**Authors Note:**

This project is an ongoing rework of a much earlier project that was originally intended for self-education. Therefore, if some of the code is less than 100% efficient, and the organization less than optimal, I will get to it at some point when I have more time.

**Purpose:**

The purpose of this project is to automate the testing of the functionality of the demoqa bookstore website. It checks things such as the login, the displayed text on various webpages, and the ability to manage the books in the user’s ‘bookshelf’ or ‘profile’. Unfortunately, at the time of writing (2/29/2024), the book webpages that the bookstore page redirects to, which contains the button to add a book to the user profile, are completely blank, preventing the user from adding the desired book. For the purposes of this project, I will skip over testing these pages, as they are virtually identical to other pages accessed through the user profile, and it is rather hard to write code tests for web elements that do not appear. Adding a book to the user’s profile will be done through automated calls to the bookstore API.

**Background:**

The code was written with the Java coding language. It uses TestNG for the testing framework, uses Selenium Webdriver in order to automate browser actions, the RestAssured library to make API calls, and implements Java database connectivity (JDBC) with Type-2 Driver to interact with a MySQL database.

**Executing Code:**

When running the code, I have the following:

jdk 1.8.0\_301

MySQL 8.0.34

chromedriver, geckodriver, or msedgedriver, that is compatible with the browser you want to use are stored in the ‘Driver’ folder.

Mysql-connection-j-8.0.33.jar in Java Build path, and user variable classpath pointing towards the jar.

Set up the ‘config.properties’ file, under the ‘Config’ folder, so that it names the browser type you wish to use for the test, and the credentials to the MySQL database.

Have a database created by running the SQL script file ‘createBookstoreDatabase’ inside the MySQL Workbench.

Maven Dependencies:

* selenium-java: 3.4.0
* testng: 6.9.9
* poi: 3.14
* poi-ooxml: 3.14
* extentreports: 2.41.2
* commons-io: 2.11.0
* reporting: 1.1.4 [test scope]
* guice: 5.0.1
* json: 20190722
* jackson-databind: 2.10.3
* rest-assured: 4.2.0

Eclipse Marketplace:

* Cucumber Eclipse Plugin 1.0.202106240526
* Eclipse Enterprise Java and Web Developer Tools 3.8.2
* Eclipse Web Developer tools 3.8.2
* Eclipse XML Editors and Tools 3.8.2
* TestNG for Eclipse

**Database Organization and Purpose:**

This database stores all the information about the books in the database, sets of real and fake login credentials, and webpage element identification information.

The database is divided into 3 tables: WebElementDataTbl, UserLoginTbl, and BookDataTbl.

WebElementDataTbl is used to store the identifying information for elements on the webpage, so that the Selenium Webdriver code can use that information to interact with the webpage elements. There are three columns. The name of the element is under element\_name, and it contains information about the page where the element is found and information about what the element is or does. The identifier\_type column specifies whether the information in the element\_data column is xpath information or an element id. This so the java code can correctly identify whether to have selenium search for the element by xpath or by element id, with the added benefit of being a potential term for table filtering if needed.

The ­UserLoginTbl is used to store both real and fake login credentials for the website. This allows for data driven test cases. The username column stores usernames, the pswrd column stores passwords. The valid\_creds column is a Boolean that identifies whether the username/password combination is valid, and will lead to login. The userid column is an id for the user that is used for user-specific API calls.

The BookDataTbl is used to store information about the various books in the bookstore that API calls and webpage text is checked against. The column names are self-descriptive: isbn, title, subtitle, author, publish data, publisher, pages, descript, and website.

**Folder and Package Structure:**

I have used a Page Object Model (POM) design pattern as the basis for my organization. The java code that I have written is stored in the ‘src’ folder. The ‘Driver’ folder stores the browser specific webdrivers, the ‘Config’ folder contains a file to allow for the defining of the browser to run the test, the URL of the bookstore login page, and the credentials of the database to be connected to. The screenshots folder will contain snapshots of the browser, as visual evidence of what is on the page, for proof of testing and as a method to help troubleshoot failed tests.

A screenshot of a computer program

Description automatically generated

Under ‘src’, there are 5 packages. ‘org.automation.seleniumdemo.assertions’ contains classes with methods that will evaluate complex conditions, and return a Boolean, for an easily evaluable Assertion method in the test case code. ‘org.automation.seleniumdemo.base’ contains the class that will instantiate and end web driver instances. ‘org.automation.seleniumdemo.pages’ contains the classes that will become objects that allow for interactions with the webpages. ‘org.automation.seleniumdemo.testcases’ contains the test cases. ‘org.automation.seleniumdemo.utility’, contains multiple utility classes used for handling snapshots of the browser, reading through the configuration text file.

**Description of Classes and Their Methods:**

org.automation.seleniumdemo.assertions:

Contains three classes: ‘Compare’, ‘ElementExists’, and ‘VerifyBookInfo’

The ‘Compare’ class contains a single method, ‘validatePageURL’ that evaluates whether the browser URL is equal to the expected URL, takes a snapshot of the page, and returns true or false.

The ‘ElementExists’ class contains two methods: ‘validateProfileBookElementExists’ and ‘validateProfileBookElementGone’. These methods check whether the element for a book exists on the profile page of the bookstore app, and are used to validate if books have been successfully added and removed.

The ‘VerifyBookInfo’ class contains two methods: ‘checkBookPage’ and ‘dbCompAPI’. ‘checkBookPage’ will compare the information about a specific book as it is shown on the webpage against the information about that book as it is stored in the database, and returns true or false. It also captures a snapshot of the webpage. ‘dbCompAPI’ compares the information about a book as taken from the API against the value stored in the database.

org.automation.seleniumdemo.base:

Contains a single class, ‘DriverInstance’, which has two methods: ‘startDriverInstance’ and ‘closeDriverInstance’. Using TestNG annotation, ‘startDriverInstance’ is set to run before each tested method, and ‘closeDriverInstance’ after each tested method. ‘startDriverInstance’ will check the config file in order determine which driver type to use, set up the driver and start the browser, set up wait timings, and go to the testing website. ‘closeDriverInstance’ will close the browser when each test method is complete.

org.automation.seleniumdemo.pages

Contains five classes: ‘BookPage’, ‘BookStorePage’, ‘LoginPage’, ‘ProfileBookPage’, and ‘ProfilePage’.

The ‘BookPage’ class represents the webpage accessed through the bookstore page, which will describe the information about any selected book. The constructor for the object will pull element locator information from the database the driver and keep the values as properties for the object. The ‘clickAddToCollection’ method will add the book that is currently displayed on the webpage to the user’s profile (assuming that the user has already logged in). The ‘returnToStore’ method will bring the user back to the bookstore page. The ‘goToProfile’ method will bring the user to the profile page for the logged in user. The ‘gatherBookInfo’ method will return information about the book that is displayed on the webpage. At time of writing, the webpage is broken, and the methods in this class are mostly non-functional.

‘BookStorePage’ class represents the bookstore webpage, and is used to select books to find information on. The constructor for the object will pull element locator information from the database the driver and keep the values as properties for the object. The ‘selectBook’ method allows the user to click a book, when given a book title. The ‘goToProfile’ method will bring the user to the profile page for the logged in user. The ‘getBookText’ will return the currently displayed information about a specified book.

A screenshot of a computer

Description automatically generated

The ’LoginPage’ class represent the login webpage, and is used for login. The constructor for the object will pull element locator information from the database the driver and keep the values as properties for the object. The ‘enterUsername’ method will input the provided username value into the username field on the webpage. The ‘enterPassword’ method will input the provided password value into the password field. The ‘clickLogin’ method will have the browser click the login button, to execute the login process.

A screenshot of a computer

Description automatically generated

The ‘ProfileBookPage’ class represents the webpage accessed through the profile page, which will describe the information about any selected book. The constructor for the object will pull element locator information from the database the driver and keep the values as properties for the object. The ‘returnToProfile’ method will click the button to bring the user back to the user’s profile page. The ‘goToProfile’ method will bring the user to the profile page for the logged in user (which is a different button than ‘returnToProfile’ uses). The ‘gatherBookInfo’ method will return information about the book that is displayed on the webpage.

A screenshot of a computer

Description automatically generated

The ’ProfilePage’ class represents the user’s profile page when they have logged in. The constructor for the object will pull element locator information from the database the driver and keep the values as properties for the object. The ‘clickBookStore’ method will bring the user to the bookstore page. The ‘showMaxRows’ method will set the number of displayed rows to 100, to maximize the number of books that can be displayed on the page at once. The ‘selectBook’ method will bring the user to a book description page. The ‘deleteBook’ method will delete one of the books from the user’s profile. The ‘clickDeleteAllBooks’ method will have the browser click the button that will remove all of the books on the user’s profile. The ‘getBookText’ method will return the information about a specific book that is displayed on the page.

A screenshot of a computer

Description automatically generated

org.automation.seleniumdemo.testcases:

Contains two classes: ‘TS\_001\_ValidateBookStoreAndProfile’ and ‘TS\_002\_CheckLoginFunction’.

‘TS\_001\_ValidateBookStoreAndProfile’ contains a series of tests meant to validate the functionality of the majority of the bookstore pages. ‘tc\_001\_checkAPIs’ will happen before any other test in this calls, and it makes a series of calls of API calls in order to validate that key APIs that are called by the webpages will function before the tests through the browser begin. ‘tc\_002\_checkStorePage’ verifies the presence of all the expected books in the bookstore page, and that the correct text is displayed for them. ‘tc\_003\_checkProfile’ adds a couple of select books from the database, verifies their presence on the user’s profile, that the selected books have the correct text, and that the different book deletion methods work. ‘tc\_004\_checkBookInfo’ adds every book from the database into the user’s profile, and verifies that the books are present on the profile page, that the displayed information is correct, and that the information displayed on each book’s description page is correct.

‘TS\_002\_CheckLoginFunction’ contains a two methods. ‘getLoginData’ pull a 2d array of valid and invalid user credentials from the database, and provides them as a dataset for ‘tc\_001\_checkLogin’ to iterate through. ‘tc\_001\_checkLogin’, runs through each set of credentials that the data provider ‘getLoginData’ gives it, and attempts to use that information to log in, to validate whether real users can log in and that invalid users cannot.

org.automation.seleniumdemo.utility:

Contains four classes: ‘APICalls’, ‘CapScreenshot’, ‘ConfigUtil’, and ‘JDBCUtil’.

‘APICalls’ contains the methods that are responsible for making calls to the bookstore API. ‘getBookStoreBookCount’ calls for a list of the books in the bookstore, and returns the count of them. ‘printBookStoreList’ prints the list of books in the bookstore to the console. ‘getBookInfo’ will return the information about a specified book ISBN. ‘isBookUnderUser’ checks if a specified book is listed as being in a specified user’s profile, and returns true or false. ‘getUserBookCount’ returns the count of books under a specified user's profile. ‘getUserBookList’ returns the list of books under a specified user's profile. ‘addBook’ will add a specified book to the user's profile, while ‘deleteBook’ will delete the specified book from the user’s profile. ‘replaceBook’ will replace a specified book from the user's profile with another book. ‘clearAllBooks’ will remove all books from the specified user's profile.

‘CapScreenshot’ contains a single method, ‘screenshot’, which captures a snapshot of the browser being automated, and saves it in the ‘Screenshots’ folder with a descriptive name.

‘ConfigUtil’ contains a single method, ‘fetchProperyValue’, which when given a key, will parse the ‘config.properties’ file and return the desired value.

‘JDBCUtil’ contains eight methods that establish and end connections to the database, as well as perfom the longer and more complex handling of the data retrieved from the database. ‘getMySQLConnection’ will establish and return a connection to the database. ‘cleanupConnection’ will perform the correct closing of the connection to the database. ‘getLogin’ will return a 2d array of all the login information pulled from the database. ‘getFunctionalLogin’ will return a 2d array of all of the login information in the database that is marked as being a valid combination. ‘getSingleFunctionalLogin’ will return an array for a single set of credentials marked as being valid. ‘getSingleBookInfo’ will return the complete set of information on a book listed in the database. ‘getTwoBookInfo’ will return a 2d array containing the complete set of information on two books in the database. ‘getAllBookInfo’ will return a 2d array containing all book information listed in the database.

**Class Relationships:**

A screenshot of a computer screen

Description automatically generated

**Method Flowcharts:**

tc\_001\_checkAPIs()

A screenshot of a phone

Description automatically generated

tc\_002\_checkStorePage()

A screenshot of a black and white screen

Description automatically generated

tc\_003\_checkProfile()

A screenshot of a computer screen

Description automatically generated

tc\_004\_checkBookInfo()

A screenshot of a phone

Description automatically generated

TS\_002\_CheckLoginFunction:

A screenshot of a cell phone

Description automatically generated

Compare - validatePageURL():

A screenshot of a cell phone

Description automatically generated

ElementExists - validateProfileBookElementExists():

A screenshot of a phone

Description automatically generated

ElementExists - validateProfileBookElementGone():

A screenshot of a phone

Description automatically generated

VerifyBookInfo - checkBookPage()

A diagram of a computer program

Description automatically generated

VerifyBookInfo - dbCompAPI()

A diagram of a computer program

Description automatically generated with medium confidence

DriverInstance - startDriverInstance():

A screenshot of a computer error

Description automatically generated

DriverInstance - closeDriverInstance()

A close-up of a sign

Description automatically generated

BookPage - BookPage Constructor:

A close-up of a black and white label

Description automatically generated

BookPage - clickAddToCollection()

A screenshot of a cell phone

Description automatically generated

BookPage - returnToStore()

A screenshot of a cell phone

Description automatically generated

BookPage - goToProfile()

A screenshot of a cell phone

Description automatically generated

BookPage - gatherBookInfo()

A white rectangular object with black text

Description automatically generated

BookStorePage - BookStorePage Constructor:

A close-up of a black and white label

Description automatically generated

BookStorePage – selectBook()

A screenshot of a black and white screen

Description automatically generated

BookStorePage – goToProfile():

A screenshot of a cell phone

Description automatically generated

BookStorePage – getBookText():

A close-up of a sign

Description automatically generated

LoginPage – LoginPage Constructor:

A close-up of a black and white label

Description automatically generated

LoginPage – enterUsername():

A close-up of a sign

Description automatically generated

LoginPage – enterPassword():

A close-up of a sign

Description automatically generated

LoginPage – clickLogin():

A screenshot of a black and white screen

Description automatically generated

ProfileBookPage – ProfileBookPage Constructor:

A close-up of a black and white label

Description automatically generated