**CS4227**

**Systems Architecture and Design**

Automated Testing Framework

*James Reidy*, 14147009

*Aoife Watson*, 14159295

*Jiasen Tian*, 16060245

*George Rickard*, 14118637

**2. Table of Contents**

**3. Requirements**

**Scenario Outline**

The Automation Testing Framework is used for testing Front End Web Applications that makes use of external 3rd party

**Use Case Descriptions**

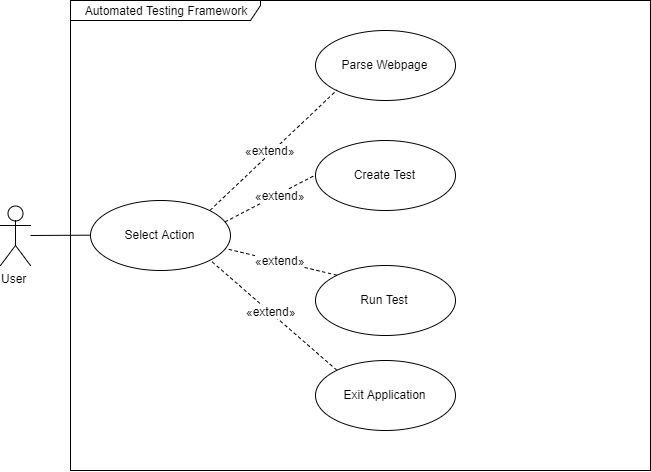
|  |  |
| --- | --- |
| **Use Case** | **Select Option** |
| *Success Scenario* | *System Response* |
| 1. User selects desired action | 1. System returns appropriate window |
| *Extension* |  |
| 1. Click “Exit” button | 1. System exits application |

|  |  |
| --- | --- |
| **Use Case** | **Parse Page** |
| *Success Scenario* | *System Response* |
| 1. Enter webpage URL 2. Enter a name for the parsed file 3. Select XML or JSON format 4. Click “Parse” button | 1. System checks validity of URL 2. System parses page into desired format 3. System returns message indicating successful parse |
| *Extension* |  |
|  | 1. System returns message indicating unsuccessful parse |
| 1. Click “Main Menu” button | 1. System presents main menu |

|  |  |
| --- | --- |
| **Use Case** | **Create Test** |
| *Success Scenario* | *System Response* |
| 1. Select page elements to test 2. Enter inputs/actions to take on each element 3. Select XML or JSON format 4. Click “Save Test” button | 1. System checks if path exists 2. System saves tests 3. Test name and path are written to text file |
| *Extension* |  |
|  | 1. System returns error message indicating that a test with that name already exists |
| 1. Click “Main Menu” button | 1. System presents main menu |

|  |  |
| --- | --- |
| **Use Case** | **Run Test** |
| *Success Scenario* | *System Response* |
| 1. Select desired test to run 2. Click “Run” button | 1. System runs tests and logs results |
| *Extension* |  |
| 1. Click “Main Menu” button | 1. System presents main menu |

**Use Case Diagram**

****

**Discussion on Architectural Use Cases**

**Discussion on Tactics to Support Architectural Use Cases**

**4. Discussion of Interceptor Architectural Pattern and Design Patterns**

**3 pages max! discuss context and consequences and their support in scenario**

**Interceptor**

**Strategy – write more about strategy than other 5**

The strategy was an obvious choice to use in our project because of the different file formats we used in our project to change at any time.

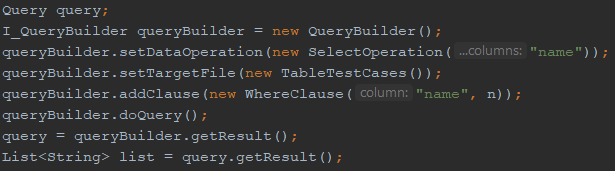
**Factory**

**Command**

**Memento**

**Builder**

Our implementation of the Builder was used in the QueryBuilder system in our application. QueryBuilder constructs simple “Queries” to be performed on CSV files in an SQL-like fashion.



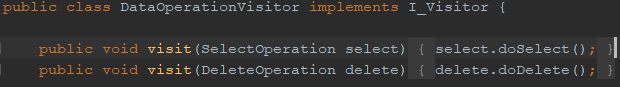
SelectOperation in the example shown takes in the columns to be selected, TableTestCases acts as a wrapper for the path to the text file containing the data, and WhereClause narrows the data down as the user desires. The SQL equivalent to the example shown would be “SELECT name FROM TestCases WHERE name = n”.

**Visitor**

As well as incorporating the Builder design pattern, the concrete Query it constructs utilizes the Visitor design patter to perform an action depending on the DataOperation being performed. Shown below is the visitor being passed into the dataOperation.



Once the visitor is passed in it performs the necessary action based on its type; select, delete, insert or update.



**Singleton**

**5. Discussion of System Architecture**

**One or more diagrams**

**Model View Controller**

**6. Structural, Runtime and Architectural Diagrams**

**Structure**

**Runtime**

**7. Demonstration of Extensibility**

**8. Documentation of Added Value**

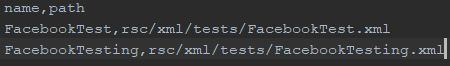
**Interceptor Priority**

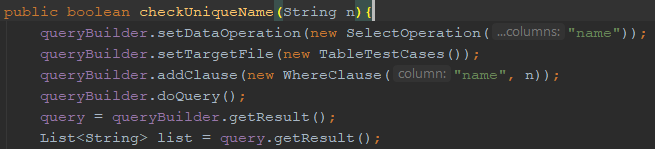
**Selenium**

**Properties**

**QueryBuilder**

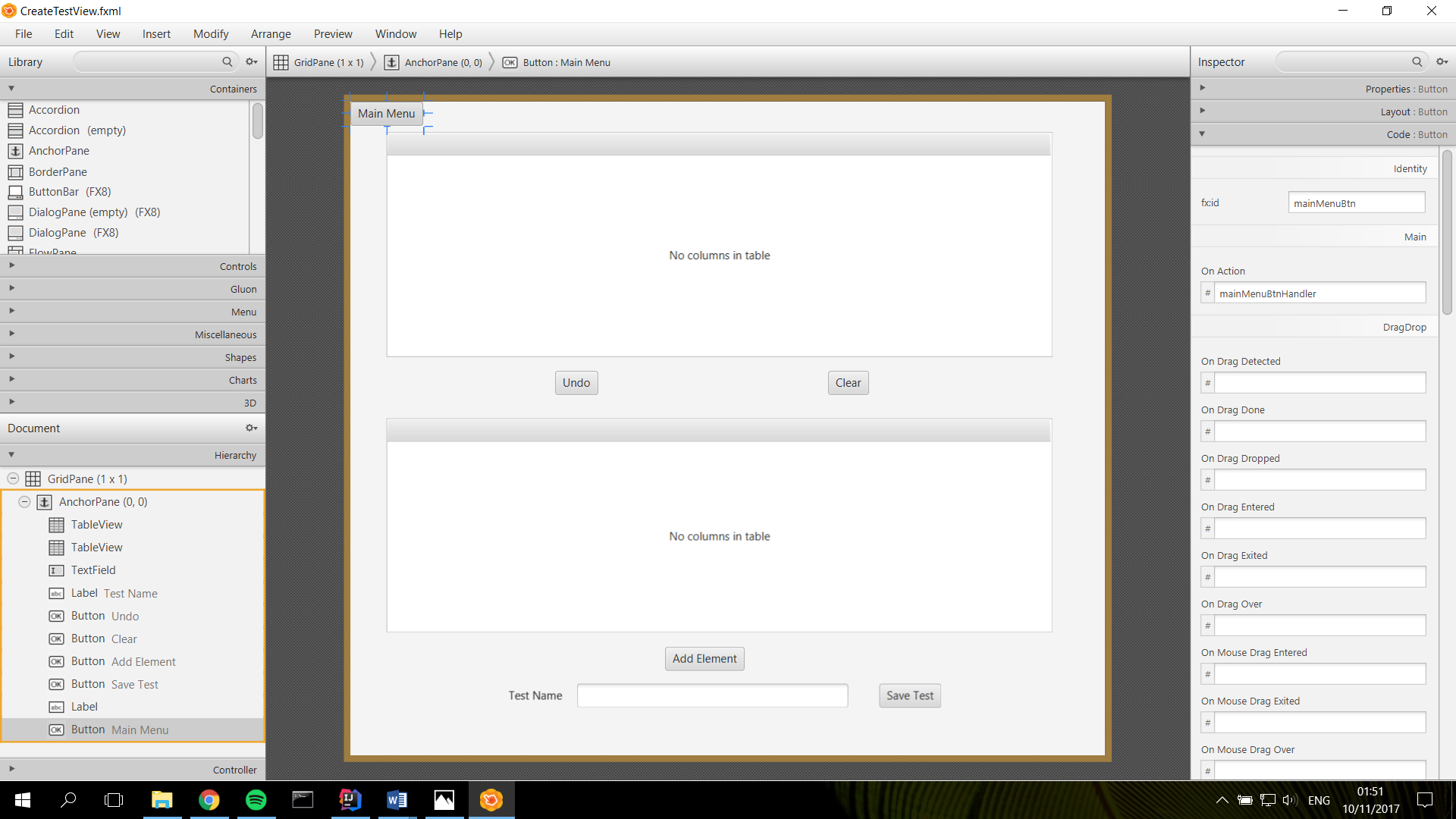
As discussed earlier in this report, QueryBuilder is an attempt to model CSV files as if they were tables in an SQL database by allowing the user to perform select, update, insert and delete operations on it and passing in where clauses when needed. Its creation was motivated from a project done last semester that also made use of CSV files. Information was pulled from a single class and team members created members to pull very specific information, which became messy, convoluted and posed a security concern. Thus, attempting to model an industry standard in its most basic form felt like a fitting solution to this problem. Below is the format of the text file required and a SelectOperation being performed on this file.





**JavaFX and Scene Builder**

JavaFX is Oracle’s latest set of tools for developing GUIs. We used it in our project for two reasons; ease of use thanks to Scene Builder, and to gain experience with newer technologies instead of working with old, soon-to-be deprecated JPanels. It allows for drag and drop capabilities of different widgets and editing different parameters all from the same window. This information is then stored on an FXML file. Shown below is Scene Builder in action on CreateTestView.fxml. Along the left-hand side are available widgets, while the right-hand side contains information about the selected widget, such as ID and handler.



Controllers use the @FXML annotation for objects to create the link between the controller and the view. The first line of code shown below is the FXML for the “Main Menu” button on the upper left-hand corner of the window above. The next line is the corresponding controller calling that instance of that button.



**Git**

Git was our version control system of choice for this project. While there are many platforms that utilize git i.e. BitBucket, GitLab, we went with GitHub since it what we were most familiar with. Using version control allowed us to seamlessly make changes to code without having to worry too much about breaking or accidentally removing work by other team members.

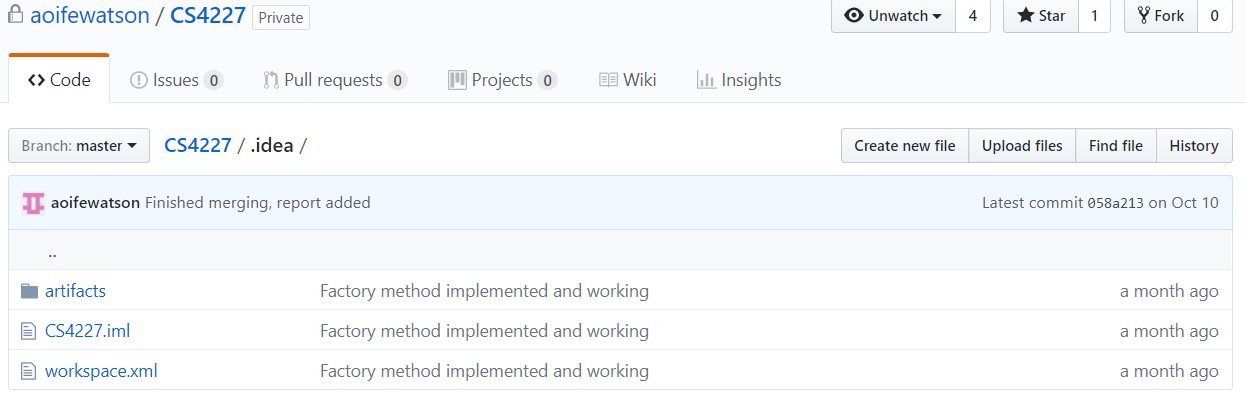
**9. Evidence of Testing**

**10. Discussion of Problems Encountered**

**Provide evidence of how all this was resolved**

**.gitignore**

Early on in implementation we experienced issues with Git. Code was failing to compile due to configuration issues within the IDE we were using, IntelliJ. The root of these issues was the misconfiguration of gitignore, which contains a list of files and/or file types to be ignore by Git. These include configuration files specific to each machine. Displayed below is an early repository with workspace.xml included in Git.



The issue was resolved by creating a new repository for the project with a .gitignore file that had been created specifically for IntelliJ.

**File Paths**

**Selenium Dependency**

**Dividing work – lots of waiting on people to finish bits before others could be completed.**

**11. Evaluation of Support for Non-Functional Requirements**

**“should be done using scenarios” whatever that means?**

**12. References**

**Give reference for any images shown or tools used (including jdom etc.)**

**13. Contributions**

**Few screenshots from github should do this**