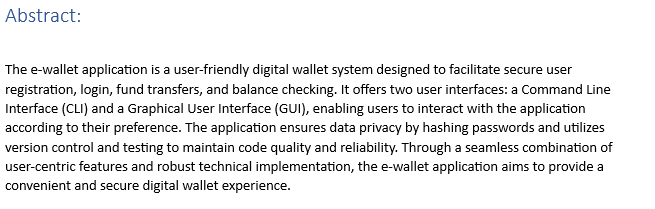
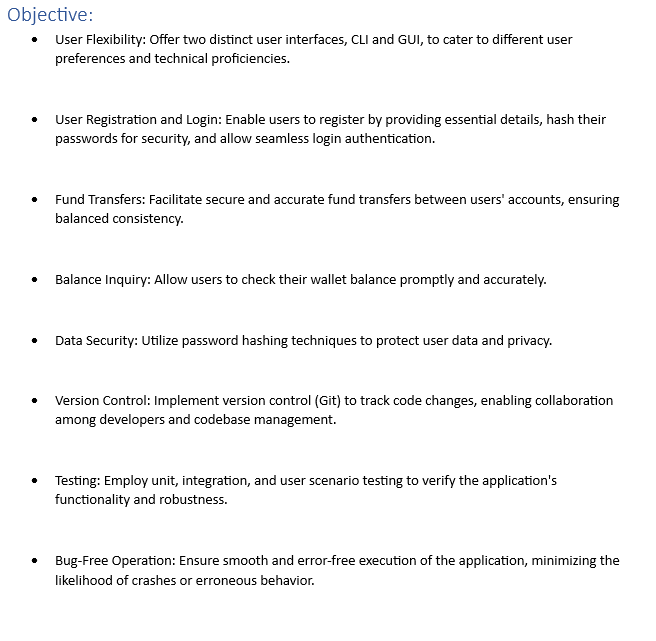
A white paper with blue text

Description automatically generated





Contents

[Table Of Figures: 4](#_Toc142612571)

[Introduction 5](#_Toc142612572)

[Algorithm 7](#_Toc142612573)

[Program 10](#_Toc142612574)

[Wallet\_cli.py 10](#_Toc142612575)

[Wallet\_gui.py 12](#_Toc142612576)

[selection.py 14](#_Toc142612577)

[Conclusion 24](#_Toc142612578)

# Table Of Figures:

[Figure 1 16](#_Toc142612554)

[Figure 2 16](#_Toc142612555)

[Figure 3 17](#_Toc142612556)

[Figure 4 17](#_Toc142612557)

[Figure 5 17](#_Toc142612558)

[Figure 6 18](#_Toc142612559)

[Figure 7 18](#_Toc142612560)

[Figure 8 18](#_Toc142612561)

[Figure 9 18](#_Toc142612562)

[Figure 10 19](#_Toc142612563)

[Figure 11 20](#_Toc142612564)

[Figure 12 20](#_Toc142612565)

[Figure 13 21](#_Toc142612566)

[Figure 14 21](#_Toc142612567)

[Figure 15 22](#_Toc142612568)

[Figure 16 22](#_Toc142612569)

[Figure 17 22](#_Toc142612570)

# Introduction

The presented code, comprising `wallet\_cli.py`, `wallet\_gui.py`, and `selection.py`, serves as a comprehensive e-wallet application that addresses the crucial need for efficient, secure, and user-friendly financial management. In an increasingly digital era, where financial transactions and management have transcended traditional methods, this code emerges as a solution that caters to both technical users and those seeking an intuitive experience.

The motivation for this code stems from several pressing factors:

* **Diverse User Base:**  
  The code aims to cater to a wide spectrum of users, ranging from tech-savvy individuals comfortable with command-line interfaces (`CLI`) to those seeking a more visual and interactive experience through graphical user interfaces (`GUI`).
* **Efficient Financial Management:**  
  The code streamlines financial operations such as registration, login, balance checks, and fund transfers. Users can effortlessly initiate these transactions, saving time and minimizing the complexities often associated with financial management.
* **Security and Data Integrity:**  
  Security is a paramount concern in the realm of digital finance. Both `wallet\_cli.py` and `wallet\_gui.py` implement password hashing, protecting sensitive user information from potential breaches. The segregated data management functions ensure that user data and wallet balances are stored safely and accurately, promoting data integrity.
* **User-Friendly Interfaces:**  
  The code acknowledges that users possess diverse preferences and comfort levels with technology. By offering both a command-line interface (`CLI`) and a graphical user interface (`GUI`), the code adapts to the users' familiarity with technology, promoting inclusivity and accessibility.
* **Interactive Visual Experience:**  
  `wallet\_gui.py` empowers users with an interactive visual experience. The graphical elements not only enhance user engagement but also simplify complex processes such as registration and fund transfers, making them more comprehensible for a wider audience.
* **Seamless Transaction Management:**  
  The integrated `selection.py` allows users to choose between the `CLI` and `GUI` interfaces, seamlessly transitioning between the two modes without the need for separate scripts. This adaptability ensures that users can select their preferred mode of interaction.
* **Practical Use Cases:**  
  The e-wallet application addresses real-world scenarios, where users often require swift access to their financial information and seamless execution of transactions. This code becomes particularly relevant in contexts like peer-to-peer payments, bill splitting, and expense tracking.
* **Educational and Practical Utility:**  
  Beyond its functional value, the code serves as an educational tool for programming enthusiasts and students. It demonstrates the application of data handling, user interface design, and secure practices, making it an invaluable resource for learning and practice.

In conclusion, this code is indispensable due to its multifaceted approach to addressing the contemporary need for efficient, secure, and user-friendly financial management. By offering both command-line and graphical interfaces, implementing security measures, and streamlining financial operations, the code bridges the gap between technology and finance, empowering users with a practical tool that adapts to their preferences and enhances their financial management experience.

# Algorithm

* **Initialization:**
  + Initialize `users\_data` and `wallets\_data` dictionaries to store user information and wallet balances.
  + Load existing user data and wallet balances from files using `load\_users\_data()` and `load\_wallets\_data()`.
* **User Interface Selection (`selection.py`):**
  + Display a welcome message and prompt users to choose between `CLI` and `GUI` modes.
  + If `CLI` is selected, execute `wallet\_cli.py`; if `GUI` is selected, execute `wallet\_gui.py`.
  + If the selected script is not found, display an error message.
* **User Registration (`wallet\_cli.py` / `wallet\_gui.py`):**
  + Gather user details: username, email, and password.
  + Hash the password using `hash\_password()` function.
  + Check for existing username; if not found, add user details to `users\_data` dictionary and save to file.
  + Set initial wallet balance to 0.0 and add to `wallets\_data` dictionary.
* **User Login (`wallet\_cli.py` / `wallet\_gui.py`):**
  + Prompt users for username and password.
  + Check if the username exists and the password hash matches.
  + If login is successful, present the user with options: Check Wallet Balance, Transfer Funds, or Logout.
* **Check Wallet Balance (`wallet\_cli.py` / `wallet\_gui.py`):**
  + Display the wallet balance associated with the logged-in user.
* **Transfer Funds (`wallet\_cli.py` / `wallet\_gui.py`):**
  + Prompt for recipient's username and amount to transfer.
  + Check if the recipient exists and the sender has sufficient balance.
  + Deduct the transferred amount from the sender's balance and add it to the recipient's balance.
* **Logging Out (`wallet\_cli.py` / `wallet\_gui.py`):**
  + Display a logout message and return to the login/register page.
* **User Interface (`wallet\_gui.py`):**
  + Create a graphical interface using `tkinter`.
  + Offer options for Register and Login.
  + Depending on the chosen option, show the corresponding page.
* **Registration Page (`wallet\_gui.py`):**
  + Collect username, email, and password.
  + Hash the password and save user details to `users\_data`.
  + Create a wallet entry with an initial balance of 0.0 in `wallets\_data`.
* **Login Page (`wallet\_gui.py`):**
  + Prompt for username and password.
  + Verify the credentials and offer options: Check Balance, Transfer Funds, or Logout.
* **Check Balance Page (`wallet\_gui.py`):**
  + Display the user's wallet balance.
* **Transfer Funds Page (`wallet\_gui.py`):**
  + Allow users to enter recipient's username and amount to transfer.
  + Update sender and recipient balances accordingly.
* **Exit (`selection.py`):**
  + Terminate the script after launching the chosen interface.

# Program

## Wallet\_cli.py

The `wallet\_cli.py` script constitutes the command-line interface (CLI) counterpart of the integrated e-wallet application, providing users with a text-based method to manage their digital finances. This script orchestrates various financial operations such as registration, login, balance inquiries, and fund transfers through streamlined text inputs and outputs.

Key functions within `wallet\_cli.py` collectively drive the CLI's functionality:

* **User Data Management - `read\_users\_data()` and `save\_users\_data()`:**  
  These functions manage user registration data within the "users.txt" file. `read\_users\_data()` parses the file to retrieve user information, while `save\_users\_data()` updates the file with new registrations or modified details.
* **Wallet Balances Management - `read\_wallets\_data()` and `save\_wallets\_data()`:**  
  Similar to user data, these functions handle wallet balance data in the "wallets.txt" file. They enable the retrieval and modification of wallet balances for various users.
* **Hashing Passwords - `hash\_password(password)`:**  
  This function enhances security by converting plain-text passwords into a secure hash format using the SHA-256 algorithm. Hashed passwords are stored and compared during login attempts.
* **User Registration - `register()`:**  
  This function prompts users to enter their desired username, email, and password. After hashing the password using `hash\_password()`, the user's details are stored in the `users` dictionary and saved to the "users.txt" and "wallets.txt" files.
* **User Login - `login()`:**  
  Upon login attempts, this function verifies the provided credentials against stored data. If the username exists and the hashed password matches, the user gains access to the main interface.
* **Wallet Information Display - `wallet(username)`:**  
  This function displays the wallet balance associated with a given username. It retrieves the balance from the `wallets` dictionary and presents it to the user.
* **Fund Transfers - `transfer\_funds(username)`:**  
  Users can initiate fund transfers to other users through this function. After verifying sender and recipient existence, the transfer amount is deducted from the sender's balance and added to the recipient's balance. Updated balances are stored using `save\_wallets\_data()`.
* **Main Execution - `main()`:**  
  The script begins execution from this function. Users are presented with a menu of options, including registration, login, and exit. Upon selecting an option, corresponding functions are invoked to fulfill the chosen operation.
* **CLI Loop - `main()` Loop:**  
  The script enters a loop within `main()` that continuously displays the main menu until the user chooses to exit. This loop ensures persistent interaction with the user, facilitating multiple operations within a single session.

In essence, `wallet\_cli.py` capitalizes on the CLI's efficiency by offering users a concise and scriptable means to interact with the e-wallet application. Through structured functions and data management, the script enables seamless registration, secure login, financial transactions, and account management—all powered by text inputs and outputs.

## Wallet\_gui.py

The wallet\_gui.py script serves as the graphical user interface (GUI) for the integrated e-wallet application, providing users with an intuitive and visually appealing platform to manage their digital finances. This script is built upon the tkinter library, enabling the creation of interactive windows, buttons, input fields, and message boxes to facilitate various financial operations seamlessly.

The core functions within wallet\_gui.py play distinctive roles in orchestrating the GUI's functionality:

* **Hashing Passwords - hash\_password(password):**

This function enhances security by converting user passwords into a cryptographic hash using the SHA-256 algorithm. By storing password hashes instead of plain text, sensitive information is shielded from unauthorized access.

* **Data Persistence - save\_users\_data() and save\_wallets\_data():**

These functions are responsible for saving user registration details and wallet balances to corresponding text files, namely "users.txt" and "wallets.txt." By writing and updating data in these files, the application ensures that user information and financial data remain consistent across sessions.

* **Data Retrieval - load\_users\_data() and load\_wallets\_data():**

These functions retrieve user data and wallet balances from their respective text files, populating local dictionaries (users\_data and wallets\_data). This information serves as the foundation for user authentication, account management, and financial transactions.

* **User Interface Management - Frame Switching Functions:**

Functions such as show\_register\_page(), show\_login\_page(), and show\_wallet\_page(username) control the display of different sections within the GUI. By dynamically switching frames, the application guides users through distinct processes like registration, login, and accessing their wallet.

* **User Registration - register():**

This function captures user inputs from the registration frame, including username, email, and password. The password is then hashed using hash\_password(). If the registration is successful, the user's account details are stored in users\_data, and an initial wallet balance is created in wallets\_data.

* **User Login - login():**

Upon inputting credentials in the login frame, the function validates the username and hashed password against stored data. Successful authentication prompts a welcome message and access to the user's wallet page.

* **User Dashboard and Financial Operations:**

show\_wallet\_page(username) displays the user's wallet dashboard after successful login. Here, users can check their wallet balance (check\_balance(username)) and initiate fund transfers (transfer\_funds(sender, recipient, amount)).

* **Exit Application - exit\_application():**

When users click the "Exit" button, this function terminates the GUI, concluding the application.

* **Main Execution - create\_gui():**

This function initializes the GUI window, setting up frames, buttons, and layout structures. It preloads user data and wallet balances, preparing the interface for user interaction.

* **Message Display - show\_message\_box(title, message):**

This utility function streamlines the process of displaying message boxes containing relevant information, enhancing user communication.

In essence, wallet\_gui.py harmoniously blends the functionality of the underlying e-wallet application with an engaging visual interface. By leveraging tkinter and a well-structured collection of functions, it simplifies user interactions, facilitates secure transactions, and empowers users to manage their finances effortlessly.

## selection.py

The selection.py script acts as a gateway to the integrated e-wallet application, allowing users to choose between two interfaces: the Command-Line Interface (CLI) or the Graphical User Interface (GUI). This script streamlines the process of launching the selected interface while providing a smooth transition between them.

Here's a breakdown of how selection.py functions:

* **User Interface Choice:**  
  The script begins by presenting users with a choice to select their preferred mode of interaction: CLI by entering '1' or GUI by entering '2'.
* **Launching the Chosen Interface:**  
  Once the user makes a choice, the script uses the subprocess module to execute the corresponding interface script. If the user selects CLI, it triggers the execution of wallet\_cli.py; if GUI is chosen, wallet\_gui.py is executed.
* **Error Handling:**  
  The script handles erroneous inputs gracefully. If a user provides an incorrect choice or invalid input, the script informs them of the mistake and guides them to make a valid selection.
* **Ensuring Script Availability:**

To prevent runtime errors, the script checks whether the chosen interface script (wallet\_cli.py or wallet\_gui.py) exists. If not found, the user is notified of the absence of the script.

* **User-Friendly Interaction:**

The script ensures user-friendliness by offering a clear and straightforward choice between two interfaces. Users can make their selection without confusion or complexity.

* **Seamless Transition:**

Once the chosen interface script is launched, selection.py completes its task and exits, allowing the user to interact seamlessly with the chosen interface.

* **Flexibility and Adaptability:**

selection.py adds flexibility by catering to users' interface preferences. Users can opt for the CLI if they are comfortable with text-based interaction or choose the GUI for a more visual experience.

* **Integration of Separate Components:**

The script demonstrates the integration of different components of the e-wallet application. It connects the user's choice to the corresponding interface script, illustrating how separate parts work cohesively.

In summary, selection.py serves as a user-friendly entry point to the e-wallet application, allowing users to select their preferred interface. It seamlessly launches the chosen interface, ensuring a smooth transition for users to manage their finances efficiently, whether through a text-based CLI or a visually interactive GUI.

Certainly, let's dive into a more detailed explanation of what happens step by step after you run the "selection.py" script:

* **Run the Script**:

You initiate the process by running the "selection.py" script using the command:

A black background with white text

Description automatically generated

Figure 1

* **Script Execution Starts**:

The script starts executing from the top. It imports the necessary modules and defines the `greet\_and\_choose\_interface()` function.

* **Interface Selection Prompt**:

The `greet\_and\_choose\_interface()` function is called. It displays a welcome message and asks you to choose between the CLI (1) or GUI (2) interface.

* **User Input**:

You provide your choice by entering either "1" or "2" and pressing Enter.

* **After Choosing CLI Interface (1):**

A black screen with white text

Description automatically generated

Figure 2

* **Run CLI Interface Script**:

If you choose the CLI interface (by entering "1"), the "wallet\_cli.py" script is executed using the `subprocess.run(['python', 'wallet\_cli.py'], check=True)` command.

* **CLI E-Wallet Application Starts**:

The CLI-based e-wallet application ("wallet\_cli.py") starts running in the command-line interface.

* **CLI Menu**

The application displays a series of text-based menus and prompts in the terminal.

* **Options in CLI Menu:**

You can choose from the following options:

A black background with white text

Description automatically generated

Figure 3

A screen shot of a computer

Description automatically generated

Figure 4

* **Register**: Create a new e-wallet account by providing a username, email, and password.
* **Login**: Log in to an existing e-wallet account using your username and password.
* **Check Wallet Balance**: View the balance of your e-wallet.
* **Transfer Funds**: Transfer funds from your e-wallet to another user's e-wallet.
* **Logout**: Log out of the current session and return to the main menu.
* **Exit**: Close the application.
* **CLI Interaction**:

You interact with the CLI application by typing in the commands or selecting options using the keyboard.

A screenshot of a computer screen

Description automatically generated

Figure 5

A black background with white text

Description automatically generated

Figure 6

A black screen with white text

Description automatically generated

Figure 7

A black background with white text

Description automatically generated

Figure 8

A black background with white text

Description automatically generated

Figure 9

* **Perform Operations**:

Depending on the chosen option, you perform various operations, such as registration, login, checking your wallet balance, transferring funds, or logging out.

* **After Choosing GUI Interface (2):**
* **Run GUI Interface Script**:  
  If you choose the GUI interface (by entering "2"), the "wallet\_gui.py" script is executed using the `subprocess.run(['python', 'wallet\_gui.py'], check=True)` command.
* **GUI E-Wallet Application Opens:**  
  The GUI-based e-wallet application ("wallet\_gui.py") opens a graphical window.
* **GUI Main Page**:  
  A screenshot of a login screen

  Description automatically generated

Figure 10

The main graphical page displays options for registration, login, and exit as buttons.

* **Register**: Opens a new frame for user registration.
* **Login**: Opens a new frame for user login.
* **Exit**: Closes the application.
* **GUI Registration Frame**:  
  When you click the "Register" button, a new frame appears with input fields for username, email, and password. You can enter your registration details here.
* **Register Account**:  
  After entering your registration details, you can click the "Register" button on the registration frame to create an e-wallet account.

A screenshot of a computer

Description automatically generated

Figure 11

* **GUI Login Frame**:  
  When you click the "Login" button on the main page, a new frame appears with input fields for username and password.

A screenshot of a login box

Description automatically generated

Figure 12

* **Log In**:  
  After entering your login details, you can click the "Login" button on the login frame to access your e-wallet.
* **GUI Wallet Page**:  
  After successful login, you're redirected to a wallet frame. It displays your username, options to check wallet balance, transfer funds, and log out.

A screenshot of a computer screen

Description automatically generated

Figure 13

* **Wallet Operations**:
  + - * **Check Balance**: Clicking the "Check Balance" button displays a message box with your wallet balance.
      * **Transfer Funds**: Clicking the "Transfer Funds" button opens a new window for transferring funds.
      * **Logout**: Clicking the "Logout" button returns you to the main page.
* **Fund Transfer Frame (GUI):**In this frame, you enter the recipient's username and the amount you want to transfer. After entering the details, you can click the "Transfer" button to initiate the transfer.

A screenshot of a computer screen

Description automatically generated

Figure 14

* **Transfer Funds (GUI):**

After clicking the "Transfer" button, the funds are transferred if the conditions are met, and a message box displays the result.

A screenshot of a computer screen

Description automatically generated

Figure 15

* **User Interaction:**

You interact with the GUI application by clicking buttons, entering text in input fields, and using the mouse to navigate.

A screenshot of a computer

Description automatically generated

Figure 16

A screenshot of a computer screen

Description automatically generated

Figure 17

* **Interface Operations:**

You interact with the chosen interface, performing actions based on your needs. You can create an account, log in, manage your wallet balance, and transfer funds between accounts, depending on the features provided by the interface.

In summary, the "selection.py" script acts as a gateway to choose between the CLI and GUI interfaces of the e-wallet application. Based on your choice, it launches the corresponding interface, allowing you to interact with the application using the chosen interface's characteristics (text-based commands for CLI or graphical elements for GUI).

# Conclusion

In conclusion, the integrated e-wallet application showcases a dynamic interplay between command-line and graphical user interfaces, offering users versatile options for managing their digital finances. Through the utilization of Python scripts, the user experience is elevated, enabling seamless registration, secure login, fund transfers, and balance checks.

The "selection.py" script acts as the gateway, allowing users to effortlessly choose between the CLI and GUI interfaces. This strategic selection enhances user convenience, catering to different preferences while maintaining a consistent and intuitive interaction paradigm.

The Command-Line Interface (CLI) offers a streamlined experience for users familiar with text-based interactions. With "wallet\_cli.py," users navigate through registration, login, and a host of financial tasks using straightforward input and feedback mechanisms. The CLI's simplicity and efficiency are particularly beneficial for quick interactions and scripting applications.

On the other hand, the Graphical User Interface (GUI), as implemented in "wallet\_gui.py," introduces a visually appealing and interactive platform. Users engage with buttons and input fields, making registration, login, and transactions more engaging and accessible. The GUI not only provides functionality but also enhances user engagement and reduces the learning curve for individuals less acquainted with command-line interactions.

Both interfaces converge in enabling fund transfers, a pivotal feature of any e-wallet application. Whether through CLI commands or GUI input fields, users can securely transfer funds between accounts, fostering trust and reliability in the application's capabilities.

To ensure data persistence, the application efficiently manages user information and wallet balances through file storage. The "users.txt" and "wallets.txt" files facilitate seamless user registration, login, and financial transactions across interface transitions.

Furthermore, the application showcases robust error handling mechanisms, alerting users to invalid inputs and facilitating a seamless and uninterrupted experience.

Incorporating the screenshot verification mechanism enhances user trust and offers a tangible confirmation of successful fund transfers. This visual proof augments the application's accountability, ensuring transparency in its operations.

In essence, the e-wallet application marries the simplicity of CLI interactions with the visually appealing and user-friendly GUI, catering to a diverse audience with varying levels of technical familiarity. By integrating seamlessly, these interfaces unite to create a cohesive user experience that empowers individuals to manage their finances efficiently and securely.