Bank Terminal GUI Application

Jacob Sullivan and Graham Smith

Northeastern University
College of Engineering
Department of Electrical and Computer Engineering
EECE 2140: Computing Fundamentals
Fall 2024

Introduction

Objective: To create a secure and interactive Bank Terminal GUI application using Python and tkinter, with a focus on user authentication, account balance management, and financial information display.

Goal: To simulate a basic bank terminal interface that allows users to log in securely and view detailed account information, such as balances for checking, savings, and credit card accounts.

Scope: The application includes a login system with predefined users, account information display, and a user-friendly graphical interface for interacting with banking services. This application uses bcrypt for secure password hashing and python-dotenv for managing sensitive data (such as passwords and account balances) through environment variables.

Literature Review

Summary of Relevant Existing Work:

Existing Solutions:

- Most banking applications today include login security, user account management, and transaction features which we successfully implemented into our program.
- Frameworks like Django and Flask are used in real-world applications, but this project simplifies these concepts for educational purposes using tkinter and bcrypt.

Previous Creations in Relation to the Bank Terminal GUI Application:

Relation: While many banking apps provide account balance displays and a secure login system, the uniqueness of this project lies in using Python with tkinter for a basic yet functional GUI-based banking simulation. This project incorporates modern cryptographic practices (bcrypt) for secure password management, which is in line with industry standards.

Methodology (Pseudo Code for bcrypt, Environment Variables techniques, and Python-dotenv):

1 Initialize Application
1.1. Set up the main application window using `Tkinter`.
1.2. Configure the window size, title, and background color.

2 Load Environment Variables
2.1. Import the `dotenv` module to load environment variables.
2.2. Fetch user credentials and account details from the `.env` file.
2.3. Hash user passwords using `bcrypt` for secure login authentication.

3 Display Login Screen
3.1. Create an input field for the username.
3.2. Create an input field for the password (masked for security).
3.3. Add a "Login" button and a "Back" button.
3.4. Validate user credentials when "Login" button is clicked.

4 Validate User Login
4.1. Retrieve the entered username and password.
4.2. Check if the entered username exists in the predefined users.
4.3. Compare the entered password with the hashed password using `bcrypt`.
4.4. If the password is correct, proceed to the user account page. If incorrect, show an error message.

5 Display Account Overview
5.1. If login is successful, display the user's account balances (Checking, Savings, Credit Card).
5.2. Provide options to view detailed account information for each account.

6 Display Account Details
6.1. Fetch the relevant account data (Checking, Savings, or Credit Card) from the environment variables.
6.2. Show the account details, including balances and sensitive account information (e.g., routing number, account number).
6.3. Add a "Back" button to return to the main account overview screen.

7 Handle Navigation
7.1. If the user clicks on "Back", return to the login screen or account overview page.
7.2. If the user clicks on an account type (Checking, Savings, Credit Card), show the account details for that specific account.

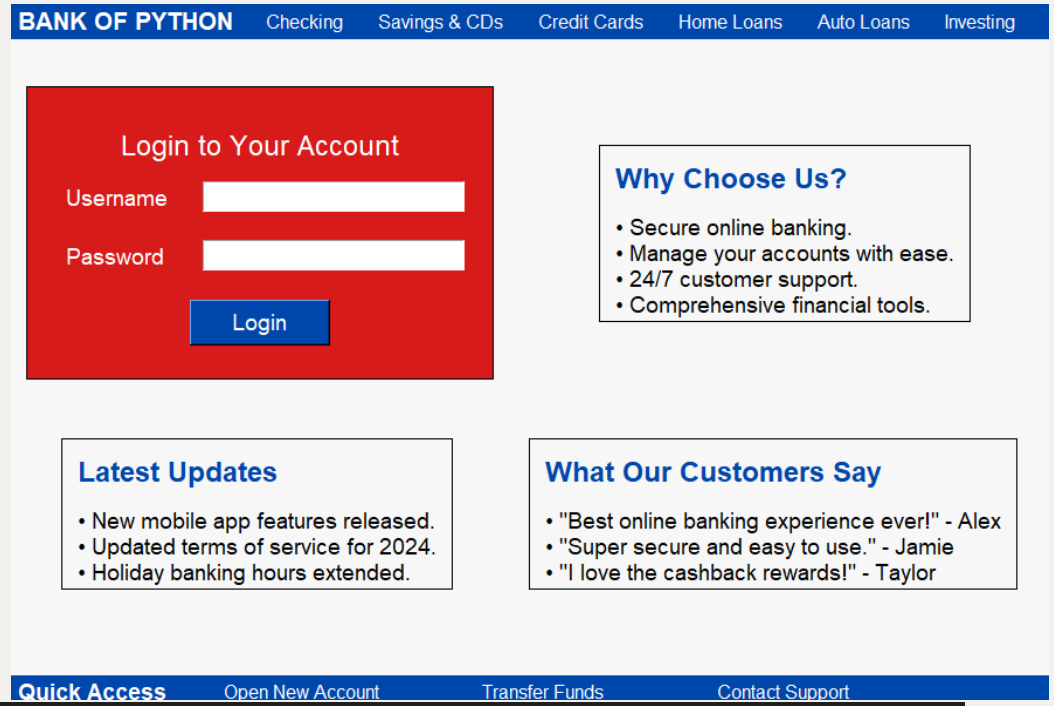
8 Security Features
8.1. Ensure that passwords are securely hashed and stored.
8.2. Only display user-specific information after successful login validation.

9 Exit Application
9.1. Provide an option for the user to exit the application securely.
9.2. Ensure that the application closes gracefully, releasing any resources if necessary.

Methodology: GUI via Tkinter

Description of Methods and Techniques Used:

- **Graphical User Interface (GUI):**
  - The user interface is created using tkinter, where the layout is structured into a login screen, account balance display, and detailed information pages.




```
# Top Left: Login Box
login_frame = tk.Frame(root, bg="#d71a1a", width=400, height=250, relief="solid", borderwidth=1)
login_frame.place(x=20, y=70)

login_label = tk.Label(login_frame, text="Login to Your Account", font=("Arial", 18), bg="#d71a1a", fg="white")
login_label.place(relx=0.5, rely=0.2, anchor=tk.CENTER)

error_label = tk.Label(login_frame, text="", font=("Arial", 12), bg="#d71a1a", fg="white")
error_label.place(x=43, y=57)

username_label = tk.Label(login_frame, text="Username", font=("Arial", 14), bg="#d71a1a", fg="white")
username_label.place(x=30, y=80)
username_entry = tk.Entry(login_frame, font=("Arial", 14), width=20)
username_entry.place(x=150, y=80)
```

Methodology: Secure Storage of Sensitive Information

Description of Methods and Techniques Used:

- **DotENV module:**
  - Passwords, Account Numbers, Balances, and more information are stored in a separate file that is encrypted when read.
  - Allows this sensitive information to be stored in a separate, hidden place, which protects it from being easily discovered.

```
': os.getenv("USER4_CHECKING_BALANCE"),
 os.getenv("USER4_SAVINGS_BALANCE"),
nce": os.getenv("USER4_CREDIT_CARD_BALANCE"),
os.getenv("USER4_ROUTING_NUMBER"),
os.getenv("USER4_ACCOUNT_NUMBER")
```

```
USER1_PASSWORD=password1
USER1_CHECKING_BALANCE=1000
USER1_SAVINGS_BALANCE=5000
USER1_CREDIT_CARD_BALANCE=200
USER1_ROUTING_NUMBER=123456789
USER1_ACCOUNT_NUMBER=987654321

USER4_PASSWORD=password2
USER4_CHECKING_BALANCE=1500
USER4_SAVINGS_BALANCE=3000
USER4_CREDIT_CARD_BALANCE=500
USER4_ROUTING_NUMBER=987654321
USER4_ACCOUNT_NUMBER=123456789

USER7_PASSWORD=password3
USER7_CHECKING_BALANCE=2000
USER7_SAVINGS_BALANCE=2500
USER7_CREDIT_CARD_BALANCE=100
USER7_ROUTING_NUMBER=192837465
USER7_ACCOUNT_NUMBER=564738291

USER3_PASSWORD=password4
USER3_CHECKING_BALANCE=3000
USER3_SAVINGS_BALANCE=4500
USER3_CREDIT_CARD_BALANCE=800
USER3_ROUTING_NUMBER=564738291
USER3_ACCOUNT_NUMBER=192837465
```

Methodology: Password Hashing Using Bcrypt

Description of Methods and Techniques Used:

- **Login Authentication:**
  - Passwords are securely hashed using bcrypt to ensure the protection of user credentials.
  - Environment variables (via .env files) store sensitive information like passwords and account balances, which are fetched and hashed at runtime.

```
USER1_PASSWORD=password1
USER1_CHECKING_BALANCE=1000
USER1_SAVINGS_BALANCE=5000
USER1_CREDIT_CARD_BALANCE=200
USER1_ROUTING_NUMBER=123456789
USER1_ACCOUNT_NUMBER=987654321
```

```
hashed_credentials = {
    "user1": bcrypt.hashpw(os.getenv("USER1_PASSWORD").encode(), bcrypt.gensalt()) if os.getenv("USER1_PASSWORD") else None,
    "user4": bcrypt.hashpw(os.getenv("USER4_PASSWORD").encode(), bcrypt.gensalt()) if os.getenv("USER4_PASSWORD") else None,
    "user7": bcrypt.hashpw(os.getenv("USER7_PASSWORD").encode(), bcrypt.gensalt()) if os.getenv("USER7_PASSWORD") else None,
    "user3": bcrypt.hashpw(os.getenv("USER3_PASSWORD").encode(), bcrypt.gensalt()) if os.getenv("USER3_PASSWORD") else None,
    "user6": bcrypt.hashpw(os.getenv("USER6_PASSWORD").encode(), bcrypt.gensalt()) if os.getenv("USER6_PASSWORD") else None
}
```

Methodology

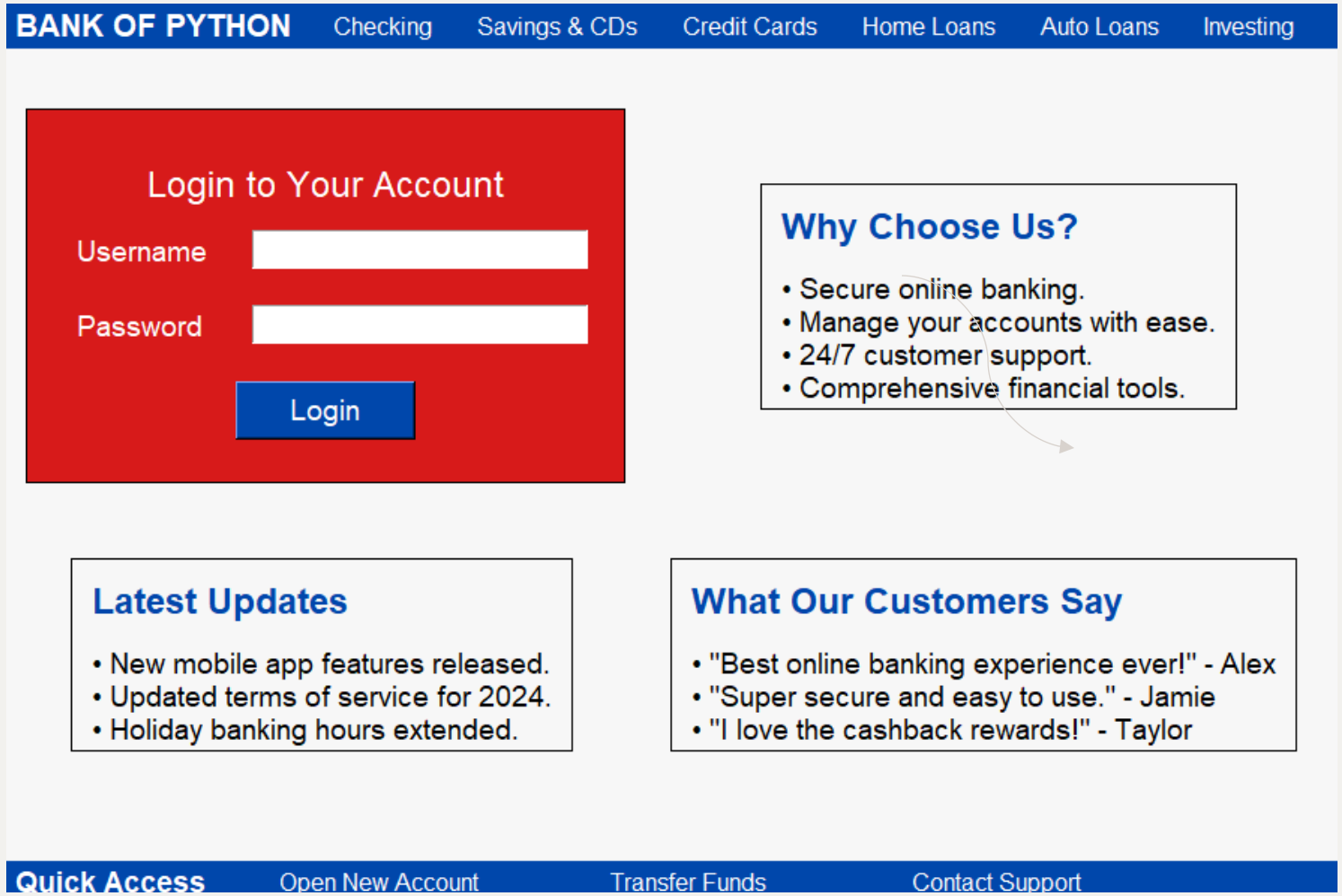
Data Structures Utilized:

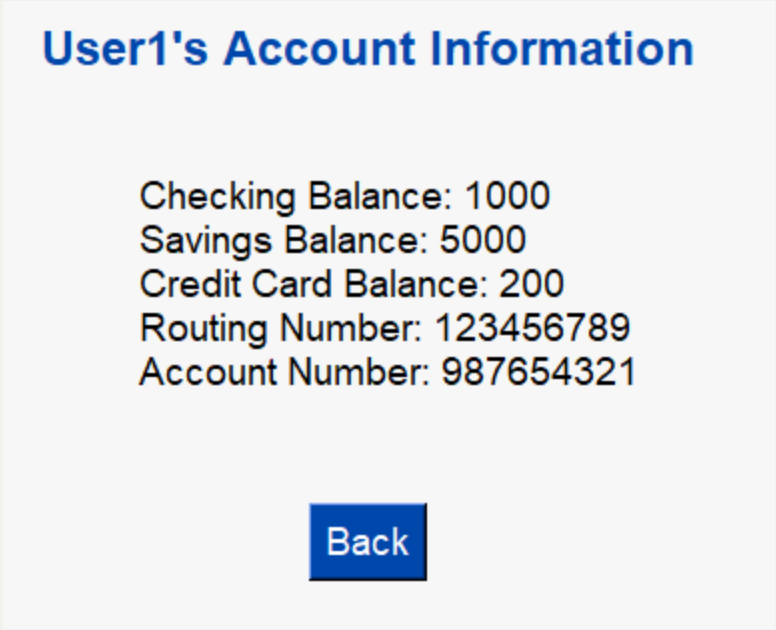
- **Dictionaries:**
  - user_accounts: Stores account balances and user-specific data (checking, savings, credit card balances).
  - Os.getenv stores environmental variables on a separate file on the computer

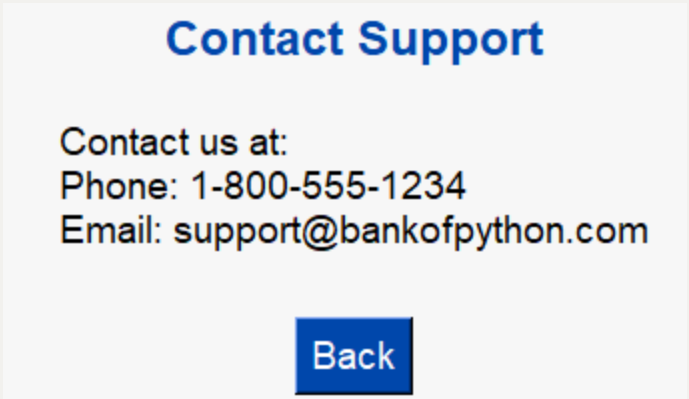
```
user_accounts = {
    "user1": {
        "checking_balance": os.getenv("USER1_CHECKING_BALANCE"),
        "savings_balance": os.getenv("USER1_SAVINGS_BALANCE"),
        "credit_card_balance": os.getenv("USER1_CREDIT_CARD_BALANCE"),
        "routing_number": os.getenv("USER1_ROUTING_NUMBER"),
        "account_number": os.getenv("USER1_ACCOUNT_NUMBER")
    },
    "user4": {
        "checking_balance": os.getenv("USER4_CHECKING_BALANCE"),
        "savings_balance": os.getenv("USER4_SAVINGS_BALANCE"),
        "credit_card_balance": os.getenv("USER4_CREDIT_CARD_BALANCE"),
        "routing_number": os.getenv("USER4_ROUTING_NUMBER"),
        "account_number": os.getenv("USER4_ACCOUNT_NUMBER")
    },
    "user7": {
        "checking_balance": os.getenv("USER7_CHECKING_BALANCE"),
        "savings_balance": os.getenv("USER7_SAVINGS_BALANCE"),
        "credit_card_balance": os.getenv("USER7_CREDIT_CARD_BALANCE"),
        "routing_number": os.getenv("USER7_ROUTING_NUMBER"),
        "account_number": os.getenv("USER7_ACCOUNT_NUMBER")
    },
```

Methodology

FinalBankTerminal.py







Analysis and Results (Interpretation of Results):

Login System:

- The login system securely verifies user passwords using bcrypt, ensuring that only authorized users can access account details.

Account Data Display:

- Upon login, users can view their checking, savings, and credit card balances. The data is displayed correctly, but the system is currently read-only.

User Interface:

- The interface is user-friendly, with clear navigation and helpful error messages. However, it lacks features for account management or transactions.

Error Handling:

- Errors are displayed for invalid logins, but there is no limit on failed login attempts, which could be a security concern.

Limitations:

- The app is basic and does not support transactions or account modifications. The use of environment variables for data storage is not scalable for larger applications.

Analysis and Results (Timetable)

Time Management:

Week	Hours Spent
Week 1	5
Week 2	5
Week 3	6
Week 4	6
Total Weeks: 4	22 Hours

Analysis and Results (Key Findings)

Finding	Description
Secure Login Method	Passwords are secure by using hashed bcrypt.
User Interface Kept Simple and Flexible	Tkinter provides the framework for the GUI while keeping all elements flexible.
Static Data for Testing	The program uses hardcoded data from the environment variables.
Limited Features	The program lacks in currency transactions and real-time updates.

Discussion

Implications of Findings:

- **Security Implications:**
  - Any real-world application must have user credentials that are securely stored and validated, which is ensured by using bcrypt.
- **Usability:**
  - Users may simply go through their account data and other financial services thanks to the tkinter GUI.

Project Limitations:

- **Limitations:**
  - Only static user data (from environment variables) is supported by the application.
  - No real-time data updates or transaction processing, for example, indicate limited interactivity.
  - More sophisticated features, such different user roles or more thorough account management, could improve the straightforward design but we were very limited with the Python resources we had.

Conclusion

Conclusions from the Project:

- The Bank Terminal GUI application effectively illustrates how to use bcrypt and tkinter to protect passwords, display account balances, and enable secure login. Despite being restricted to simple features, the project is an excellent foundation for a more intricate financial system.

Future Enhancements:

- Put dynamic account changes into practice by retrieving real-time data from an API or database.
- Include transactional functionality like transfers, withdrawals, and deposits.
- Enhance the user interface with more eye-catching visuals and a more user-friendly layout.
- Include error handling for unforeseen circumstances such as false account information or network outages.

References

- **bcrypt**. *Python Software Foundation*, 2019, https://pypi.org/project/bcrypt/.
- **"os — Miscellaneous operating system interfaces"**. *Python Software Foundation*, 2020, https://docs.python.org/3/library/os.html#os.environ.
- **"python-dotenv"**. *Python Software Foundation*, 2020, https://pypi.org/project/python-dotenv/.