Energy Billing System

Module Title: Visual Object Software
Module Code: CSYM025
Tutor Name: Suraj Ajit
Assignment Title: Energy Billing System
Student Name: Parastoo Shojaei Sarcheshmeh
Student ID: 25828319

Contents

1 Login Details (Username and Password)

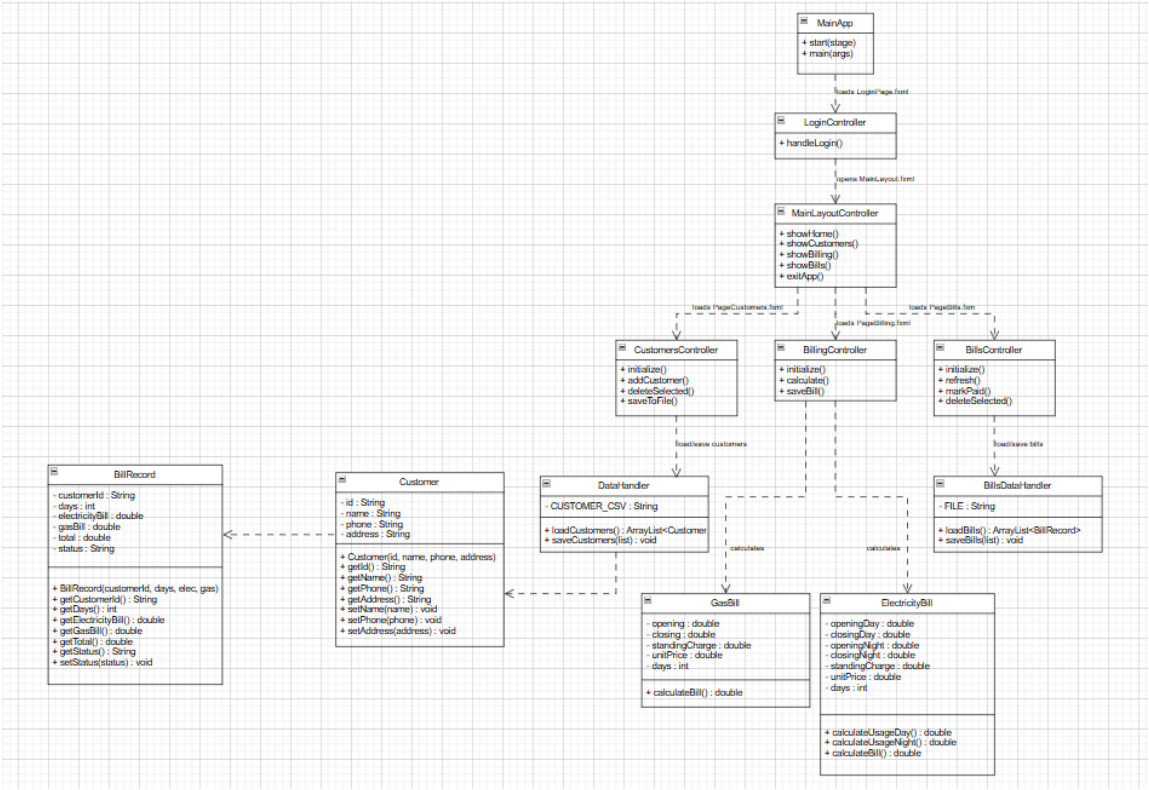
The application starts with a login page.
The login is only used to control access to the system before the main menu is opened.
Login credentials used in this system:

- Username: admin
- Password: 1234

After a successful login, the system opens the main layout with the menu and all pages.
If the login details are incorrect, the system shows an error message.
The full implementation and detailed explanation of the login code are provided in Document 2 – Full Source Code Listing.

2 UML Class Diagram

The diagram shows the main structure of the Energy Billing System. The design is based on separating responsibilities into different classes. The model classes (Customer and BillRecord) represent the main data of the system. The calculation classes (ElectricityBill and GasBill) contain the billing formulas, while help classes (DataHandler and BillsDataHandler) manage CSV files. The controllers connect the JavaFX user interface to the logic and data. Full code and detailed explanations for all classes are provided in Document 2 – Full Source Code Listing.



3 User Stories

This section describes the main user stories that were implemented in the Energy Billing System.

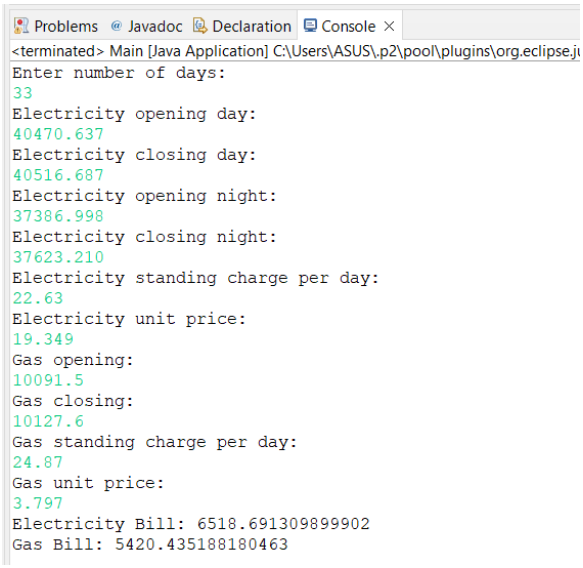
User Story	Implemented	Comments
I want to log in to the system so that I can access the application.	Full	Simple login using username and password.
I want to see a home page after login so that I know the system is ready.	Full	Home page shows module and information.
I want to add new customers so that I can create bills for them.	Full	Customers added.
I want to view all customers in a table so that I can manage customer data easily.	Full	Customer data is stored in a CSV file
I want to delete customers so that I can remove incorrect data	Full	Delete Customers.
I want to calculate electricity and gas bills so that I can see the total cost.	Full	Calculations follow billing rules.
I want to save calculated bills so that I can access them later.	Full	Bills are stored in a CSV file.
I want to view all saved bills so that I can manage billing records.	Full	Bills shown in a Table View.
I want to mark bills as paid so that I can track payment status.	Full	Status updated and saved.
I want to delete bills so that I can remove old records.	Full	CSV file updated after deletion.
I want the system to save data to files so that information is not lost.	Full	File-based storage used.
I want to move between pages using a menu so that the system is easy to use.	Full	Handled by main layout controller.

4 Design and Implementation Process

This section explains how the system was designed and implemented step by step.

4.1 Phase 1 – Console Application

The project started as a simple console application. In this phase, the goal was to make sure that the electricity and gas billing calculations were correct.
The program asked the user to enter meter readings, prices, and the number of days, and then printed the electricity bill and gas bill in the console.
At this stage, there was no graphical interface.
The full console code and explanation are provided in Document 2 – Full Source Code Listing.



4.2 Phase 2 – JavaFX GUI Development

After the console version was working correctly, the project was converted into a JavaFX application.
In this phase, I designed the user interface using Scene Builder. Each page of the system was first created visually in Scene Builder by placing buttons, labels, text fields, tables, and layouts.
After designing the layout, Scene Builder automatically generated the FXML code for each page.
The controllers were used to connect the interface to the program logic. Each controller is responsible for one page only.
The main layout uses a menu to switch between pages, so the user stays in one window and the content changes dynamically.
The full JavaFX code and FXML explanations are provided in Document 2 – Full Source Code Listing.

4.3 Phase 3 – File Storage and Data Handling

In the next phase, file-based storage was added to the system. Instead of using a database, I used CSV files.
Customer data is stored in customers.csv, and bill data is stored in bills.csv.
Special classes were created to handle file reading and writing.
When the application starts, data is loaded from the files, and when the user saves or updates data, the files are updated automatically.

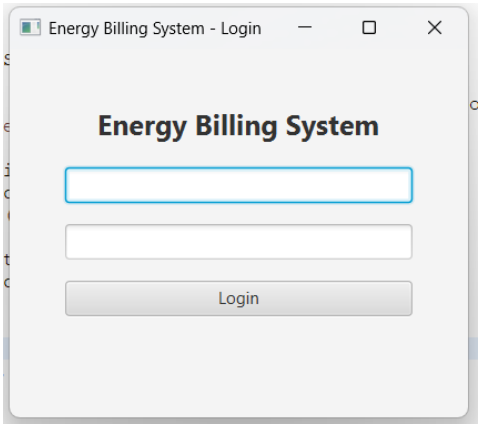
4.4 Phase 4 – Final Integration and Object-Oriented Design

In the final, all parts of the system were connected together. The controllers, model classes, calculation classes, and file handlers were integrated to create one complete application.
Each class has a clear role: model classes store data, calculation classes perform billing logic, controllers manage user interaction, and data handler classes manage file storage.

5 Screenshots of the System

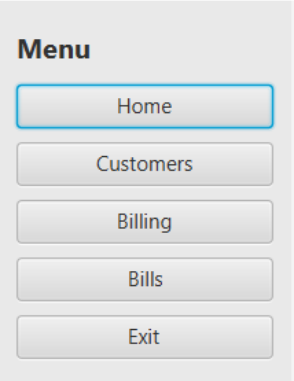
This section shows screenshots of the main features of the System.

5.1 Login Page



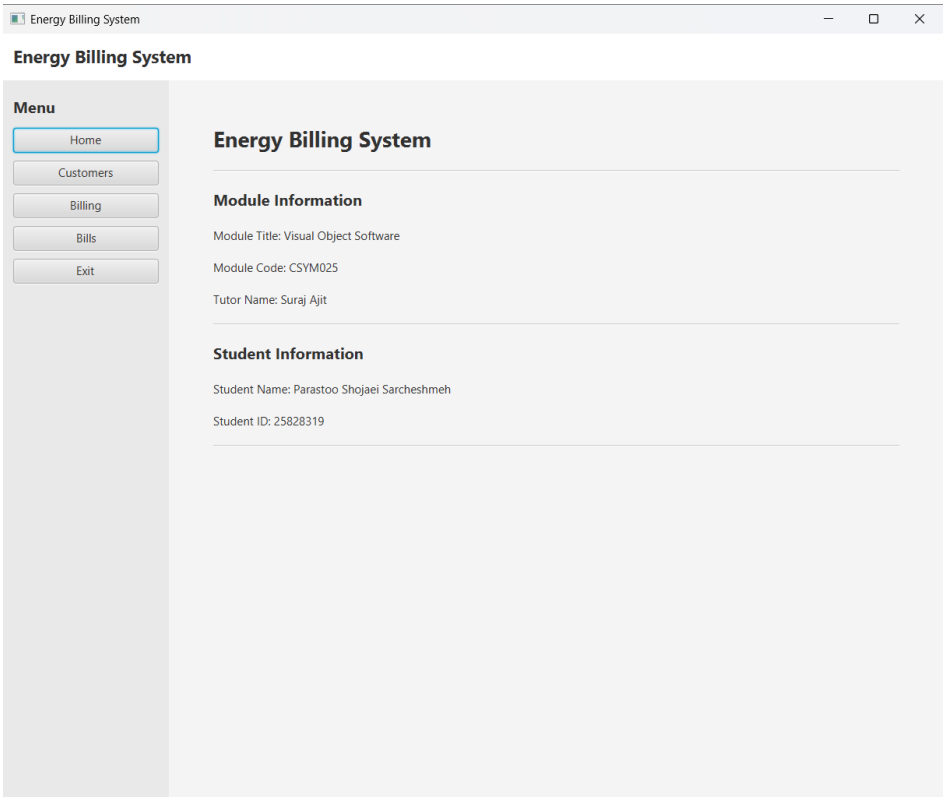
This shows the login page of the system. The user must enter a username and password to access the application. If the login details are correct, the system opens the main menu.
If the details are wrong, an error message is shown.

5.2 Main Layout and Menu



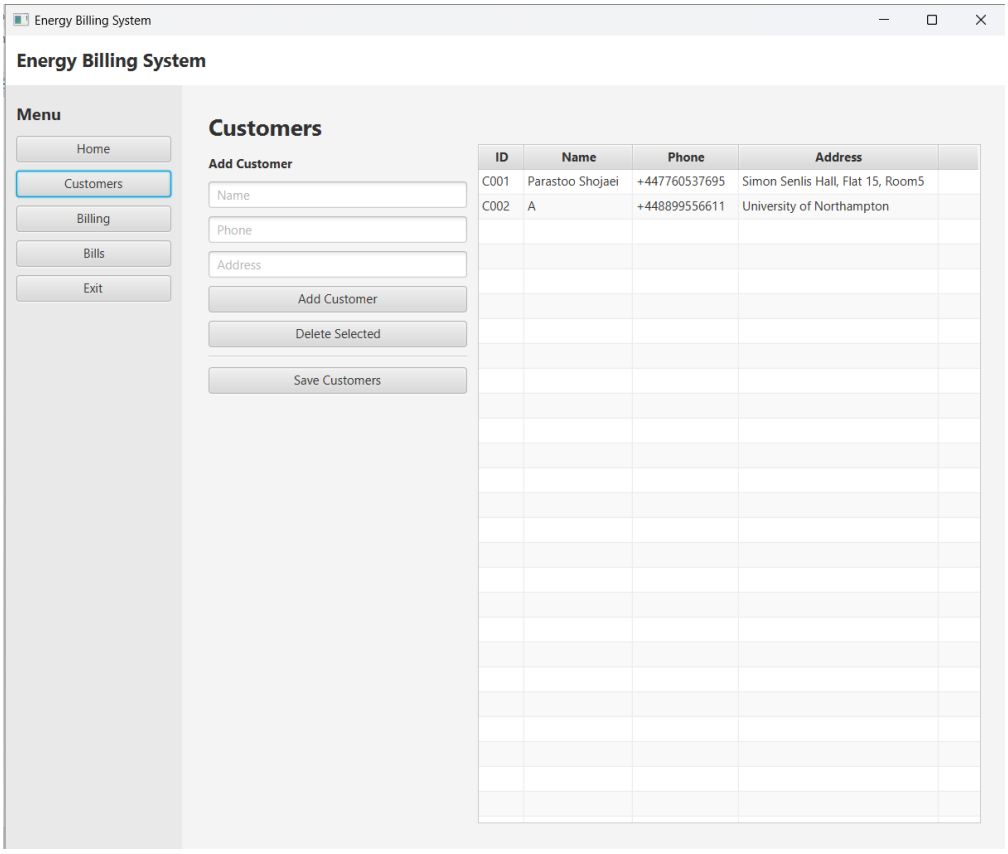
This shows the main layout of the system after login. The menu is displayed on the left side of the window and allows the user to navigate between pages.

5.3 Home Page



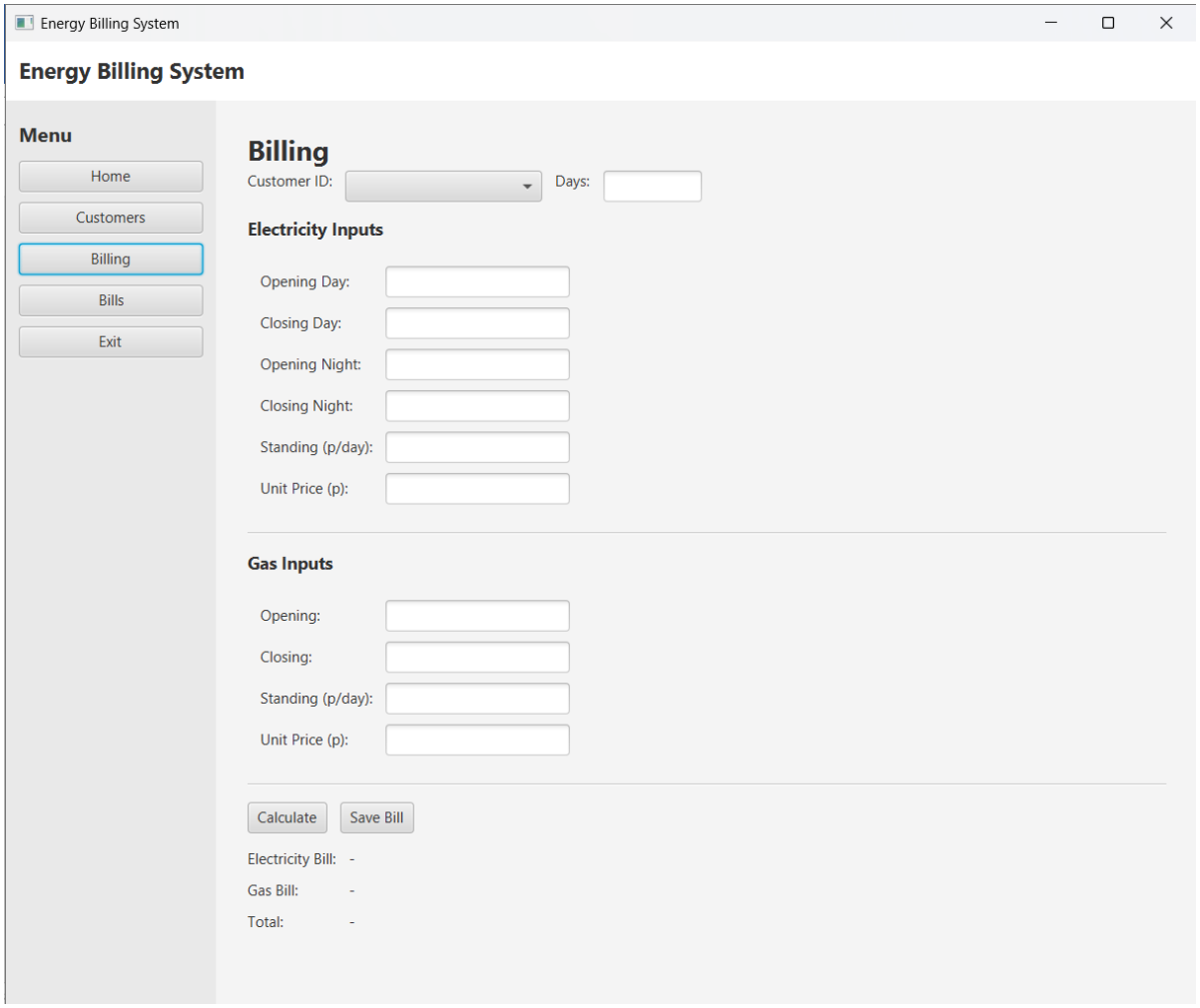
The Home page is the first page shown after login.

5.4 Customers Page



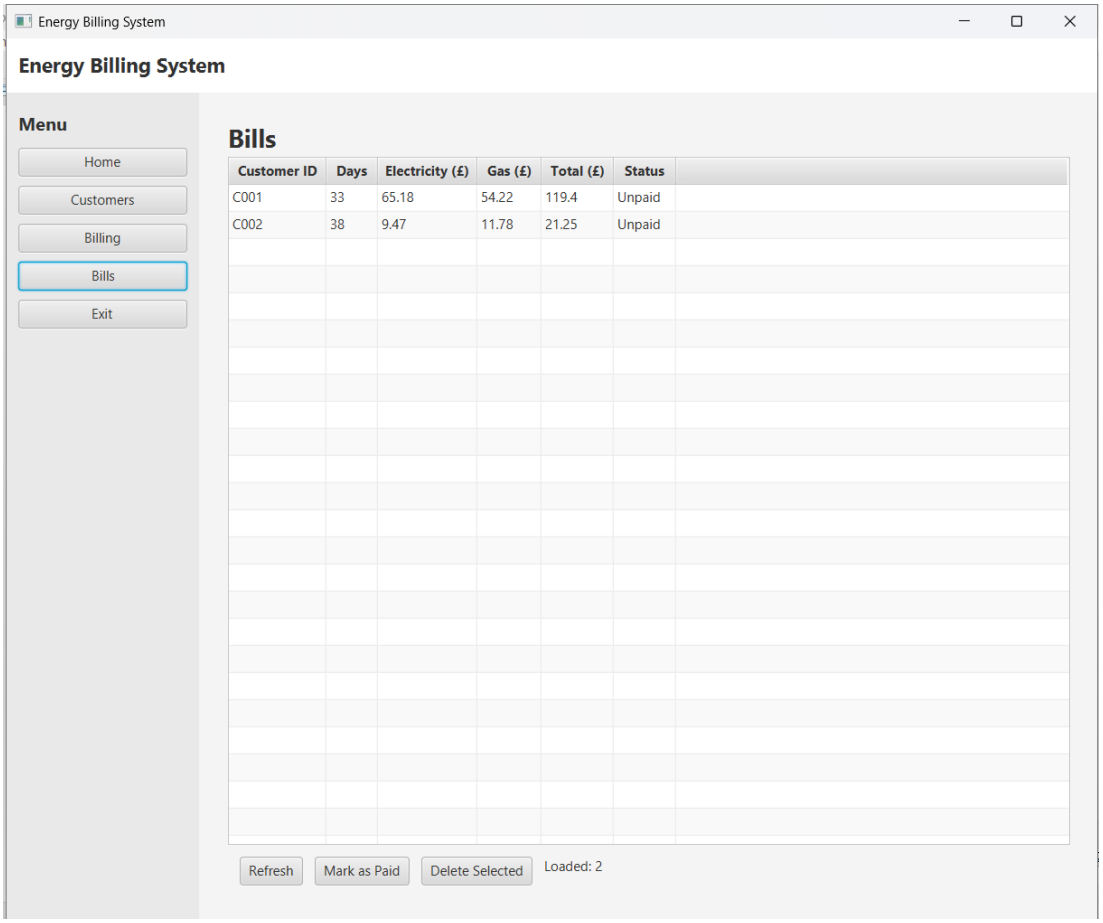
This shows the Customers page where the user can add new customers by entering name, phone, and address. All customers are displayed in a Table View. The user can also delete customers and save the data to a CSV file.

5.5 Billing Page



This shows the Billing page where the user can calculate electricity and gas bills.
The user selects a customer, enters meter readings, prices, and number of days, and then clicks the calculate button. The system displays the electricity bill, gas bill, and total amount in pounds.

5.6 Bills Page



This screenshot shows the Bills page where all saved bills are displayed in a table.
The user can refresh the list, mark bills as paid, or delete selected bills.

6 Evidence of Testing

First, I did blackbox testing by using the system like a user and checking the output on the GUI. Second, I did whitebox testing using JUnit to test the main calculation classes with sample values.

6.1 Blackbox Testing

Blackbox testing checks the system from the user's point of view. In this testing, I entered different inputs in the GUI and checked if the system behaved correctly.

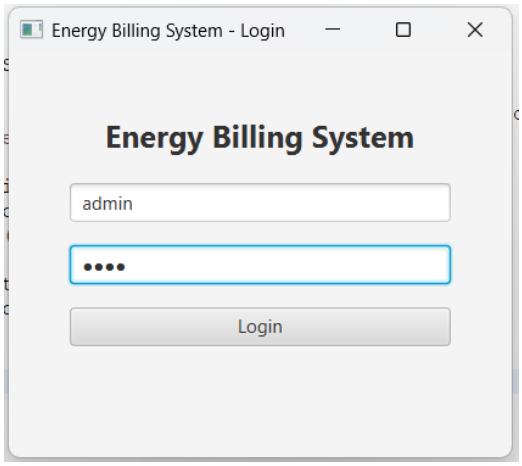
Feature Tested	Test Steps	Result	Pass/Fail
Login (correct)	Enter admin / 1234 and click Login	Main menu opens	Pass
Login (wrong)	Enter wrong details and click Login	Error message shown	Pass
Navigation	Click Home / Customers / Billing / Bills	Correct page loads	Pass
Add Customer	Fill all fields, click Add	Customer appears in table	Pass
Add Customer (missing)	Leave one field empty, click Add	Warning shown	Pass
Delete Customer	Select a customer and delete	Customer removed	Pass
Save Customers	Click Save	customers.csv updated	Pass
Billing Calculate (valid)	Enter valid numeric inputs, click Calculate	Bills shown	Pass
Billing Calculate (invalid)	Enter non-numeric input, click Calculate	"Invalid input"	Pass
Save Bill (no calculation)	Select customer but do not calculate	"Calculate first"	Pass
Save Bill (valid)	Select customer + calculate + save	Bill saved to file	Pass
Bills Refresh	Open Bills page and click Refresh	Table reloads	Pass
Mark Paid (no selection)	Click Mark Paid without selecting	Message shown	Pass
Mark Paid (selected)	Select bill row and click Mark Paid	Status becomes Paid	Pass
Delete Bill (selected)	Select bill row and delete	Removed + file updated	Pass

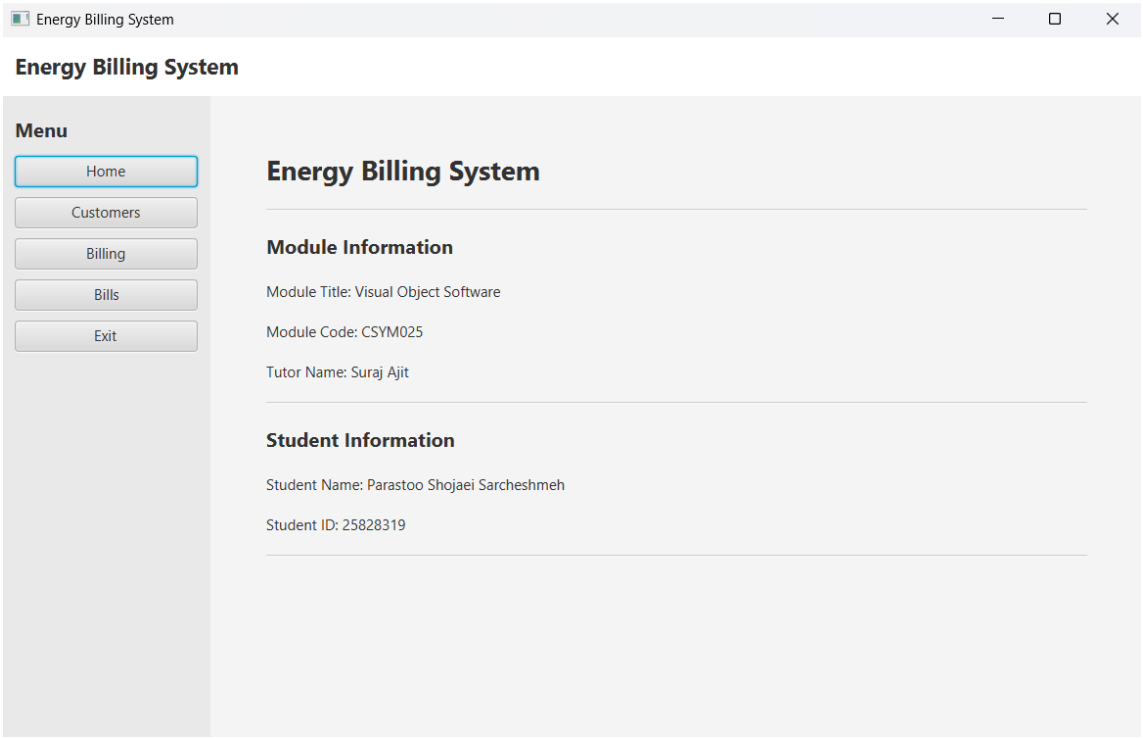
6.1.1 Blackbox Testing Screenshots

The following screenshots show evidence of the Blackbox tests listed in the table.

Login Success

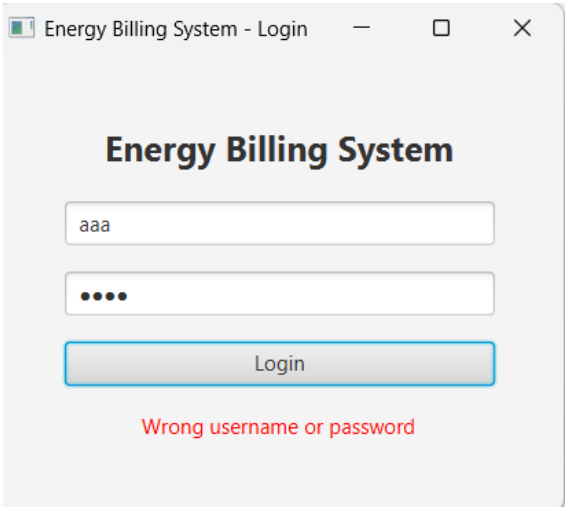
This screenshot shows a successful login after entering the correct username and password.





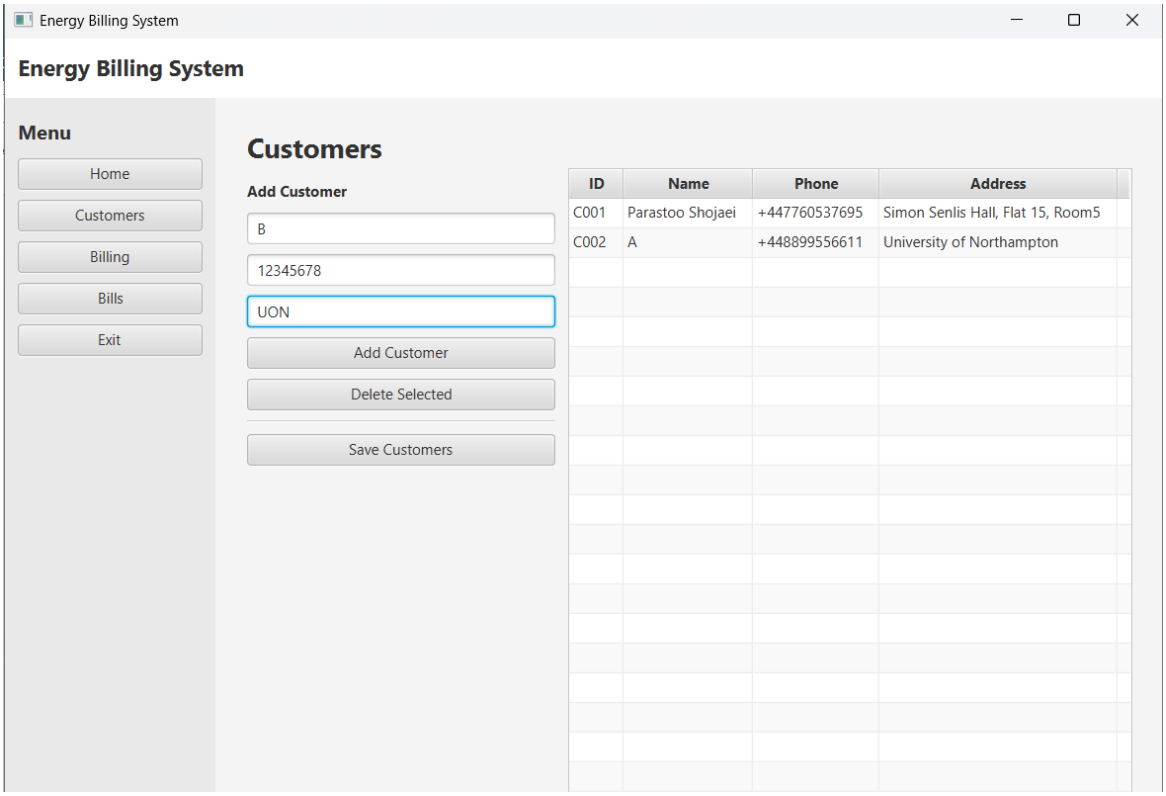
Login Failure Message

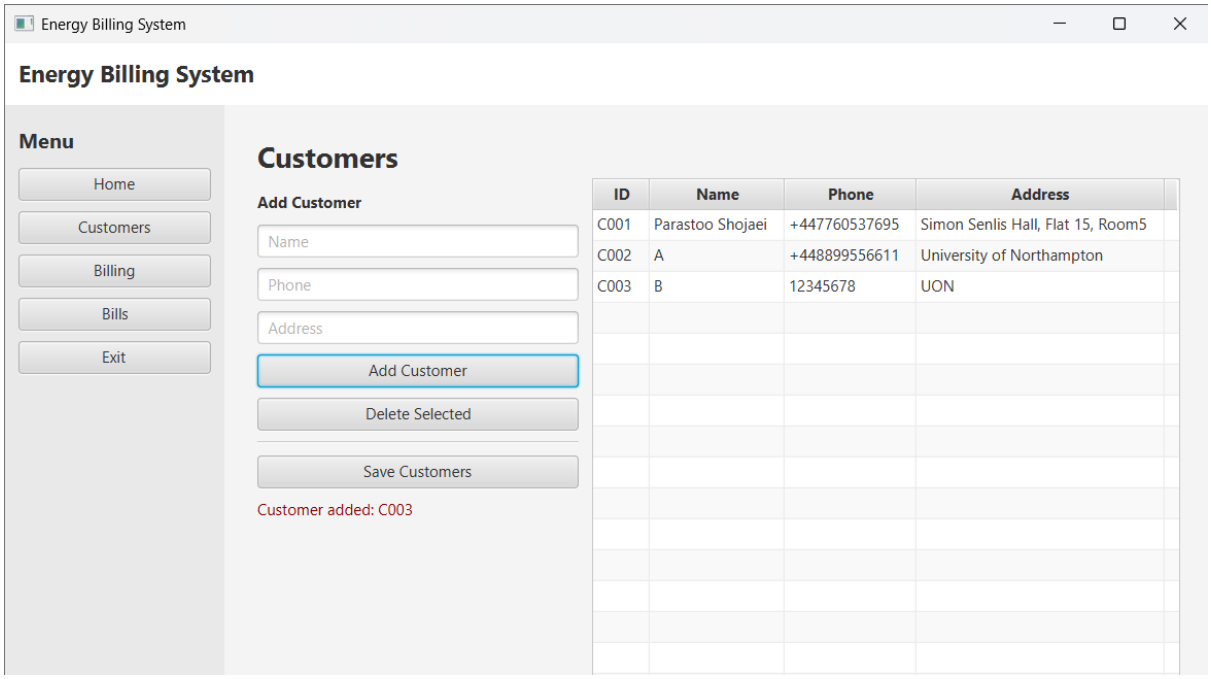
This shows the error message when incorrect login details are entered.



Add Customer

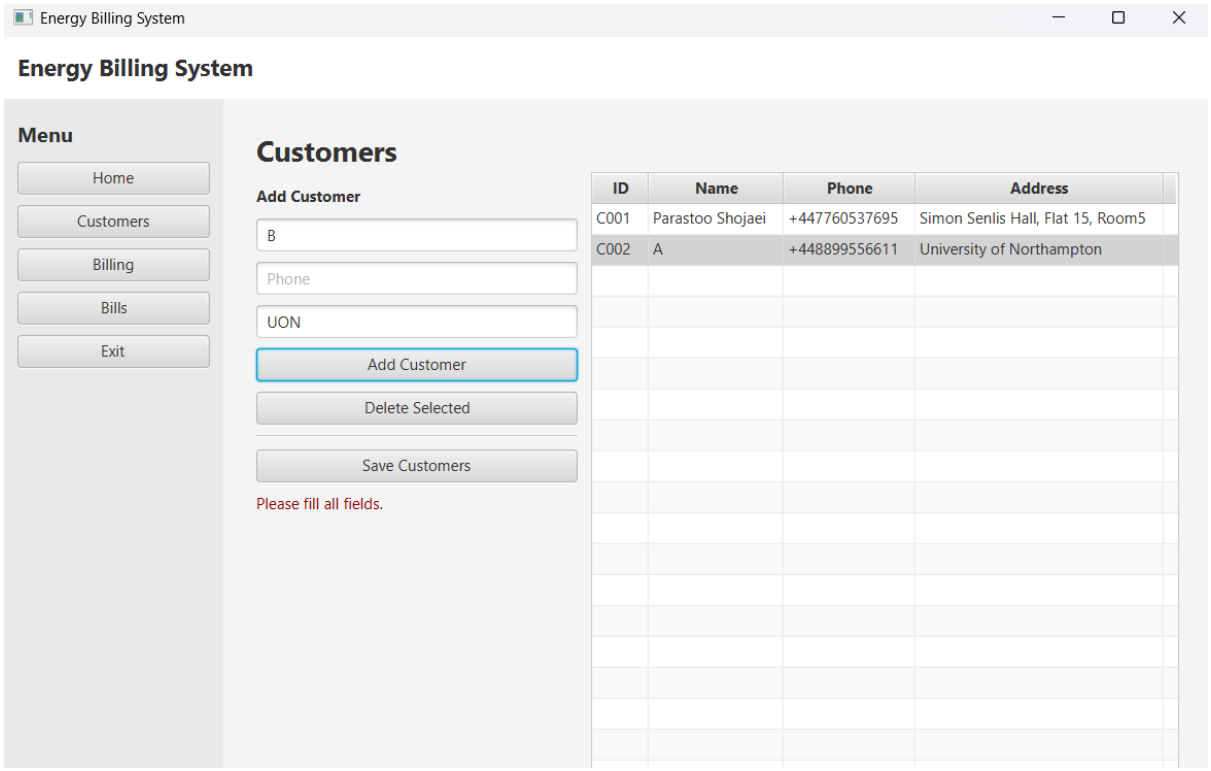
This shows a new customer added to the table and the confirmation message displayed by the system.





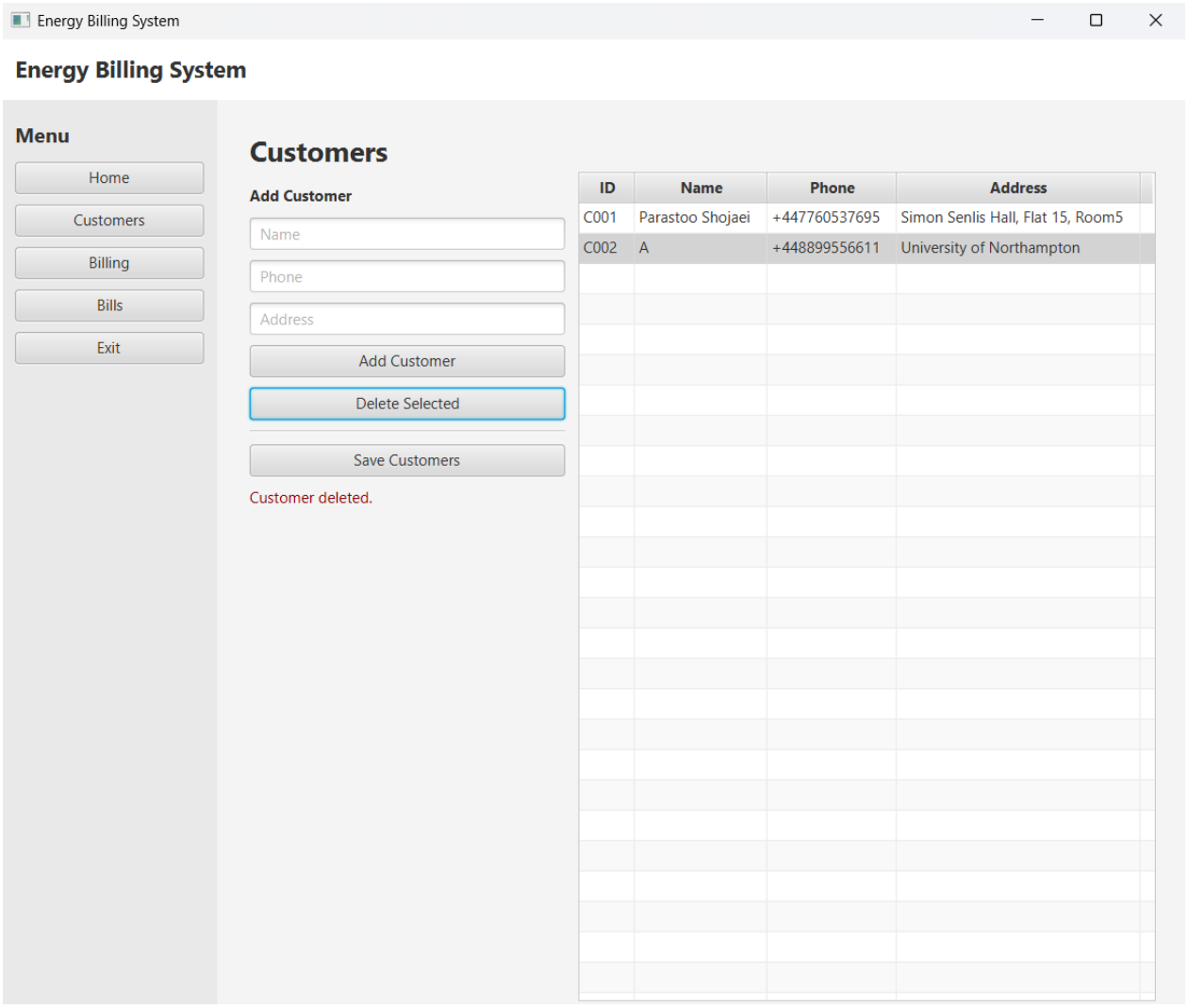
Missing Fields Validation

This shows the warning message when add a customer without filling all required fields.



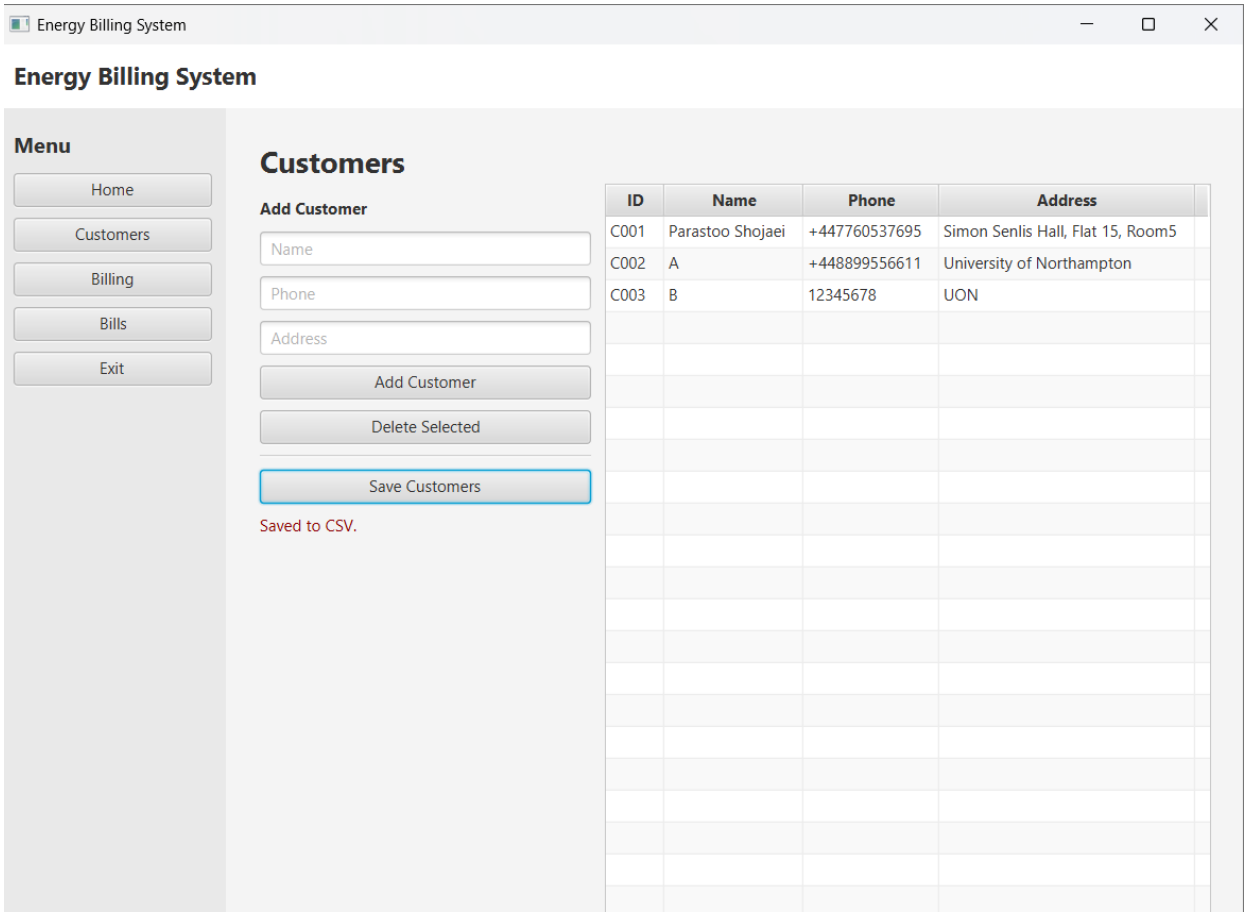
Delete Customer

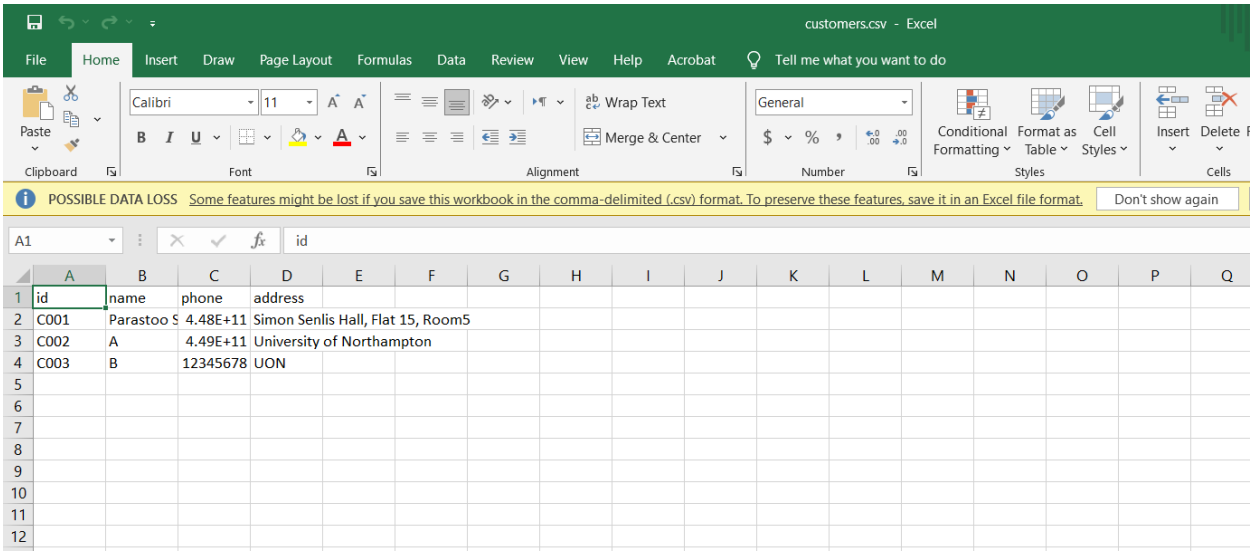
This shows the customer table after a selected customer has been deleted.



Save Customers to CSV

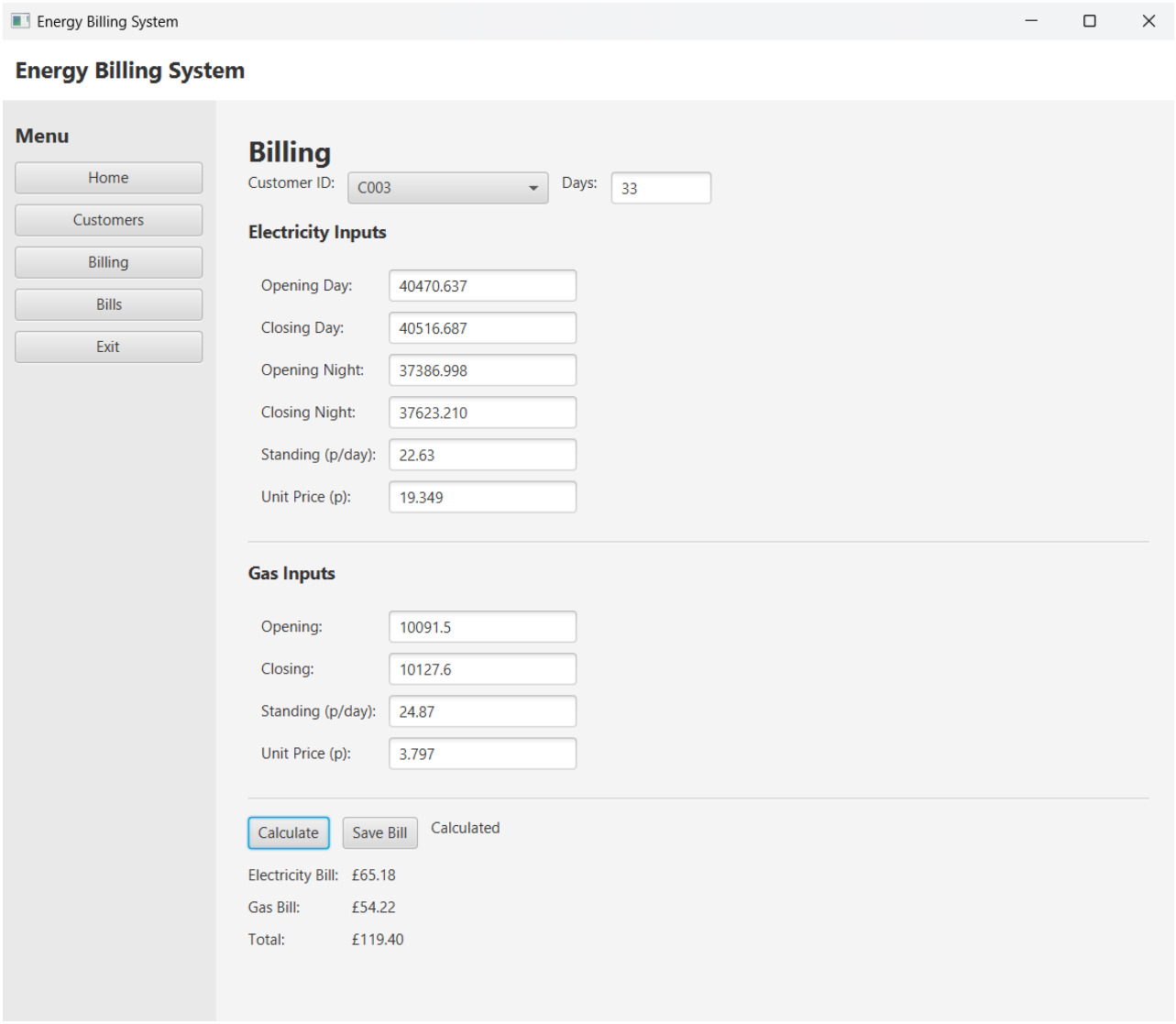
This shows the confirmation message after customer data is saved to the CSV file.





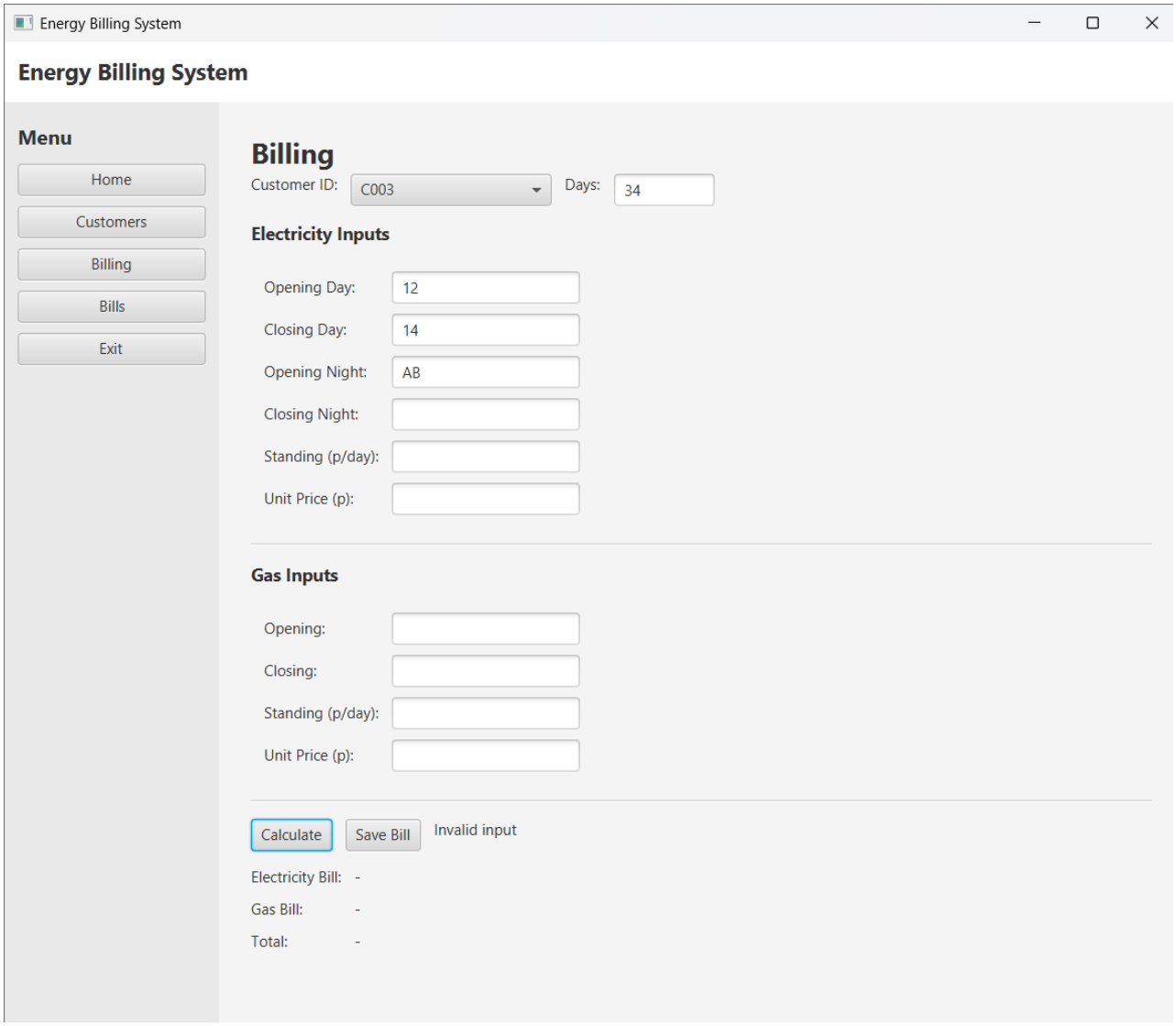
Bill Calculation Result

This shows the electricity bill, gas bill, and total amount after calculation.



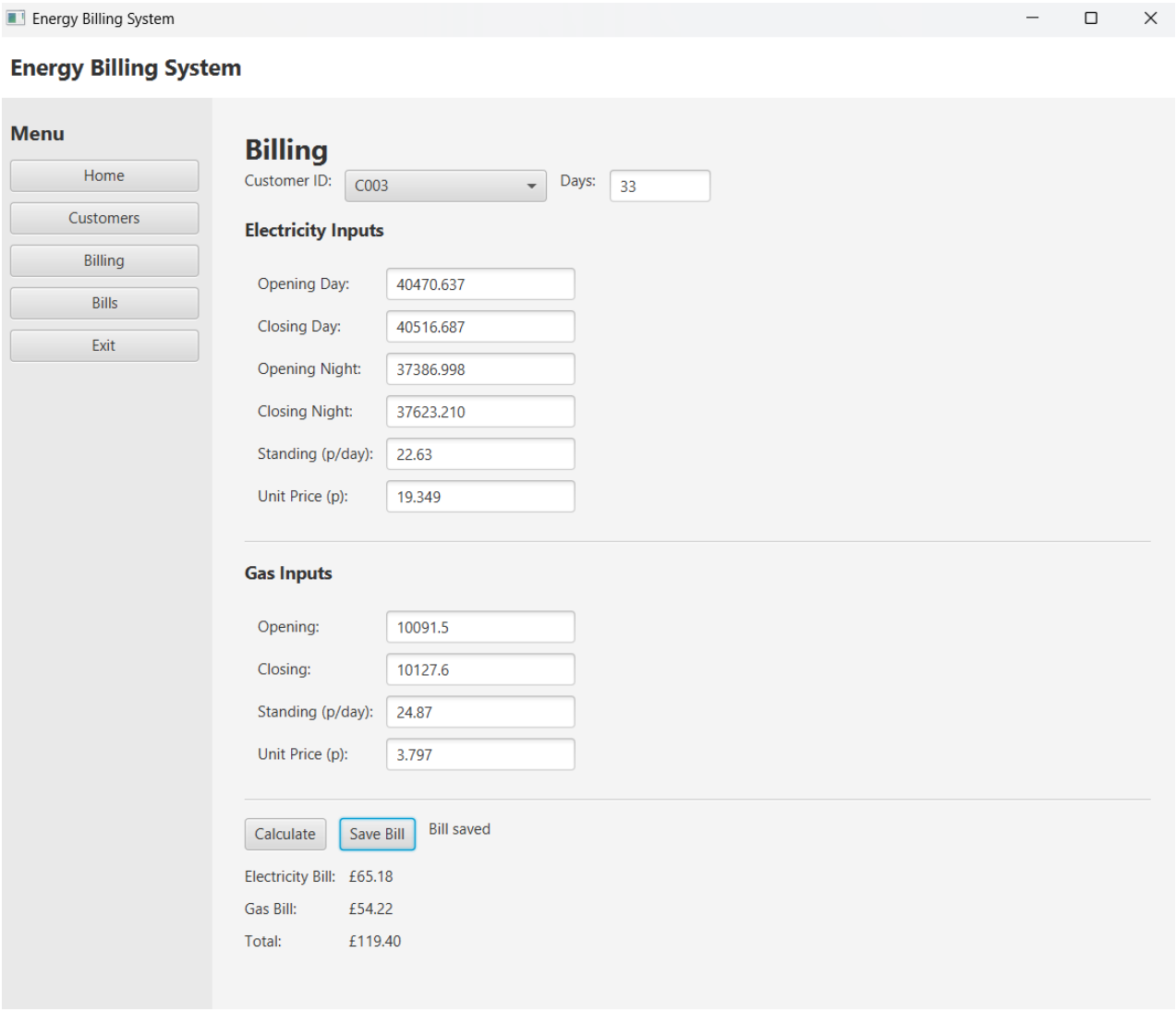
Invalid Input Handling

This shows the error message when invalid (non-numeric) input is entered in billing fields.



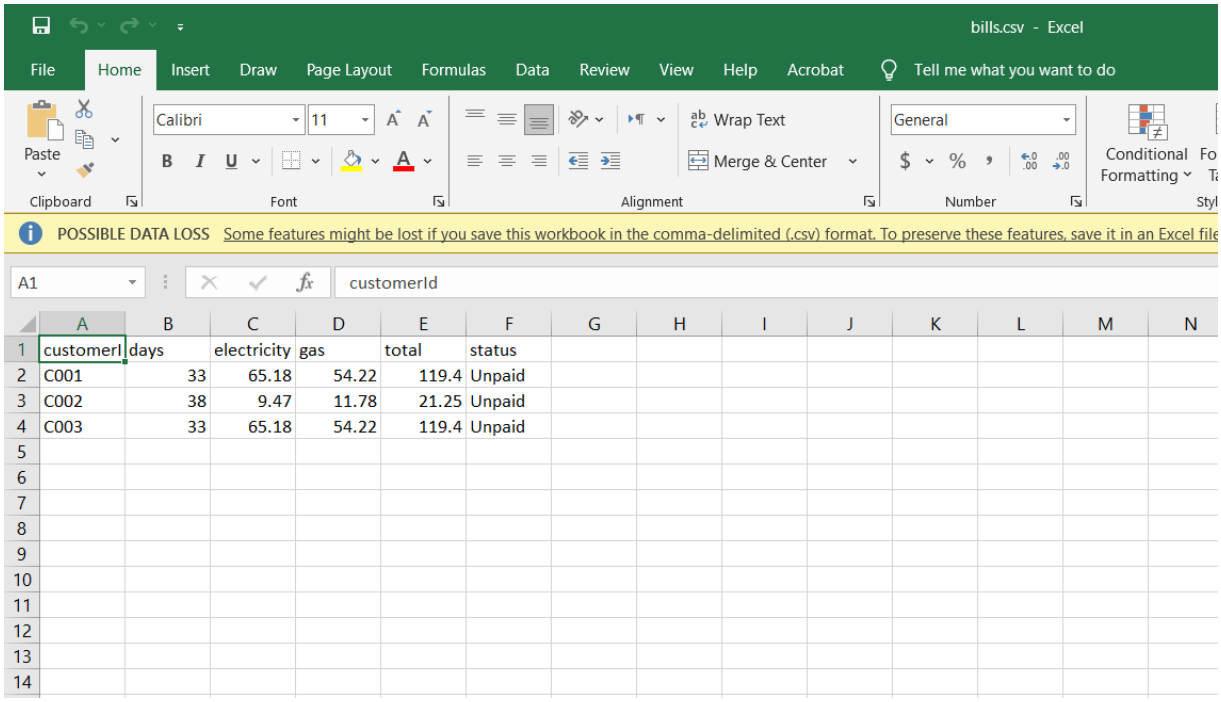
Bill Saved

This shows the confirmation message after a bill is saved successfully.



Bills Page Loaded

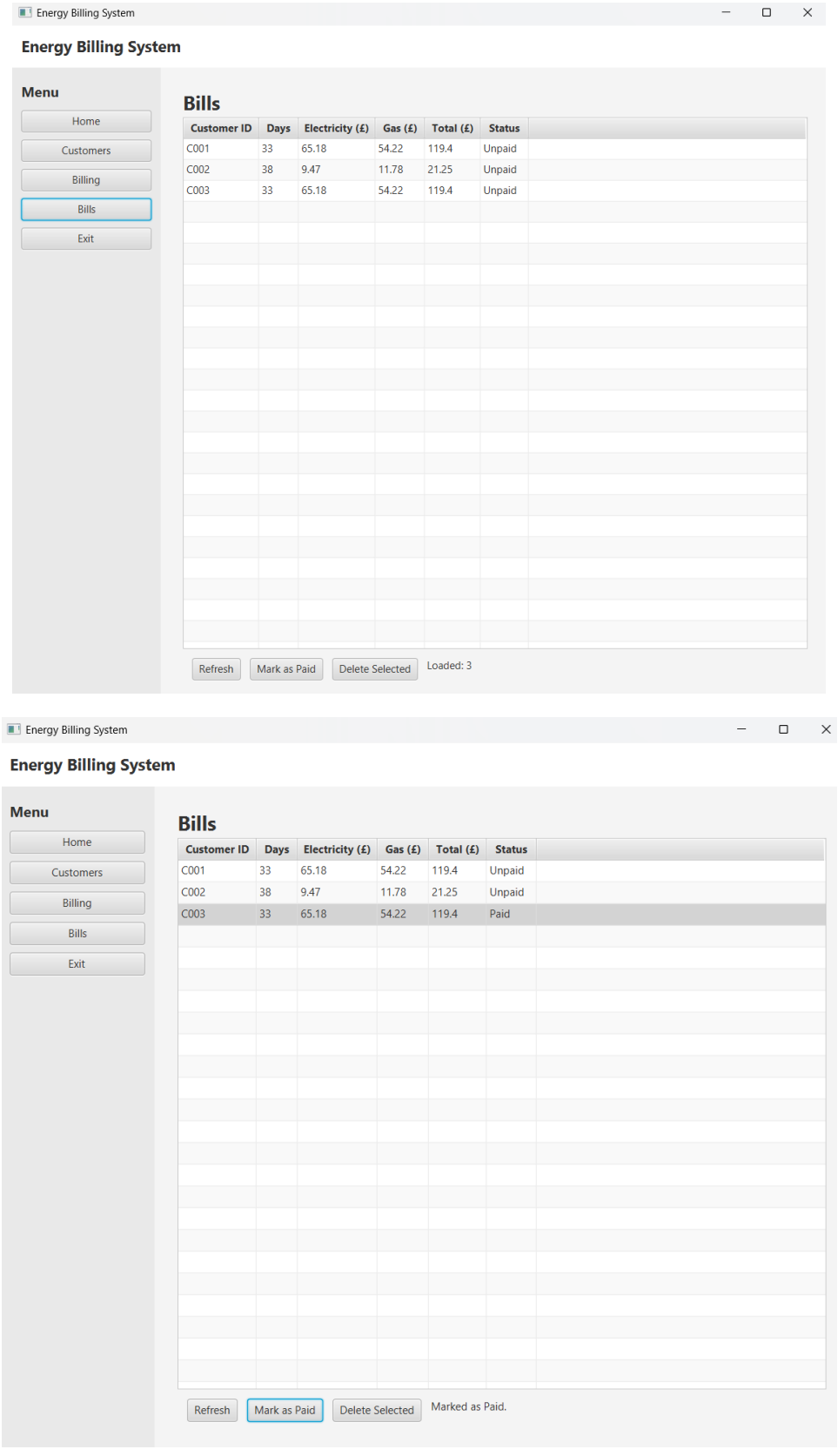
This shows the table loaded from the CSV file and the number of records displayed.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	customerI	days	electricity	gas	total	status								
2	C001	33	65.18	54.22	119.4	Unpaid								
3	C002	38	9.47	11.78	21.25	Unpaid								
4	C003	33	65.18	54.22	119.4	Unpaid								
5														
6														
7														
8														
9														
10														
11														
12														
13														
14														

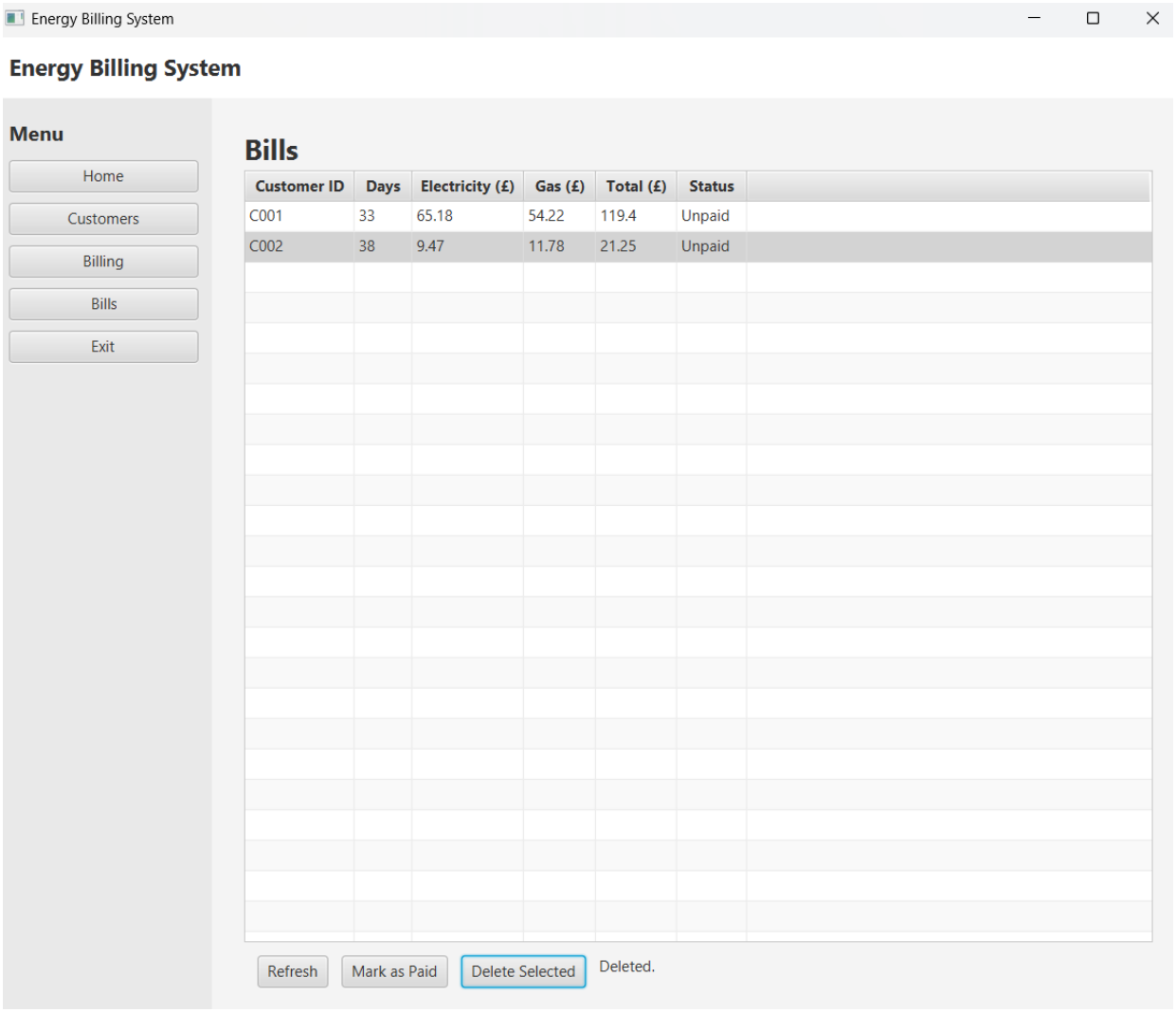
Bill Marked as Paid

This screenshot shows a bill after its status has been changed to “Paid”.



Bill Deleted

This shows the table after a selected bill has been deleted from the system.



6.2 Whitebox Testing

In this project, the most important logic is the bill calculation code, so I created JUnit tests for ElectricityBill and GasBill.

The main reason for using JUnit was to confirm that the calculation results match expected values using fixed inputs, and to make sure rounding and VAT are applied correctly.

I created a separate package called application.tests to keep unit tests separate from the main application code.

For each test, I create a bill object with known input values.

Then I call calculateBill() and compare the result to the expected value in pounds

If the expected value matches, the test passes. If not, the test fails.

ElectricityBillTest.java:

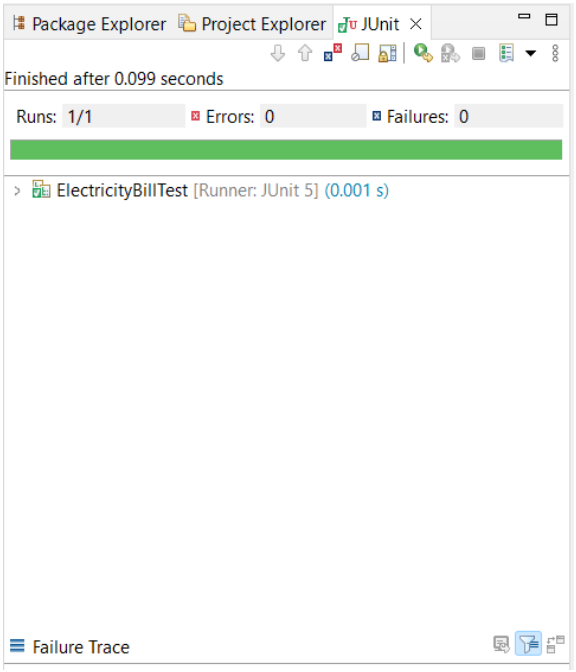
```
package application.tests;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import application.ElectricityBill;
public class ElectricityBillTest {
    @Test
    void shouldMatchSampleStatement_electricity() {
        int days = 33;
        ElectricityBill e = new ElectricityBill(
                40470.637, 40516.687,
                37386.998, 37623.210,
                22.63,   // standing charge (p/day)
                19.349,  // unit price (p)
                days
        );
        double pence = e.calculateBill();
        double pounds = pence / 100.0;
        assertEquals(65.18, pounds, 0.0001);
    }
}
```

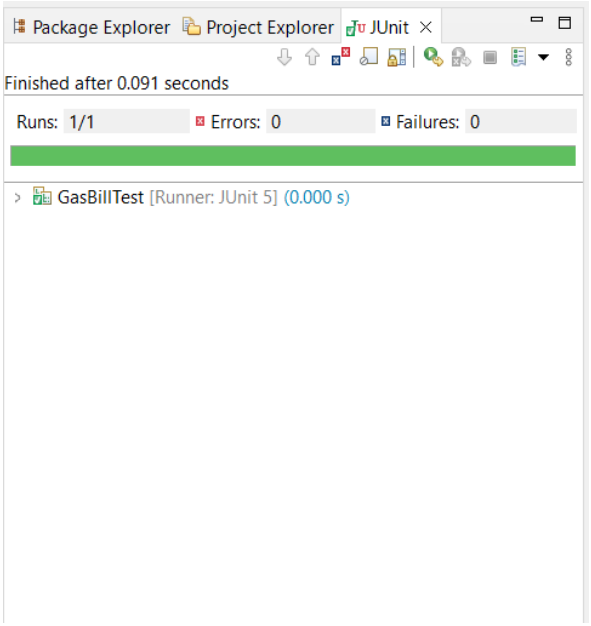
GasBillTest.java:

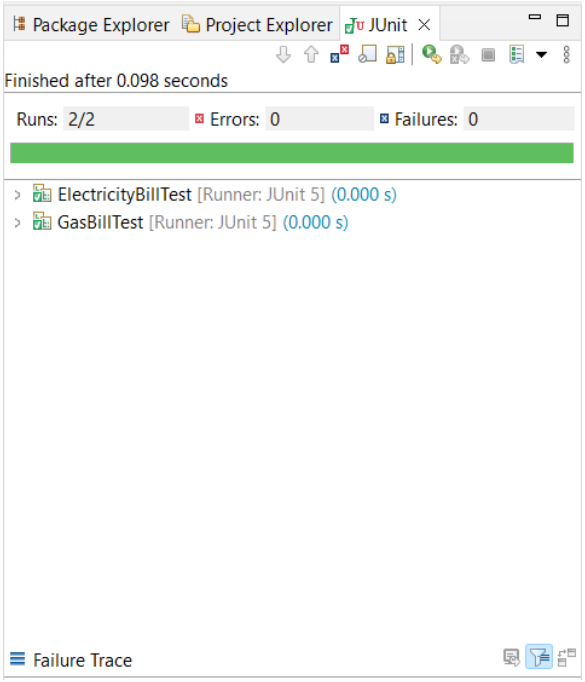
```
package application.tests;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import application.GasBill;
public class GasBillTest {

    @Test
    void shouldMatchSampleStatement_gas() {
        int days = 33;
        GasBill g = new GasBill(
                10091.5, 10127.6,
                24.87,
                3.797,
                days
```

```
        );
        double pence = g.calculateBill();
        double pounds = pence / 100.0;
        assertEquals(54.22, pounds, 0.0001);
    }
}
```

I also included screenshots showing the unit tests running successfully.







6.3 Bugs, Weaknesses

The main system functions work correctly, but the current version has some limitations. These are not critical errors, but they are areas that could be improved in a future version.

- **Login is fixed (no user management):** The system uses one fixed username and password (admin/1234). The user cannot create new accounts, change passwords, or manage different roles (admin/user).
- **No "Edit/Update" feature for customers:** Customers can be added and deleted, but there is no feature to edit an existing customer's name, phone, or address in the GUI.
- **No "Edit/Update" feature for bills:** Bills can be saved, deleted, and marked as paid, but bills cannot be edited after they are created.
- **Customer deletion does not check related bills:** A customer can be deleted even if bills exist for that customer. In a real system, the system should warn the user or prevent deletion.
- **No search or filter:** The system does not include searching customers or filtering bills (for example by status Paid/Unpaid).
- **No confirmation dialogs:** Delete actions remove rows immediately. A confirmation popup could be added to avoid accidental deletion.

7 References

I developed this project using the concepts and examples taught during lectures.
I used some example code provided by the tutor as a reference.
I also used the official Java and JavaFX documentation when I needed help with layouts, TableView, event handling, and application structure. These resources were mainly used for learning syntax.
The following resources were used during the development of this project:

- Oracle (2024). Java Platform, Standard Edition Documentation.
  https://docs.oracle.com/javase/
- Oracle (2024). JavaFX Documentation.
  https://openjfx.io/javadoc/
- JUnit Team (2024). JUnit 5 User Guide.
  https://junit.org/junit5/docs/current/user-guide/
- University of Northampton. CSYM025 – Visual Object Software