Technical report

1.0 Project Overview

1.1 Description of the problem being solved

- The problem that my code aims to solve is the creation of an e-commerce GUI, that implements correct coding practices. This is achieved by implementing Java practices such as inheritance, polymorphism and abstraction. These practices help to reduce repeated code as much as possible. I also tried to ensure that the final product was as user friendly as possible, and easy to understand. This has been achieved by ensuring that buttons and labels are placed and named correctly, and that any errors that may occur are dealt with “gracefully” and provide the user with a brief description of what may be causing the error.

1.2 Objectives and scope of the project

- The objectives of my project included:
    - Maintaining good coding standards
    - The target user is able to use the software easily
    - The code is of high quality, secure and performs well
    - Utilising key java practices such as inheritance,polymorphism, abstraction and encapsulation
    - The project can use information stored in a file/database and perform CRUD (Create, Read, Update, Delete)
    - The code is well commented, and easy to follow
    - Exceptions/errors are dealt with gracefully
    - The use of OOP (Object Oriented Programming) principles

- The scope of the project was to produce code that was written in Java and was functional on any OS (Operating System) whether it be Windows, MacOS or Linux. I was given the option to use any IDE of my choice, however I decided to use IntelliJ as it streamlines the process of writing Java and is much more updated, and user-friendly than an IDE such as Eclipse. I was given roughly two months to complete this project with a deadline of 09/01/2023 @ 3:00pm.

1.3 Target users and stakeholders

- Whilst creating the software, the target user I had in mind was any potential customers that were looking to do some online grocery shopping. Whilst creating the code, I tried to make it as basic as possible for the user so that the shopping experience was as straight-foward as possible. I took inspiration from other e-commerce retailers such as Amazon,Currys and Tescos to see how their online shopping was conducted.

- The stakeholders of my project include myself, my lecturer and any potential customers online.

1.4 Overview of the proposed solution

- In the end, I decided to settle on creating as little buttons/labels as possible and making the GUI as simple as possible whilst looking professional. This was beneficial as it streamlines the process of adding products to the users basket, and checking out. Additionally, keeping the process as simple as possible benefits me as the amount of errors that could occur is minimised and any errors that do occur can be solved with exception handling methods such as try/catches.

2.0 Functional requirements

2.1 Detailed functional requirements of the system

- The first functional requirement provided to me was that the project must have a GUI(graphical user interface) that is built with either Java Swing or JavaFX. I personally chose to use Java Swing as I would prefer to hard code my product.

- The second functional requirement provided to me was that the project should have data that is stored in a file or database. For my project, I chose to use a file as I have previously used files in coding, whereas I have never really used a database before. Additionally, the code must be able to perform CRUD (Create, Read, Update and Delete) functions. My code performs all of them apart from the delete function.

- The third functional requirement was that the core entities in the application should be modelled as classes. One example of how I achieved this is my Basket class. The basket class is initialised at the start of the code, and stores a value of “0” for each item. This is until the value is incremented or decreased by the user later in the code. The Basket class is passed through all of the other classes, as it was a pivotal piece of my code.

- The fourth functional requirement was that the code should implement principles such as abstraction,inheritance and polymorphism when the are appropriate. I believe that my code implements abstraction as some variables and classes are private/final. For example, the user hashmap is protected for security reasons. My code implements inheritance and polymorphism through the Basket class, as it is passed through each of the individual classes.

- The fifth functional requirement was that the code should have features such as searching, reporting and notifications. My code has notifications that prompt the user when certain actions have been completed, or when errors have occurred.

- The sixth functional requirement was that the code should have sufficient documentation and comments for readability and maintenance. All of the classes within my project folder are commented to a high standard ensuring that anyone can

understand it. I ensured this by getting someone who does not understand code to any level to read the code and try to make basic sense of it which they could due to the comments added.

- The seventh functional requirement was that common utility functions could be kept in a separate util package/module for improved reuse and modularity.

- The eighth functional requirement was that exceptions should be handled properly to make the application resilient and user-friendly. I believe I achieved this by using exception handling methods such as try/catches. For example, if the users.dat file could not be found in my code the user would get an error that specified the file could not be located. Furthermore, if any other errors related to the file occurred the user would get a general error message that lets them know what the error is associated with.

- The ninth functional requirement was that input data validation should be done to reject invalid/incorrect data. I believe that my code achieves this, as certain mandatory fields cannot be left blank otherwise the user will get an error and the code will not proceed without it. However, optional fields are not required.

- The tenth functional requirement was that the code should follow standard Java coding guidelines and best practices for quality and maintenance. My code is simplified as much as possible, and is well commented for easy understanding. As well as that, any errors that could occur are dealt with with exception handling.

2.2 Cases and user stories covering key features

- I was able to receive feedback on my code by allowing two people, one who is knowledgeable in coding, and one who is not to try my application, and read the source code. The first person (my partner who is not knowledgeable in code), found that the application was easy to use, however she said that it was not visually appealing and commented that the placements of some of the buttons could be improved. With this feedback, I edited the code to her requests and she reused the code and was much happier with the experience. When she read the source code, she was able to understand it to a basic level and tell me roughly what some lines of code did. This was helpful in understanding if my comments detailed what the code was doing/what it meant. The second person, my father, who has been a developer for 40+ years tested my application and said that functionally it was adequate, however when he read my source code he demonstrated ways that the code could be improved to reduce repeated code. He also recommended changing some of the variable names as they were a little confusing. With this feedback, I changed variable names, and also changed some of the class names.

2.3 Non-functional requirements

- The first non functional requirement that I set for myself was that the code performed quickly and effectively. For example, I wanted frames to load instantly and I wanted the users.dat file to be updated as soon as a command was sent. This was achieved,

for example when a user registers a new account their new account is instantly made and is ready to be logged into.

- The second non functional requirement I set for myself was that certain variables and parts of code were either private, final or protected. This stops the code from being edited in any ways that may pose a security risk. In particular, code that is associated with the users.dat file.

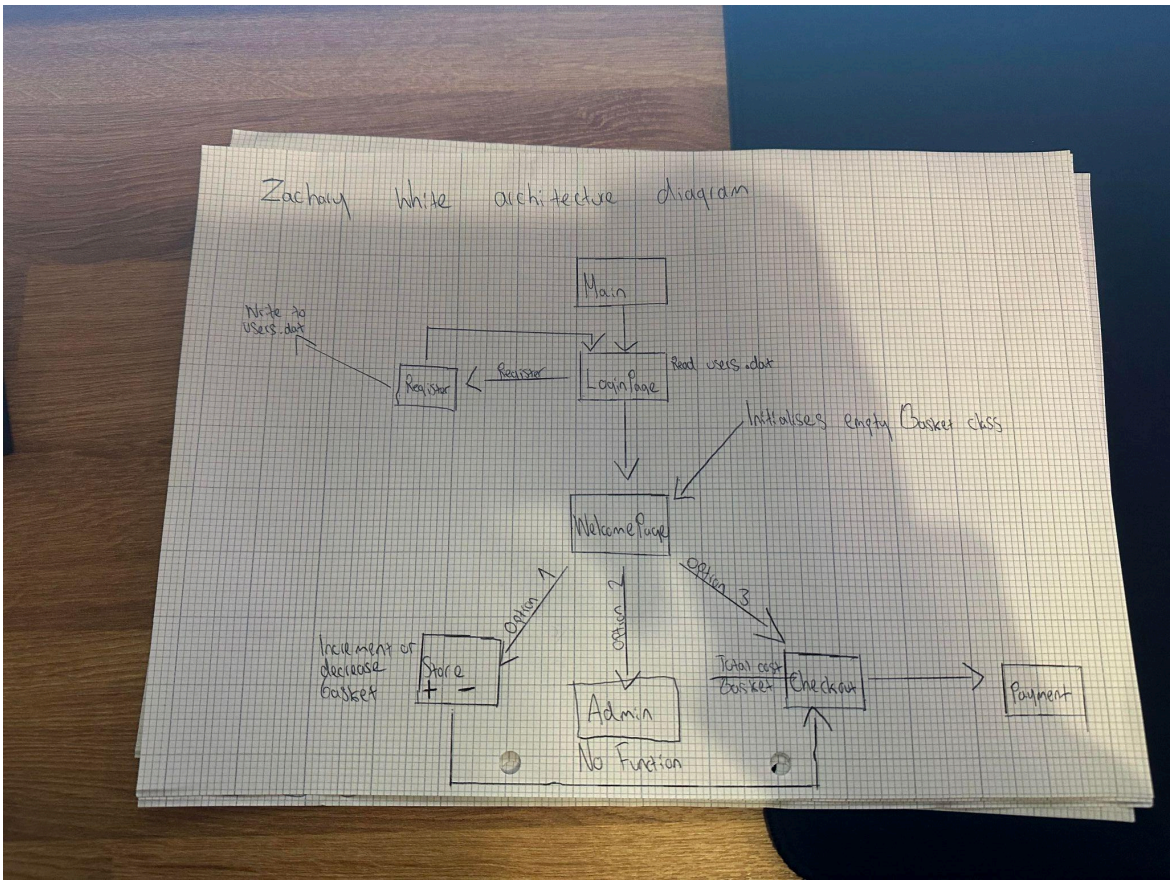
- The third non-functional requirement was that the GUI was easy for a user to use and understand. I believe this was achieved as I attempted to reduce clutter on the screen, and keep it as basic as possible to understand and use.

2.4 Prioritised list of requirements

- 1. Code functions properly
- 2. The GUI is simple to understand, and user experience is streamlined
- 3. Good coding standards/practices are maintained throughout the code
- 4. Any errors are dealt with and provide an error code that the user can understand
- 5. Code is well commented, and easy to follow

3.0 System architecture

3.1 High Level Architecture diagram



3.2 Description of key components and technologies used

- For this assignment I used Java Swing to create the GUIs. I chose to use Java Swing over JavaFX as I would much prefer to hardcode my GUI as it feels I have more control over the placement of elements. Also, I am somewhat familiar in Java Swing compared to JavaFX so if I was to choose to code with JavaFX I would have to learn it before starting the project.

- The most key component of my assignment is the Basket class. This is because the Basket class is virtually used in all other classes apart from the loginPage and registerPage. The Basket class allows the user to increase or decrease the amount of products they want within their basket.

3.3 Database scheme and descriptions

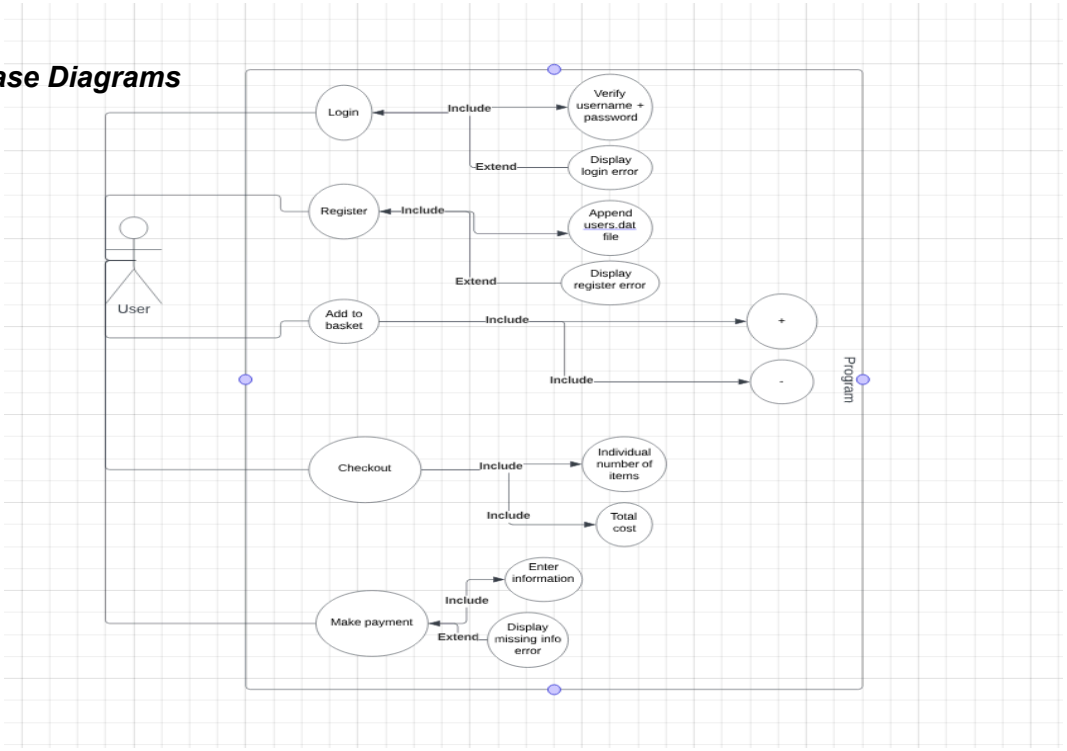
- N/A as I did not use a database

3.4 Identifying types of files that were handled + specifying operations used

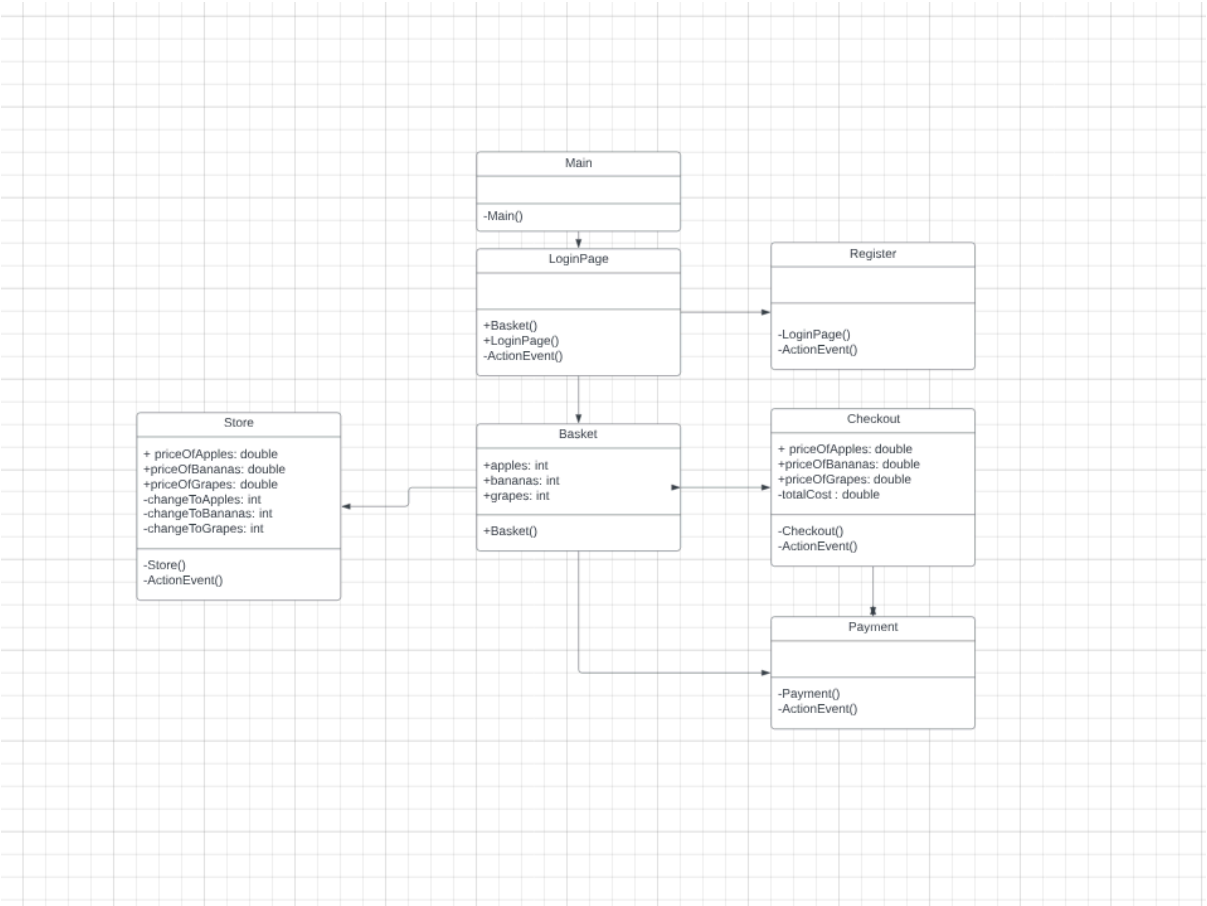
- The types of file I used for my project were .dat files. I chose to use .dat files for the following reasons:
    - Very flexible as they allow information of any kind, and are perfect for programmes to store data.
    - Easy to identify
    - Automatically created
- The operations that my program could perform on the .dat files were:
    - Read - Reading the information inside the file allows the program to accept usernames and passwords that are already inside the file
    - Write - Writing new usernames and passwords that the user has registered to the file
    - Create - My program creates a unique invoice/receipt for each completed order creating a new .dat file

4.0 UML Diagrams

4.1 Use Case Diagrams



4.2 Class Diagrams



4.3 Sequence diagrams for critical user workflows

5.0 Implementation

5.1 Source code of critical components

I believe that the two most important components of my project are the welcomePage class and the Basket class. This is because these two classes are continually used throughout the project.

welcomePage source code:

```
//Java packages that are used throughout the code
import javax.swing.*;
```

```java
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.beans.PropertyChangeListener;

public class WelcomePage implements ActionListener {


    JFrame welcomeFrame = new JFrame(); //Creating a new welcomeFrame

    JLabel title = new JLabel("Shop");

    //Creating the buttons
    JButton catalog = new JButton("Catalog");
    JButton basketButton = new JButton("Basket");
    JButton checkout = new JButton("Checkout");
    JButton admin = new JButton("Admin");

    Store store=null;

    int totalApples=0;
    int totalBananas=0;
    int totalGrapes=0;

    Basket basket = new Basket();
    public WelcomePage(Basket basket){

        this.basket= basket;
        //Setting the location and the action listener for the buttons
        catalog.setBounds(415,200,125,50);
        catalog.addActionListener(this);
        catalog.setFocusable(false);

        basketButton.setBounds(415,300,125,50);
        basketButton.addActionListener(this);
        basketButton.setFocusable(false);

        checkout.setBounds(415,400,125,50);
        checkout.addActionListener(this);
        checkout.setFocusable(false);

        admin.setBounds(415,500,125,50);
        admin.addActionListener(this);
        admin.setFocusable(false);

        title.setFont(new Font("Serif", Font.BOLD,50));
        title.setBounds(425,50,200,100);

        checkout.setBackground(new Color(59, 89, 182));
        checkout.setForeground(Color.WHITE);
        checkout.setFocusPainted(false);
        checkout.setFont(new Font("Tahoma", Font.BOLD, 12));
```

```
        basketButton.setBackground(new Color(59, 89, 182));
        basketButton.setForeground(Color.WHITE);
        basketButton.setFocusPainted(false);
        basketButton.setFont(new Font("Tahoma", Font.BOLD, 12));

        catalog.setBackground(new Color(59, 89, 182));
        catalog.setForeground(Color.WHITE);
        catalog.setFocusPainted(false);
        catalog.setFont(new Font("Tahoma", Font.BOLD, 12));

        admin.setBackground(new Color(59, 89, 182));
        admin.setForeground(Color.WHITE);
        admin.setFocusPainted(false);
        admin.setFont(new Font("Tahoma", Font.BOLD, 12));



        //Adding the elements to the welcomeFrame + adding default behaviour
changes
        welcomeFrame.add(catalog);
        welcomeFrame.add(basketButton);
        welcomeFrame.add(checkout);
        welcomeFrame.add(admin);
        welcomeFrame.add(title);
        welcomeFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        welcomeFrame.setSize(1000,800);
        welcomeFrame.setLayout(null);
        welcomeFrame.setLocationRelativeTo(null);
        welcomeFrame.setVisible(true);
        welcomeFrame.setResizable(false);
        welcomeFrame.setTitle("Home");


        ImageIcon home = new ImageIcon("home.png");
        welcomeFrame.setIconImage(home.getImage());



    }
//The events that are performed when a button is pressed
    @Override
    public void actionPerformed(ActionEvent e) {

        if(e.getSource()==catalog) {
            welcomeFrame.dispose(); //The current open frame closes when the
new one opens
            store = new Store(basket); //Opens the new frame
        }
```

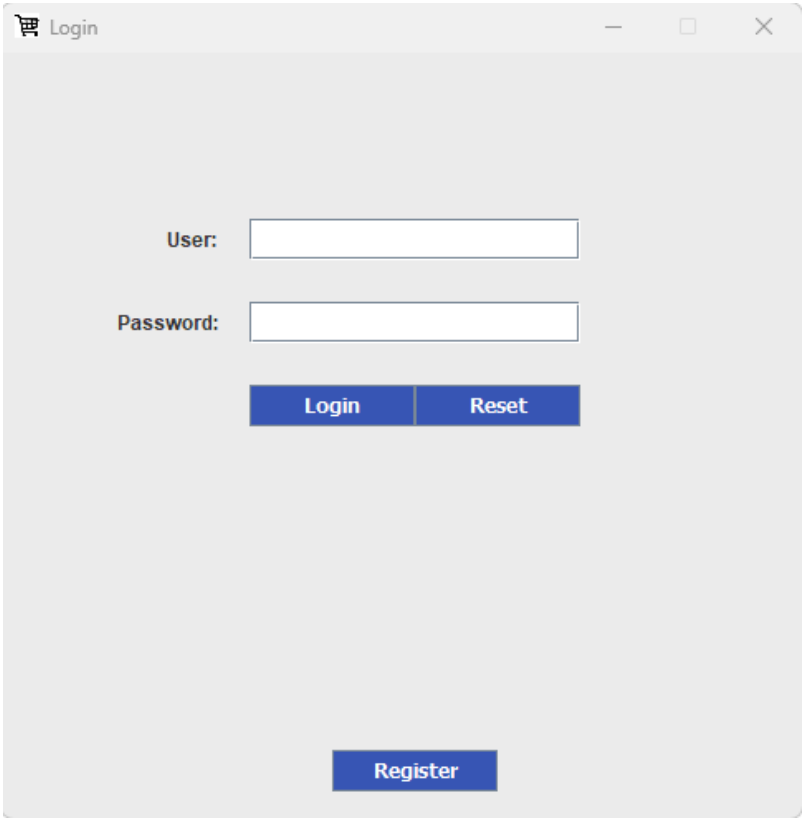
```
        if(e.getSource()==checkout) {
            welcomeFrame.dispose();
            Checkout checkout1 = new Checkout(basket);
        }

        if(e.getSource()==admin) {
            welcomeFrame.dispose();
            Admin admin1 = new Admin();
        }

    }
}
```

Basket Class sourcecode:

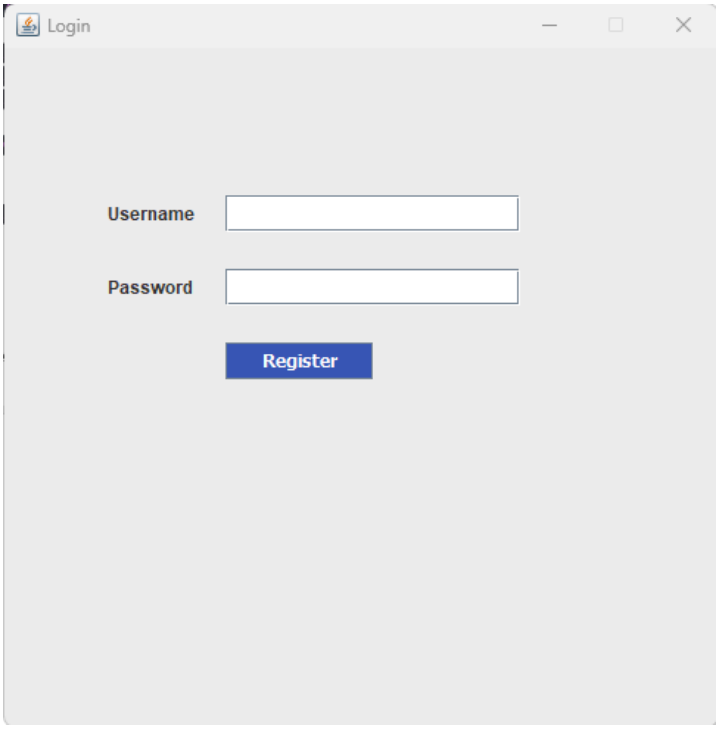
```
//Java packages that are used throughout the code
import javax.swing.*;
import java.util.EventListener;

public class Basket {

    public int apples = 0;
    public int bananas = 0;
    public int grapes = 0;

    double cost = 0;


    public Basket() {


    }
}
```

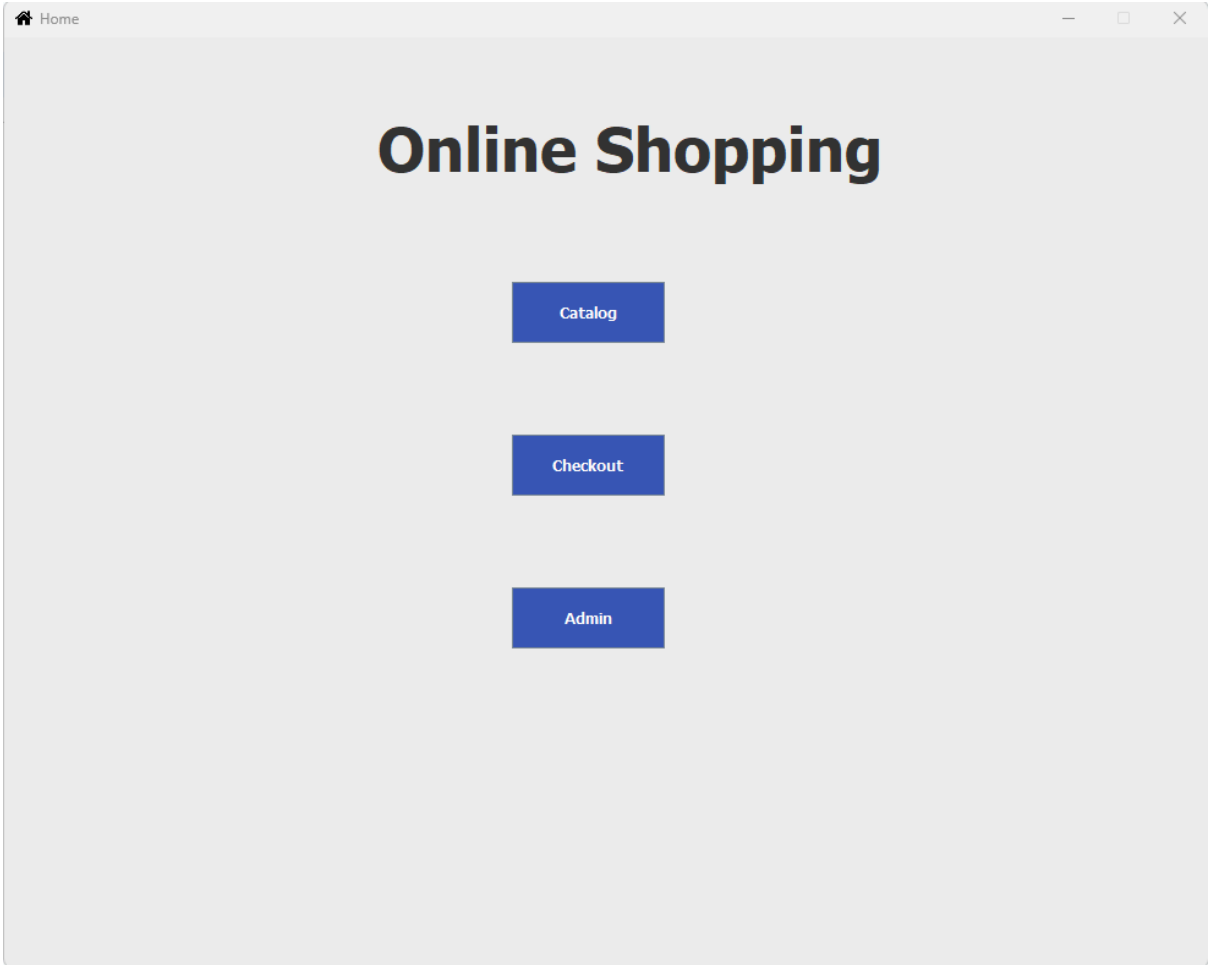
5.2 Screenshots of user interface

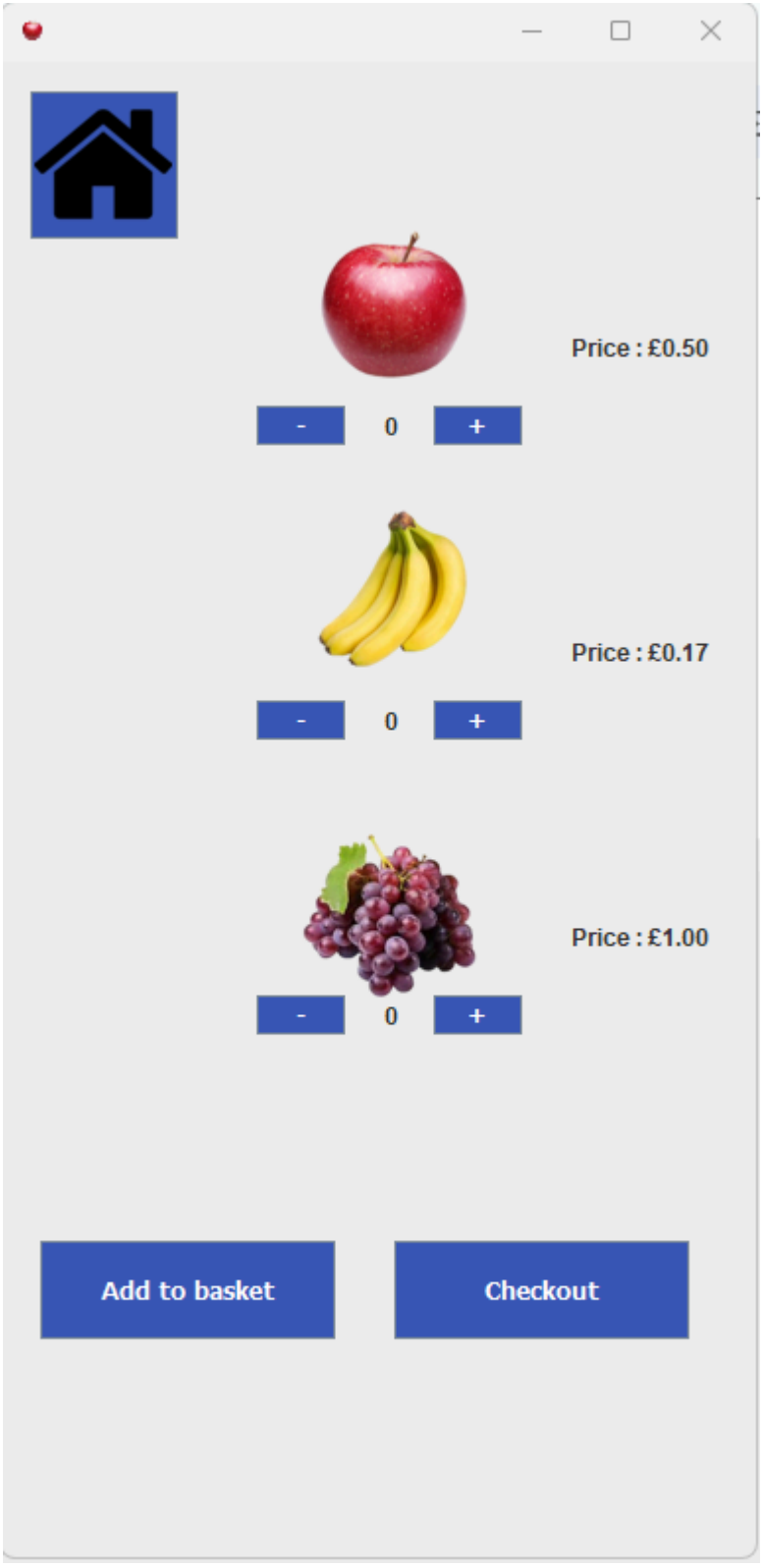
Login page:

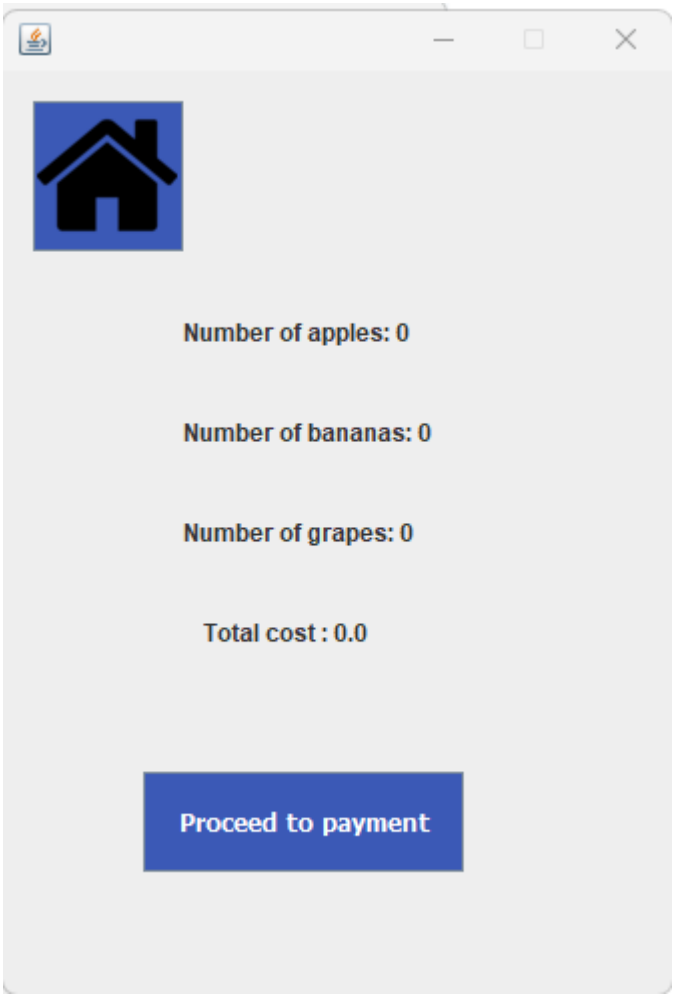


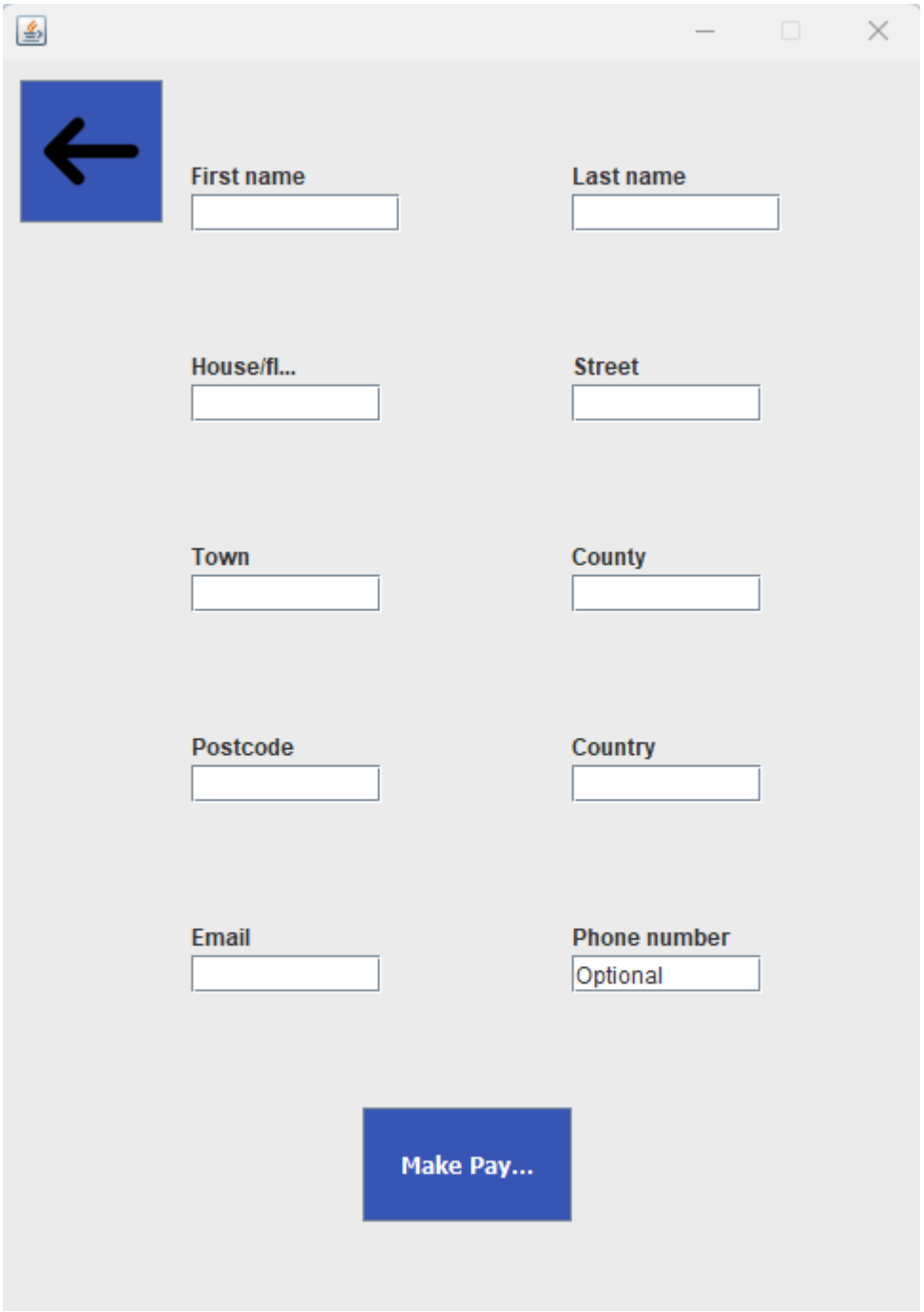
Register page:






Price : £0.50
-
0
+
Price : £0.17
-
0
+
Price : £1.00
-
0
+
Add to basket
Checkout


Number of apples: 0
Number of bananas: 0
Number of grapes: 0
Total cost : 0.0
Proceed to payment



5.3 Sample data for testing

5.4 Installation and deployment instructions

Step 1: Install zip file

Step 2: Unpack zip file

Step 3: Run main file

BONUS STEPS

1.To test errors change users.dat file name

5.5 Conclusion

- After completing the assignment I am moderately happy with my work. I believe that the code functions to an acceptable standard and I am happy with the Java skills that I have learnt through completing the assignment. I have enjoyed the progress, and the learning curve that the language presents. The code completes all required functions that were asked.

- However, I am unhappy with the way that the GUI is styled as I believe it is bland, and not very interesting to look at.

5.6 Summary of deliverables

5.7 Discussion of potential enhancements and future work

- If I were to do this project again I would start by changing the visual appearance of the GUI. I believe that with the help of others I could make the GUI more visually appealing and stand out more. Furthermore, I would also alter the way that the items are implemented. This is because, as of right now, if I were to add more items to the Store class it would require a lot of changes to the code of each individual class. Additionally, I would add features to the admin page, such as being able to delete users or delete invoices etc…