*5.5 Post Prototype*

**Investor Centre Ltd Post Prototype System**

**Feedback from Prototype**

After completing the prototype for the **Investor Centre Ltd trading system**, I decided to gather feedback from potential users, including staff at **Investor Centre Ltd** and individuals with experience in **financial trading**. The goal was to understand their opinions on the system’s functionality, usability, and areas needing improvement.

To collect feedback, I created a **Google Forms survey** that participants could access to provide constructive insights on the system’s **interface, features, and overall user experience**. The survey was designed to identify usability issues, missing functionality, and potential enhancements that would improve the effectiveness of the system.

To ensure realistic feedback, I did **not** provide a walkthrough of the system before participants used it. This approach was intentional, as real users would not have a guided tutorial when interacting with the system for the first time. Instead, I wanted to assess how intuitive the interface was and whether users could navigate and utilize the features without prior instruction.

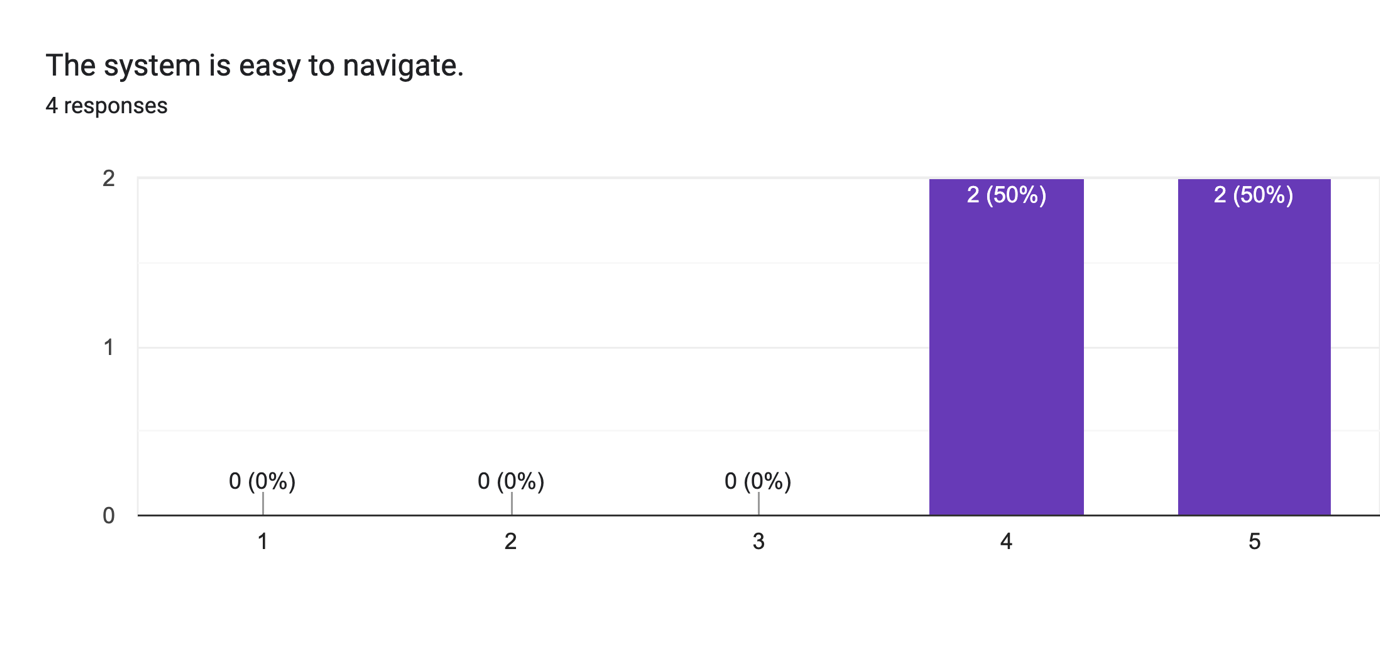
**The Survey**

The following is a list of the statements and questions included in the survey:

* **The system is easy to navigate** – If not, please provide details below.
* **Did you encounter any errors?**
* **Does the system calculate profit/loss correctly?**
* **Does the system store customer trades effectively?**
* **Does the system have sufficient validation?** – If so, please describe your experience with validation.
* **The system is slow to load** – Rated on a scale from **1 (Strongly Disagree) to 5 (Strongly Agree)**.
* **If you were a customer, would you use this system over the current system?** – If yes, explain why.
* **Do you have any additional feedback on the system?**

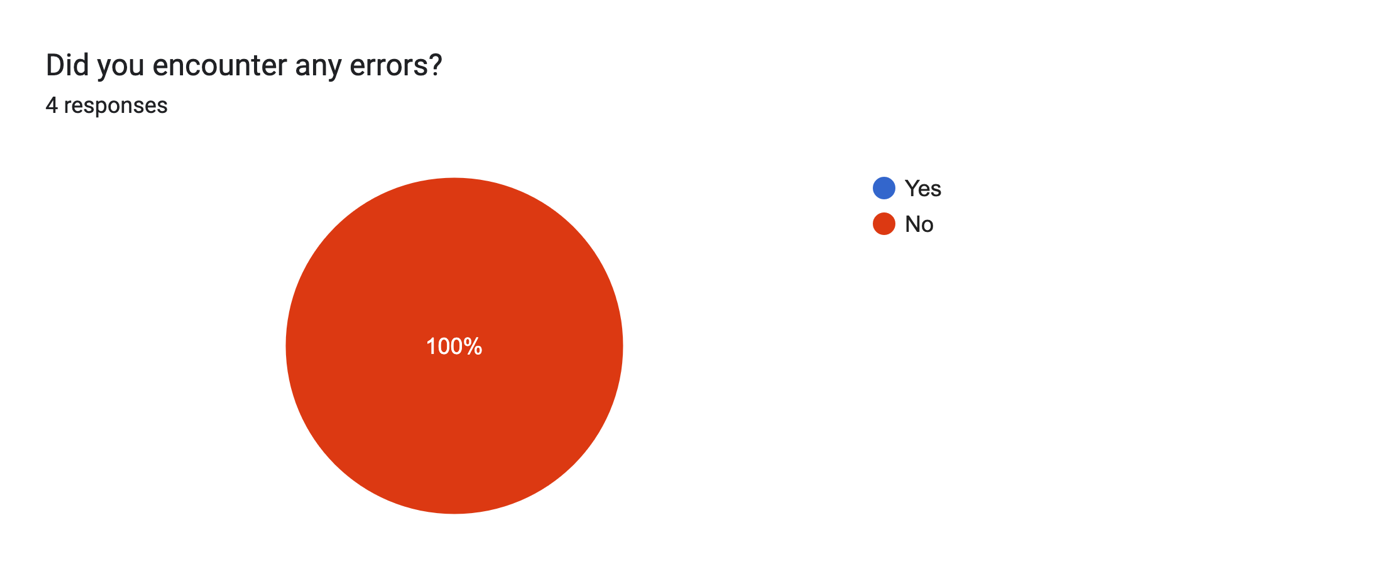
Users had the opportunity to **rate, provide comments, and highlight areas for improvement**, helping to identify usability issues, missing functionality, and potential refinements.

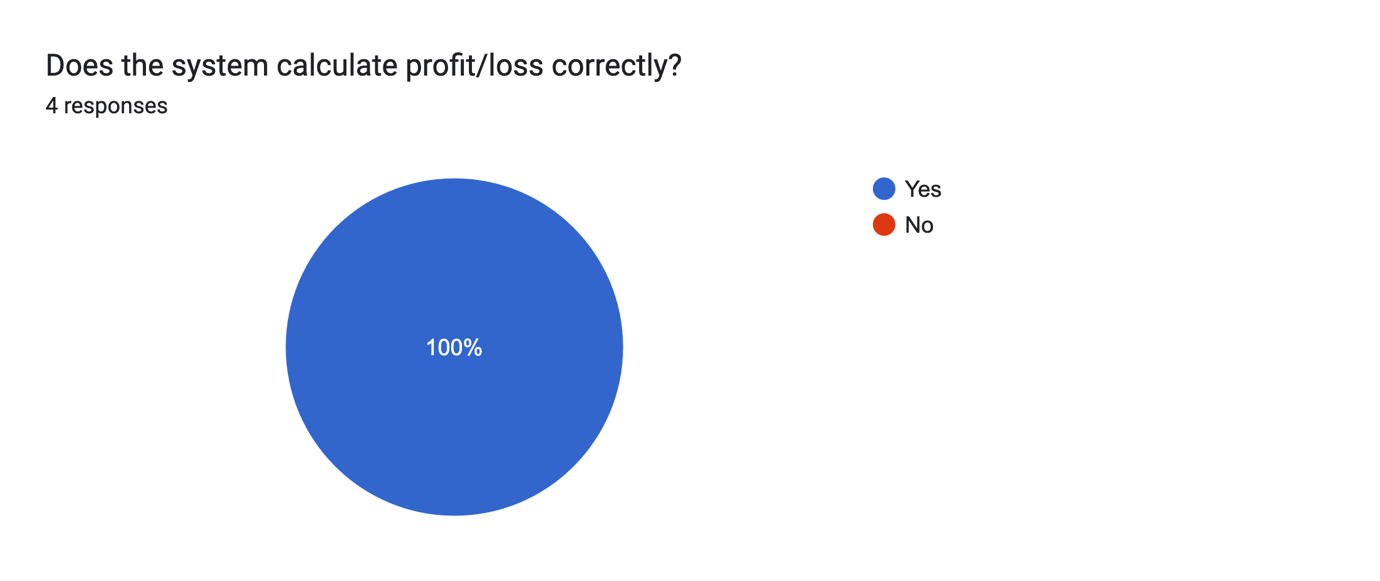
**Responses**

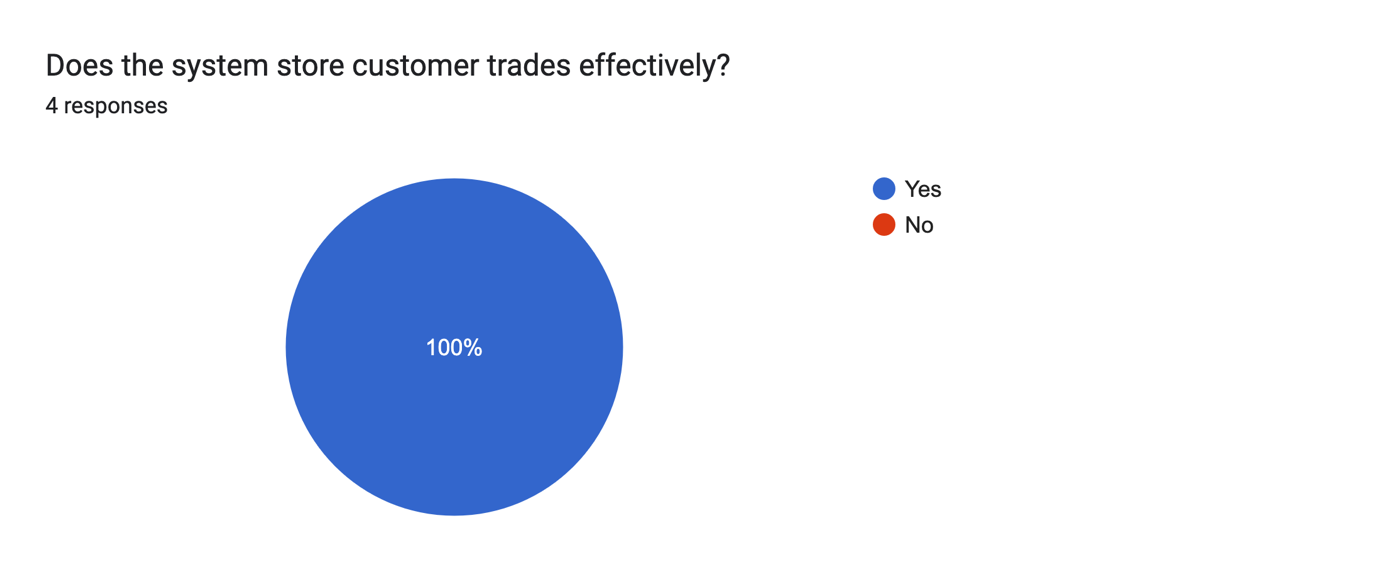


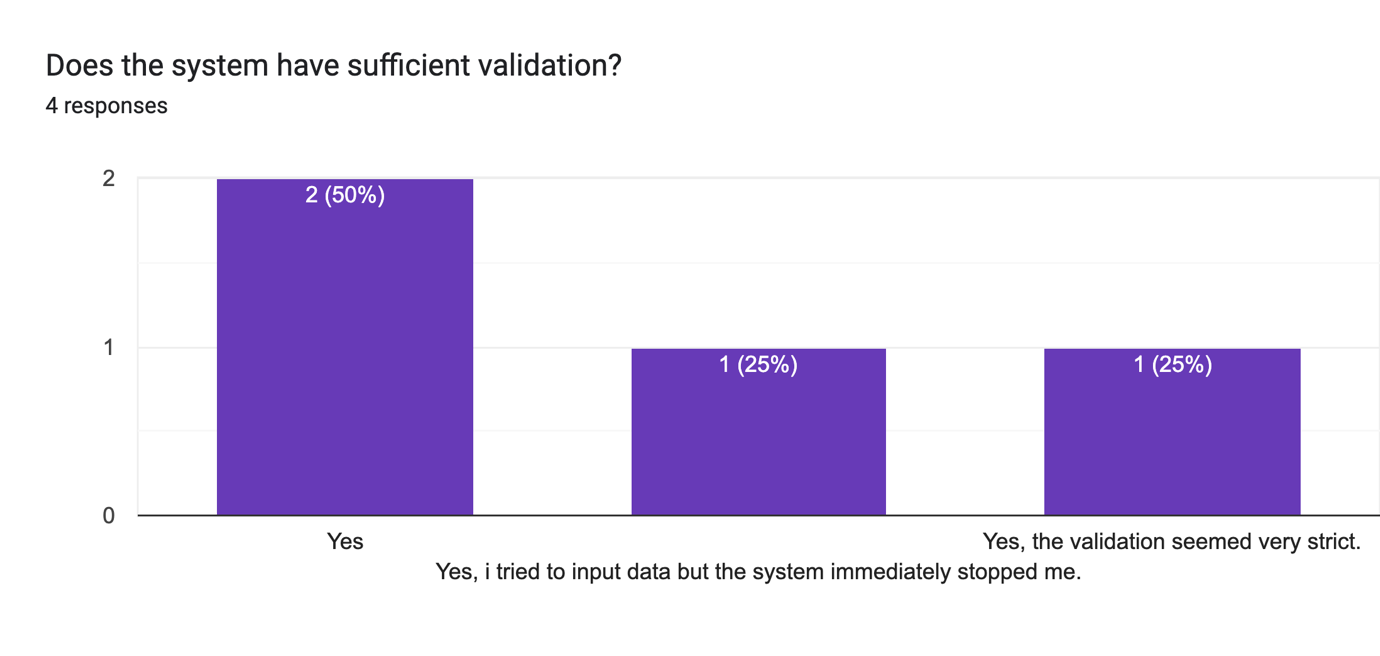
**A screenshot of a computer

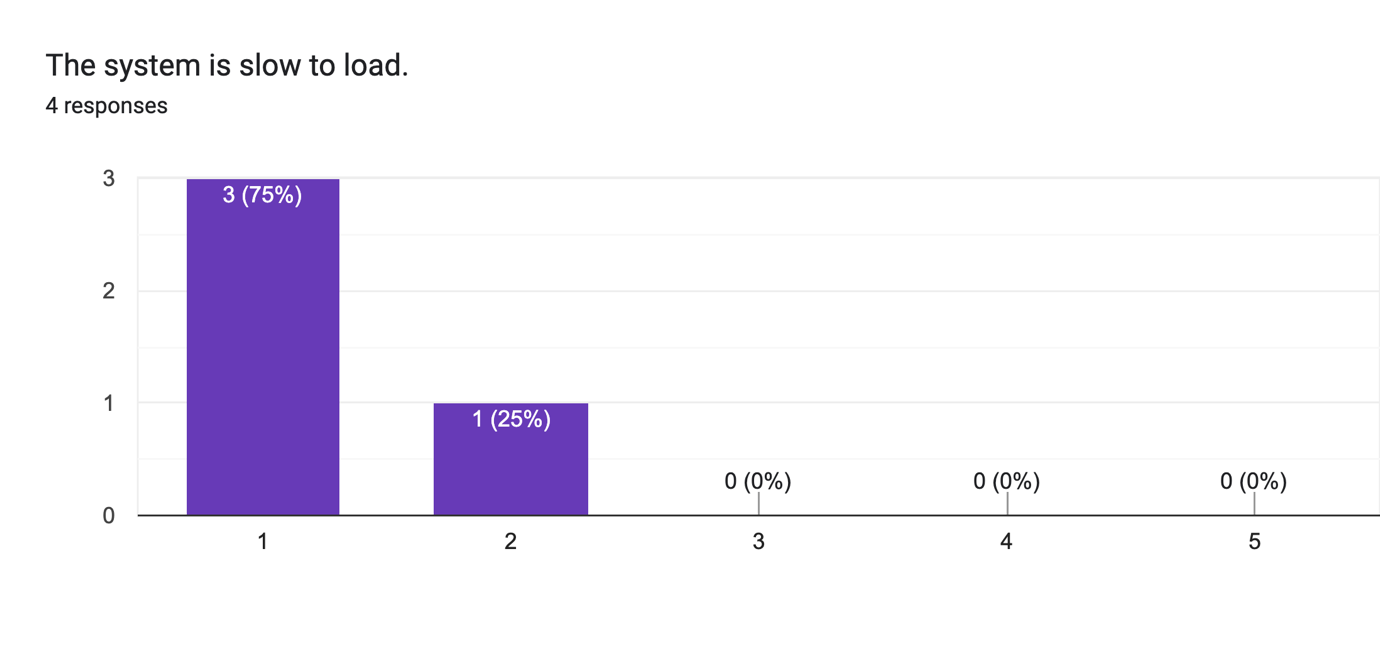
Description automatically generated**

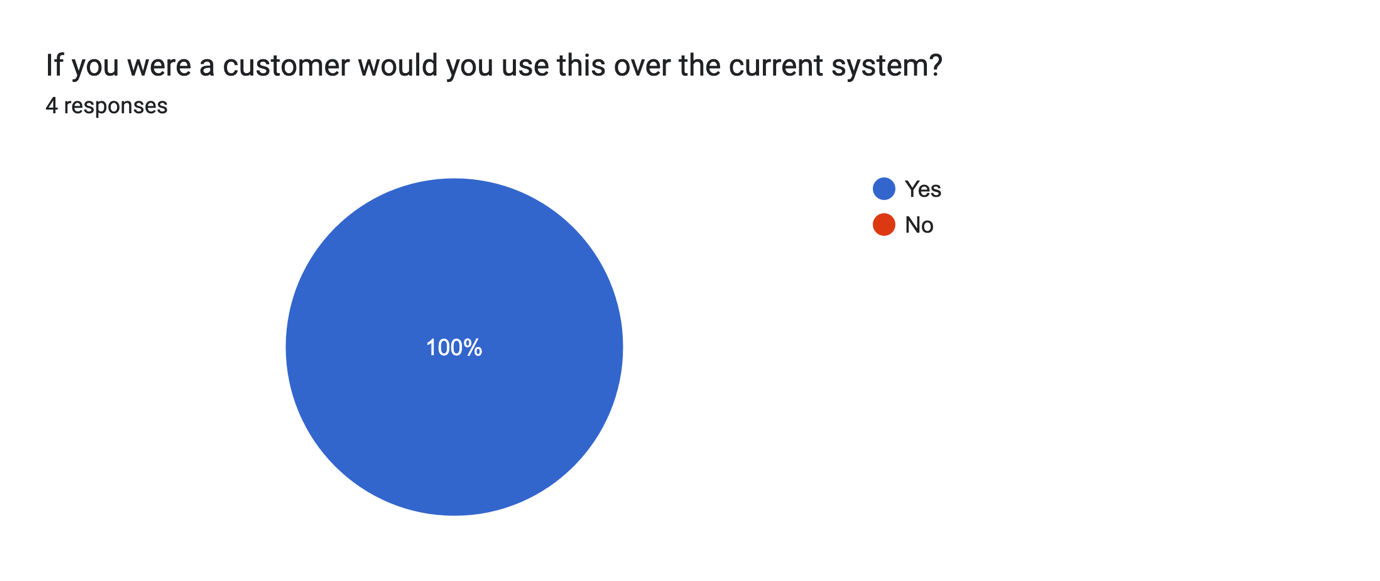












**A screenshot of a white box

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

### ****Analysis of Responses****

#### **The system is easy to navigate.**

Users generally found the system easy to navigate, with an average rating of **4.5 out of 5**. However, some feedback highlighted areas for improvement:

* One user noted that **some buttons do not seem to be working**, which may cause confusion.
* Another user mentioned that **pages load quickly**, but the **previous page remains open**, leading to multiple windows cluttering their desktop.
* One user pointed out that the **chart button has no functionality**, suggesting that missing features affect the navigation experience.

#### **Did you encounter any errors?**

**100% of users reported that they did not encounter any errors** while using the system. This indicates that the core functionality is stable.

#### **Does the system calculate profit/loss correctly?**

All users confirmed that **profit/loss calculations are accurate** (**100% "Yes" responses**), meaning the financial logic behind trade execution is functioning correctly.

#### **Does the system store customer trades effectively?**

Again, **100% of users agreed** that the system **correctly stores trade data**, suggesting that database interactions are working as expected.

#### **Does the system have sufficient validation?**

Users had mixed feedback regarding validation:

* **50% of users simply answered "Yes,"** indicating they found the validation sufficient.
* **One user mentioned that the system immediately stopped them from entering incorrect data**, suggesting that input validation is effective.
* **Another user found the validation "very strict,"** implying that while effective, some flexibility or clearer error messages might be needed.

#### **The system is slow to load.**

Most users **disagreed that the system is slow**, with a mean rating of **1.25** (where 1 = Strongly Disagree).

* **75% rated 1 (Strongly Disagree), and 25% rated 2**, suggesting that load times are **not a major concern** for users.

#### **If you were a customer, would you use this system over the current one?**

All users (**100% "Yes" responses**) preferred the prototype over the existing system. Reasons included:

* The system **allows customers to manage trades more quickly**, reducing reliance on intermediaries.
* It **gives users more control over their trades**, improving efficiency.
* The **simple design minimizes misunderstandings**, making trade execution clearer and more direct.

#### **Additional Feedback**

Users provided **two key improvement suggestions**:

* **Enable the chart and balance functions** to make the system more useful.
* **Implement encryption** to improve security, which is critical for financial applications.

### ****Conclusion****

The feedback provided **valuable insights into areas for improvement**. Users generally found the system **intuitive, functional, and an improvement over the existing process**, but **several key issues need to be addressed**:

* **Fix non-functional buttons** to improve navigation.
* **Ensure previous windows close properly** to prevent screen clutter.
* **Implement missing features** like charts and balance tracking to provide a complete trading experience.
* **Refine validation** to be strict but user-friendly.
* **Enhance security with encryption**, as financial systems require data protection.

By incorporating these improvements, the system will offer a **more seamless, efficient, and secure trading experience** for Investor Centre Ltd users.

**Fix of non-functional buttons**

**A screen shot of a computer program

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

The **balance\_window()** function is responsible for displaying the user's account balance in a dedicated window. It first retrieves the **customer's current balance** using the get\_customer\_balance\_display() function. The window is designed with a **modern, professional UI**, using Tkinter’s ttk module along with sv\_ttk for improved styling. The interface includes a **header, balance display, and three key action buttons**: **Add Funds**, **Withdraw Funds**, and **Manage Payment Methods**. Each button is styled distinctly to indicate its purpose, enhancing user experience. Additionally, a **Back button** is provided to allow users to return to the home screen smoothly. The function ensures proper **grid configuration** for a structured layout, preventing misalignment of UI elements.

The **chart()** function is designed to allow users to **configure and view financial charts** based on selected currency pairs, time periods, and intervals. It first opens a **chart configuration window**, where users can choose from predefined options using dropdown menus. The function fetches **historical financial data** from **Yahoo Finance (yfinance)** based on the selected parameters. Once the user confirms their selections, the data is processed and validated to ensure completeness. If all required fields are correctly filled, the function generates a **candlestick chart using mplfinance**, applying a **customized visual style** for clarity. The function also implements **error handling** to manage missing data or incorrect inputs, ensuring that the system remains stable even if an issue occurs.

**Ensuring previous windows close properly**

****

The button labelled "← Back" in a new window. When clicked, the button triggers two actions: it closes the current window (new\_window.destroy()) and opens the home window (open\_home()) using a lambda function. The button has a width of 10 and is placed within the window with padding for a better layout. This functionality is essential for improving navigation within the application, addressing the feedback to ensure windows close properly and enhancing the overall user experience by preventing screen clutter.

**Refined validation**

**A screen shot of a computer program

Description automatically generated**

This block of code performs several validation checks on the input fields in the balance\_window() for managing customer credit card payments. Here's a breakdown of the validation process:

1. **Check Required Fields**: It first ensures that all essential fields are filled, except for the optional "Address Line 2." If any required field is empty, an error message is displayed.
2. **Cardholder Name Validation**: It checks that the cardholder\_name contains only letters and spaces, ensuring valid input for the cardholder’s name.
3. **Card Number Validation**: The code ensures that the card\_number is exactly 16 digits long, as required by credit card standards.
4. **Start and End Date Validation**: The start\_date and end\_date are validated to ensure they follow the MM/YY format using a regular expression. This guarantees that users enter valid expiration dates.
5. **Card Provider Validation**: It ensures that the card\_provider field is not empty, confirming that a card provider is selected or provided.
6. **Building Number Validation**: The building\_number is checked to ensure it contains only digits, preventing any non-numeric characters.
7. **Address Line 1 Validation**: This ensures that line1, the primary address field, is not empty.
8. **Postcode Validation**: The postcode is validated using a regular expression to match the specific format ensuring a valid postcode is entered.
9. **Password Validation**: It ensures that the password has a minimum length of 8 characters to meet security requirements.

Each validation is followed by an error message using tk.messagebox.showerror if the input is invalid, prompting the user to correct the error before proceeding. These validations help improve data accuracy, prevent user errors, and ensure a more secure and efficient payment management system for customers.

**Enhanced security with encryption**

**A computer screen shot of a program

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

The **create\_account()** function is responsible for creating a new customer account in the database. It begins by establishing a connection to the SQLite database and creating a cursor for executing SQL queries.

The function then generates a **salt** using os.urandom(16).hex(), which creates a unique random string to be added to the password before hashing. This salt is used with the user's plaintext password and is hashed using the SHA-256 algorithm (hashlib.sha256).

The resulting hashed password, along with the salt and other user details such as the date of birth, name, email, phone number, and username, are then inserted into the customers table. The function handles any potential errors, such as duplicate usernames, by catching the exception and returning False if an error occurs.

Finally, the connection to the database is closed. This process ensures that the customer’s password is securely stored and cannot be easily accessed, as the plaintext password is never saved directly in the database.