Scenario 1

Browser-based end user testing using Selenium WebDriver

Steps:

1. Created a Maven Project (FlyAwayCucumber)
2. Pom.xml created to add Maven Dependencies that satisfy Browser based Selenium WebDriver Testing
3. Project Structure maintained to follow POM (Screenshot1\_FlyAway\_ProjectStructure)
4. Project is made a Cucumber project creating feature files and stepdefinitions
5. Testng.xml is created and the test is run through it
6. Pages created are (SRC/MAIN/JAVA/pages)

* LoginPage
* SearchFlightPage
* BookFlightPage
* ConfirmBookingPage
* MainExecution – this calls all other programs and executes the functions and does the validation.

1. Utilities (SRC/MAIN/JAVA/utilities)

DriverSetUp.Java is created. This Program setsup the driver initialization function and called from all functions before start of the class.

1. Feature File – FlyAway.feature(SRC/TEST/RESOURCES)

Scenarioncreated for login, search , booking and confirm booking

1. Stepdefinitions – FlyAwayStepDefinitions.java(SRC/TEST/JAVA/Stepdefinitions)

Step definitions file corresponding to the feature file is created

1. Console output screen shots are captured and attached(Screenshot2\_EmailableReport, Screenshot3\_MainExecution\_Consoleoutput, Screenshot4\_MainExecutionTest\_htmlpage)

Scenario 2

Unit testing for backend elements of the website using TestNG

MainExecutionPage contains testing components and suite is tested creating testing.xml. The reports are obtained in test-output folder of the project directory

Scenario 3

UserRegistration using RESTAPI and data are retrieved from feature file created in BDD

Steps:

1. Maven Cucumber Project created(FlyAwayRestAPI)
2. Pom.xml created with rest assured maven dependencies and other necessary ones
3. Feature file – RegisterUser.feature file created and data is sent from feature file to stepdefinitons
4. Stepdefinitions File – RegisterUserSteps.java corresponding to feature file with RestAssured code execution. Data is retreieved from feature file and then posted and validation is done
5. TestRunner File – CucumberTestRunner.java is created to run the Programs mentioned in the feature file and glue
6. Output is validated cross checking the database after post is done(screenshot of database screen attached.
7. Data sent in feature file is

| Email | test@example.com |

| Password | secret|

| ConfirmPassword | secret |

|Name |John |

| Address |123 Main Street |

| City |Anytown |

1. Data output in SQL console output after post operation is (Screenshot5\_RegistrationSucces\_SiteScreen, Screenshot6\_RegistrationConf\_SQLconsole)

Scenario 4

API testing with Postman on AWS cloud

1. AWS cloud logged in thru Simplilearn portal
2. Postman opened through AWS cloud

Steps to do this

* Log into AWS Management Console
* Navigate to the EC2 service
* Click on “Launch Instances” to create a new virtual machine
* Choose an Amazon Linux machine
* Select the instance type, Configure the instance details and add storage as needed
* Configure security groups to allow traffic on the ports intended to use for PostMan(typically 80)
* Launch the instance. Create or select an existing key pair for SSH access
* Once the instance is running, you can connect to it via SSH using a terminal.
* Use the private key from the key pair you selected during instance creation
* After connecting to EC2 instance download Postman and install it
* Start running postman. By default Postman should open its GUI. We can access it by connecting through our EC2 instance

Steps in Postman

1. Collection name Flyware is created

Two folders created

* Login
* Booking

1. Login

* Post – LoginAction User details are given as parameter and loginaction is posted. After that successful validation is tested. (Screenshot7\_Postman\_PostLoginAction)
* Post- Register New user details are sent as parameters and post action is requested. The test is written for validation(Screenshot8\_Postman\_Register)

1. Booking

* Get - HomePage request is sent and output is seen in console(Status code 200 obtained)
* Get - SearchFlight Page run with search for id=1 as src and id=4 as destination set as collection variables(Statuscode 200 is obtained as result)
* Get – BookFlight request is executed and test script is written for status code validation and receiving output of Complete purchase on the screen.( Screenshot9\_postman\_bookflight)
* Get – CompleteBooking – Request is executed and test script written for status code validation and receiving output on console as Your Bookings see your Bookings(Screenshot 10\_Postman\_CompBooking)
* Get – MemberBookings - Request is executed and test script written for status code validation and receiving output on console as Text output of total number of bookings done by that member(Screenshot 11\_postman\_mememberbookings)

Scenario 5

Automating the whole testing process by creating Jenkins job

Step 1 - **Create a New Jenkins Job**

1. java -jar jenkins.war Jenkins is started
2. Click on "New Item" to create a new Jenkins job.
3. Enter a name for your job(JENKINS\_FLYAWAY), select the "Freestyle project" option, and click "OK."

Step 2 - **Configure General Settings**

1. In the job configuration page, scroll down to the "General" section.
2. Check the "GitHub project" option if your Maven project is hosted on GitHub. Provide the GitHub project URL.
3. Optionally, you can add a description for your job.

Step 3 - **Build**

1. In the "Build" section, click "Add build step" and select "Invoke top-level Maven targets."
2. In the "Goals" field, specify the Maven goals you want to execute. For example, you can use clean install to clean the project and build it.

Step 4 – **Save the Job Configuration**

1. Scroll to the bottom of the page and click "Save" to save your Jenkins job configuration.

Step 5 – **Build Now**

1. Back on the Jenkins dashboard, select your newly created job.
2. Click "Build Now" to manually trigger the build and test your Maven project.

Step 6 – **Monitor the Build**

1. Once the build is triggered, Jenkins will execute the Maven goals specified in the job configuration.
2. You can monitor the build progress and view the console output to see the Maven build and test results.

This setup will run Maven project, which includes browser-based end user testing with Selenium WebDriver and unit testing for backend elements using TestNG, whenever a build is triggered based on your configured triggers.

GITHUB REPOSITORIES:

1. FlyAwayCucumber - <https://github.com/mohanadesi/FlyAwayCucumber>
2. FlyAwayRestAPI - <https://github.com/mohanadesi/FlyAwayRestAPI>
3. POSTMAN\_FLYAWAY - <https://github.com/mohanadesi/POSTMAN_FLYAWAY>
4. Project FlyAway Documents and Screenshots - <https://github.com/mohanadesi/Project-FlyAway-Documents-and-Screenshots>

Tools Used are:

**The following tools have been used:**

1. Eclipse IDE: Source code editing and modification
2. Selenium WebDriver: A browser testing framework for end-user black box testing
3. Postman: A standalone application for API testing
4. TestNG: A testing framework for unit testing of the backend elements of the website
5. Git: To connect and push files from the local system to GitHub
6. GitHub: To store the application code and track its versions
7. Specification document: Word document for Project Report

To create an end-to-end automation project following the specified steps, you'll need to implement various concepts and technologies. Here's a high-level overview of the concepts used in the project:

\*\*Concepts Used in the Project:\*\*

1. \*\*Page Object Design Pattern\*\*: This design pattern helps in structuring the automation code by separating the page elements and their interactions from the test scripts. Each page in the application has its own Page Object, which contains locators and methods to interact with the page's elements.

2. \*\*Selenium WebDriver\*\*: Selenium WebDriver is used for automating interactions with web elements. I used it to navigate through web pages, interact with elements, and perform various actions.

3. \*\*TestNG\*\*: TestNG is used for unit testing and test case management. It allows you to organize and execute test cases, handle dependencies, and generate detailed test reports.

4. \*\*REST API Testing\*\*: This involves testing the functionality of RESTful web services. I create d API tests to verify the behaviour of the endpoints: post the new user registration details

5. \*\*Postman\*\*: Postman is a popular tool for testing REST APIs. I used it to create and execute API tests for all the links in the pages

6. \*\*Jenkins\*\*: Jenkins is a continuous integration and continuous delivery (CI/CD) tool that helps automate the build, test, and deployment processes. I created a Jenkins job to automate various phases of your project.

\*\*Project Structure and Flow:\*\*

Here's the project structure and flow for the automation project:

1. \*\*Page Object Model (POM) Structure\*\*:

- Create separate Page Object classes for each web page in your application (e.g., HomePage, LoginPage, BookingPage etc).

- Each Page Object class should contain web element locators and methods to interact with those elements.

2. \*\*Selenium Scripts\*\*:

- Created test scripts using Selenium WebDriver to interact with the web pages. Use the Page Object design pattern to maintain the separation between test code and page interactions.

- Implemented test cases to cover various scenarios, such as logging in, searching for flights, booking a flight, and confirming the booking.

3. \*\*Unit Testing (TestNG)\*\*:

- Create unit tests for your backend classes and methods using TestNG.

- Define test suites and test cases to validate the correctness of your backend code.

4. \*\*REST API Module\*\*:

- Implement the REST API module with endpoint to register new user

5. \*\*Postman Scripts\*\*:

- Create Postman scripts to test the various endpoints.

- Define test cases, assertions, and data inputs to verify the behavior of the APIs.

6. \*\*Jenkins Automation\*\*:

- Created a Jenkins job that automates the entire testing process:

- Checkout the project code from your version control system (e.g., Git).

- Build the project (e.g., compile code, resolve dependencies).

- Execute Selenium tests against the web application.

- Run unit tests for backend classes and methods.

- Trigger Postman scripts to test the API endpoints.

- Generate test reports and logs.

By following these concepts and steps, i created a comprehensive automation project that covers both web application testing and REST API testing while integrating with Jenkins for automation.

**Conclusion on enhancing the application and defining the USPs (Unique Selling Points)**

 **Improved User Experience (UX)**:

* Enhance the user interface (UI) to make it more intuitive, visually appealing, and user-friendly.
* Streamline navigation and reduce the number of clicks required to perform common tasks.
* Implement responsive design to ensure the application works seamlessly across different devices and screen sizes.

 **Feature Expansion**:

* Add new features that provide unique value to users and address pain points. Consider features that competitors may not offer.
* Look for opportunities to integrate with third-party services or APIs to extend functionality.
* Implement user-requested features to increase user satisfaction.

 **Scalability and Reliability**:

* Ensure the application is scalable to handle increasing user loads.
* Implement redundancy and failover mechanisms to improve reliability and minimize downtime.

 **Customer Support and Engagement**:

* Offer excellent customer support channels, such as chat support, email support, or a knowledge base.
* Implement user engagement features like notifications, newsletters, or forums to keep users engaged and informed.

The USPs of our enhanced application should be centered around what makes it stand out in the market and how it addresses users' specific needs and pain points. By continually improving and innovating, we can create a compelling and competitive application that attracts and retains users.

*Project Prepared and Reported by – Mohana Venkatraman.*