. cucumber-junit 1.2.5
5. cucumber-picocontainer 1.2.5

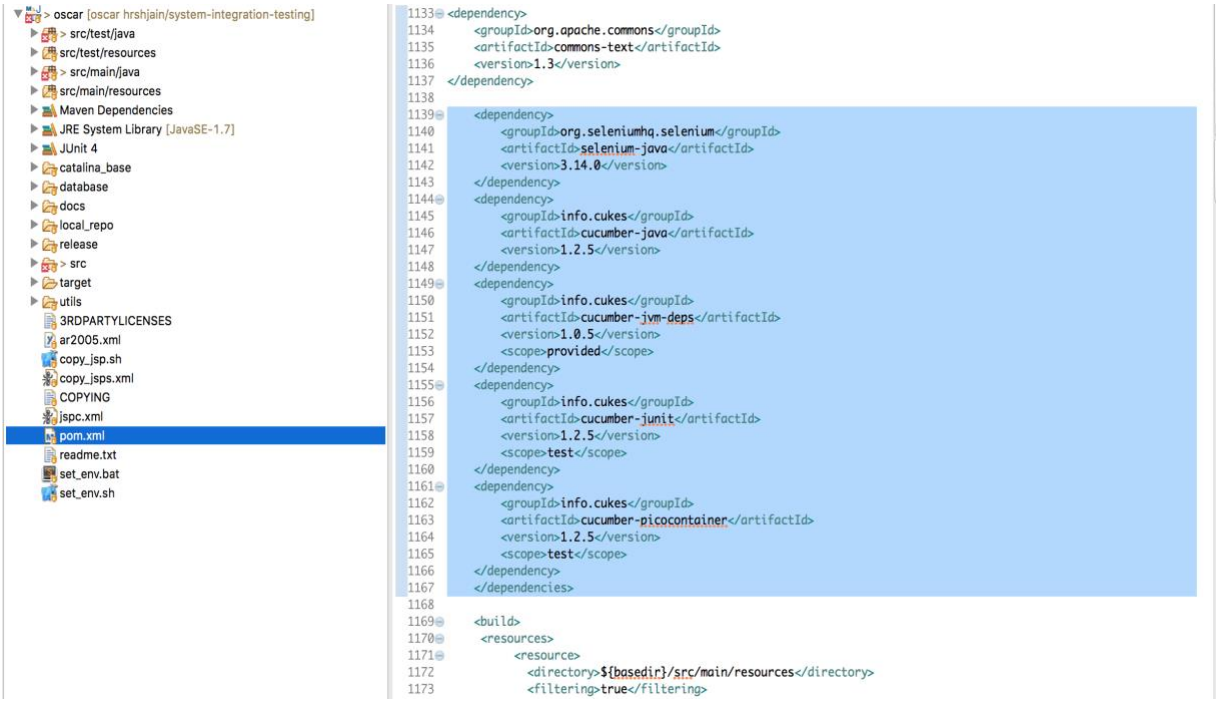


Figure 22 - pom.xml

Code snippet of dependencies added to pom.xml file:

```
      <dependency>

<groupId>org.seleniumhq.selenium</
groupId>
            <artifactId>selenium-
java</artifactId>

<version>3.14.0</version>
      </dependency>
      <dependency>

<groupId>info.cukes</groupId>
            <artifactId>cucumber-
java</artifactId>
            <version>1.2.5</version>
      </dependency>
      <dependency>

<groupId>info.cukes</groupId>
            <artifactId>cucumber-
jvm-deps</artifactId>
            <version>1.0.5</version>
            <scope>provided</scope>
      </dependency>
      <dependency>

<groupId>info.cukes</groupId>
            <artifactId>cucumber-
junit</artifactId>
            <version>1.2.5</version>
            <scope>test</scope>
      </dependency>
      <dependency>

<groupId>info.cukes</groupId>
            <artifactId>cucumber-
picocontainer</artifactId>
            <version>1.2.5</version>
            <scope>test</scope>
      </dependency>
      </dependencies>
```

Figure 23 - Adding Selenium and Cucumber dependencies in pom.xml

After updating pom.xml file, oscar maven project can be updated with these by performing following two operations:

a. Right Click on Project -> Select “Run as” -> Select “Maven Clean”

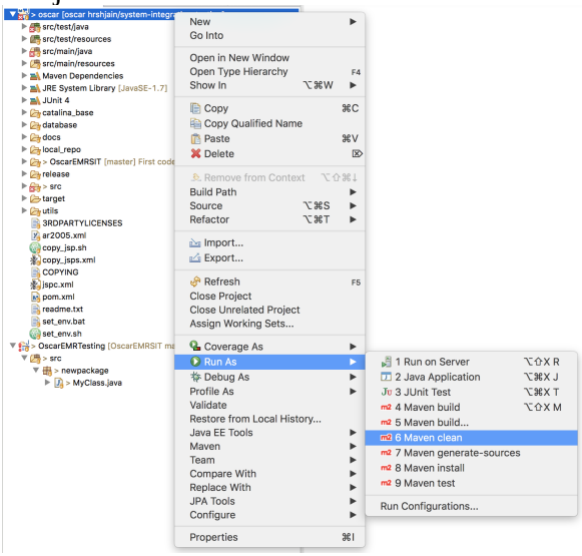


Figure 24 - Maven Clean

b. Right Click on Project -> Select “Maven” -> Select “Update Project”

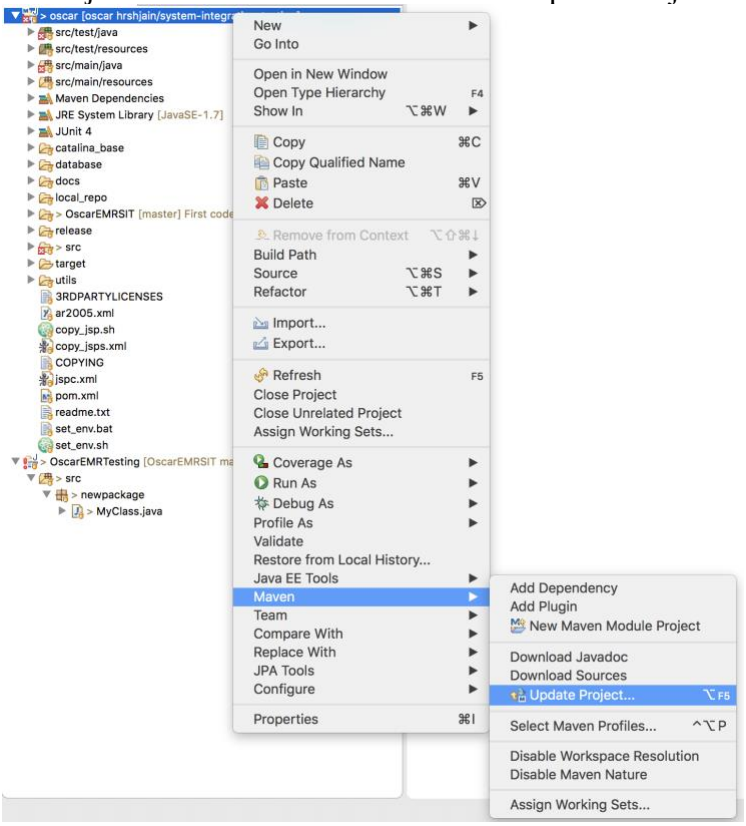


Figure 25 - Maven Update Project

Select Test Cases to Automate from Oscar Wiki

For this report, the following highlighted manual test case has been picked from *Oscar Note Screen Test Cases* to demonstrate how they will be converted to Automated BDD Test scripts using Cucumber-Selenium Framework.

1.0 Test Consultation Response/ Request Letterhead Defaults to the Logged in User

Step ID	Description	Precondition	Action	Expected Results	Actual Results	Pass/Fail	Notes
#1	Consultation Response/ Request Letterhead should default to the logged-in user	1. Consultation Response/ Request screen is present 2. User is logged in with correct credentials	Verify that when new Consult Note is started, the default letterhead selection displays for the current Oscar user logged-in	Default letterhead selection should be for the current Oscar user logged-in			
#2	Ability to select other Letterheads selection	Provider records are registered within the OSCAR database	1. Select Letterhead drop down 2. Verify that other Letterheads can be selected using drop down Letterhead field	1. Letterhead is selectable from the drop down list 2. User should be able to select other letterhead name from the drop down			
#3	Ability to save Consultation Response/ Request		Select Save button	1. Save button is available on the floating bar 2. Consultation Request/ Response is saved			
#4	Ability to display Consultation Response/ Request print preview screen		Select Print Preview button	1. Consultation Request/ Response Print preview window pops up 2. Selected Letterhead should populate in FROM section			

Figure 26 - Sample Test Case to Demonstrate Selenium Cucumber Framework

The test case can be found at link:
https://oscaremr.atlassian.net/wiki/spaces/OS/pages/54001741/OSCAR+15+Consultation+Note+Screen+Test+Cases.

Convert Test Case to Cucumber Scenario and Steps

The above highlighted test case will now be converted to Cucumber Scenario and Steps. Under *src/test/resources* in oscar repository, create a new folder *cucumberSeleniumFramework*. Within this folder, create the Cucumber feature file with name *consultationNoteScreen.feature*. This feature file will contain the Cucumber Scenario and Steps.

The Cucumber feature file [6] contains:

1. Header with Author information
2. *Feature:* A single statement describing the feature which is being tested
3. *Scenario:* The scenario which is being tested. Annotation is also recommended at the top of Scenario.
4. *Gherkin Test Steps*: These are the When/Then/And/But statements written in Gherkin language. They specify the test steps which need to be taken to test a particular scenario [3].

A feature file has one Feature specification and each feature specification can have multiple scenarios with each Scenario having multiple Gherkin steps. Annotation are used for scenarios to give additional information and their running sequence with respect to Hooks. The heading of the selected test case *1.0TestConsultationResponseRequestLetterhead* is being used as the Scenario annotation for convenience and tracking purpose.

Following is a snapshot of the Feature file and the Scenario/Steps derived from the selected test case:

Screenshot:

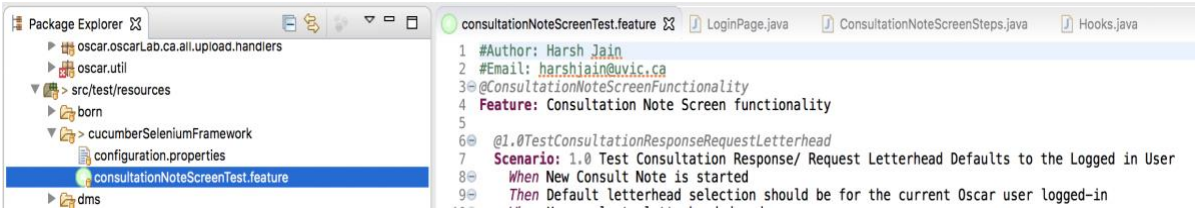


Figure 27 - Scenario and Gherkin Steps in Cucumber Feature File

Code Snippet:

```
#Author: Harsh Jain
```

#Email: harshjain@uvic.ca

@ConsultationNoteScreenFunctionality

Feature: Consultation Note Screen functionality

@1.0TestConsultationResponseRequestLetterhead

Scenario: 1.0 Test Consultation Response/ Request Letterhead Defaults to the Logged in User

When New Consult Note is started

Then Default letterhead selection should be for the current Oscar user logged-in

Configure JUnit Test Runner

Create a JUnit test runner to run the above steps in the form of JUnit test cases. Create package *org.oscarehr.consultations.runners* under *src/test/java* folder of the oscar repository. Create a new file CucumberSeleniumTestRunner.java. in the package. Following is the screenshot and code of the JUnit test runner:

Screenshot:

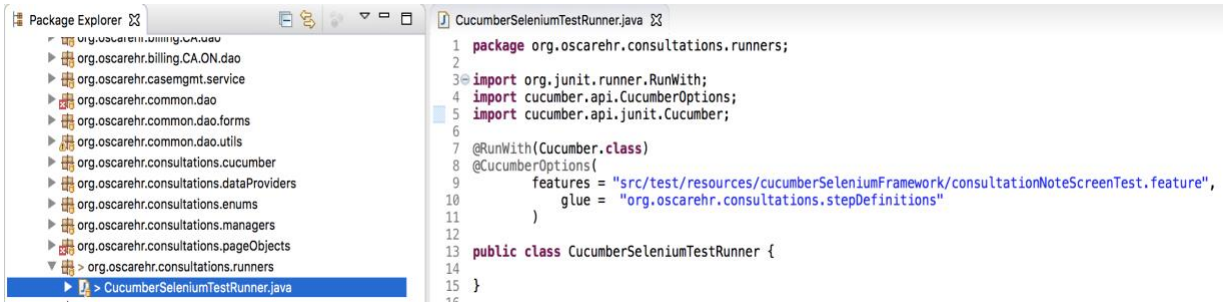


Figure 28 - Design JUnit Test Runner

Code Snippet:

```
1.  package org.oscarehr.consultations.runners;
2.
3.  import org.junit.runner.RunWith;
4.  import cucumber.api.CucumberOptions;
5.  import cucumber.api.junit.Cucumber;
6.
7.  @RunWith(Cucumber.class)
8.  @CucumberOptions(
9.          features = "src/test/resources/cucumberSeleniumFramework/consultationNoteScreenTest.feature",

10.             glue =  "org.oscarehr.consultations.stepDefinitions"
11.         )
12.
13. public class CucumberSeleniumTestRunner {
14.
15. }
```

@RunWith(Cucumber.**class**) annotation is used to ensure that the JUnit runner runs with Cucumber class. @CucumberOptions annotation is used to specify the feature files that will be covered as part of the JUnit test. Only one feature file *consultationNoteScreenTest.feature* is used for now. However, in case of multiple feature files, relative path of feature files should be added to features option separated by a comma.

Run the JUnit test by selecting Right Click and selecting Run as -> JUnit test.

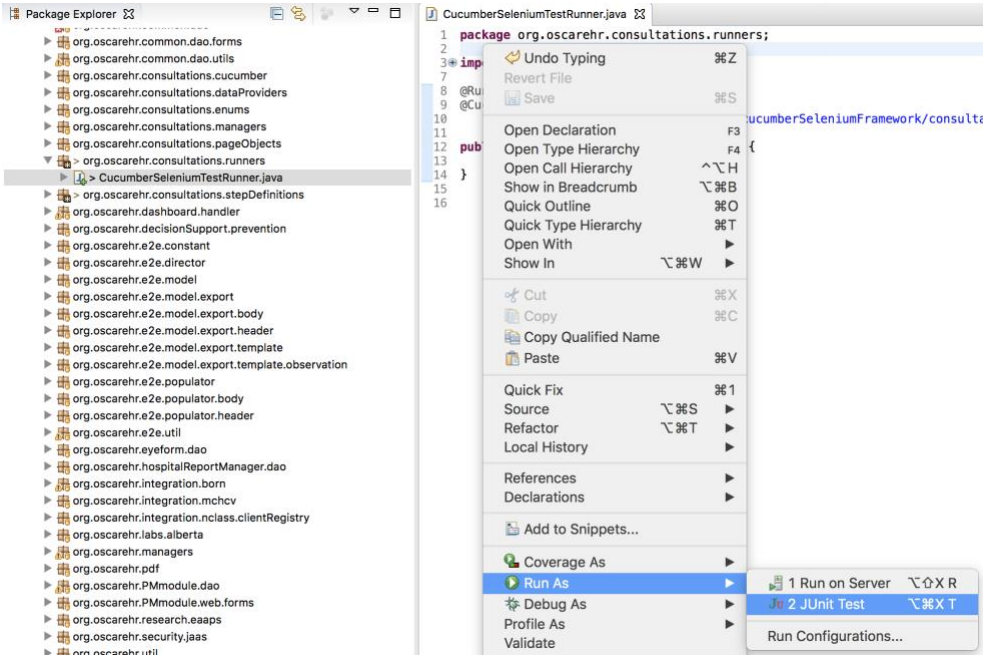


Figure 29 - Run JUnit Test

Following output should be displayed in the console logs:

```
You can implement missing steps with the snippets below:

@When("^New Consult Note is started$")
public void new_Consult_Note_is_started() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Then("^Default letterhead selection should be for the current Oscar user logged-in$")
public void default_letterhead_selection_should_be_for_the_current_Oscar_user_logged_in() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```

Figure 30 - Sample Step Definition Methods matching Gherkin Steps

The above message indicates that JUnit test runner was successfully able to pick and run the Cucumber steps. However, these steps have not been implemented using a programming language.

Implementing the missing methods using Step Definition files

To implement the above missing steps using Java programming language, create a package with name *org.oscarehr.consultations.stepDefinitions* in *src/test/java* folder.

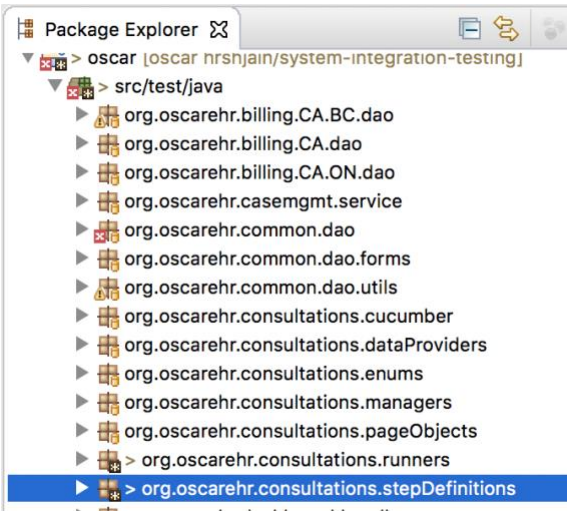


Figure 31 - Step Definition Package

In this package, create a new class *ConsultationNoteScreenSteps.java* and paste the missing methods from console logs to the class.

Screenshot:

Figure 32 - Step Definition Methods

Code Snippet:

```
1.  package org.oscarehr.consultations.stepDefinitions;
2.
3.  import cucumber.api.java.en.Then;
4.  import cucumber.api.java.en.When;
5.
6.  public class ConsultationNoteScreenSteps {
7.
8.      @When("^New Consult Note is started$")
9.      public void new_Consult_Note_is_started() {
10.         // Write code here that turns the phrase above into concrete actions
11.     }
```

```
12.
13.     @Then("^Default letterhead selection should be for the current Oscar user logged-in$")
14.     public void default_letterhead_selection_should_be_for_the_current_Oscar_user_logged_in(){
15.         // Write code here that turns the phrase above into concrete actions
16.     }
17. }
```

Navigate to JUnit test runner and add glue option in @CucumberOptions annotation. Glue option is used by Cucumber to locate the Step Definition file.

Screenshot:

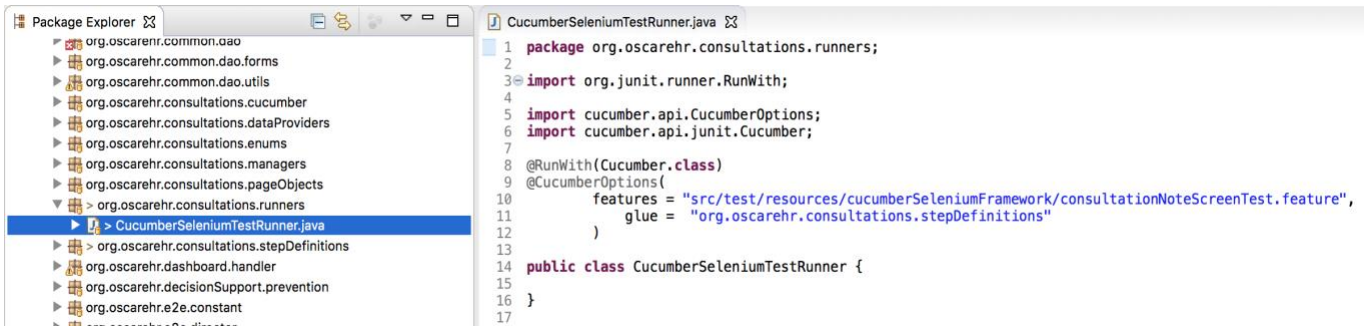


Figure 33 - Glue added in JUnit Test Runner

Code Snippet:

```
1.  package org.oscarehr.consultations.runners;
2.
3.  import org.junit.runner.RunWith;
4.
5.  import cucumber.api.CucumberOptions;
6.  import cucumber.api.junit.Cucumber;
7.
8.  @RunWith(Cucumber.class)
9.  @CucumberOptions(
10.         features = "src/test/resources/cucumberSeleniumFramework/consultationNoteScreenTest.feature",
11.             glue = "org.oscarehr.consultations.stepDefinitions"
12.         )
13.
14. public class CucumberSeleniumTestRunner {
15.
16. }
```

Run the JUnit test again. In the JUnit tab, you can observe that the Scenario and two steps are now passing.

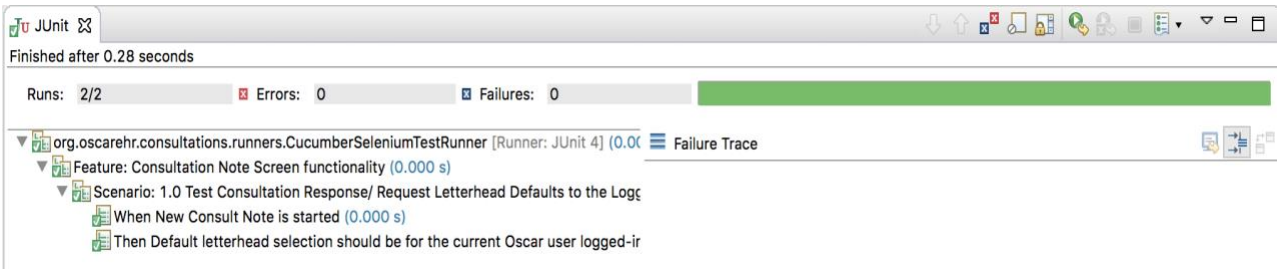


Figure 34 - Successful JUnit Test Run

Fill the Step Definition methods with Java and Selenium code

JUnit runner is now successfully running the cucumber steps implemented using StepDefinition files. However, the StepDefinition methods are not performing any automatic browser actions.

Add Selenium Java code in the two methods to perform automatic browser actions. Below code will automatically start Firefox web browser, log into Oscar EMR, navigate to Consultations requests and verify whether Consultation request letterhead is as expected or not.

Screenshot:

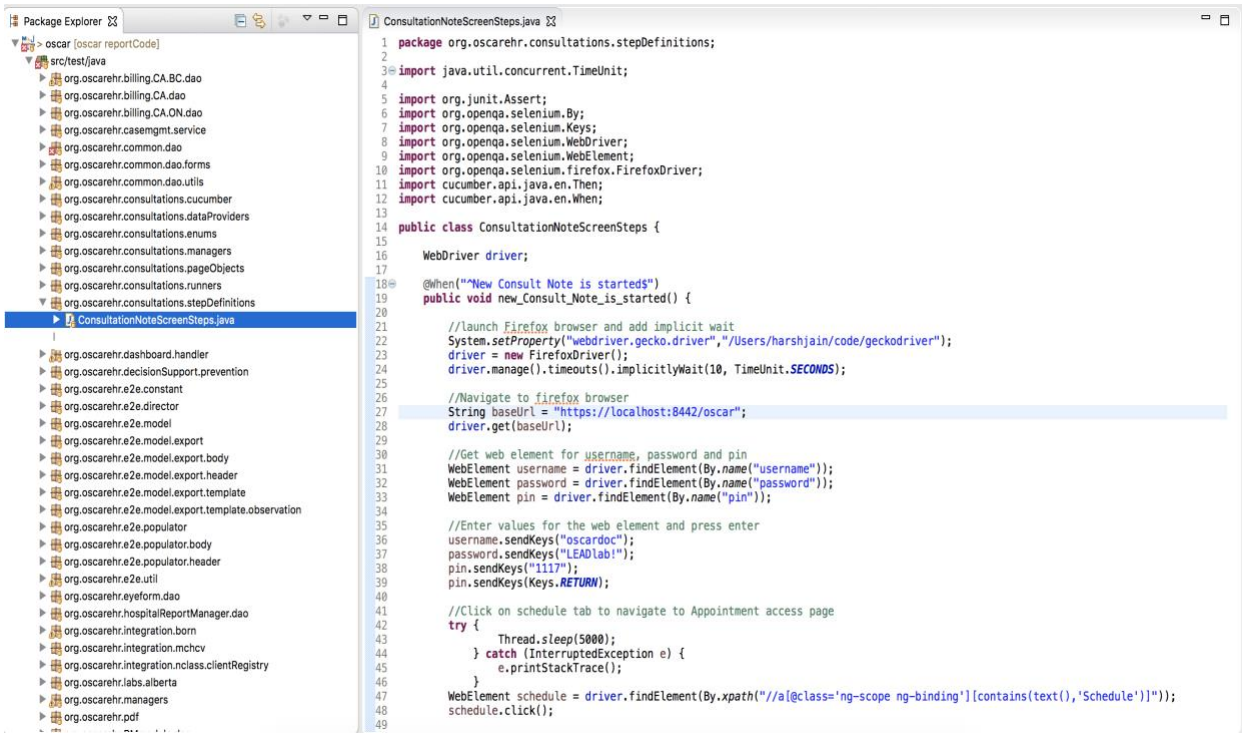


Figure 35 - Step Definition File Part 1

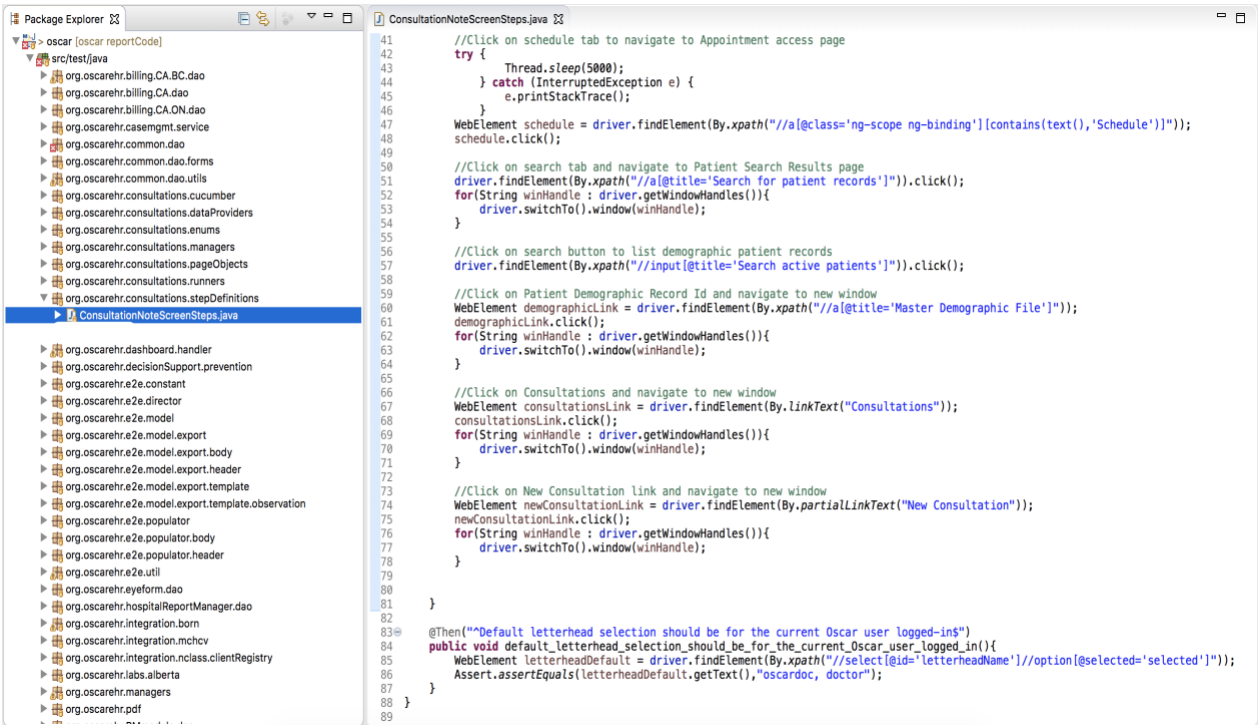


Figure 36 - Step Definition File Part 2

Code Snippet:

```
1.  package org.oscarehr.consultations.stepDefinitions;
2.
3.  import java.util.concurrent.TimeUnit;
4.
5.  import org.junit.Assert;
6.  import org.openqa.selenium.By;
7.  import org.openqa.selenium.Keys;
8.  import org.openqa.selenium.WebDriver;
9.  import org.openqa.selenium.WebElement;
10. import org.openqa.selenium.firefox.FirefoxDriver;
11. import cucumber.api.java.en.Then;
12. import cucumber.api.java.en.When;
13.
14. public class ConsultationNoteScreenSteps {
15.
16.     WebDriver driver;
17.
18.     @When("^New Consult Note is started$")
19.     public void new_Consult_Note_is_started() {
20.
21.         //launch Firefox browser and add implicit wait
22.         System.setProperty("webdriver.gecko.driver","/Users/harshjain/code/geckodriver");
23.         driver = new FirefoxDriver();
24.         driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
25.
26.         //Navigate to firefox browser
27.         String baseUrl = "https://localhost:8442/oscar";
28.         driver.get(baseUrl);
29.
```

```
30.          //Get web element for username, password and pin
31.          WebElement username = driver.findElement(By.name("username"));
32.          WebElement password = driver.findElement(By.name("password"));
33.          WebElement pin = driver.findElement(By.name("pin"));
34.
35.          //Enter values for the web element and press enter
36.          username.sendKeys("oscardoc");
37.          password.sendKeys("LEADlab!");
38.          pin.sendKeys("1117");
39.          pin.sendKeys(Keys.RETURN);
40.
41.          //Click on schedule tab to navigate to Appointment access page
42.          try {
43.                  Thread.sleep(5000);
44.              } catch (InterruptedException e) {
45.                  e.printStackTrace();
46.              }
47.          WebElement schedule = driver.findElement(By.xpath("//a[@class='ng-scope ng-
     binding'][contains(text(),'Schedule')]"));
48.          schedule.click();
49.
50.          //Click on search tab and navigate to Patient Search Results page
51.          driver.findElement(By.xpath("//a[@title='Search for patient records']")).click();
52.          for(String winHandle : driver.getWindowHandles()){
53.              driver.switchTo().window(winHandle);
54.          }
55.
56.          //Click on search button to list demographic patient records
57.          driver.findElement(By.xpath("//input[@title='Search active patients']")).click();
58.
59.          //Click on Patient Demographic Record Id and navigate to new window
60.          WebElement demographicLink = driver.findElement(By.xpath("//a[@title='Master Demographic File'
     ]"));
61.          demographicLink.click();
62.          for(String winHandle : driver.getWindowHandles()){
63.              driver.switchTo().window(winHandle);
64.          }
65.
66.          //Click on Consultations and navigate to new window
67.          WebElement consultationsLink = driver.findElement(By.linkText("Consultations"));
68.          consultationsLink.click();
69.          for(String winHandle : driver.getWindowHandles()){
70.              driver.switchTo().window(winHandle);
71.          }
72.
73.          //Click on New Consultation link and navigate to new window
74.          WebElement newConsultationLink = driver.findElement(By.partialLinkText("New Consultation"));
75.          newConsultationLink.click();
76.          for(String winHandle : driver.getWindowHandles()){
77.              driver.switchTo().window(winHandle);
78.          }
79.      }
80.
81.      @Then("^Default letterhead selection should be for the current Oscar user logged-in$")
82.      public void default_letterhead_selection_should_be_for_the_current_Oscar_user_logged_in(){
83.          //Verify letterhead is equal to current logged in user
84.          WebElement letterheadDefault = driver.findElement(By.xpath("//select[@id='letterheadName']//op
     tion[@selected='selected']"));
85.          Assert.assertEquals(letterheadDefault.getText(),"oscardoc, doctor");
86.
87.          //Close all browser instances
```

```
88.          for(String child : driver.getWindowHandles()) {
89.              driver.switchTo().window(child);
90.              driver.close();
91.          }
92.      }
93.  }
```

ConfigFileReader for Test Parameters and Data

Above code contains many hardcoded parameters values like oscar url and login credentials which have been used directly in the code. The hardcoded values will now be parameterised using *configuration.properties* file and *ConfigFileReader* class [9]. In the *cucumberSeleniumFramework* folder in *src/test/resources/* directory, create a new file *configuration.properties* and place all the parameter values in this file in the form of key, value pairs.

Screenshot:

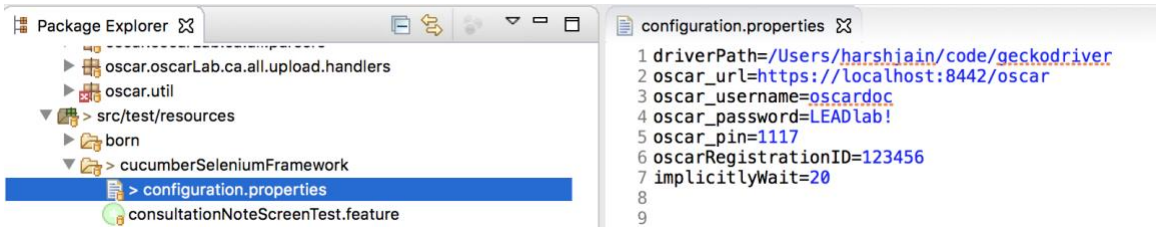


Figure 37 - Configuration.properties

Code Snippet:

```
1.  driverPath=/Users/harshjain/code/geckodriver
2.  oscar_url=https://localhost:8442/oscar
3.  oscar_username=oscardoc
4.  oscar_password=LEADlab!
5.  oscar_pin=1117
6.  oscarRegistrationID=123456
7.  implicitlyWait=20
```

To access these key values, create a new package *org.oscarehr.consultations.dataProviders* in *src/test/java* and create a new Class *ConfigFileReader*.java in the package.

In the ConfigFileReader constructor, inbuilt java classes File Reader and Buffered Reader are used to read key values from configuration.properties file. For every key in the *configuration.properties* file, a corresponding get method should be present in *ConfigFileReader.java* class.

Screenshot:

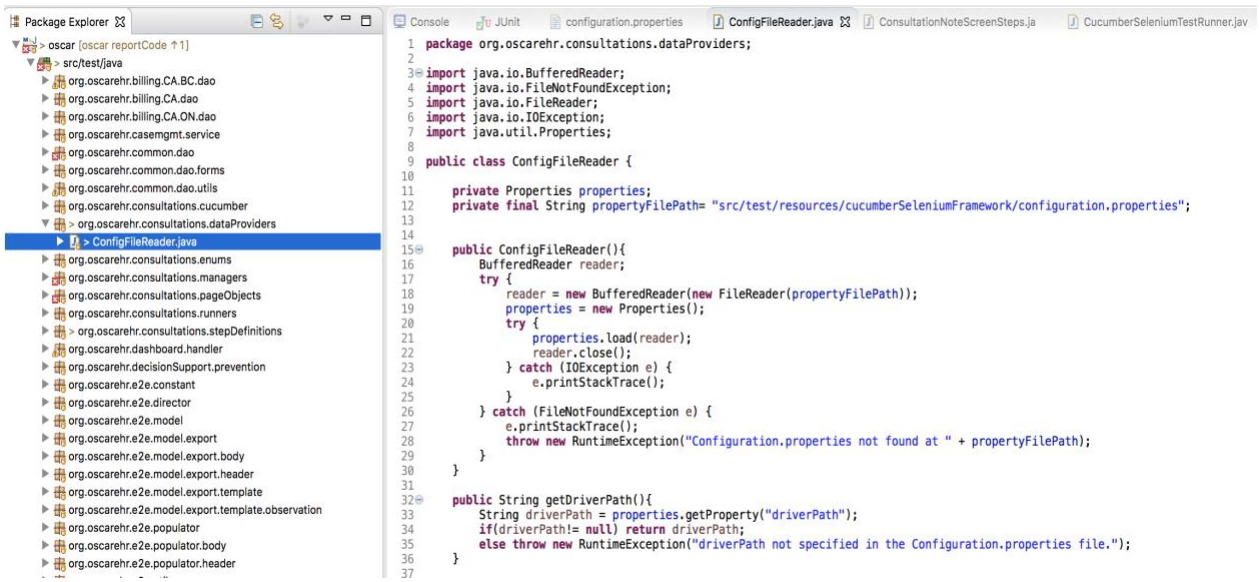


Figure 38 - ConfigFileReader.java

Code Snippet:

```
1.  package org.oscarehr.consultations.dataProviders;
2.
3.  import java.io.BufferedReader;
4.  import java.io.FileNotFoundException;
5.  import java.io.FileReader;
6.  import java.io.IOException;
7.  import java.util.Properties;
8.
9.  public class ConfigFileReader {
10.
11.     private Properties properties;
12.     private final String propertyFilePath= "src/test/resources/cucumberSeleniumFramework/configuration
    .properties";
13.
14.
15.     public ConfigFileReader(){
16.         BufferedReader reader;
17.         try {
18.             reader = new BufferedReader(new FileReader(propertyFilePath));
19.             properties = new Properties();
20.             try {
21.                 properties.load(reader);
22.                 reader.close();
23.             } catch (IOException e) {
24.                 e.printStackTrace();
25.             }
26.         } catch (FileNotFoundException e) {
27.             e.printStackTrace();
28.             throw new RuntimeException("Configuration.properties not found at " + propertyFilePath);
29.         }
30.     }
31.
32.     public String getDriverPath(){
33.         String driverPath = properties.getProperty("driverPath");
```

```
34.          if(driverPath!= null) return driverPath;
35.          else throw new RuntimeException("driverPath not specified in the Configuration.properties file
     .");
36.      }
37.
38.      public long getImplicitlyWait() {
39.          String implicitlyWait = properties.getProperty("implicitlyWait");
40.          if(implicitlyWait != null) {
41.              try{
42.                  return Long.parseLong(implicitlyWait);
43.              }catch(NumberFormatException e) {
44.                  throw new RuntimeException("Not able to parse value : " + implicitlyWait + " in to Lon
     g");
45.              }
46.          }
47.          return 30;
48.      }
49.
50.      public String getApplicationUrl() {
51.          String url = properties.getProperty("oscar_url");
52.          if(url != null) return url;
53.          else throw new RuntimeException("url not specified in the Configuration.properties file.");
54.      }
55.
56.      public String getOscarUsername() {
57.          String oscar_username = properties.getProperty("oscar_username");
58.          if(oscar_username != null) return oscar_username;
59.          else throw new RuntimeException("url not specified in the Configuration.properties file.");
60.      }
61.
62.      public String getOscarPassword() {
63.          String url = properties.getProperty("oscar_password");
64.          if(url != null) return url;
65.          else throw new RuntimeException("url not specified in the Configuration.properties file.");
66.      }
67.
68.      public String getOscarPin() {
69.          String url = properties.getProperty("oscar_pin");
70.          if(url != null) return url;
71.          else throw new RuntimeException("url not specified in the Configuration.properties file.");
72.      }
73.
74.      public String getRegistrationID() {
75.          String url = properties.getProperty("oscarRegistrationID");
76.          if(url != null) return url;
77.          else throw new RuntimeException("url not specified in the Configuration.properties file.");
78.      }
79.  }
```

In the StepDefinition file, ConfigFileReader class is instantiated and hard coded values are replaced by their respective ConfigFileReader methods to fetch the parameter values.

Screenshot:

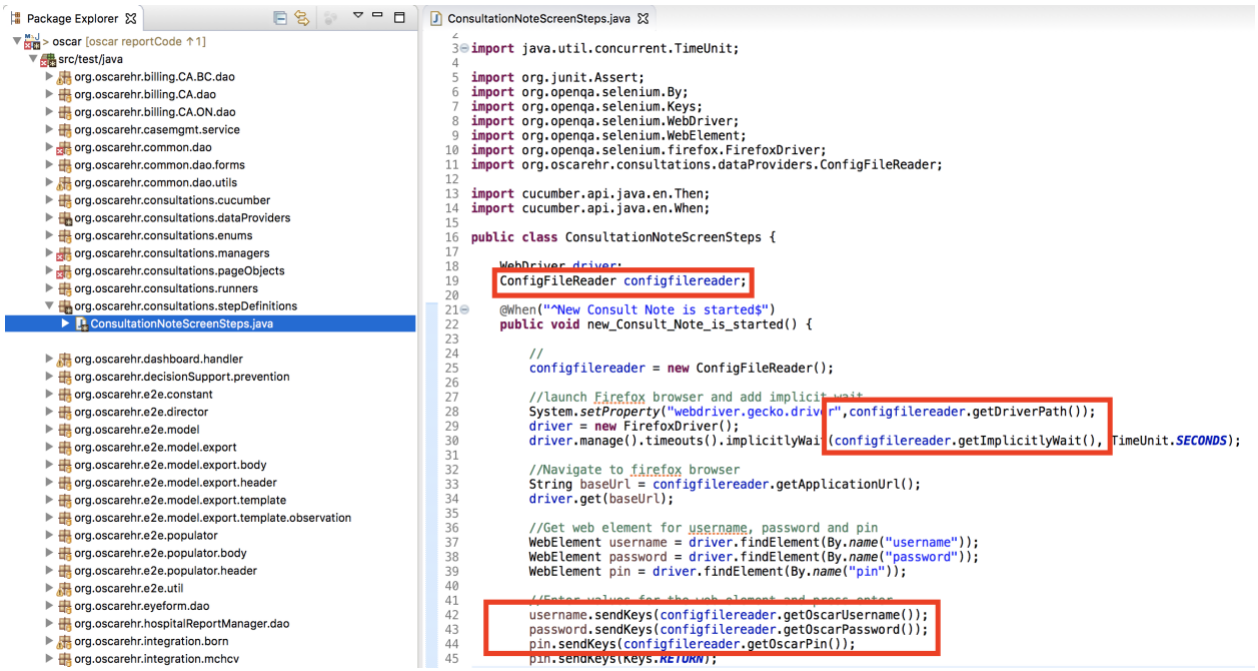


Figure 39 - ConfigFileReader in StepDefinition

Code Snippet:

```
1.  package org.oscarehr.consultations.stepDefinitions;
2.
3.  import java.util.concurrent.TimeUnit;
4.
5.  import org.junit.Assert;
6.  import org.openqa.selenium.By;
7.  import org.openqa.selenium.Keys;
8.  import org.openqa.selenium.WebDriver;
9.  import org.openqa.selenium.WebElement;
10. import org.openqa.selenium.firefox.FirefoxDriver;
11. import org.oscarehr.consultations.dataProviders.ConfigFileReader;
12.
13. import cucumber.api.java.en.Then;
14. import cucumber.api.java.en.When;
15.
16. public class ConsultationNoteScreenSteps {
17.
18.     WebDriver driver;
19.     ConfigFileReader configfilereader;
20.
21.     @When("^New Consult Note is started$")
22.     public void new_Consult_Note_is_started() {
23.
24.         //
25.         configfilereader = new ConfigFileReader();
26.
27.         //launch Firefox browser and add implicit wait
28.         System.setProperty("webdriver.gecko.driver",configfilereader.getDriverPath());
29.         driver = new FirefoxDriver();
30.         driver.manage().timeouts().implicitlyWait(configfilereader.getImplicitlyWait(), TimeUnit.SECON
    DS);
31.
```

```
32.          //Navigate to firefox browser
33.          String baseUrl = configfilereader.getApplicationUrl();
34.          driver.get(baseUrl);
35.
36.          //Get web element for username, password and pin
37.          WebElement username = driver.findElement(By.name("username"));
38.          WebElement password = driver.findElement(By.name("password"));
39.          WebElement pin = driver.findElement(By.name("pin"));
40.
41.          //Enter values for the web element and press enter
42.          username.sendKeys(configfilereader.getOscarUsername());
43.          password.sendKeys(configfilereader.getOscarPassword());
44.          pin.sendKeys(configfilereader.getOscarPin());
45.          pin.sendKeys(Keys.RETURN);
46.
47.          //Click on schedule tab to navigate to Appointment access page
48.          try {
49.                  Thread.sleep(5000);
50.              } catch (InterruptedException e) {
51.                  e.printStackTrace();
52.              }
53.          WebElement schedule = driver.findElement(By.xpath("//a[@class='ng-scope ng-
    binding'][contains(text(),'Schedule')]"));
54.          schedule.click();
55.
56.          //Click on search tab and navigate to Patient Search Results page
57.          driver.findElement(By.xpath("//a[@title='Search for patient records']")).click();
58.          for(String winHandle : driver.getWindowHandles()){
59.              driver.switchTo().window(winHandle);
60.          }
61.
62.          //Click on search button to list demographic patient records
63.          driver.findElement(By.xpath("//input[@title='Search active patients']")).click();
64.
65.          //Click on Patient Demographic Record Id and navigate to new window
66.          WebElement demographicLink = driver.findElement(By.xpath("//a[@title='Master Demographic File'
    ]"));
67.          demographicLink.click();
68.          for(String winHandle : driver.getWindowHandles()){
69.              driver.switchTo().window(winHandle);
70.          }
71.
72.          //Click on Consultations and navigate to new window
73.          WebElement consultationsLink = driver.findElement(By.linkText("Consultations"));
74.          consultationsLink.click();
75.          for(String winHandle : driver.getWindowHandles()){
76.              driver.switchTo().window(winHandle);
77.          }
78.
79.          //Click on New Consultation link and navigate to new window
80.          WebElement newConsultationLink = driver.findElement(By.partialLinkText("New Consultation"));
81.          newConsultationLink.click();
82.          for(String winHandle : driver.getWindowHandles()){
83.              driver.switchTo().window(winHandle);
84.          }
85.
86.
87.      }
88.
89.      @Then("^Default letterhead selection should be for the current Oscar user logged-in$")
90.      public void default_letterhead_selection_should_be_for_the_current_Oscar_user_logged_in(){
```

```
91.         //Verify letterhead is equal to current logged in user
92.         WebElement letterheadDefault = driver.findElement(By.xpath("//select[@id='letterheadName']//option[@selected='selected']"));
93.         Assert.assertEquals(letterheadDefault.getText(),configfilereader.getOscarUsername() + ", doctor");
94.
95.         //Close all browser instances
96.         for(String child : driver.getWindowHandles()) {
97.             driver.switchTo().window(child);
98.             driver.close();
99.         }
100.    }
101. }
```

FileReaderManager

When the project gets bigger and multiple StepDefinition files are used, ConfigFileReader will need to be instantiated in each of the StepDefinition files. This will create unnecessary objects and is against the Singleton Design Pattern. To resolve this issue, FileReaderManager will be used to make sure there is a single *ConfigFileReader* instance used throughout the test run and there is a single point to access it.
FileReaderManager uses getInstance() method to return its own static instance and uses getConfigReader() method to return static instance of *ConfigFileReader* class. Create a new package *org.oscarehr.consultations.managers* in *src/test/java* directory and create a new class *FileReaderManager.java* in the package.

Screenshot:

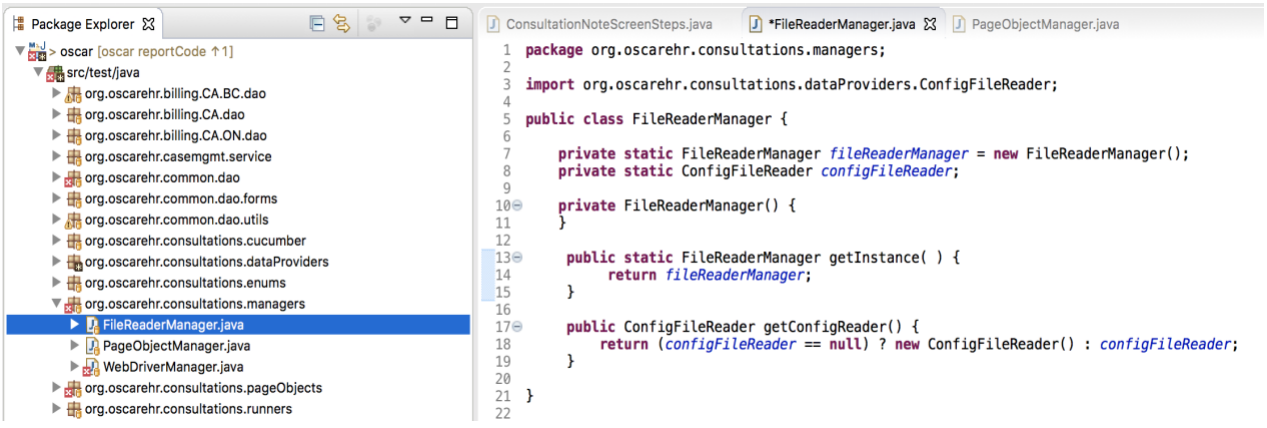


Figure 40 - FileReaderManager

Code Snippet:

```
1.  package org.oscarehr.consultations.managers;
2.
3.  import org.oscarehr.consultations.dataProviders.ConfigFileReader;
4.
5.  public class FileReaderManager {
```

```
6.
7.      private static FileReaderManager fileReaderManager = new FileReaderManager();
8.      private static ConfigFileReader configFileReader;
9.
10.     private FileReaderManager() {
11.     }
12.
13.      public static FileReaderManager getInstance( ) {
14.           return fileReaderManager;
15.      }
16.
17.      public ConfigFileReader getConfigReader() {
18.          return (configFileReader == null) ? new ConfigFileReader() : configFileReader;
19.      }
20. }
```

Update the code in Step Definition File to use FileReaderManager instead of ConfigFileReader.

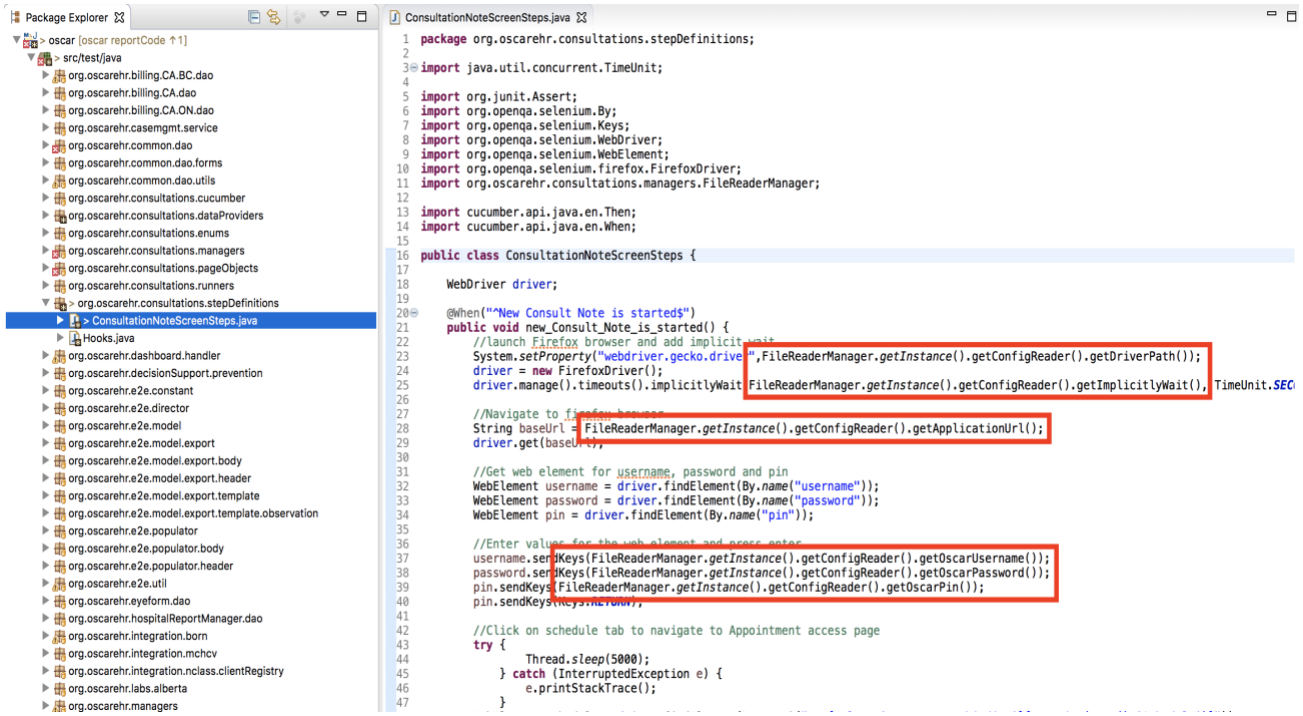


Figure 41 - Using FileReaderManager in Step Definition File

Code Snippet:

```
1.  package org.oscarehr.consultations.stepDefinitions;
2.
3.  import java.util.concurrent.TimeUnit;
4.
5.  import org.junit.Assert;
6.  import org.openqa.selenium.By;
7.  import org.openqa.selenium.Keys;
8.  import org.openqa.selenium.WebDriver;
9.  import org.openqa.selenium.WebElement;
10. import org.openqa.selenium.firefox.FirefoxDriver;
```

```
11. import org.oscarehr.consultations.managers.FileReaderManager;
12. 
13. import cucumber.api.java.en.Then;
14. import cucumber.api.java.en.When;
15. 
16. public class ConsultationNoteScreenSteps {
17. 
18.     WebDriver driver;
19. 
20.     @When("^New Consult Note is started$")
21.     public void new_Consult_Note_is_started() {
22.         //launch Firefox browser and add implicit wait
23.         System.setProperty("webdriver.gecko.driver",FileReaderManager.getInstance().getConfigReader().
    getDriverPath());
24.         driver = new FirefoxDriver();
25.         driver.manage().timeouts().implicitlyWait(FileReaderManager.getInstance().getConfigReader().ge
    tImplicitlyWait(), TimeUnit.SECONDS);
26. 
27.         //Navigate to firefox browser
28.         String baseUrl = FileReaderManager.getInstance().getConfigReader().getApplicationUrl();
29.         driver.get(baseUrl);
30. 
31.         //Get web element for username, password and pin
32.         WebElement username = driver.findElement(By.name("username"));
33.         WebElement password = driver.findElement(By.name("password"));
34.         WebElement pin = driver.findElement(By.name("pin"));
35. 
36.         //Enter values for the web element and press enter
37.         username.sendKeys(FileReaderManager.getInstance().getConfigReader().getOscarUsername());
38.         password.sendKeys(FileReaderManager.getInstance().getConfigReader().getOscarPassword());
39.         pin.sendKeys(FileReaderManager.getInstance().getConfigReader().getOscarPin());
40.         pin.sendKeys(Keys.RETURN);
41. 
42.         //Click on schedule tab to navigate to Appointment access page
43.         try {
44.                 Thread.sleep(5000);
45.             } catch (InterruptedException e) {
46.                 e.printStackTrace();
47.             }
48.         WebElement schedule = driver.findElement(By.xpath("//a[@class='ng-scope ng-
    binding'][contains(text(),'Schedule')]"));
49.         schedule.click();
50. 
51.         //Click on search tab and navigate to Patient Search Results page
52.         driver.findElement(By.xpath("//a[@title='Search for patient records']")).click();
53.         for(String winHandle : driver.getWindowHandles()){
54.             driver.switchTo().window(winHandle);
55.         }
56. 
57.         //Click on search button to list demographic patient records
58.         driver.findElement(By.xpath("//input[@title='Search active patients']")).click();
59. 
60.         //Click on Patient Demographic Record Id and navigate to new window
61.         WebElement demographicLink = driver.findElement(By.xpath("//a[@title='Master Demographic File'
    ]"));
62.         demographicLink.click();
63.         for(String winHandle : driver.getWindowHandles()){
64.             driver.switchTo().window(winHandle);
65.         }
66. 
67.         //Click on Consultations and navigate to new window
```

```
68.          WebElement consultationsLink = driver.findElement(By.linkText("Consultations"));
69.          consultationsLink.click();
70.          for(String winHandle : driver.getWindowHandles()){
71.              driver.switchTo().window(winHandle);
72.          }
73.
74.          //Click on New Consultation link and navigate to new window
75.          WebElement newConsultationLink = driver.findElement(By.partialLinkText("New Consultation"));
76.          newConsultationLink.click();
77.          for(String winHandle : driver.getWindowHandles()){
78.              driver.switchTo().window(winHandle);
79.          }
80.
81.
82.      }
83.
84.      @Then("^Default letterhead selection should be for the current Oscar user logged-in$")
85.      public void default_letterhead_selection_should_be_for_the_current_Oscar_user_logged_in(){
86.          //Verify letterhead is equal to current logged in user
87.          WebElement letterheadDefault = driver.findElement(By.xpath("//select[@id='letterheadName']//op
    tion[@selected='selected']"));
88.          Assert.assertEquals(letterheadDefault.getText(),FileReaderManager.getInstance().getConfigReade
    r().getOscarUsername() + ", doctor");
89.
90.          //Close all browser instances
91.          for(String child : driver.getWindowHandles()) {
92.              driver.switchTo().window(child);
93.              driver.close();
94.          }
95.      }
96.  }
```

WebDriver Manager

Currently, Firefox driver instance is initiated within the StepDefinition file. In case of multiple Stepdefinition files and PageObjects, a single instance of driver should be used throughout the test run. Additionally, WebDriver functionality and managing browser instances should be handled not be handled within StepDefinition files but by a separate class.

Hence, WebDriverManager class is used to ensure that same browser instance is used in multiple Step Definitions and PageObjects. Having a separate WebDriver Manager will also improve flexibility and coverage by allowing testing in different browsers (chrome, internet explorer) and in different environments (local, remote).

Enums.java class is used to provide different browser and environment options to users. Create a new package *org.oscarehr.consultations.enums* in *src/test/java* directory. Within the package, create two classes DriverType.java and EnvironmentType.java to manage browser type and environments respectively.

Screenshots:

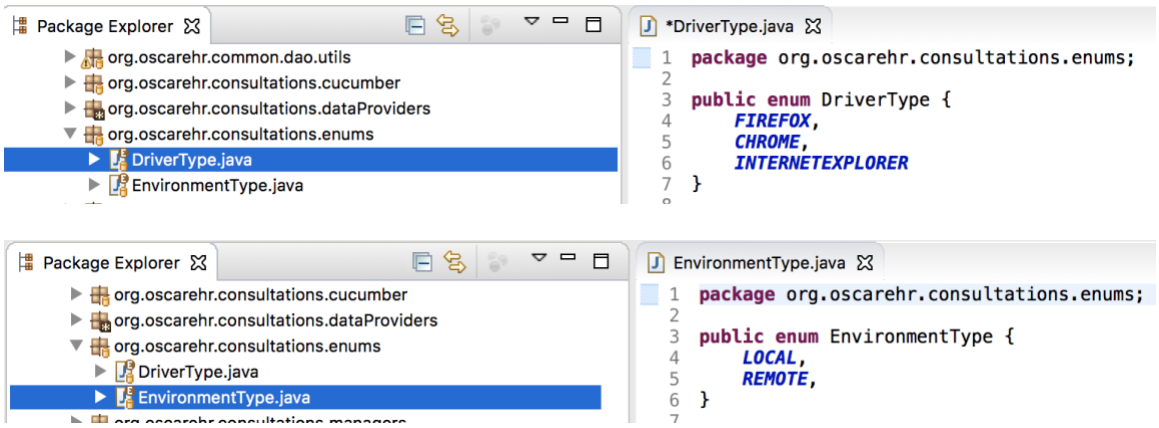


Figure 42 - Enums (DriverType and EnvironmentType)

Code Snippets:

```
1.  package org.oscarehr.consultations.enums;
2.
3.  public enum DriverType {
4.      FIREFOX,
5.      CHROME,
6.      INTERNETEXPLORER
7.  }
```

```
1.  package org.oscarehr.consultations.enums;
2.
3.  public enum EnvironmentType {
4.      LOCAL,
5.      REMOTE,
6.  }
```

Add browser and environment parameters in Configuration.properties file and their corresponding methods in ConfigFileReader class.

Screenshots:

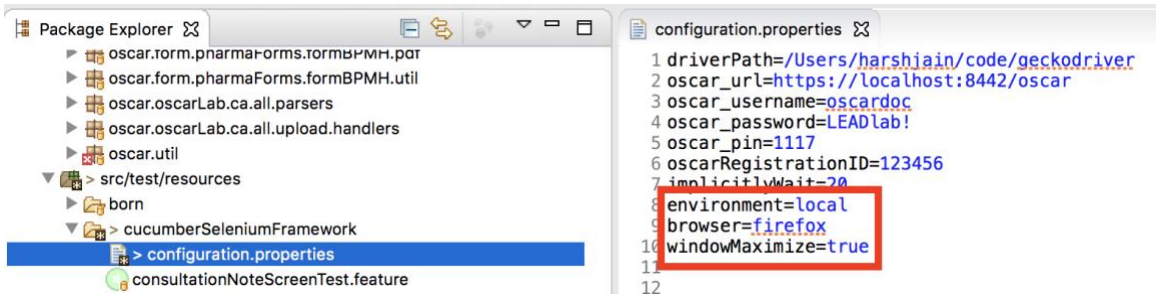


Figure 43 - ConfigFileReader to set Driver and Environment

Testers can easily switch to different browsers and environments just by changing values of browser and environment keys in configuration.properties file. The windowMaximize option is used to maximize window when the browser starts and is set to true by default.

The WebDriver Manager creates a new browser instance if there no existing driver instance is running. If the driver instance has already been initiated, it will return the existing running instance.

Screenshot:

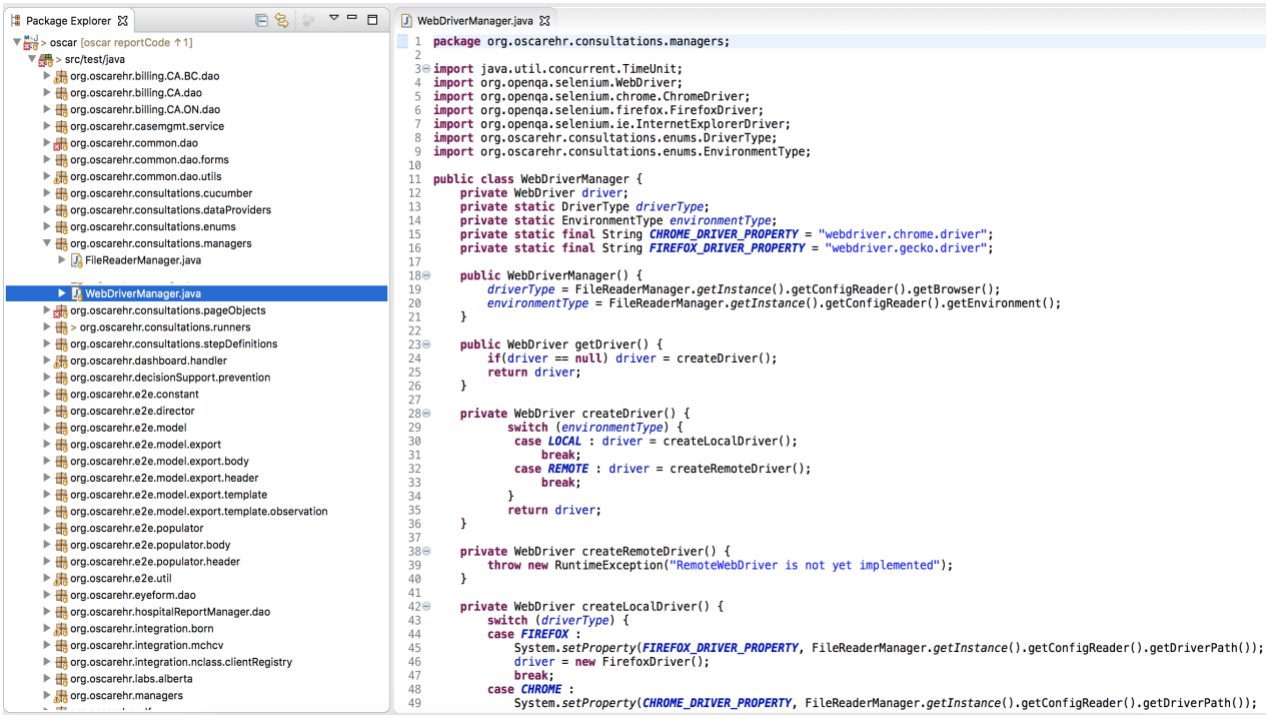


Figure 44 - WebDriverManager

Code Snippet:

```
1.  package org.oscarehr.consultations.managers;
2.
3.  import java.util.concurrent.TimeUnit;
4.  import org.openqa.selenium.WebDriver;
5.  import org.openqa.selenium.chrome.ChromeDriver;
6.  import org.openqa.selenium.firefox.FirefoxDriver;
7.  import org.openqa.selenium.ie.InternetExplorerDriver;
8.  import org.oscarehr.consultations.enums.DriverType;
9.  import org.oscarehr.consultations.enums.EnvironmentType;
10.
11. public class WebDriverManager {
12.     private WebDriver driver;
13.     private static DriverType driverType;
14.     private static EnvironmentType environmentType;
15.     private static final String CHROME_DRIVER_PROPERTY = "webdriver.chrome.driver";
16.     private static final String FIREFOX_DRIVER_PROPERTY = "webdriver.gecko.driver";
17.
18.     public WebDriverManager() {
19.         driverType = FileReaderManager.getInstance().getConfigReader().getBrowser();
20.         environmentType = FileReaderManager.getInstance().getConfigReader().getEnvironment();
21.     }
22.
23.     public WebDriver getDriver() {
24.         if(driver == null) driver = createDriver();
25.         return driver;
26.     }
27.
28.     private WebDriver createDriver() {
29.            switch (environmentType) {
30.             case LOCAL : driver = createLocalDriver();
31.                 break;
32.             case REMOTE : driver = createRemoteDriver();
33.                 break;
34.            }
35.            return driver;
36.     }
37.
38.     private WebDriver createRemoteDriver() {
39.         throw new RuntimeException("RemoteWebDriver is not yet implemented");
40.     }
41.
42.     private WebDriver createLocalDriver() {
43.         switch (driverType) {
44.         case FIREFOX :
45.             System.setProperty(FIREFOX_DRIVER_PROPERTY, FileReaderManager.getInstance().getConfigReade
    r().getDriverPath());
46.             driver = new FirefoxDriver();
47.             break;
48.         case CHROME :
49.             System.setProperty(CHROME_DRIVER_PROPERTY, FileReaderManager.getInstance().getConfigReader
    ().getDriverPath());
50.             driver = new ChromeDriver();
```

```
51.             break;
52.         case INTERNETEXPLORER : driver = new InternetExplorerDriver();
53.             break;
54.         }
55.
56.         if(FileReaderManager.getInstance().getConfigReader().getBrowserWindowSize()) driver.manage().w
    indow().maximize();
57.         driver.manage().timeouts().implicitlyWait(FileReaderManager.getInstance().getConfigReader().ge
    tImplicitlyWait(), TimeUnit.SECONDS);
58.         return driver;
59.     }
60.
61.     public void closeDriver() {
62.         for(String child : driver.getWindowHandles()) {
63.             driver.switchTo().window(child);
64.             driver.close();
65.         }
66.     }
67.
68.     public void quitDriver() {
69.         driver.quit();
70.     }
71. }
```

If multiple scenarios are present in a feature file and when a new scenario starts running during test execution, a separate browser window instance is created by the cucumber library for the new scenario. This ensures that all the scenarios are running independent of each other.

The StepDefinition file will now be updated to use WebDriverManager instead of initiating the browser instance itself.

Screenshots:

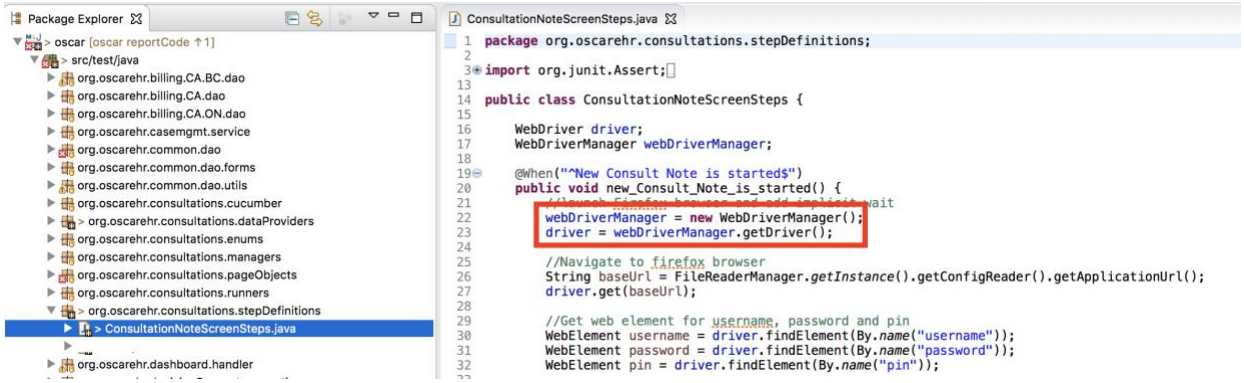


Figure 45 - Use WebDriverManager in Step Definition File

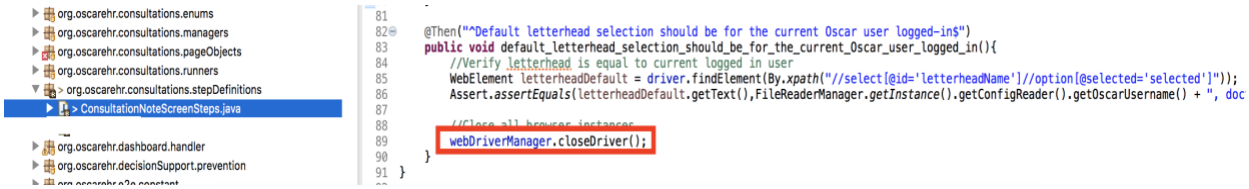


Figure 46 - Use WebDriverManager in Step Definition File

Code Snippet:

```
1.  package org.oscarehr.consultations.stepDefinitions;
2.
3.  import org.junit.Assert;
4.  import org.openqa.selenium.By;
5.  import org.openqa.selenium.Keys;
6.  import org.openqa.selenium.WebDriver;
7.  import org.openqa.selenium.WebElement;
8.  import org.oscarehr.consultations.managers.FileReaderManager;
9.  import org.oscarehr.consultations.managers.WebDriverManager;
10.
11. import cucumber.api.java.en.Then;
12. import cucumber.api.java.en.When;
13.
14. public class ConsultationNoteScreenSteps {
15.
16.     WebDriver driver;
17.     WebDriverManager webDriverManager;
18.
19.     @When("^New Consult Note is started$")
20.     public void new_Consult_Note_is_started() {
21.         //launch Firefox browser and add implicit wait
22.         webDriverManager = new WebDriverManager();
23.         driver = webDriverManager.getDriver();
24.
25.         //Navigate to firefox browser
26.         String baseUrl = FileReaderManager.getInstance().getConfigReader().getApplicationUrl();
27.         driver.get(baseUrl);
28.
29.         //Get web element for username, password and pin
30.         WebElement username = driver.findElement(By.name("username"));
31.         WebElement password = driver.findElement(By.name("password"));
32.         WebElement pin = driver.findElement(By.name("pin"));
33.
34.         //Enter values for the web element and press enter
35.         username.sendKeys(FileReaderManager.getInstance().getConfigReader().getOscarUsername());
36.         password.sendKeys(FileReaderManager.getInstance().getConfigReader().getOscarPassword());
37.         pin.sendKeys(FileReaderManager.getInstance().getConfigReader().getOscarPin());
38.         pin.sendKeys(Keys.RETURN);
39.
40.         //Click on schedule tab to navigate to Appointment access page
41.         try {
42.                 Thread.sleep(5000);
43.             } catch (InterruptedException e) {
44.                 e.printStackTrace();
45.             }
46.         WebElement schedule = driver.findElement(By.xpath("//a[@class='ng-scope ng-
    binding'][contains(text(),'Schedule')]"));
47.         schedule.click();
48.
49.         //Click on search tab and navigate to Patient Search Results page
50.         driver.findElement(By.xpath("//a[@title='Search for patient records']")).click();
51.         for(String winHandle : driver.getWindowHandles()){
52.             driver.switchTo().window(winHandle);
53.         }
54.
55.         //Click on search button to list demographic patient records
```

```
56.          driver.findElement(By.xpath("//input[@title='Search active patients']")).click();
57.
58.          //Click on Patient Demographic Record Id and navigate to new window
59.          WebElement demographicLink = driver.findElement(By.xpath("//a[@title='Master Demographic File'
     ]"));
60.          demographicLink.click();
61.          for(String winHandle : driver.getWindowHandles()){
62.              driver.switchTo().window(winHandle);
63.          }
64.
65.          //Click on Consultations and navigate to new window
66.          WebElement consultationsLink = driver.findElement(By.linkText("Consultations"));
67.          consultationsLink.click();
68.          for(String winHandle : driver.getWindowHandles()){
69.              driver.switchTo().window(winHandle);
70.          }
71.
72.          //Click on New Consultation link and navigate to new window
73.          WebElement newConsultationLink = driver.findElement(By.partialLinkText("New Consultation"));
74.          newConsultationLink.click();
75.          for(String winHandle : driver.getWindowHandles()){
76.              driver.switchTo().window(winHandle);
77.          }
78.
79.
80.      }
81.
82.      @Then("^Default letterhead selection should be for the current Oscar user logged-in$")
83.      public void default_letterhead_selection_should_be_for_the_current_Oscar_user_logged_in(){
84.          //Verify letterhead is equal to current logged in user
85.          WebElement letterheadDefault = driver.findElement(By.xpath("//select[@id='letterheadName']//op
     tion[@selected='selected']"));
86.          Assert.assertEquals(letterheadDefault.getText(),FileReaderManager.getInstance().getConfigReade
     r().getOscarUsername() + ", doctor");
87.
88.          //Close all browser instances
89.          webDriverManager.closeDriver();
90.      }
91. }
```

Page Objects Design Pattern and Page Object Manager

To cover just the two Cucumber steps, large lines of code have been added in StepDefinition file. Every new Cucumber step will require new methods to be added in the StepDefinition file. This will make the Step Definition files bloated and make the code difficult to understand, debug and maintain.

To resolve this issue, Page Object Design Pattern [10] will be implemented and the code in StepDefinition file will be divided among multiple PageObjects. Following are the 7 pages which are accessed in sequence while running the automation script:

1. Login Page
2. Appointment Access Page
3. Patient Search Results Page
4. Patient Detail Info Page
5. View Consultation Requests Page

6. Oscar Consultation Request Page
7. Print Preview Page

For each of the above pages, a corresponding Page Object can be created and the code in StepDefinition files can be transferred to these Page Object classes.

For explanation purpose, LoginPage Page Object has been demonstrated in this section. Create a package with name *org.oscarehr.consultations.pageObjects* within *src/test/java* directory. Within this package create class *LoginPage.java* and transfer all the login page operations in Stepdefinition file to *LoginPage.java* class.

Screenshots:

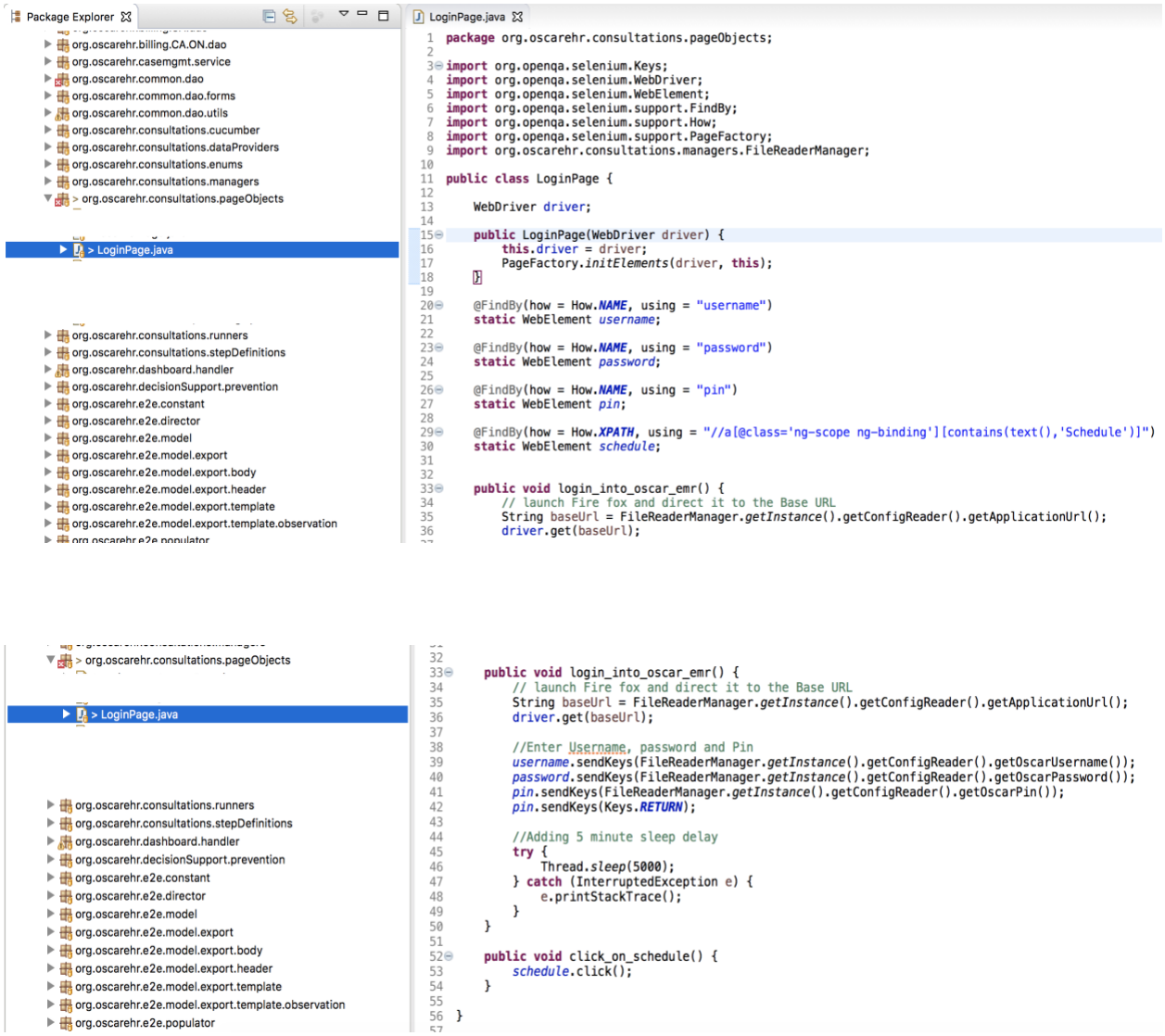


Figure 47 - Login PageObject

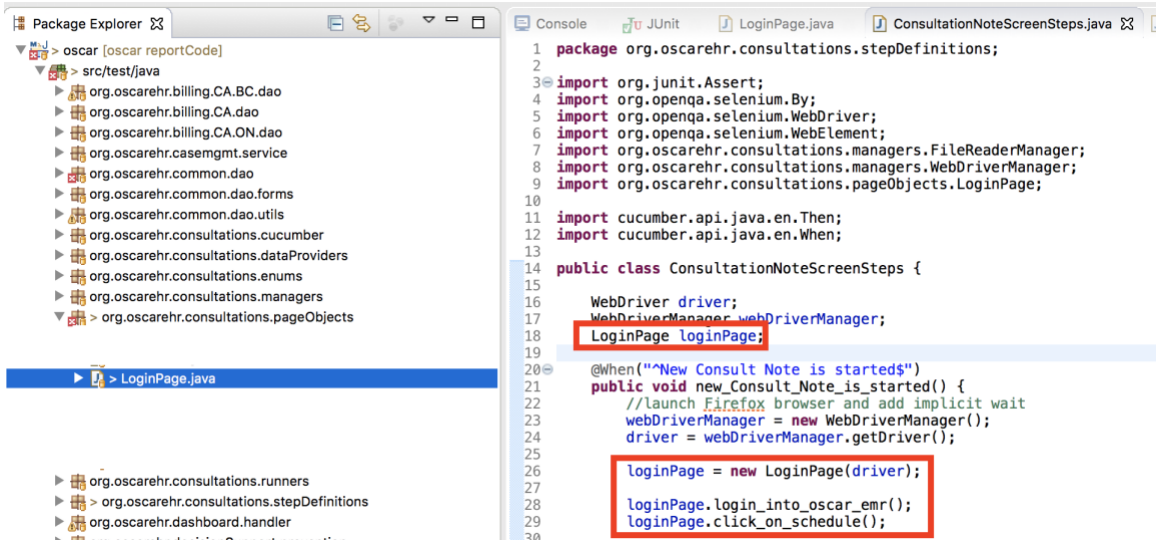
Code Snippet:

```
1.  package org.oscarehr.consultations.pageObjects;
2.
3.  import org.openqa.selenium.Keys;
4.  import org.openqa.selenium.WebDriver;
5.  import org.openqa.selenium.WebElement;
6.  import org.openqa.selenium.support.FindBy;
7.  import org.openqa.selenium.support.How;
8.  import org.openqa.selenium.support.PageFactory;
9.  import org.oscarehr.consultations.managers.FileReaderManager;
10.
11. public class LoginPage {
12.
13.     WebDriver driver;
14.
15.     public LoginPage(WebDriver driver) {
16.         this.driver = driver;
17.         PageFactory.initElements(driver, this);
18.     }
19.
20.     @FindBy(how = How.NAME, using = "username")
21.     static WebElement username;
22.
23.     @FindBy(how = How.NAME, using = "password")
24.     static WebElement password;
25.
26.     @FindBy(how = How.NAME, using = "pin")
27.     static WebElement pin;
28.
29.     @FindBy(how = How.XPATH, using = "//a[@class='ng-scope ng-
    binding'][contains(text(),'Schedule')]")
30.     static WebElement schedule;
31.
32.
33.     public void login_into_oscar_emr() {
34.         // launch Fire fox and direct it to the Base URL
35.         String baseUrl = FileReaderManager.getInstance().getConfigReader().getApplicationUrl();
36.         driver.get(baseUrl);
37.
38.         //Enter Username, password and Pin
39.         username.sendKeys(FileReaderManager.getInstance().getConfigReader().getOscarUsername());
40.         password.sendKeys(FileReaderManager.getInstance().getConfigReader().getOscarPassword());
41.         pin.sendKeys(FileReaderManager.getInstance().getConfigReader().getOscarPin());
42.         pin.sendKeys(Keys.RETURN);
43.
44.         //Adding 5 minute sleep delay
45.         try {
46.             Thread.sleep(5000);
47.         } catch (InterruptedException e) {
48.             e.printStackTrace();
49.         }
50.     }
51.
52.     public void click_on_schedule() {
53.         schedule.click();
54.     }
55.
56. }
```

In the above code *@FindBy* annotation is used to locate web elements instead of using *driver.findElement(By.)* method. Selenium's Page Factory (its InitElements method), FindBy and How libraries are used to implement PageObject functionality. Using PageObjects makes the codebase clean, organised and easier to understand and maintain.

The LoginPage will be instantiated and its methods login_into_oscar_emr() and click_on_schedule() will be called in the Step Definition file to perform the WebDriver operations.

Screenshot:



Code Snippet:

```
1.  package org.oscarehr.consultations.stepDefinitions;
2.
3.  import org.junit.Assert;
4.  import org.openqa.selenium.By;
5.  import org.openqa.selenium.WebDriver;
6.  import org.openqa.selenium.WebElement;
7.  import org.oscarehr.consultations.managers.FileReaderManager;
8.  import org.oscarehr.consultations.managers.WebDriverManager;
9.  import org.oscarehr.consultations.pageObjects.LoginPage;
10.
11. import cucumber.api.java.en.Then;
12. import cucumber.api.java.en.When;
13.
14. public class ConsultationNoteScreenSteps {
15.
16.     WebDriver driver;
17.     WebDriverManager webDriverManager;
18.     LoginPage loginPage;
19.
20.     @When("^New Consult Note is started$")
21.     public void new_Consult_Note_is_started() {
```

```
22.         //launch Firefox browser and add implicit wait
23.         webDriverManager = new WebDriverManager();
24.         driver = webDriverManager.getDriver();
25.
26.         loginPage = new LoginPage(driver);
27.
28.         loginPage.login_into_oscar_emr();
29.         loginPage.click_on_schedule();
```

In case of multiple step definition files using the same pages, every Page Object has to be instantiated multiple time. This method is again inefficient and is against the Singleton Design Pattern principle. To avoid this, PageObjectManager.java class will be implemented. Page Object Manager will ensure that only one object is created per page and all the Page Objects will have a single point of access throughout the test execution.

In the package *org.oscarehr.consultations.managers*, create a new Class *PageObjectManager.java*. The PageObjectManager.java will accept Web driver as parameter in its constructor and will instantiate a PageObject only if its null. If the PageObject is not null, it will return existing PageObject.

Screenshot:

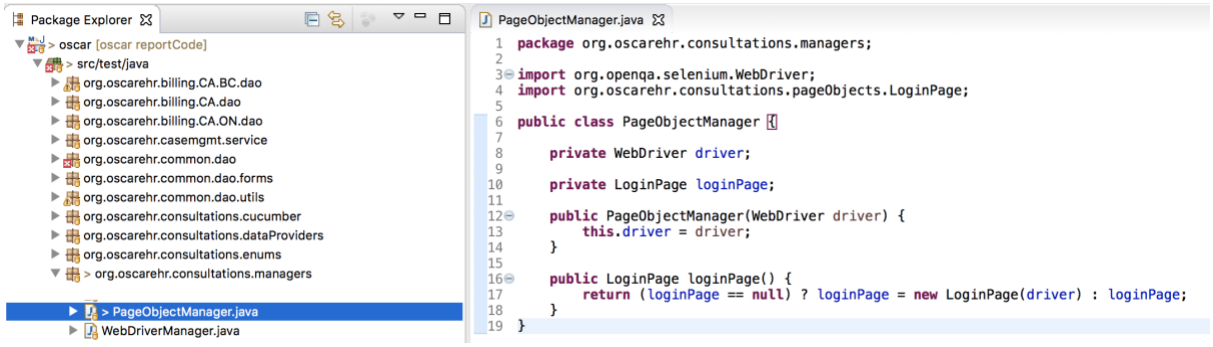


Figure 48 - Managing LoginPage in Page Object Manager

Code Snippet:

```
1.  package org.oscarehr.consultations.managers;
2.
3.  import org.openqa.selenium.WebDriver;
4.  import org.oscarehr.consultations.pageObjects.LoginPage;
5.
6.  public class PageObjectManager {
7.
8.      private WebDriver driver;
9.
10.     private LoginPage loginPage;
11.
12.     public PageObjectManager(WebDriver driver) {
13.         this.driver = driver;
14.     }
15.
16.     public LoginPage loginPage() {
```

```
17.         return (loginPage == null) ? loginPage = new LoginPage(driver) : loginPage;
18.     }
19. }
```

The Step Definition file can now be updated to use Page Object Manager to access LoginPage methods instead of calling LoginPage directly.

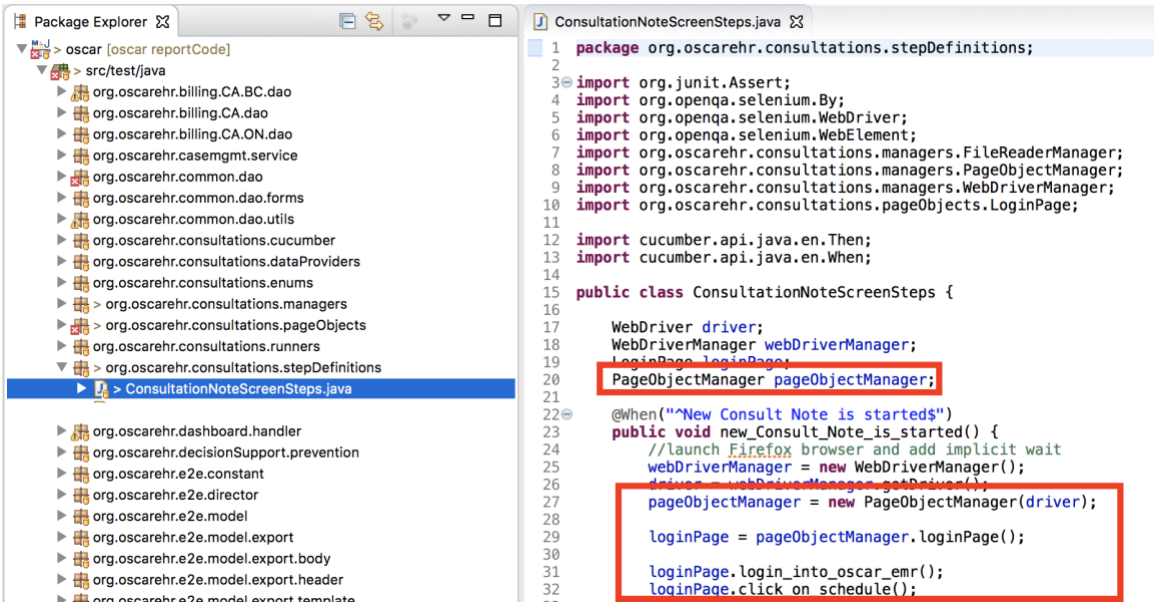


Figure 49 - Using Page Object Manager in StepDefinition File

Code Snippet:

```
1.  package org.oscarehr.consultations.stepDefinitions;
2.
3.  import org.junit.Assert;
4.  import org.openqa.selenium.By;
5.  import org.openqa.selenium.WebDriver;
6.  import org.openqa.selenium.WebElement;
7.  import org.oscarehr.consultations.managers.FileReaderManager;
8.  import org.oscarehr.consultations.managers.PageObjectManager;
9.  import org.oscarehr.consultations.managers.WebDriverManager;
10. import org.oscarehr.consultations.pageObjects.LoginPage;
11.
12. import cucumber.api.java.en.Then;
13. import cucumber.api.java.en.When;
14.
15. public class ConsultationNoteScreenSteps {
16.
17.     WebDriver driver;
18.     WebDriverManager webDriverManager;
19.     LoginPage loginPage;
20.     PageObjectManager pageObjectManager;
21.
22.     @When("^New Consult Note is started$")
23.     public void new_Consult_Note_is_started() {
24.         //launch Firefox browser and add implicit wait
25.         webDriverManager = new WebDriverManager();
26.         driver = webDriverManager.getDriver();
```

```
27.         pageObjectManager = new PageObjectManager(driver);
28.
29.         loginPage = pageObjectManager.loginPage();
30.
31.         loginPage.login_into_oscar_emr();
32.         loginPage.click_on_schedule();
```

Similar to implementing the LoginPage above, following steps can be taken to implement Page Object Design Pattern for remaining 6 pages:

1. Create PageObject class for each page.
2. Create methods in the PageObject class to perform the browser actions that are done in StepDefinition fil. Use Selenium Page Factory and @FindBy annotation to locate web elements in these PageObjects.
3. Add variables and methods in PageObjectManager.java class to instantiate and return the PageObjects
4. Update the StepDefinition file to call the PageObject methods using Page Object Manager.

Test Context

When the project gets bigger and the codebase has multiple Step Definition files, it is not ideal to instantiate WebDriverManager and Page Object Manager for every step file. To resolve this issue, Test Context class is used [12]. Test Context uses *cucumber-picocontainer* dependency to ensure that same instance of WebDriverManager and Page Object Manager is used.

Screenshot:

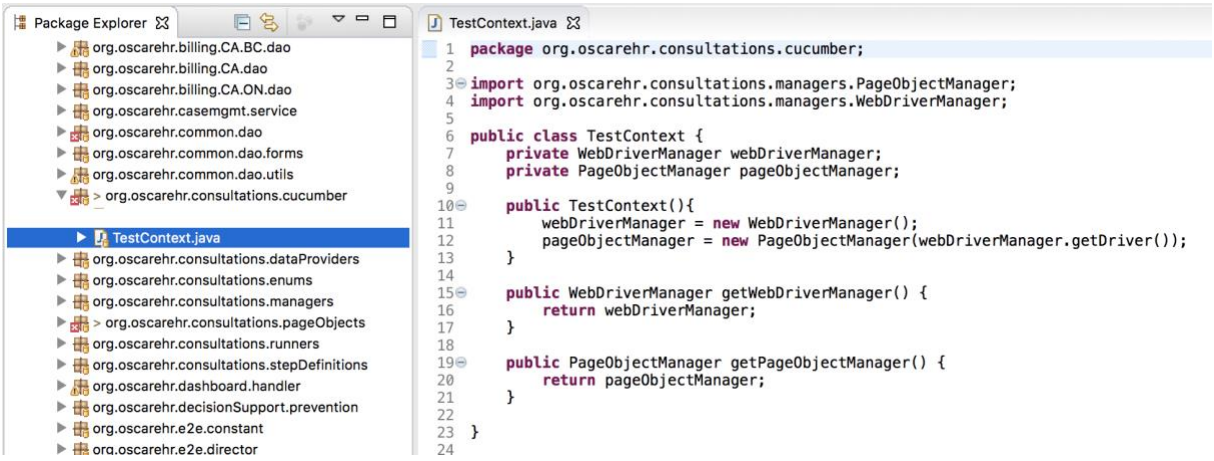


Figure 50 - TextContext.java

Code Snippet:

```
1.  package org.oscarehr.consultations.cucumber;
```

```
2.
3.  import org.oscarehr.consultations.managers.PageObjectManager;
4.  import org.oscarehr.consultations.managers.WebDriverManager;
5.
6.  public class TestContext {
7.      private WebDriverManager webDriverManager;
8.      private PageObjectManager pageObjectManager;
9.
10.     public TestContext(){
11.         webDriverManager = new WebDriverManager();
12.         pageObjectManager = new PageObjectManager(webDriverManager.getDriver());
13.     }
14.
15.     public WebDriverManager getWebDriverManager() {
16.         return webDriverManager;
17.     }
18.
19.     public PageObjectManager getPageObjectManager() {
20.         return pageObjectManager;
21.     }
22. }
```

The StepDefinition will now be updated to use Test Context to access Web Driver Manager and Page Object Manager. In the Step Definition file, LoginPage PageObject and driver object is initiated/accessed from the class constructor using Test Context.

Screenshot:

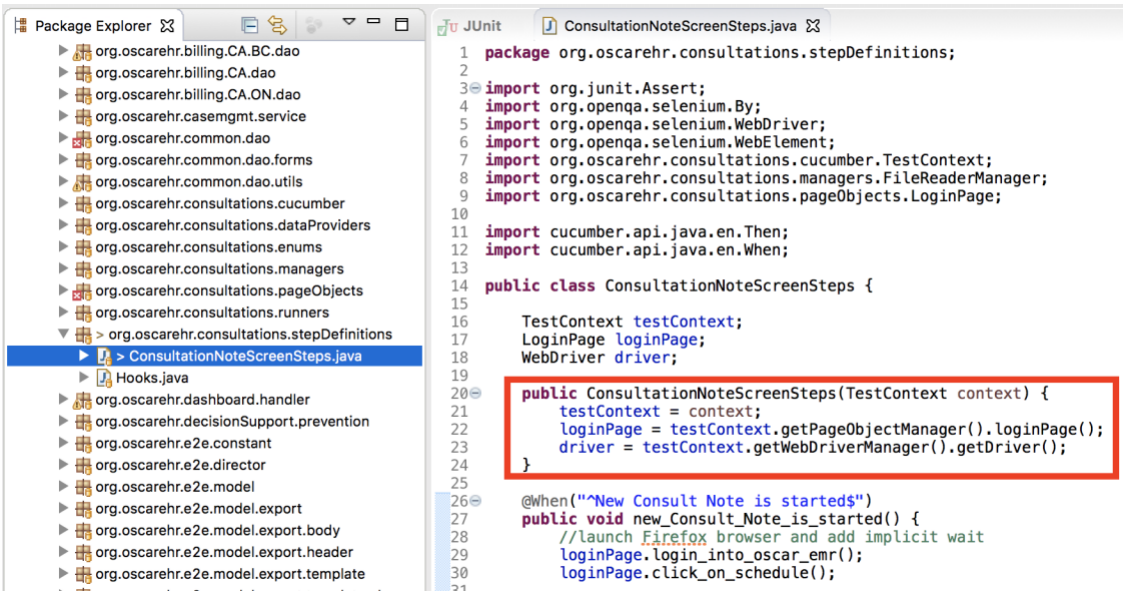


Figure 51 - Implementing Test Context

Code Snippet:

```
1.  package org.oscarehr.consultations.stepDefinitions;
2.
3.  import org.junit.Assert;
4.  import org.openqa.selenium.By;
5.  import org.openqa.selenium.WebDriver;
```

```
6.  import org.openqa.selenium.WebElement;
7.  import org.oscarehr.consultations.cucumber.TestContext;
8.  import org.oscarehr.consultations.managers.FileReaderManager;
9.  import org.oscarehr.consultations.pageObjects.LoginPage;
10.
11. import cucumber.api.java.en.Then;
12. import cucumber.api.java.en.When;
13.
14. public class ConsultationNoteScreenSteps {
15.
16.     TestContext testContext;
17.     LoginPage loginPage;
18.     WebDriver driver;
19.
20.     public ConsultationNoteScreenSteps(TestContext context) {
21.         testContext = context;
22.         loginPage = testContext.getPageObjectManager().loginPage();
23.         driver = testContext.getWebDriverManager().getDriver();
24.     }
25.
26.     @When("^New Consult Note is started$")
27.     public void new_Consult_Note_is_started() {
28.         //launch Firefox browser and add implicit wait
29.         loginPage.login_into_oscar_emr();
30.         loginPage.click_on_schedule();
```

Once, the complete code has been converted into PageObjects, the driver instance is not required in StepDefinition file and can be removed from its constructor.

Hooks

As of now, it can be observed that the first Step Definition method *@When("^New Consult Note is started$")* takes care of setting up preconditions logging into Oscar EMR. Similarly, the second glue method:
@Then("^Default letterhead selection should be for the current Oscar user logged-in$") takes care of after step like closing the browser.

```
    }

    @When("^New Consult Note is started$")
    public void new_Consult_Note_is_started() {
        //launch Firefox browser and add implicit wait
        loginPage.login_into_oscar_emr();
        loginPage.click_on_schedule();

        //Click on search tab and navigate to Patient Search Results page
        driver.findElement(By.xpath("//a[@title='Search for patient records']"
```

```
@Then("^Default letterhead selection should be for the current Oscar user logged-in$")
public void default_letterhead_selection_should_be_for_the_current_Oscar_user_logged_in(){
    //Verify letterhead is equal to current logged in user
    WebElement letterheadDefault = driver.findElement(By.xpath("//select[@id='letterheadName']//option[@selected='selected']"));
    Assert.assertEquals(letterheadDefault.getText(),FileReaderManager.getInstance().getConfigReader().getOscarUsername() + ", doctor");

    //Close all browser instances
    testContext.getWebDriverManager().closeDriver();
}
```

Figure 52 - Code Before Using Hooks

However, these preconditions and after steps should not be part of the StepDefinition methods. Instead, they should be part of Hooks. Hooks are used to setup preconditions and after steps that are required to run before and after a scenario [11]. @Before and @After annotations are used to specify whether a Hook method is run before or after scenario. These preconditions and after steps will now be transferred from StepDefinition file to Hook methods.

Create a new Class Hooks.java in *org.oscarehr.consultations.stepDefinitions* package. The preconditions\after-steps are removed from StepDefinition files and added to Hooks.java class.

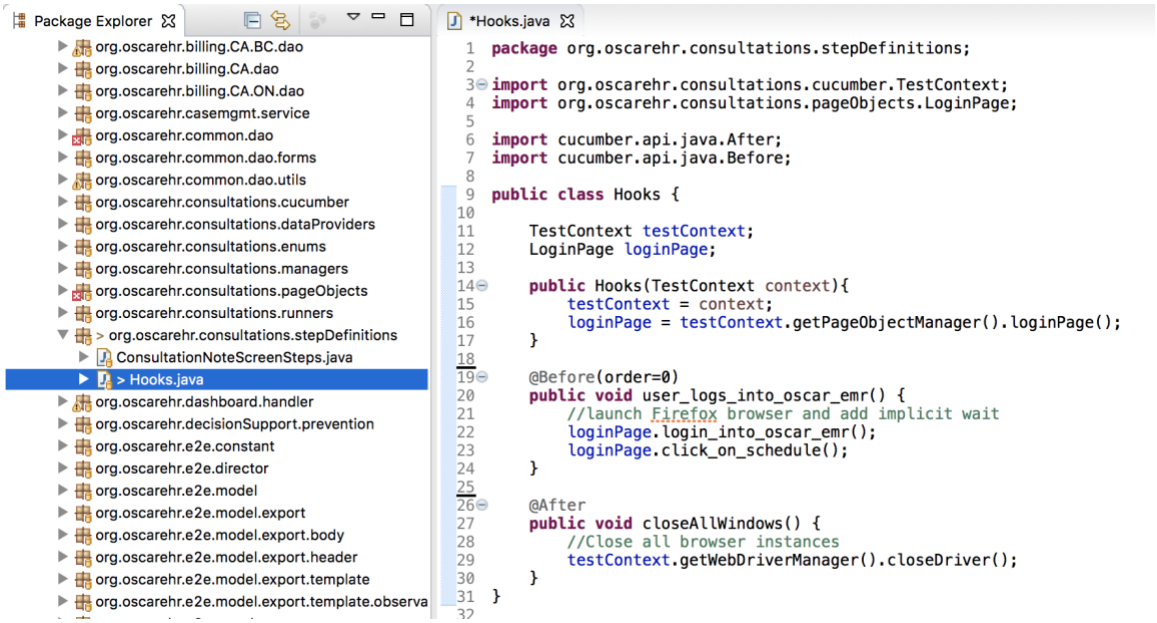


Figure 53 - Hooks.java

Code Snippet:

```
1.  package org.oscarehr.consultations.stepDefinitions;
2.
3.  import org.oscarehr.consultations.cucumber.TestContext;
4.  import org.oscarehr.consultations.pageObjects.LoginPage;
5.
6.  import cucumber.api.java.After;
7.  import cucumber.api.java.Before;
8.
9.  public class Hooks {
10.
11.     TestContext testContext;
12.     LoginPage loginPage;
13.
14.     public Hooks(TestContext context){
15.         testContext = context;
16.         loginPage = testContext.getPageObjectManager().loginPage();
17.     }
18.
```

```
19.     @Before(order=0)
20.     public void user_logs_into_oscar_emr() {
21.         //launch Firefox browser and add implicit wait
22.         loginPage.login_into_oscar_emr();
23.         loginPage.click_on_schedule();
24.     }
25.
26.     @After
27.     public void closeAllWindows() {
28.         //Close all browser instances
29.         testContext.getWebDriverManager().closeDriver();
30.     }
31. }
```

Scenario Context

ScenarioContext class is used to hold and share dynamic test data (generated during test run) among the several Stepdefinition files. ScenarioContext uses Test Context and Context.java to share data among StepDefinition files [13].

For the particular test case covered in this section, there is no specific requirement to use ScenarioContext. However, Scenario Context has been used in the project to store and transfer data like WebDriver Window Handle and Letterheads of Consultation Request.

Screenshot:

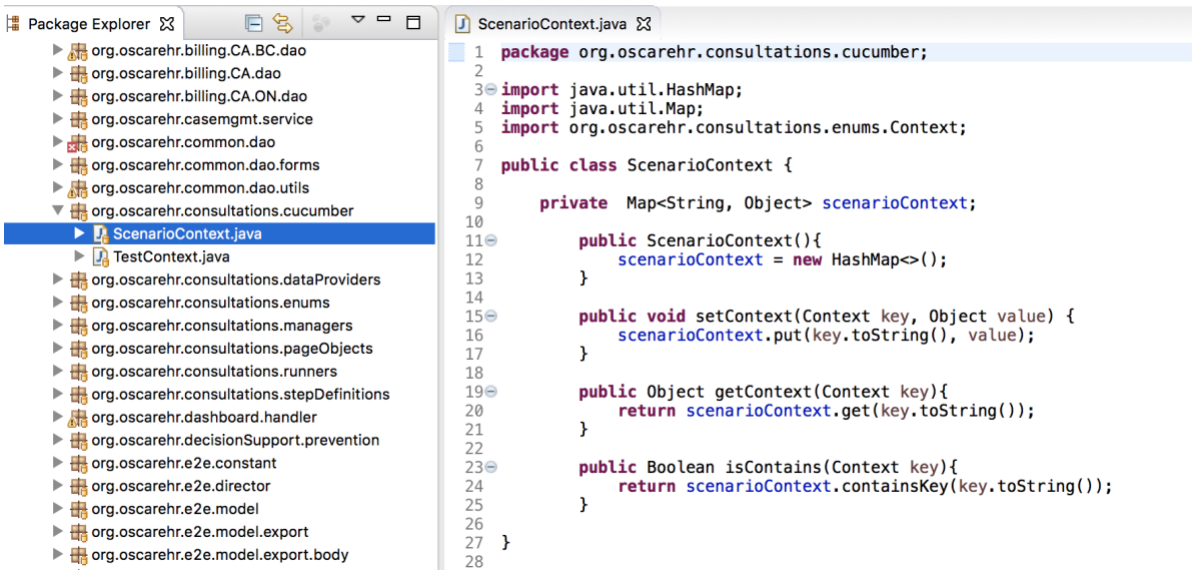


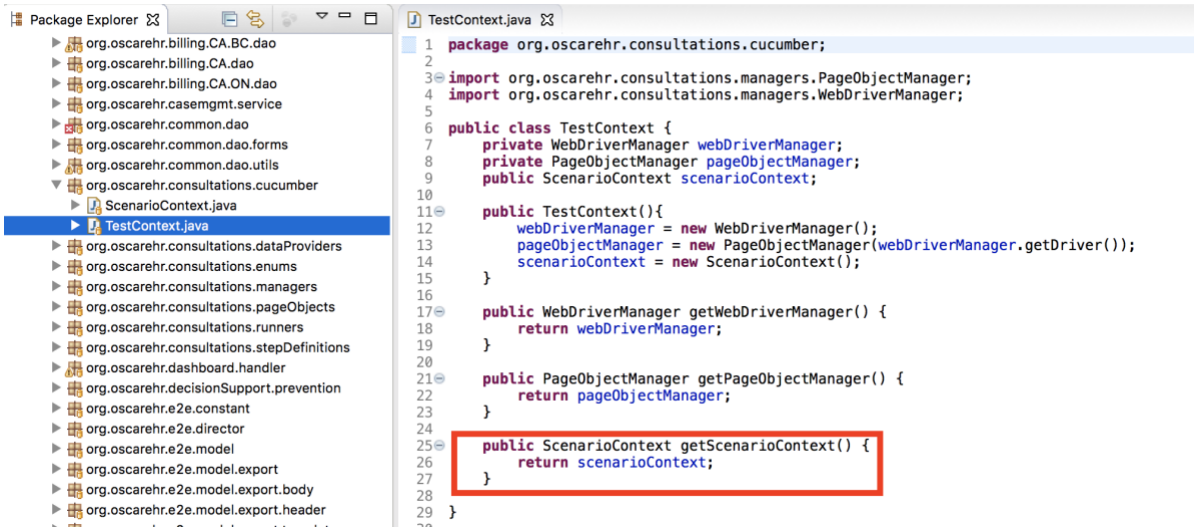
Figure 54 - ScenarioContext.java

Code Snippet:

```
1.  package org.oscarehr.consultations.cucumber;
2.
3.  import java.util.HashMap;
4.  import java.util.Map;
```

```
5.  import org.oscarehr.consultations.enums.Context;
6.
7.  public class ScenarioContext {
8.
9.      private  Map<String, Object> scenarioContext;
10.
11.         public ScenarioContext(){
12.             scenarioContext = new HashMap<>();
13.         }
14.
15.         public void setContext(Context key, Object value) {
16.             scenarioContext.put(key.toString(), value);
17.         }
18.
19.         public Object getContext(Context key){
20.             return scenarioContext.get(key.toString());
21.         }
22.
23.         public Boolean isContains(Context key){
24.             return scenarioContext.containsKey(key.toString());
25.         }
26. }
```

ScenarioContext uses Inbuilt HashMap object, TextContext.java and Context.java Enums to store and transfer values from one Stepdefinition object to other.



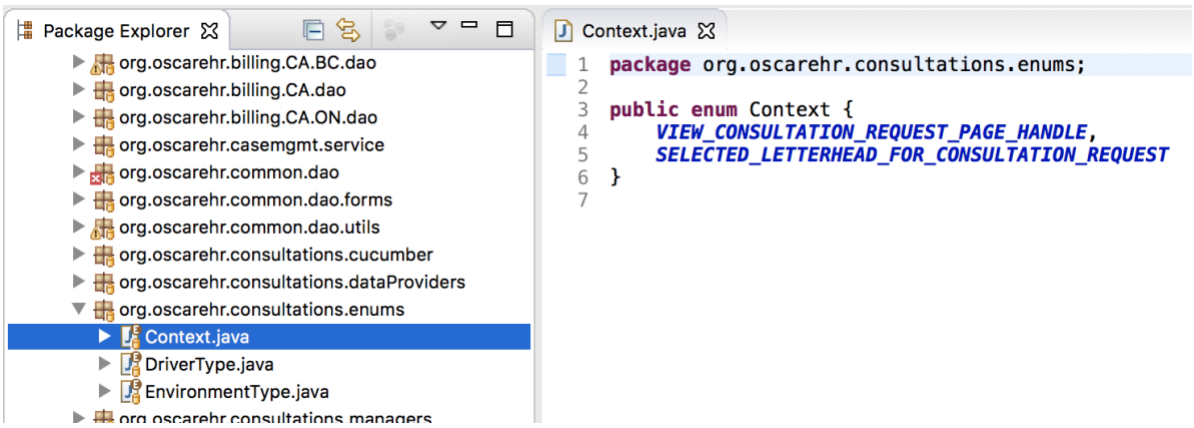


Figure 55 - Setting Up ScenarioContext

In the screenshots above it can be observed that View Consultation Page Request Webpage Handle and the Letterhead of Consultation request are stored using ScenarioContext and used during later part of the test run.

To store data during the test run, *setContext(Context key, Object value)* method is used and to retrieve the data, *getContext(Context key)* method is used. For example, Oscar Consultation Request Letterhead is stored by:

```
testContext.scenarioContext.setContext(Context.SELECTED_LETTERHEAD_FOR_CONSULTATION_REQUEST, oscarConsultationRequestPage.get_value_of_selected_letterhead());
```

The Letterhead can retrieve later during the test run by:

```
testContext.scenarioContext.getContext(Context.SELECTED_LETTERHEAD_FOR_CONSULTATION_REQUEST);
```

Cucumber Report Generation

Currently the test results of the sample test case are displayed in JUnit test console.

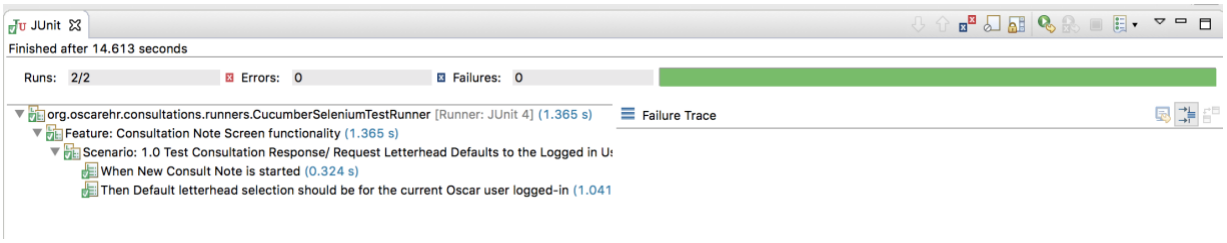


Figure 56 - Test Results using JUnit

There are many different ways users can generate and analyze test results and reports. One of the options to display report is to use pretty *plugin* (*monochrome=true* option makes the report easier to read and understand).

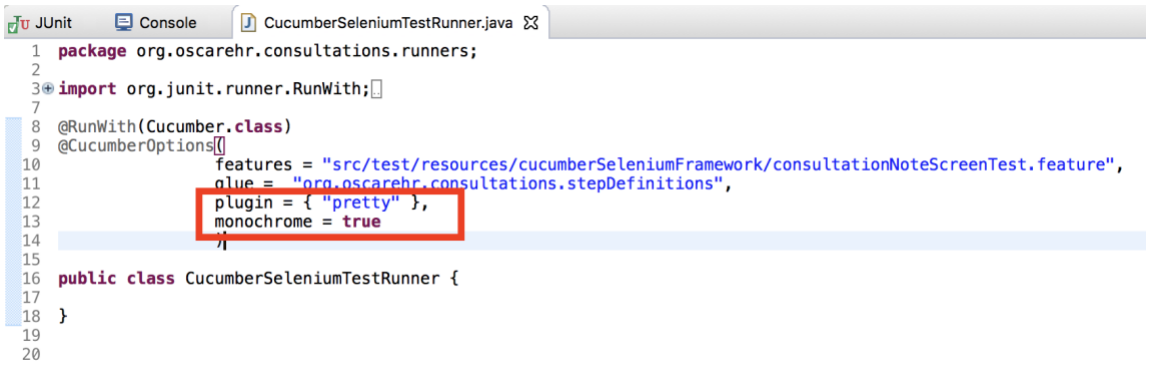


Figure 57 - Configure Pretty Plugin

Following is the report generated using pretty plugin:

Figure 58 - Pretty Plugin Results

Usage plugin option is used to display time taken to execute test scenario and steps.

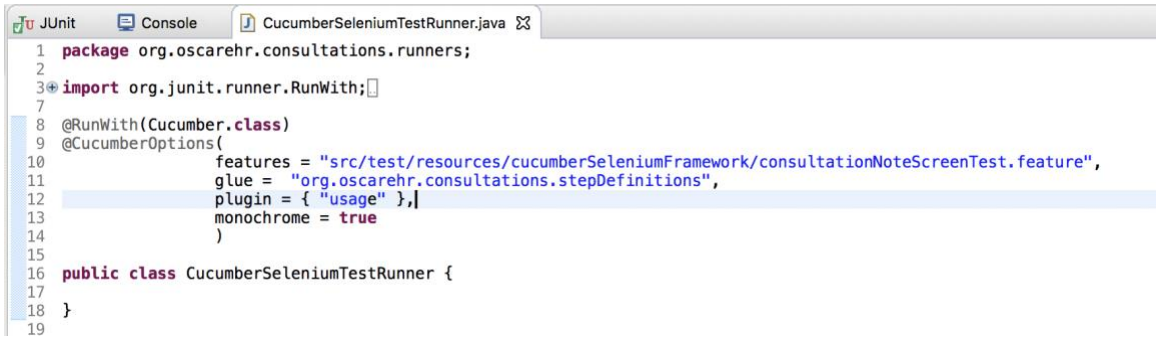


Figure 59 - Configure Usage Plugin

Following is the report generated using Usage plugin:

Figure 60 - Usage Plugin Results

Reports can also be generated in the several output forms like HTML, JSON and TXT by specifying the type of output form and target path. In below screenshots, all the three forms of report have been generated together:

JUnit configuration:

```
@RunWith(Cucumber.class)
@CucumberOptions(
                features = "src/test/resources/cucumberSeleniumFramework/consultationNoteScreenTest.feature",
                glue =  "org.oscarehr.consultations.stepDefinitions",
                plugin = { "pretty", "json:target/cucumber-reports/Cucumber.json",
                        "junit:target/cucumber-reports/Cucumber.xml",
                        "html:target/cucumber-reports" },
                monochrome = true
                )
```

Figure 61 - Configure JUnit to Generate Different Forms of Report

HTML Output:

Figure 62 - Results in HTML Format

JSON Output:

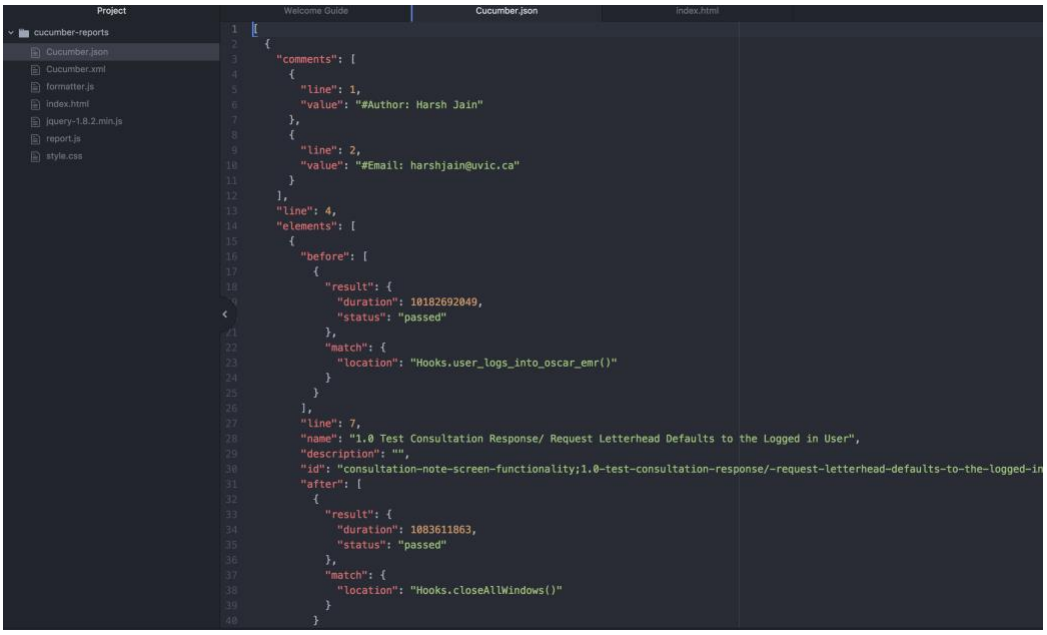


Figure 63 - Results in JSON Format

XML Output:

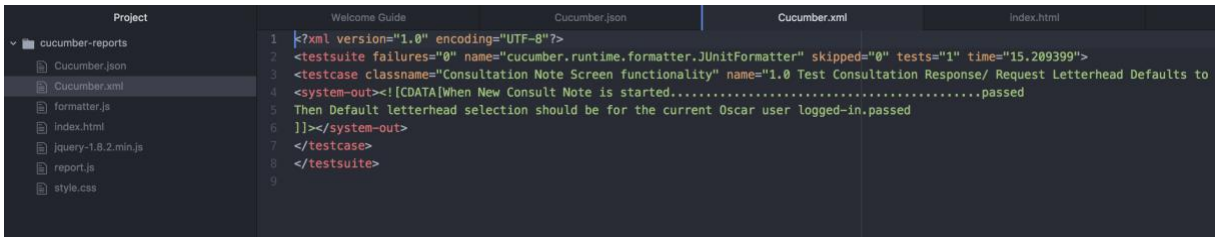


Figure 64 - Results in XML Format

There are many custom third-party plugins to create beautiful reports while running Cucumber Framework. One such plugin is Cucumber Extent Reporter [14]. To configure this plugin, first add *extentreports* and *cucumber-extentsreport* dependencies in *pom.xml* file.

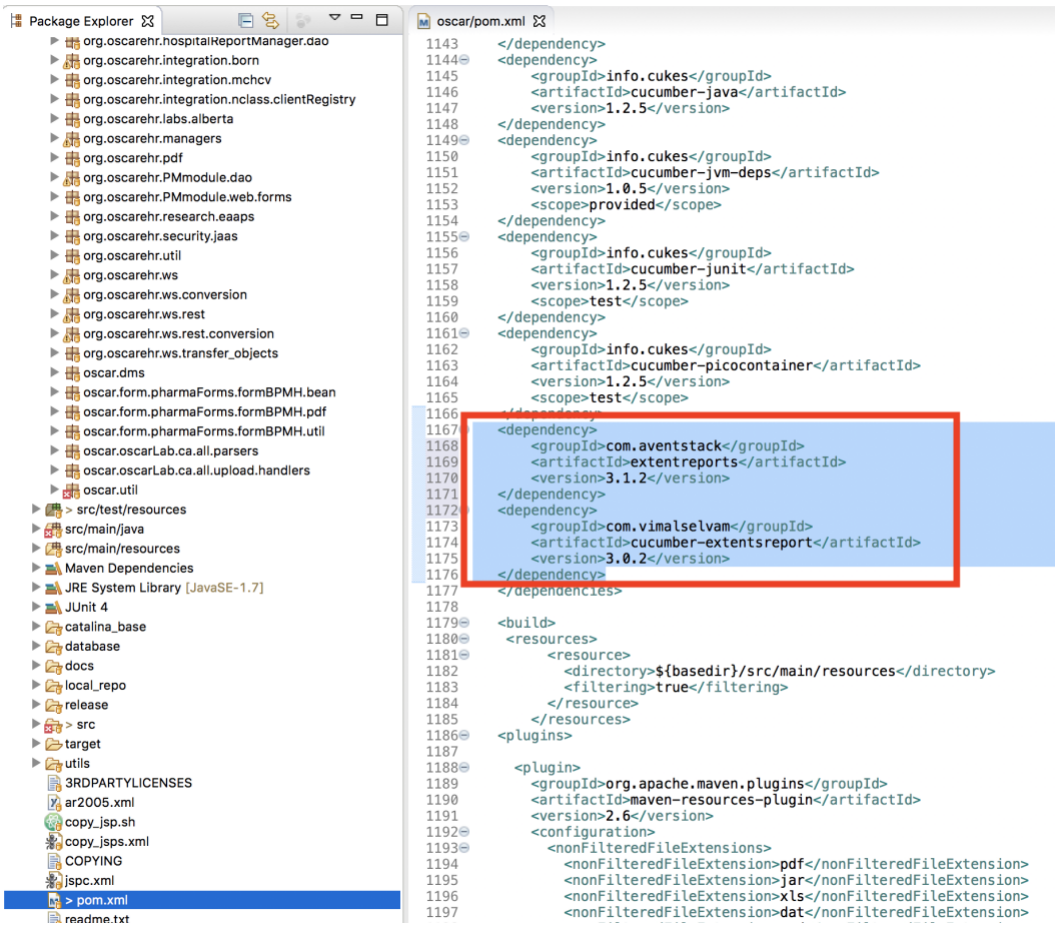
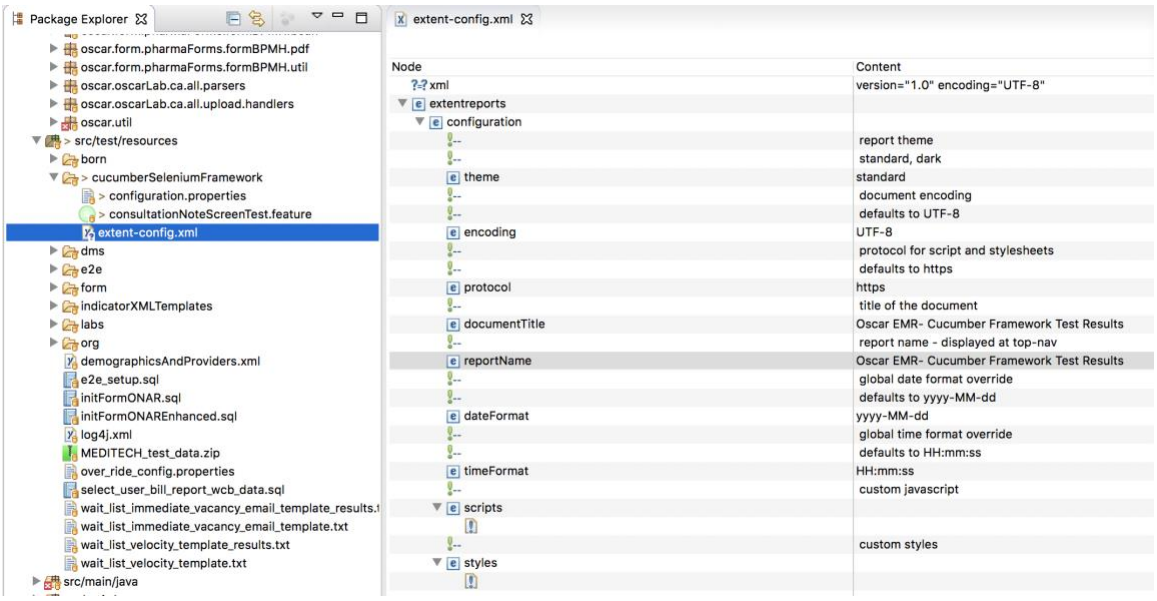


Figure 65 - Adding Extents Report Dependency

Create *extent-config.xml* in folder *cucumberSeleniumFramework* under *src/test/resources* directory to specify configuration settings for Cucumber Extent Report.



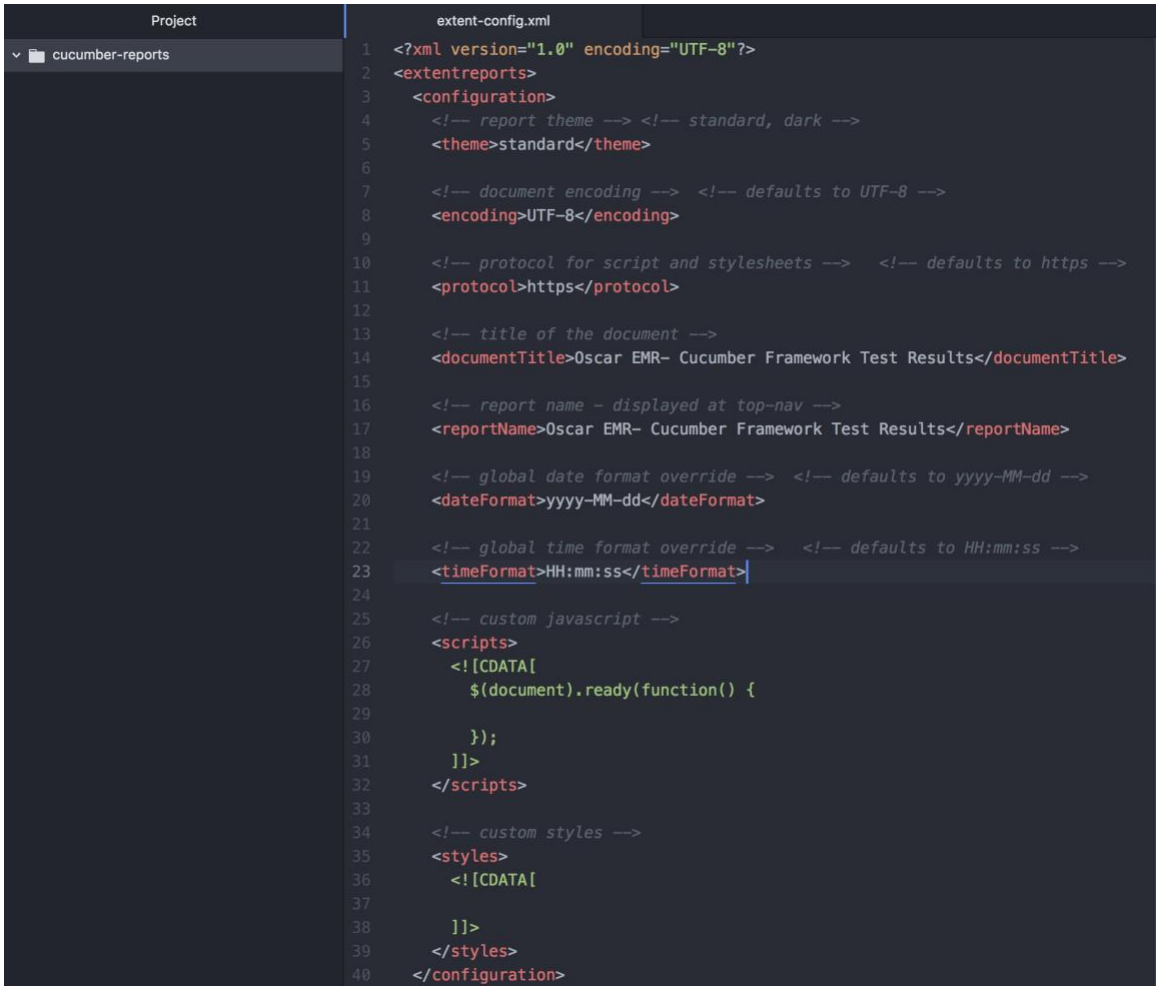


Figure 66 - Configure Extend Report Plugin

Add the location of *extent-config.xml* file location to configuration.properties file and ConfigFileReader.java file.

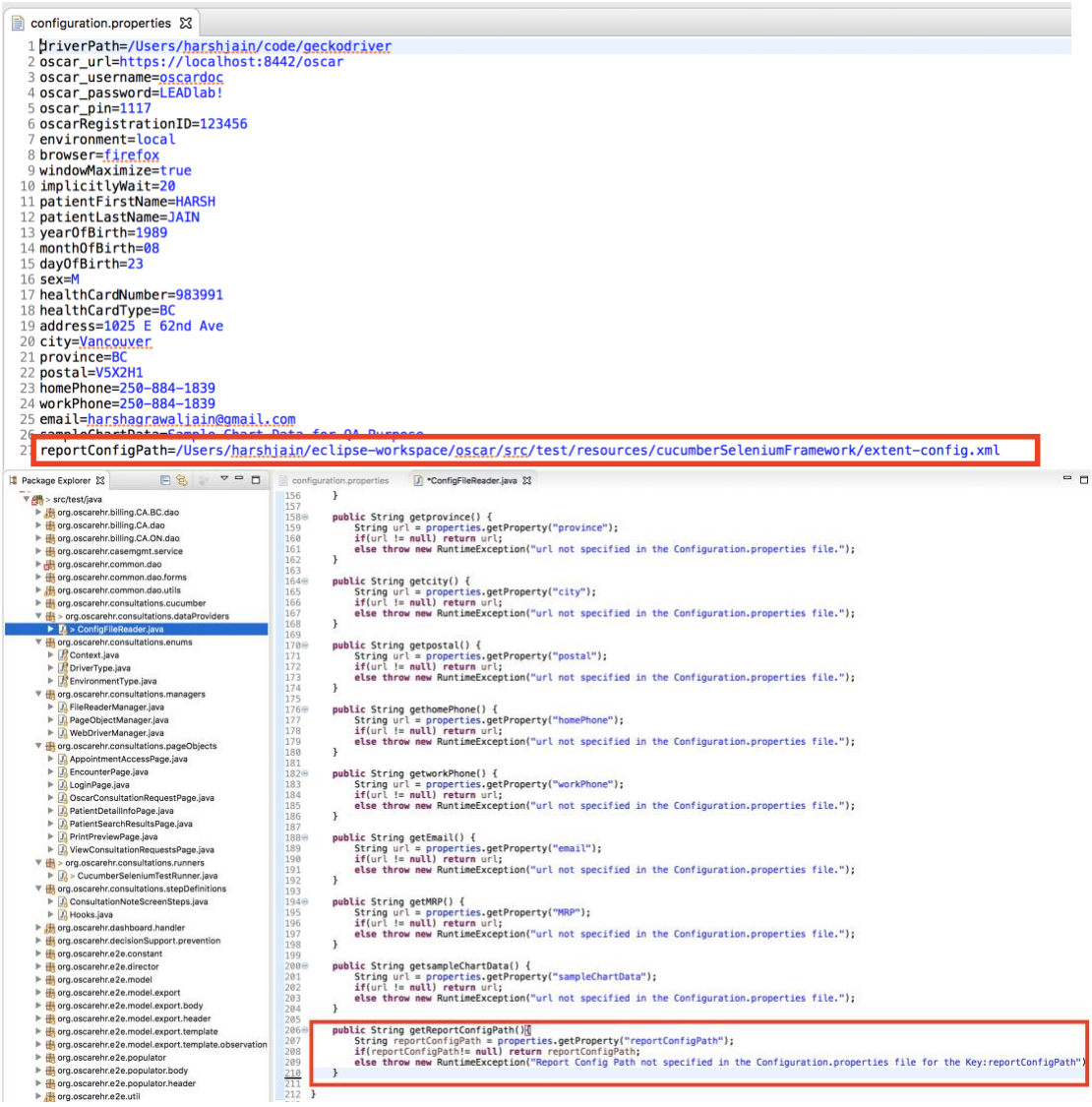


Figure 67 - Parameterising Extend Report Config Path

Update the JUnit runner to use *extendcucumber* library and use @AfterClass annotation to use extent-config.xml config file.

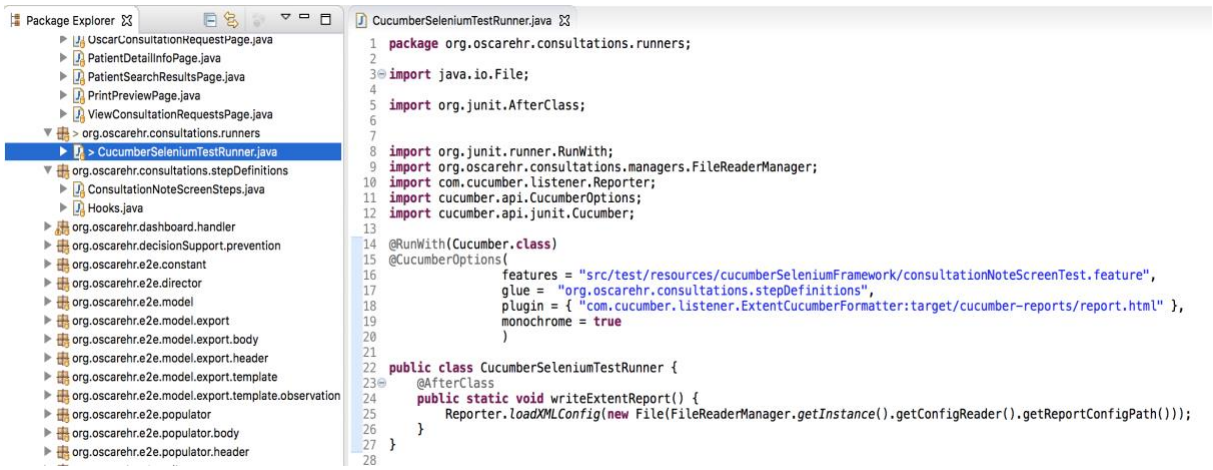


Figure 68 - Updating JUnit Runner to Use Extend Report

After running the test, navigate to *target/cucumber-reports* folder and open the report.html file. Following is a snapshot of the results displayed:

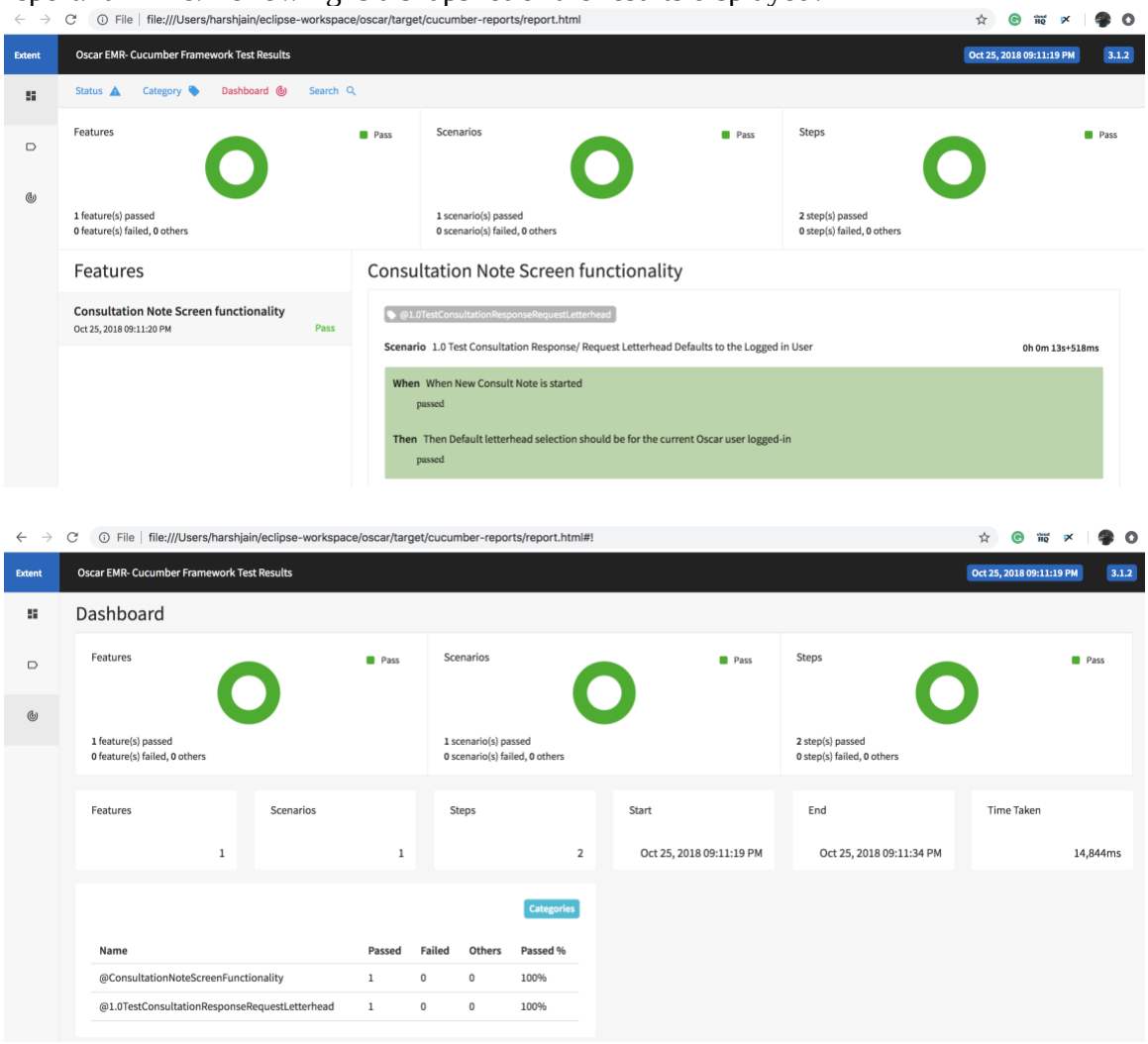


Figure 69 - Test Results from Extend Report Plugin

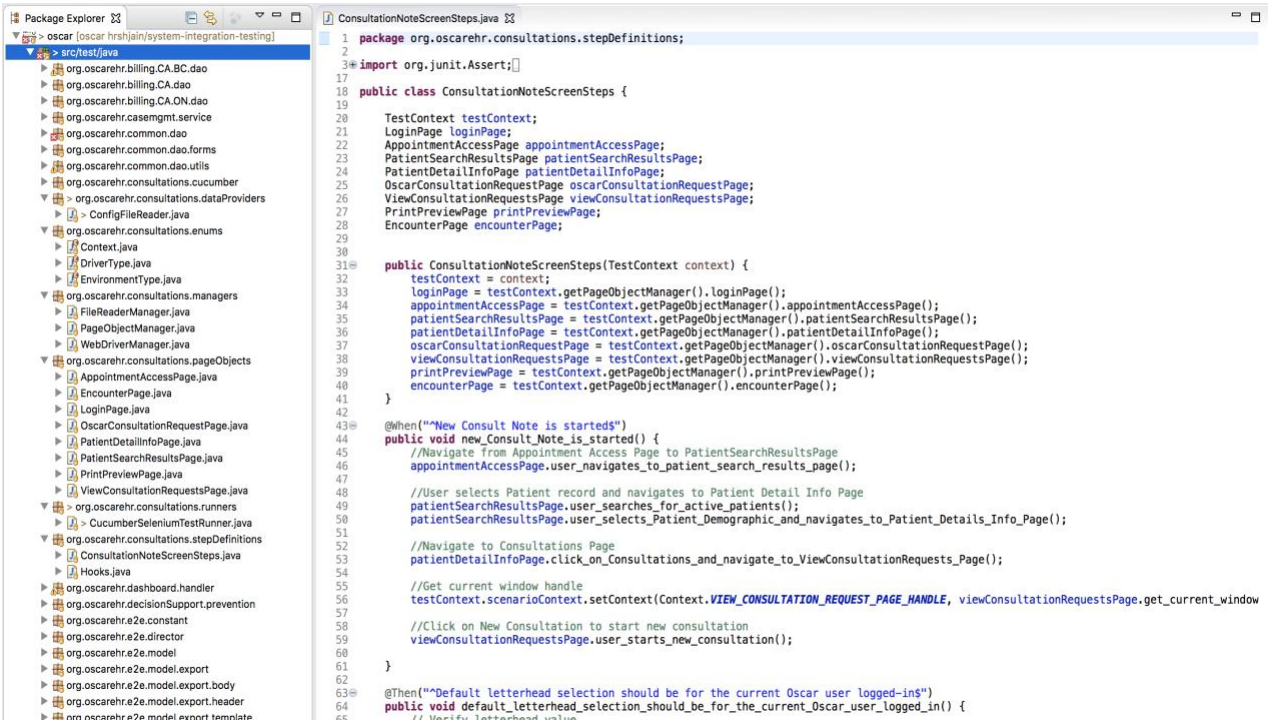
Scaling Test Steps for Consultation Note Screen Functionality and Final Results

Coverage

For the scope of this project, all the test cases of Consultation Note Screen Functionality present at the link: https://oscaremr.atlassian.net/wiki/spaces/OS/pages/54001741/OSCAR+15+Consultation+Note+Screen+Test+Cases have been covered and converted to Cucumber-Selenium Automated Test scripts. These test cases are part of single Cucumber feature file *consultationNoteScreenTest.feature*. 4 test scenarios in this webpage have been converted to their corresponding Cucumber scenarios in the feature file. Every scenario runs in a separate driver instance and all the scenarios are independent of each other.

Latest code can be accessed from forked branch (*hrshjain/system-integration-testing*) at link https://github.com/hrshjain/oscar/tree/hrshjain/system-integration-testing and the pull request can be reviewed at link https://github.com/hrshjain/oscar/pull/1.

Following is the snapshot of the package structure and Cucumber file:



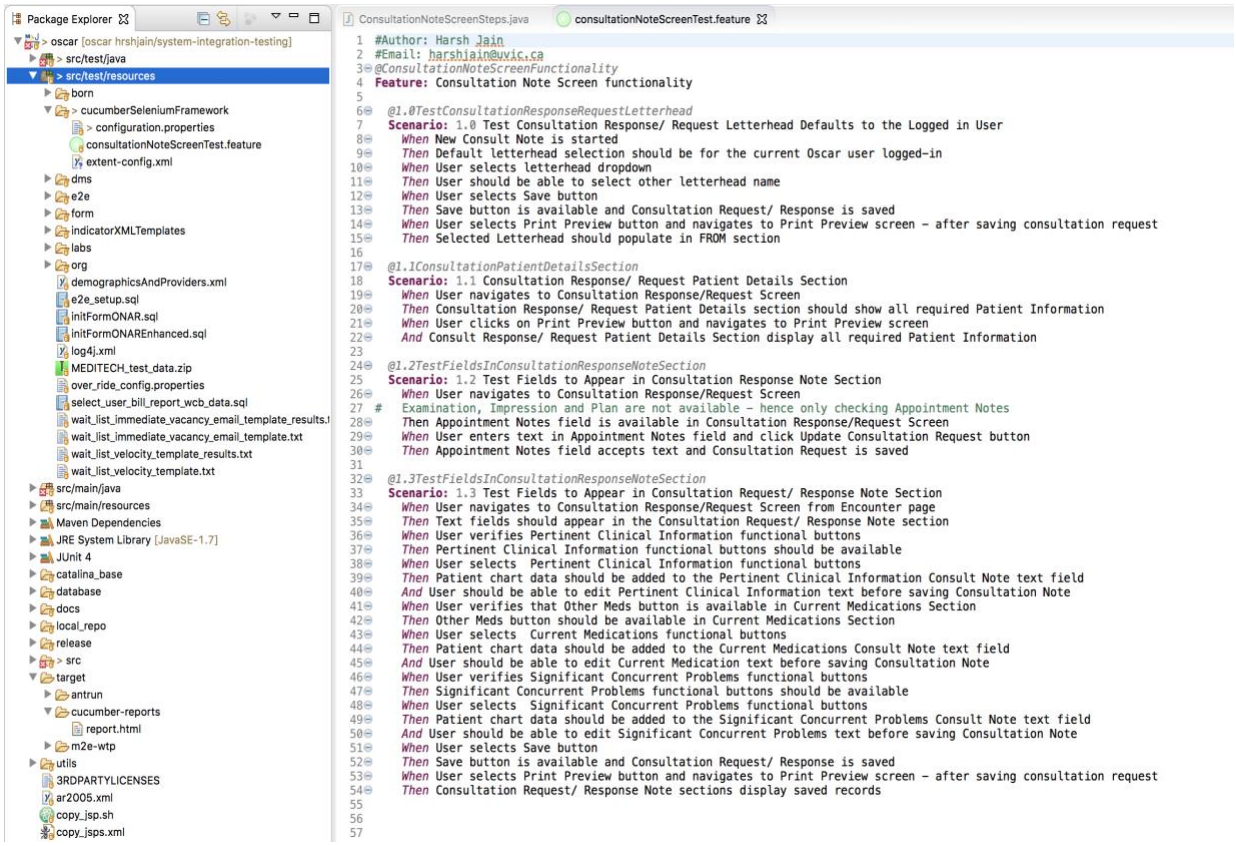
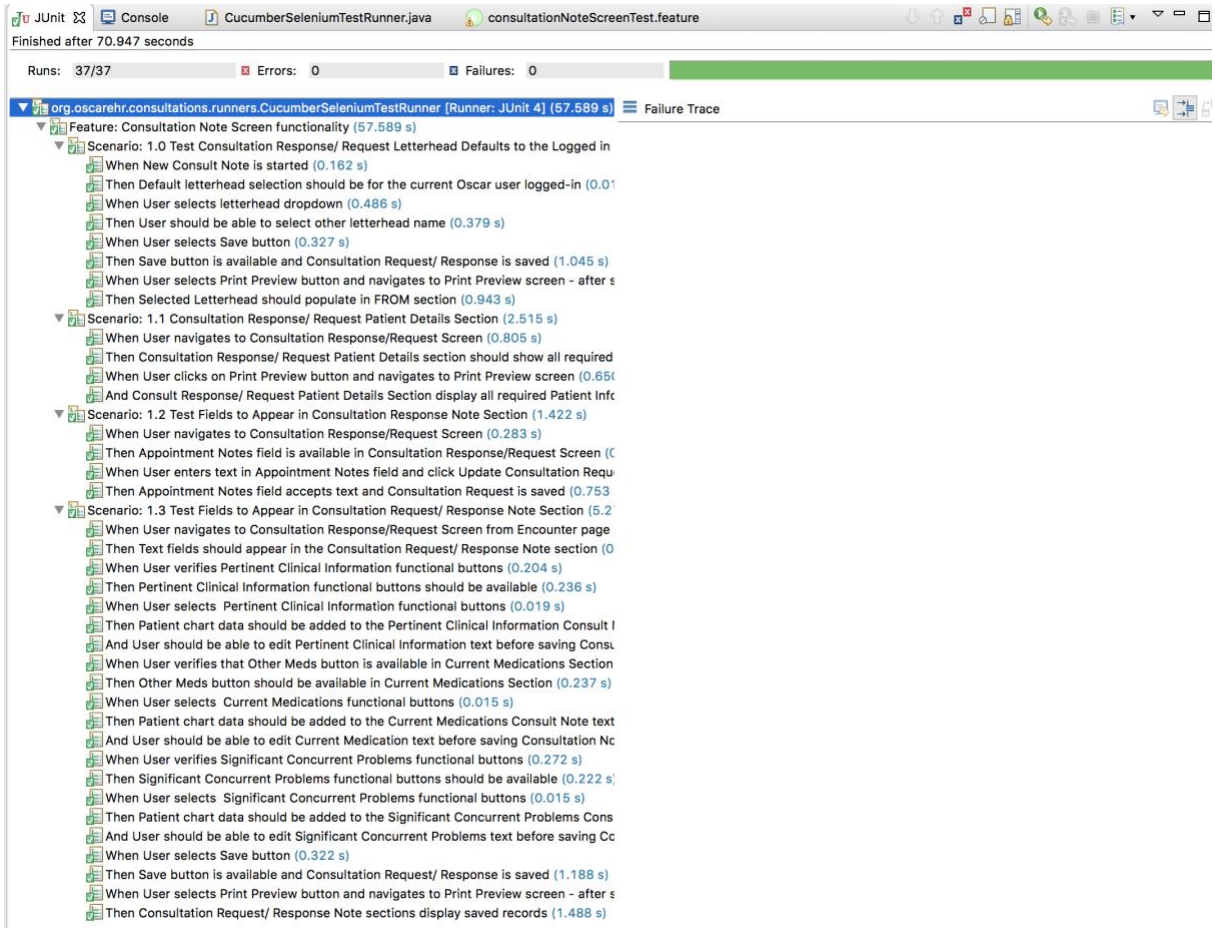
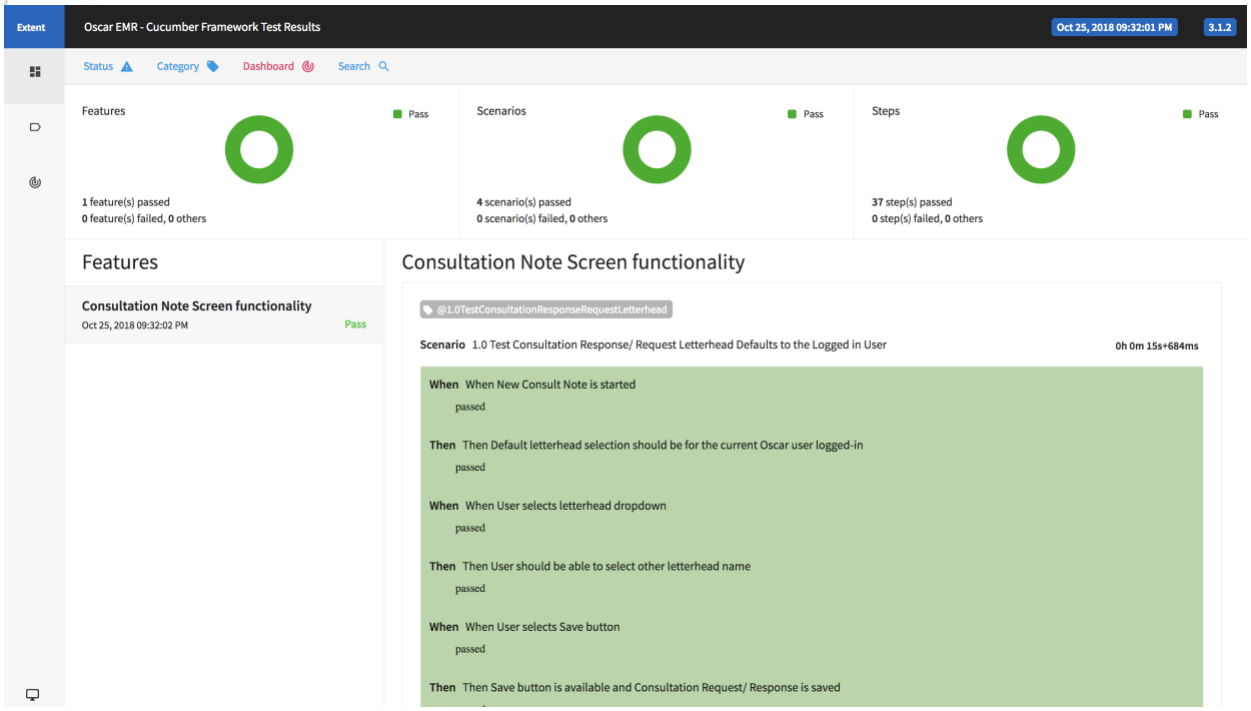


Figure 70 - Snapshot of Package Structure and final Cucumber Feature File

Final Result and Report

Following is the snapshot of final results generated after all the test cases of the Consultation Note Screen Functionality are run:


JUnit
Console
CucumberSeleniumTestRunner.java
consultationNoteScreenTest.feature
Finished after 70.947 seconds
Runs: 37/37
Errors: 0
Failures: 0
org.oscarehr.consultations.runners.CucumberSeleniumTestRunner [Runner: JUnit 4] (57.589 s)
Failure Trace
Feature: Consultation Note Screen functionality (57.589 s)
Scenario: 1.0 Test Consultation Response/ Request Letterhead Defaults to the Logged in
When New Consult Note is started (0.162 s)
Then Default letterhead selection should be for the current Oscar user logged-in
When User selects letterhead dropdown (0.486 s)
Then User should be able to select other letterhead name (0.379 s)
When User selects Save button (0.327 s)
Then Save button is available and Consultation Request/ Response is saved (1.045 s)
When User selects Print Preview button and navigates to Print Preview screen - after
Then Selected Letterhead should populate in FROM section (0.943 s)
Scenario: 1.1 Consultation Response/ Request Patient Details Section (2.515 s)
When User navigates to Consultation Response/Request Screen (0.805 s)
Then Consultation Response/ Request Patient Details section should show all required
When User clicks on Print Preview button and navigates to Print Preview screen
And Consult Response/ Request Patient Details Section display all required Patient Info
Scenario: 1.2 Test Fields to Appear in Consultation Response Note Section (1.422 s)
When User navigates to Consultation Response/Request Screen (0.283 s)
Then Appointment Notes field is available in Consultation Response/Request Screen
When User enters text in Appointment Notes field and click Update Consultation Requ
Then Appointment Notes field accepts text and Consultation Request is saved (0.753
Scenario: 1.3 Test Fields to Appear in Consultation Request/ Response Note Section
When User navigates to Consultation Response/Request Screen from Encounter page
Then Text fields should appear in the Consultation Request/ Response Note section
When User verifies Pertinent Clinical Information functional buttons (0.204 s)
Then Pertinent Clinical Information functional buttons should be available (0.236 s)
When User selects Pertinent Clinical Information functional buttons (0.019 s)
Then Patient chart data should be added to the Pertinent Clinical Information Consult
And User should be able to edit Pertinent Clinical Information text before saving Consu
When User verifies that Other Meds button is available in Current Medications Section
Then Other Meds button should be available in Current Medications Section (0.237 s)
When User selects Current Medications functional buttons (0.015 s)
Then Patient chart data should be added to the Current Medications Consult Note text
And User should be able to edit Current Medication text before saving Consultation Nc
When User verifies Significant Concurrent Problems functional buttons (0.272 s)
Then Significant Concurrent Problems functional buttons should be available
When User selects Significant Concurrent Problems functional buttons (0.015 s)
Then Patient chart data should be added to the Significant Concurrent Problems Cons
And User should be able to edit Significant Concurrent Problems text before saving Cc
When User selects Save button (0.322 s)
Then Save button is available and Consultation Request/ Response is saved (1.188 s)
When User selects Print Preview button and navigates to Print Preview screen - after
Then Consultation Request/ Response Note sections display saved records (1.488 s)


Extent
Oscar EMR - Cucumber Framework Test Results
Oct 25, 2018 09:32:01 PM
3.1.2
Status
Category
Dashboard
Search
Features
Pass
1 feature(s) passed
0 feature(s) failed, 0 others
Scenarios
Pass
4 scenario(s) passed
0 scenario(s) failed, 0 others
Steps
Pass
37 step(s) passed
0 step(s) failed, 0 others
Features
Consultation Note Screen functionality
Oct 25, 2018 09:32:02 PM
Pass
Consultation Note Screen functionality
@1.0TestConsultationResponseRequestLetterhead
Scenario 1.0 Test Consultation Response/ Request Letterhead Defaults to the Logged in User
0h 0m 15s+684ms
When When New Consult Note is started
passed
Then Then Default letterhead selection should be for the current Oscar user logged-in
passed
When When User selects letterhead dropdown
passed
Then Then User should be able to select other letterhead name
passed
When When User selects Save button
passed
Then Then Save button is available and Consultation Request/ Response is saved

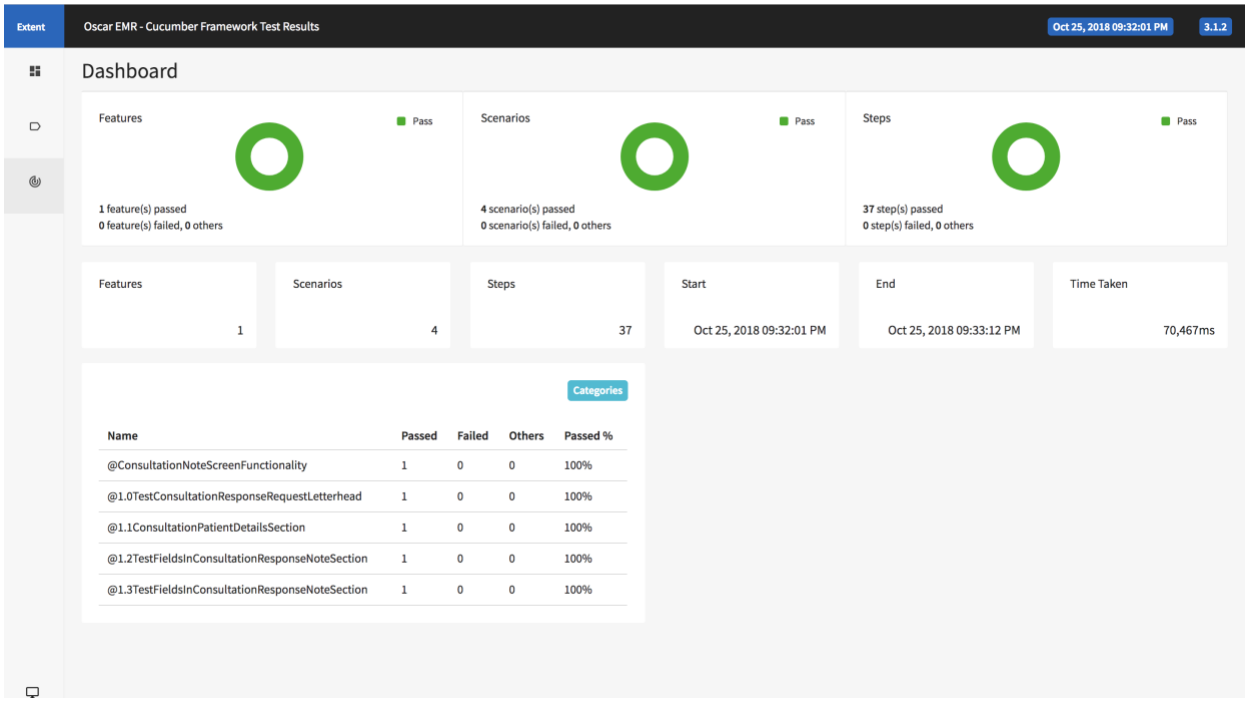
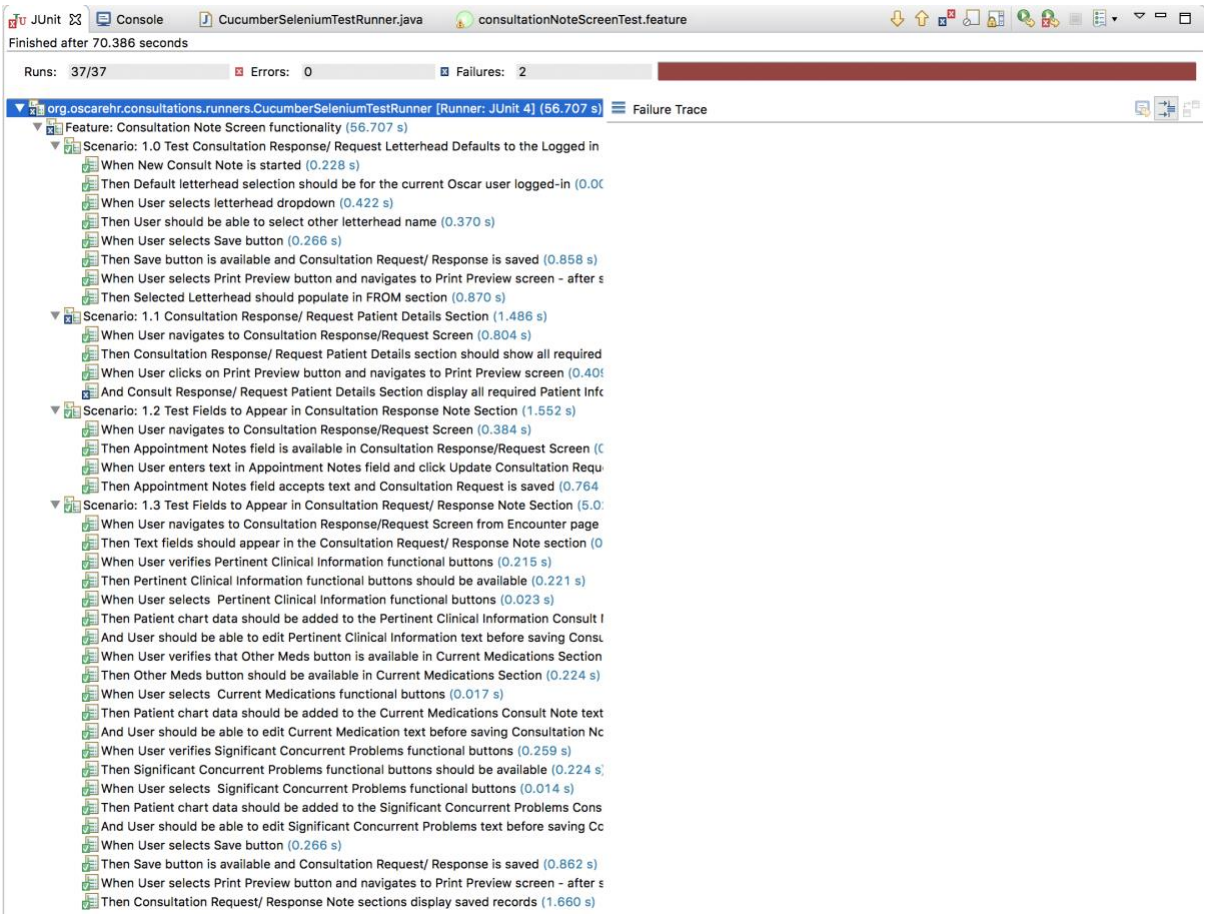
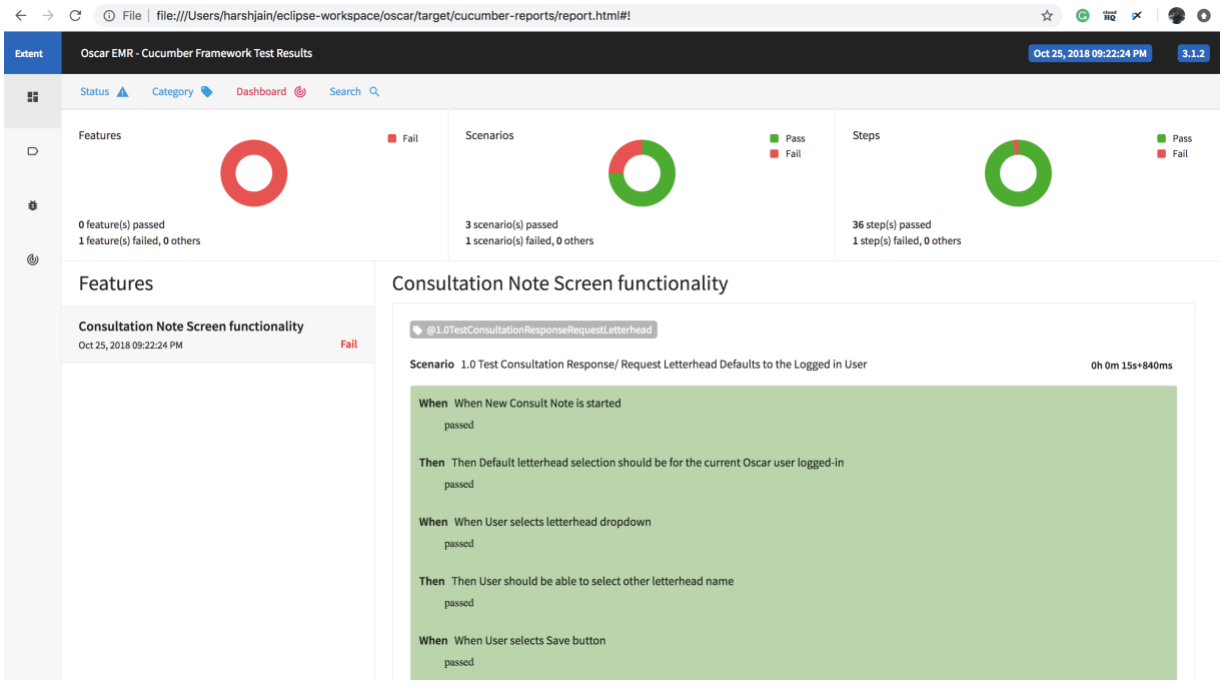


Figure 71 - Final Report of the Test Results

In case any of the test steps fail, Results will be displayed in following way:


JUnit
Console
CucumberSeleniumTestRunner.java
consultationNoteScreenTest.feature
Finished after 70.386 seconds
Runs: 37/37
Errors: 0
Failures: 2
org.oscarehr.consultations.runners.CucumberSeleniumTestRunner [Runner: JUnit 4] (56.707 s)
Failure Trace
Feature: Consultation Note Screen functionality (56.707 s)
Scenario: 1.0 Test Consultation Response/ Request Letterhead Defaults to the Logged in
When New Consult Note is started (0.228 s)
Then Default letterhead selection should be for the current Oscar user logged-in
When User selects letterhead dropdown (0.422 s)
Then User should be able to select other letterhead name (0.370 s)
When User selects Save button (0.266 s)
Then Save button is available and Consultation Request/ Response is saved (0.858 s)
When User selects Print Preview button and navigates to Print Preview screen - after
Then Selected Letterhead should populate in FROM section (0.870 s)
Scenario: 1.1 Consultation Response/ Request Patient Details Section (1.486 s)
When User navigates to Consultation Response/Request Screen (0.804 s)
Then Consultation Response/ Request Patient Details section should show all required
When User clicks on Print Preview button and navigates to Print Preview screen
And Consult Response/ Request Patient Details Section display all required Patient Inf
Scenario: 1.2 Test Fields to Appear in Consultation Response Note Section (1.552 s)
When User navigates to Consultation Response/Request Screen (0.384 s)
Then Appointment Notes field is available in Consultation Response/Request Screen
When User enters text in Appointment Notes field and click Update Consultation Requ
Then Appointment Notes field accepts text and Consultation Request is saved (0.764
Scenario: 1.3 Test Fields to Appear in Consultation Request/ Response Note Section
When User navigates to Consultation Response/Request Screen from Encounter page
Then Text fields should appear in the Consultation Request/ Response Note section
When User verifies Pertinent Clinical Information functional buttons (0.215 s)
Then Pertinent Clinical Information functional buttons should be available (0.221 s)
When User selects Pertinent Clinical Information functional buttons (0.023 s)
Then Patient chart data should be added to the Pertinent Clinical Information Consult
And User should be able to edit Pertinent Clinical Information text before saving Consu
When User verifies that Other Meds button is available in Current Medications Section
Then Other Meds button should be available in Current Medications Section (0.224 s)
When User selects Current Medications functional buttons (0.017 s)
Then Patient chart data should be added to the Current Medications Consult Note text
And User should be able to edit Current Medication text before saving Consultation Nc
When User verifies Significant Concurrent Problems functional buttons (0.259 s)
Then Significant Concurrent Problems functional buttons should be available
When User selects Significant Concurrent Problems functional buttons (0.014 s)
Then Patient chart data should be added to the Significant Concurrent Problems Cons
And User should be able to edit Significant Concurrent Problems text before saving Cc
When User selects Save button (0.266 s)
Then Save button is available and Consultation Request/ Response is saved (0.862 s)
When User selects Print Preview button and navigates to Print Preview screen - after
Then Consultation Request/ Response Note sections display saved records (1.660 s)


file:///Users/harshjain/eclipse-workspace/oscar/target/cucumber-reports/report.html#!
Extent
Oscar EMR - Cucumber Framework Test Results
Oct 25, 2018 09:22:24 PM
3.1.2
Status
Category
Dashboard
Search
Features
Fail
0 feature(s) passed
1 feature(s) failed, 0 others
Scenarios
Pass
Fail
3 scenario(s) passed
1 scenario(s) failed, 0 others
Steps
Pass
Fail
36 step(s) passed
1 step(s) failed, 0 others
Features
Consultation Note Screen functionality
Oct 25, 2018 09:22:24 PM
Fail
Consultation Note Screen functionality
@1.0TestConsultationResponseRequestLetterhead
Scenario 1.0 Test Consultation Response/ Request Letterhead Defaults to the Logged in User
0h 0m 15s+840ms
When When New Consult Note is started
passed
Then Then Default letterhead selection should be for the current Oscar user logged-in
passed
When When User selects letterhead dropdown
passed
Then Then User should be able to select other letterhead name
passed
When When User selects Save button
passed

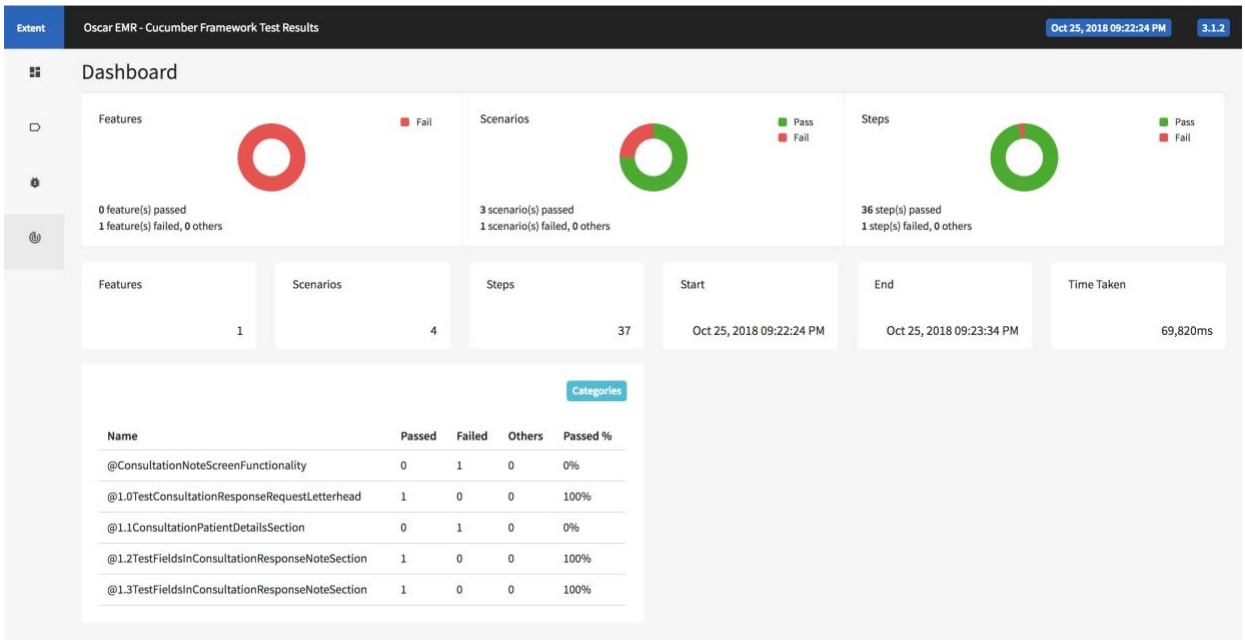


Figure 72 - Test Results (When any Step fails)

Conclusions and Recommendations

The author was successfully able to introduce BDD using Cucumber and Selenium for Oscar EMR. The project will be presented to McMaster University and will be integrated with its sprint release cycle. The current Cucumber-Selenium Testing Suite framework can be implemented in several parts of release cycle before and after deployment in several environments.

The Testing Suite can also be used as part of Regression and Smoke testing suite during every release. During regression, if any of the tests fail, testers can look at the results and logs to identify whether recent code changes have introduced a new bug in Oscar EMR. Sometimes, tests may fail because of test data or timeout issue. In such cases, testers can rerun the failing automated tests or can log into Oscar EMR and manually validate whether a new defect has been introduced or not.

Code Design and Architecture Decisions

While creating the design and architecture of codebase, several design decisions had to be made, with every decision having a list of advantages as well as disadvantages. These decisions will be discussed below. In future, however, future contributors can feel free to update the design and architecture keeping in mind the speed and efficiency of the test and making sure that codebase is easy to read, understand and it follows the standard Cucumber guidelines and Testing procedures.

Cucumber Feature File

Currently, Test Cases for 9 modules are present in the Oscar Atlassian wiki space. Every Module has few Submodules. For example, Consultation Module has 5 Submodules: Note Screen, List Screen, Preview Screen, Request/Response and Settings. As mentioned earlier, the Submodule Note Screen has been covered to fulfill the scope of this project.

There are several patterns in which these modules and submodules can be covered in a Cucumber feature file. After some analysis, it was decided that one Submodule should be mapped to one unique Cucumber Feature file for the project. This pattern keeps the code clean, organised and every Feature file can be easily tracked to its Oscar Wiki test Case. Keeping this in mind, all the test cases of Consultation Note Screen functionality have been covered in single feature file *consultationNoteScreenTest.feature*. In order to add automations test scripts for List Screen Submodule, a new feature file specifically for List Screen submodule will be created and this feature file will have its own unique Step Definition file.

Scenario and Steps

Consultation Note Screen in Oscar Wiki Space has 4 Test scenarios. Each scenario has been mapped to its corresponding scenario in Cucumber Feature File.

For example, the Scenario "*1.0 Test Consultation Response/ Request Letterhead Defaults to the Logged in User"* has its own scenario with matching annotation "*@1.0TestConsultationResponseRequestLetterhead"* in the *consultationNoteScreenTest.feature.* All Test Cases of this scenario have been converted to When/Then/And Gherkin steps in the Cucumber Feature file.

This ensures that Test Cases in Oscar Note Screen can be easily mapped to the Steps in Cucumber Feature file. This has one drawback which is that there are multiple When/Then/And Gherkin steps within a single scenario. However, all these Gherkin Steps are dependent of each and separating them will go against Cucumber Best Practices principle that Scenarios should be independent of each other [15].

Hence, for now, each scenario in Oscar Note Screen Test Case has been mapped to a Scenario in Cucumber feature file (highlighted in red box) and all the test cases in Oscar Wiki has been mapped to respective Gherkin steps (highlighted in blue box).

1.0 Test Consultation Response/ Request Letterhead Defaults to the Logged in User

Step ID	Description	Precondition	Action	Expected Results	Actual Results	Pass/Fail	Notes
#1	Consultation Response/ Request Letterhead should default to the logged-in user	1. Consultation Response/ Request screen is present 2. User is logged in with correct credentials	Verify that when new Consult Note is started, the default letterhead selection displays for the current Oscar user logged-in	Default letterhead selection should be for the current Oscar user logged-in			
#2	Ability to select other Letterheads selection	Provider records are registered within the OSCAR database	1. Select Letterhead drop down 2. Verify that other Letterheads can be selected using drop down Letterhead field	1. Letterhead is selectable from the drop down list 2. User should be able to select other letterhead name from the drop down			
#3	Ability to save Consultation Response/ Request		Select Save button	1. Save button is available on the floating bar 2. Consultation Request/ Response is saved			
#4	Ability to display Consultation Response/ Request print preview screen		Select Print Preview button	1. Consultation Request/ Response Print preview window pops up 2. Selected Letterhead should populate in FROM section			

```
1  #Author: Harsh Jain
2  #Email: harshjain@uvic.ca
3  @ConsultationNoteScreenFunctionality
4  Feature: Consultation Note Screen functionality
5
6    @1.0TestConsultationResponseRequestLetterhead
7    Scenario: 1.0 Test Consultation Response/ Request Letterhead Defaults to the Logged in User
8      When New Consult Note is started
9      Then Default letterhead selection should be for the current Oscar user logged-in
10     When User selects letterhead dropdown
11     Then User should be able to select other letterhead name
12     When User selects Save button
13     Then Save button is available and Consultation Request/ Response is saved
14     When User selects Print Preview button and navigates to Print Preview screen - after saving consultation request
15     Then Selected Letterhead should populate in FROM section
16
```

Figure 73 - Test Case mapped to Cucumber Feature File

Multiple Stepdefinition Files Alternative

One more design alternative is to divide the one Step Definition File into multiple Step Definition files on the basis of PageObjects accessed during test execution. It means that the Step Definition file ConsultationNoteScreenSteps.java can be divided into 7 Step Definition files like LoginSteps.java, OscarConsultationRequestSteps.java, EncounterSteps.java etc. on the basis of pages accesses during test execution.

However, in such case, the Gherkin steps and corresponding StepDefinition methods will need to be rewritten and become page specific. For example, the Gherkin Step *New Consult Note is started* will need to be divided into multiple page specific steps. Similarly, Stepdefinition methods for each of these Gherkin steps will need to be updated. To consider this alternative, test cases in Oscar Wiki will be first converted into highly detailed page-specific test scripts. These test scripts can then be converted to page-specific Gherkin steps and subsequently to page-specific StepDefinition methods.

Separate Repository

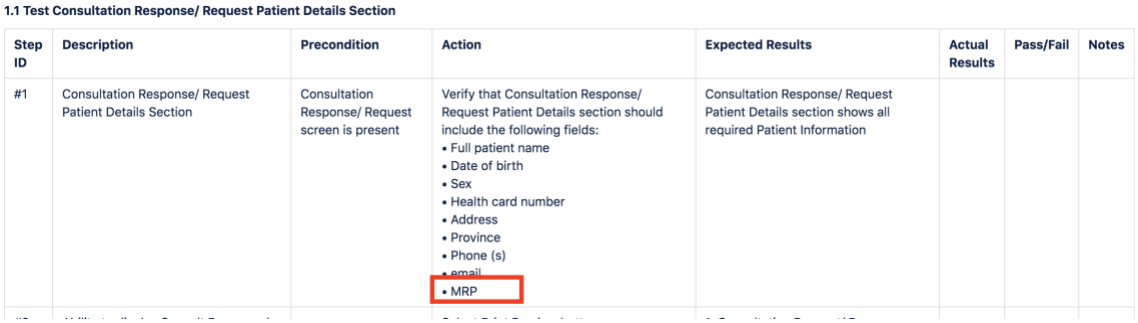
Currently, the framework has 7 packages and 1 folder which have been added to already existing oscar repository. However, it is highly recommend creating a separate standalone maven repository for this project. This will make the codebase easy to understand and contribute. The 7 Automation testing packages of the project do not fit into Unit test packages of oscar repository and the integration of Cucumber JUnit test runner with Oscar's native JUnit runner most probably will not approved by McMaster University.

The project can be easily transferred to a standalone repository by creating a maven project in eclipse and move all the packages and files from oscar to the new repository. There might be some additional restructuring of code and configuration changes required to get the new repository running. For example, in the new project, it is better to keep StepDefinition package in *src/main/java* directory instead of *src/main/test* directory. Also, the prefix *org.oscarehr* can be removed from all the package names.

Defects Discovered during Test Execution

During the execution of Cucumber-Selenium Test Suite, 2 defects were discovered. These defects are most probably Test Case Documentation issues and will be reported to Oscar EMR development team. Currently, the test scripts have been modified such that Cucumber steps are passing. Following are the defects:

1. For scenario *1.1 Test Consultation Response/ Request Patient Details Section*, Consultation Request\Response Page does not have MRP details of the patient.



1.1 Test Consultation Response/ Request Patient Details Section

Step ID	Description	Precondition	Action	Expected Results	Actual Results	Pass/Fail	Notes
#1	Consultation Response/ Request Patient Details Section	Consultation Response/ Request screen is present	Verify that Consultation Response/ Request Patient Details section should include the following fields: • Full patient name • Date of birth • Sex • Health card number • Address • Province • Phone (s) • email • MRP	Consultation Response/ Request Patient Details section shows all required Patient Information			

Figure 74 - Failing Test Scenario I

2. For Scenario *1.2 Test Fields to Appear in Consultation Response Note Section,* the text fields Examination, Impression, Plan are not available in Consultation Request\Response section (Scenario 3).

1.2 Test Fields to Appear in Consultation Response Note Section

Step ID	Step Description	Precondition	Action	Expected Results	Actual Results	Pass/Fail	Notes
#1	Fields to Appear in Consultation Response Note Section	Consultation Response screen is present	Verify that text fields should appear in Consultation Response/ Request Note Section: • Examination • Impression • Plan	Section with fields examination, impression and plan are available in Consult Response Form			

Figure 75 - Failing Test Scenario II

For Future Developers – Steps to add new functionality and shortcuts

Once, developers are familiar with the code structure, and concepts like PageObjects, Managers and Test Context, they can use the following steps to add automation scripts:

1. Determine Oscar EMR test cases which need to be covered for Automation using BDD.
2. Convert these test cases to Gherkin Scenario and Steps and add these Scenario and Steps TO respective Cucumber feature files.
3. For each Gherkin Step added in Step 2, create its corresponding method in Stepdefinition file.
4. Determine the webpages where each of these StepDefinition methods will be tested.
5. For each of these webpages, create a PageObject (if not already present) and create methods in the PageObject which will perform the web driver operations. Update Stepdefinition methods to call these PageObject methods.
6. Add Selenium code in the PageObject methods to perform the browser operations. A few things to take care of while adding Selenium test cases:

   a. To find web element locators, there are several plugins like Chropath available in Chrome and Firefox. These plugins can help in finding complex XPath and Css selector for a web element. One more way to figure out locators is to right click on element and copy its XPath\Css Selector.

   b. Make sure to utilize WebDriverManager, FileReaderManager and Text Context while writing the Selenium test scripts.

Developers can also add following features to the current codebases to improve the flexibility and coverage of the Testing Suite.

1. Currently the Tests are running successfully in local machine using Firefox. The tests can be further configured to run in remote servers and can be made more compatible on run in different Web browsers like Chrome, Internet Explorer and Safari.

2. Currently, before starting the test run, it is assumed that Patient Demographic is already present in Oscar EMR. However, the tests can be made more independent if Patient Demographic Test Cases are run before Consultation Note Screen test. This can also be done by creating Demographic Patient as part of Hooks.
3. Cucumber Extent Reports can be further enhanced to take screenshots whenever a test step fails.

Acknowledgements

I would like to express special thanks to my supervisor, Dr. Jens Weber for assigning me the project and providing me invaluable guidance throughout the project. I would also like to thank Dr. Neil Ernst for his constructive suggestions and feedback for success of the project. At last, I would also like thank Wendy Beggs, Graduate Secretary for her help and encouragement in keeping my progress on schedule.

References

[1] L. A. Winters-Miner, "Chapter 6 - Open-Source EMR and Decision Management Systems," in *Practical predictive analytics and decisioning systems for medicine : informatics accuracy and cost-effectiveness for healthcare administration and delivery including medical research*, London, Elsevier Inc, 2015, pp. 87-95.

[2] J. F. Smart, BDD in Action: Behavior-driven development for the whole software lifecycle, Shelter Island, NY: Manning Publications Co., 2015.

[3] A. H. Matt Wynne, "Gherkin Basics," in *The Cucumber Book: Behaviour-Driven Development for Testers and Developers, 2nd Edition*, Raleigh, NC: Pragmatic Programmers, LLC, 2017, pp. 25-35.

[4] M. F. Dorothy Graham, Experiences of Test Automation: Case Studies of Software Test Automation, Crawfordsville, Indiana: Pearson Education Inc., 2012.

[5] S. Laskawiec, "Brief comparison of BDD frameworks - DZone DevOps," DevOps Zone, Feb 2014. [Online]. Available: https://dzone.com/articles/brief-comparison-bdd. [Accessed Nov 2017].

[6] A. H. Matt Wynne, The Cucumber Book: Behaviour-Driven Development for Testers and Developers 2nd Edition, Raleigh, NC: The Pragmatic Programmers, LLC, 2017.

[7] H. Kaur and D. G. Gupta, "Comparative Study of Automated Testing Tools: Selenium, Quick Test Professional and Testcomplete," *Int. Journal of Engineering Research and Applications ISSN : 2248-9622*, vol. 3, no. 5, pp. 1739-1743, Sep 2013.

[8] V. Sarcar, "Singleton Patterns," in *Java Design Patterns*, New York, NY: Apress Media, LLC, 2016, pp. 17-21.

[9] C. Cocchiaro, Selenium Framework Design in Data-Driven Testing, Birmingham: Packt Publishing, 2018.

[10] U. Gundecha, "Chapter 5. Using the Page Object Model," in *Selenium Testing Tools Cookbook*, Birmingham, Packt Publishing Ltd, 2012, pp. 195-203.

[11] A. H. Matt Wynne, "Using Hooks," in *The Cucumber Book: Behaviour-Driven Development for Testers and Developers 2nd Edition*, Raleigh, NC: The Pragmatic Programmers, LLC, 2017, pp. 143-146.

[12] L. Sharma, "How to Sharing Test Context between Cucumber Step Definitions," [Online]. Available: http://toolsqa.com/selenium-cucumber-framework/sharing-test-context-between-cucumber-step-definitions/. [Accessed 15 09 2018].

[13] L. Sharma, "How to Share data between steps in Cucumber using Scenario Context," Tools QA, 20 09 2018. [Online]. Available: http://toolsqa.com/selenium-cucumber-framework/share-data-between-steps-in-cucumber-using-scenario-context/.

[14] V. Selvam, "Cucumber Extents Report," [Online]. Available: https://mvnrepository.com/artifact/com.vimalselvam/cucumber-extentsreport. [Accessed 27 10 2018].

[15] R. Parashchenko, "Cucumber Best Practices," [Online]. Available: https://github.com/strongqa/howitzer/wiki/Cucumber-Best-Practices. [Accessed 15

10 2018].

[16] L. Sharma, "Cucumber Automation Framework," [Online]. Available: http://toolsqa.com/cucumber-automation-framework/. [Accessed 29 10 2018].