An Introductory Python Bank Terminal

Graham Smith, Jacob Sullivan, and Yamin Mahmood

Northeastern University

EECE 2140. Computing Fundamentals for Engineers Fall 2024

11/22/2024

Abstract

This technical report presents the development, implementation, and evaluation of a Python-made Bank Terminal. The goal of this project was to create a functional bank terminal. It centered around 4 main parts: A complete and user-friendly GUI, a secure way of storing, editing, maintaining information, an advanced way of encrypting data, and a fool proof system with different error handling methods. We looked at other frameworks like Django and Flask, as well as visited public information on different modules GITHUBs. With this information, we were able to successfully complete our GUI. When opened, critical information is securely loaded into the program. A variety of different UI elements load which as well gives the appearance of a bank terminal. Altogether, our project effectively completes these four parts and the overall requirements.

The success of this project should be evaluated based on a couple critical factors. Firstly, the project's ability to function without errors is essential. If the project does not work flawlessly, then no other points matter. Secondly, the project should be judged on its ability to effectively mask critical information. Thirdly, the project should be judged on appearance. Though it is an entry level project, it is important that the GUI component is appealing and functional. These main points, combined with other small details, underscore the success of our bank terminal.

Contents

Chapter 1

Introduction

1.1 Background

The assignment that inspired this project had a couple of components to consider when picking out a project. Firstly, it had to be something that required skills not previously known. It was important that we challenge ourselves, and in that we did. There was also the requirement for our project to have some scalability and maintainability in addition to the need for completeness. After researching different ideas from task managers to our eventual project, a bank terminal, we decided what we were going to do.

In researching our project, we broke it down into a couple of components that we saw as essential parts of a bank terminal. We saw that all bank terminals had the following: A user-friendly interface, a login component, a

help component, different ways of presenting essential information like contact numbers and more, as well to view account information once logged in. Behind the scenes, we noted banks data structures and security measures as other things that we needed to consider and dive into further.

1.2 Problem Statement

The goal of our project is to develop an interactive bank terminal using python, its built-in modules, and other advanced tools. It is designed to provide users with a seamless, secure, and efficient banking experience. The target audience includes any individual needing access to their bank accounts but especially those who prefer a straightforward and user-friendly terminal interface. This project is particularly focused on offering advanced security, reliability, and an overall intuitive experience.

The project addresses the critical need for a secure and encrypted login system that ensures the users' information like account numbers, balances, and passwords are protected. To achieve this, we incorporate the bcrypt library for encrypting account data, pythons os module to navigate and find local files, and dotenv to read and manage information stored in different places. Additionally, tkinter was used to create an accessible and appealing graphical interface that looks like a normal bank.

While there are no restraints on the cost, size, or real scope of the project, the project must be completed in a strict 4-week timeline. The system must be well coded and use object-oriented programming and different functions to ensure clear organization and modularity. It should be free of bugs, capable of quickly being updated and expanded upon, and able to work on both Windows and Mac machines.

Extensive research was done on different existing solutions both in the bank terminal and login screen areas. We deeply investigated DJANGO and FLASK as well as the other python modules and libraries. This foundational approach allowed us to expand on our instruction in class while focusing on developing our own skills in a way that interested us.

The bank terminal is simple enough for all users to quickly understand and be able to take advantage of its features. It has solid features for handling essential banking operations in a secure manner and incorporates robust error handling and data storage methods. By addressing these needs, we aim to create a secure, accessible and impactful solution to our assignment.

1.3 Objectives

The objectives of this project were the following: create a secure, reliable, user-friendly, and robust bank terminal that replicated real world banking functionalities. This system must have an encrypted login screen that hides critical data. It also should utilize different python libraires for data management and more. The interface will provide a visually appealing a simple user experience. The code will be structured using Object Oriented Programming and separate functions to ensure it is clear and able to be eas-ily scaled and updated.

1.4 Scope

The development of this project began with research and the creation of a problem definition. We made sure to understand the requirements of both assignments and the requirements of a secure and interactive banking terminal. The final deliverable is fully functional prototype software that demonstrates all these primary objectives and even has features that go above and beyond.

The initial stages of the engineering design process were dedicated to studying the different libraries and modules, namely bcrypt, os, dotenv, and tkinter. We integrated this research into the plan we created around this project. The scope of this report includes a detailed discussion of the research, the rationale behind different decisions, and an analysis of the system's functionality and final design.

However, it will not dive to deep into specific parts of the libraries used or any backend work that is widely available online to be viewed.

This document will provide an overview of the process, from the idea of the banking terminal to the final product, showcasing the practical application of the different lessons taught over the course of the semester thus far.

Chapter 2

Technical Approaches and Code UML

2.1 Development Environment

This project was developed in VScode, or Visual Studio Code. It was an environment that we were comfortable with from previous work with different languages. We opened our own window and directory using the built-in git controls. Overall, the code was made in Python version 3.12.4. It has many

libraries already built into it, but we did have to add some others. Key libraries included bcrypt 4.2.0 which was used for encrypting sensitive account information. DotENV version 0.4.27 was used for securely managing environmental variables, and then tkinter was used for building the GUI. VSCode has built in Gitt integration that allowed us to easily manage our version control from the same applications. We used commands such as git pull, and git push to download or upload our work to our group GitHub.

The terminal in VSCode had an important role in debugging and testing our code. Errors and exceptions were displayed in the terminal where we could go back and adjust stuff later. To maintain a structed workflow, the project begain with a file called Banktester.py which was used as an exploratory file where new features were trialed and updated. There was then BankRunner.py which was the main file with only working code. Only edits were made to BankTester.py. Also, VSCode allowed us to have our .env file present in the same window which allowed us to edit it without having to switch screens. Once done, the code was migrated to a file called FinalBankTerminal.py which would be the file sent out to others who wish to use it.

2.2 Data Collection and Preparation

Though there is important data present in this project, there was none that needed to be individually collected or generated from a dataset. The only data we had was made up account numbers, routing numbers, balances, usernames

and passwords. There was important things done to the data once present in our system, but there was no particular procedure to discuss when collecting data.

2.3 Implementation Details

The coding process began with creating the GUI. We ignored more specific features first like the encryption until we could get the GUI down. First, we started by creating the window. We made it a random 600x300 size, which we knew we could change later. From that, we adjusted the color from the default grey color of tkinter, to the white background that you see now. Other rectangle shapes were created for the header, footer, and login box. At this point, we could start to see how it was coming out. We then created an outline of the different prompts you see in 3 of the 4 quadrants. After we had finished working with the shapes of tkinter, we dove into adding text. Each time we had made a shape, we noted its position. This is because tkinter requires you to place each thing based on the coordinate system. Since we had recorded where each of the rectangles were, we were able to place the text in roughly the right area before adding some padding. For some of the text, we needed to make it a button, so we added that feature too. With all of this done, we developed our home landing page complete with text input, buttons, and other features.

Once we had the landing page, we moved to create each additional page that the buttons would lead to. It was a basic copy and paste at this point. We created one page and connected it to one button before moving onto the next. For each page,

we had to change the information on it from support number to credit card info etc. At this point in time, we had developed the entire UI. It was able to be used for the correct purpose and made it look like something was happening though there was no backend to support it.

The next step was adding the data management system and DOTenv library. This was relatively simple. The first thing we did was import the module and then in the hashed credential dictionary, add a line of code that read the information. Additionally, where we had added placeholder for account details, we replaced it with the line that read the appropriate information and connected it to the specific variable. At this point, we had the UI and reading information from a couple lines in a hidden ENV file.

The next step of our code was to encrypt the information that we had just read from the correct file. We did this by adding parentheses around the lines we had just added, and then added the encryption phrase just before it. We did this for the user passwords, account information, and more. At this point we also added 4 more users, bringing the total number to 5. This mainly just means that we duplicated some lines changing stuff like “user1” to “user3” as we went. This expansion was crucial to get right as it would be a massive security breach to read the wrong users’ information.

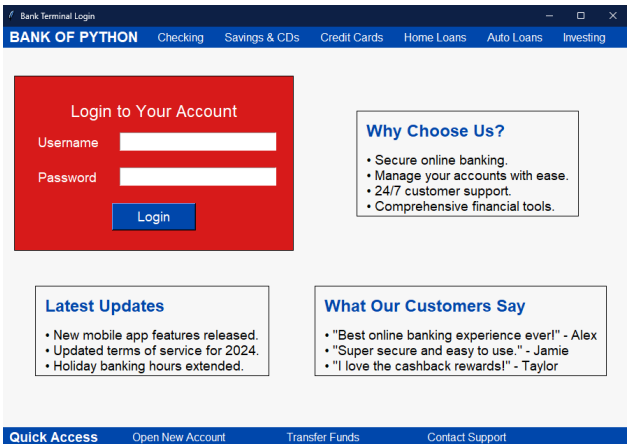
After reading and encrypting the data, we were mostly done. We had successfully made the code compare the entered username and password to real encrypted information and added the appropriate navigational tools to our UI. A more in-depth report on this can be found in Chapter 3, Project Demonstration.

Chapter 3

Project Demonstration

3.1 Screenshots and Code Snippets

As soon as the program is run the landing page of our fake bank appears. It shows cool facts, has links to different sources, and overall simulates what would be found on a real bank. It looks like the following.



As previously described, the landing page has all the different buttons, input fields, and other information. You can see on top the header and the footer which will be talked more about below. Then there are the login elements, and other text boxes. The format for most of the components is only a single line and roughly follows the text box below.

```
News_frame = tk.frame(root,bg="f7f7f7", width = 400, height = 300, relief = "solid", borderwidth = 1)
News_frame.place(x=450,y=230)
```

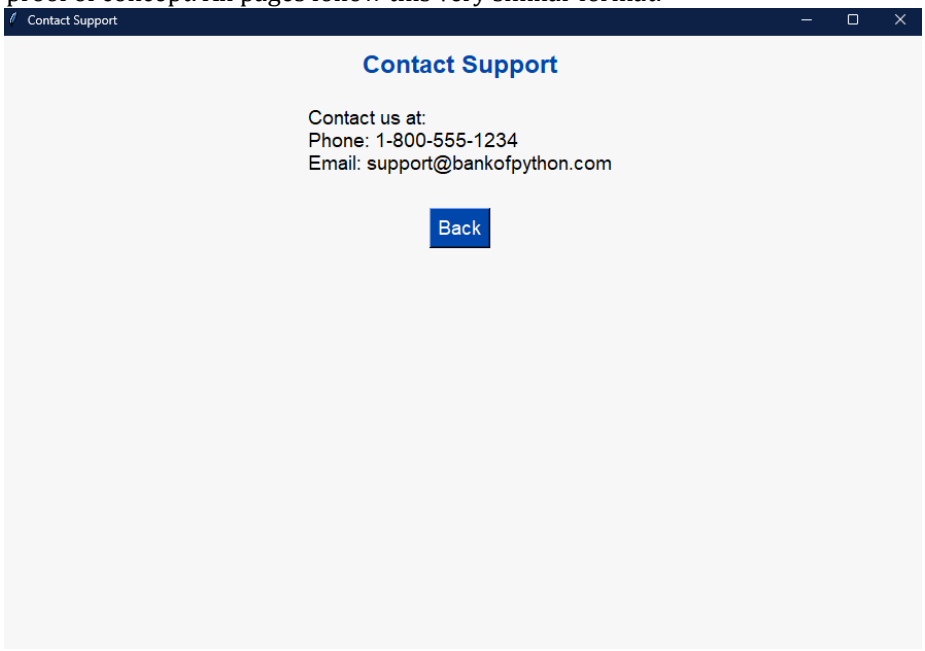
All of what you see here comes from the long block of code on the next two pages. It covers all formatting, design, layout, but none of the backend features. It is purely visual.

```
import tkinter as tk

# Function to show informational pages
def show_info_page(title, content):
    for widget in root.winfo_children():
        widget.destroy()
    root.title(title)
    root.configure(bg="#f7f7f7")

    title_label = tk.Label(root, text=title, font=("Arial", 18, "bold"), bg="#f7f7f7", fg="#0047ab")
    title_label.pack(pady=10)

    content_label = tk.Label(root, text=content, font=("Arial", 14), bg="#f7f7f7", fg="black", justify="left")
    content_label.pack(padx=20, pady=10)

    back_button = tk.Button(root, text="Back", font=("Arial", 14), bg="#0047ab", fg="white", command=show_main_page)
    back_button.pack(pady=20)

# Function to display the main page
def show_main_page():
    for widget in root.winfo_children():
        widget.destroy()

    root.title("Bank Terminal Login")
    root.geometry("900x600")
    root.configure(bg="#f7f7f7")

    # Header Section
    header_frame = tk.Frame(root, bg="#0047ab", height=50)
    header_frame.pack(fill=tk.X, side=tk.TOP)

    logo_label = tk.Label(header_frame, text="BANK OF PYTHON", font=("Arial", 16, "bold"), bg="#0047ab", fg="white")
    logo_label.pack(side=tk.LEFT, padx=10)

    nav_links = [
        ("Checking", "Information about Checking Accounts."),
        ("Savings & CDs", "Information about Savings Accounts and CDs."),
        ("Credit Cards", "Information about Credit Cards."),
        ("Home Loans", "Information about Home Loans."),
        ("Auto Loans", "Information about Auto Loans."),
        ("Investing", "Information about Investing Opportunities.")
    ]
    for link, content in nav_links:
```

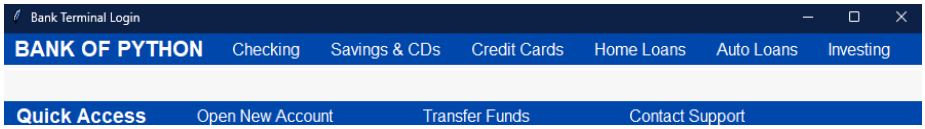
```
# Login Section
    login_frame = tk.Frame(root, bg="#d71a1a", width=400, height=250, relief="solid", borderwidth=1)
    login_frame.place(x=20, y=70)

    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=tk.CENTER, y=30)

    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)

    password_label = tk.Label(login_frame, text="Password", font=("Arial", 14), bg="#d71a1a", fg="white")
    password_label.place(x=30, y=130)
    password_entry = tk.Entry(login_frame, show="*", font=("Arial", 14), width=20)
    password_entry.place(x=150, y=130)

    login_button = tk.Button(login_frame, text="Login", font=("Arial", 14), width=10, bg="#0047ab", fg="white", command=None)
    login_button.place(relx=0.5, y=200, anchor=tk.CENTER)

    # Benefits Section
    benefits_frame = tk.Frame(root, bg="#f7f7f7", width=400, height=200, relief="solid", borderwidth=1)
    benefits_frame.place(x=510, y=120)

    benefits_label = tk.Label(benefits_frame, text="Why Choose Us?", font=("Arial", 18, "bold"), bg="#f7f7f7", fg="#0047ab")
    benefits_label.pack(anchor="w", padx=10, pady=10)

    benefits_text = """• Secure online banking.
• Manage your accounts with ease.
• 24/7 customer support.
• Comprehensive financial tools."""
    benefits_content = tk.Label(benefits_frame, text=benefits_text, font=("Arial", 14), bg="#f7f7f7", fg="black", justify="left")
    benefits_content.pack(anchor="w", padx=10)

    # Latest Updates Section
    news_frame = tk.Frame(root, bg="#f7f7f7", width=400, height=200, relief="solid", borderwidth=1)
    news_frame.place(x=50, y=370)

    news_label = tk.Label(news_frame, text="Latest Updates", font=("Arial", 18, "bold"), bg="#f7f7f7", fg="#0047ab")
    news_label.pack(anchor="w", padx=10, pady=10)

    news_text = """• New mobile app features released.
• Updated terms of service for 2024.
```

```
    news_text = """• New mobile app features released.
• Updated terms of service for 2024.
• Holiday banking hours extended."""
    news_content = tk.Label(news_frame, text=news_text, font=("Arial", 14), bg="#f7f7f7", fg="black", justify="left")
    news_content.pack(anchor="w", padx=10)

    # Testimonials Section
    testimonials_frame = tk.Frame(root, bg="#f7f7f7", width=400, height=200, relief="solid", borderwidth=1)
    testimonials_frame.place(x=450, y=370)

    testimonials_label = tk.Label(testimonials_frame, text="What Our Customers Say", font=("Arial", 18, "bold"), bg="#f7f7f7",
fg="#0047ab")
    testimonials_label.pack(anchor="w", padx=10, pady=10)

    testimonial_text = """• "Best online banking experience ever!" - Alex
• "Super secure and easy to use." - Jamie
• "I love the cashback rewards!" - Taylor"""
    testimonials_content = tk.Label(testimonials_frame, text=testimonial_text, font=("Arial", 14), bg="#f7f7f7", fg="black", justify="left")
    testimonials_content.pack(anchor="w", padx=10)

    # Footer Section
    footer_frame = tk.Frame(root, bg="#0047ab", height=50)
    footer_frame.pack(fill=tk.X, side=tk.BOTTOM)

    quick_access_label = tk.Label(footer_frame, text="Quick Access", font=("Arial", 14, "bold"), bg="#0047ab", fg="white")
    quick_access_label.pack(side=tk.LEFT, padx=10)

    buttons = [
        ("Open New Account", "Information on how to open a new account."),
        ("Transfer Funds", "Instructions for transferring funds between accounts."),
        ("Contact Support", "Contact us at:\nPhone: 1-800-555-1234\nEmail: support@bankofpython.com")
    ]
    for text, content in buttons:
        button = tk.Button(footer_frame, text=text, font=("Arial", 12), bg="#0047ab", fg="white", command=lambda t=text, c=content:
show_info_page(t, c), width=20, borderwidth=0)
        button.pack(side=tk.LEFT, padx=10)

# Initialize the main application window
root = tk.Tk()
show_main_page()
root.mainloop()
```

Our code is only 230 lines overall and what is above accounts for over half of it (128 lines). There are a couple of features that you can not see from the screenshot, but may notice in the code. Each of the items on the header and footer are buttons which when pressed lead to another page. An example of one of those pages is below. It has basic support information but serves more as proof of concept. All pages follow this very similar format.



The way that you navigate to this as stated is through the footer. The header and footer look like the following and are only made of a couple lines of code.



While simple looking, there are actually many lines of code that run behind each of these objects. For the header for example, the following code is what runs it.

```
# Header Section
header_frame = tk.Frame(root, bg="#0047ab", height=50)
header_frame.pack(fill=tk.X, side=tk.TOP)

# Bank Logo Placeholder
logo_label = tk.Label(header_frame, text="BANK OF PYTHON", font=("Arial", 16, "bold"), bg="#0047ab", fg="white")
logo_label.pack(side=tk.LEFT, padx=10)

# Navigation Links
nav_links = [
    ("Checking", "Information about Checking Accounts."),
    ("Savings & CDs", "Information about Savings Accounts and CDs."),
    ("Credit Cards", "Information about Credit Cards."),
    ("Home Loans", "Information about Home Loans."),
    ("Auto Loans", "Information about Auto Loans."),
    ("Investing", "Information about Investing Opportunities.")
]

# Create navigation buttons
for link, content in nav_links:
    nav_button = tk.Button(header_frame, text=link, font=("Arial", 12), bg="#0047ab", fg="white", borderwidth=0,
                           command=lambda l=link, c=content: show_info_page(l, c))
```

After the GUI had been established, there is a lot of backend work that occurs. For example, the reading and encryption of information. There are two sections where this occurs. The first is when reading the password. The second is the reading of account information. The code that runs this part of the project is below.

```
#hash plain text passwords from environment variables
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
}
```

This snippet also shows how information is read from the env file. The env file is formatted in the following way.

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

As you can see, it is very transparent and easy to read and update. This is good for developers, but it does lack the security needed if it were to be accessed. That is why there is the encryption in the previous step.

The following is the entire code:

```
import tkinter as tk
import bcrypt
from dotenv import load_dotenv
import os

#environment variables from .env file
load_dotenv()

#hash plain text passwords from environment variables
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
}

#user account information from environment variables
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")
```

```
  "user3": {
        "checking_balance": os.getenv("USER3_CHECKING_BALANCE"),
        "savings_balance": os.getenv("USER3_SAVINGS_BALANCE"),
        "credit_card_balance": os.getenv("USER3_CREDIT_CARD_BALANCE"),
        "routing_number": os.getenv("USER3_ROUTING_NUMBER"),
        "account_number": os.getenv("USER3_ACCOUNT_NUMBER")
    },
    "user6": {
        "checking_balance": os.getenv("USER6_CHECKING_BALANCE"),
        "savings_balance": os.getenv("USER6_SAVINGS_BALANCE"),
        "credit_card_balance": os.getenv("USER6_CREDIT_CARD_BALANCE"),
        "routing_number": os.getenv("USER6_ROUTING_NUMBER"),
        "account_number": os.getenv("USER6_ACCOUNT_NUMBER")
    }
}

def show_info_page(title, content):
    # Clear the existing content in the root window
    for widget in root.winfo_children():
        widget.destroy()

    # Create the info page layout
    root.title(title)
    root.configure(bg="#f7f7f7")

    # Display title
    title_label = tk.Label(root, text=title, font=("Arial", 18, "bold"), bg="#f7f7f7", fg="#0047ab")
    title_label.pack(pady=10)

    # Display content
    content_label = tk.Label(root, text=content, font=("Arial", 14), bg="#f7f7f7", fg="black", justify="left")
    content_label.pack(padx=20, pady=10)

    # Add back button
    back_button = tk.Button(root, text="Back", font=("Arial", 14), bg="#0047ab", fg="white", command=show_main_page)
    back_button.pack(pady=20)

def show_account_page(username):
    # Fetch account information for the logged-in user
    account_info = user_accounts.get(username, {})

    # Prepare account information content
    content = f"""
```

```
    """
    show_info_page(f"{username.capitalize()}'s Account Information", content)

def show_main_page():
    global username_entry, password_entry, error_label
    # Clear the existing content in the root window
    for widget in root.winfo_children():
        widget.destroy()

    # Rebuild the main page layout
    root.title("Bank Terminal Login")
    root.geometry("900x600")
    root.configure(bg="#f7f7f7")

    # Create the top header bar
    header_frame = tk.Frame(root, bg="#0047ab", height=50)
    header_frame.pack(fill=tk.X, side=tk.TOP)

    # Add a bank logo placeholder
    logo_label = tk.Label(header_frame, text="BANK OF PYTHON", font=("Arial", 16, "bold"), bg="#0047ab", fg="white")
    logo_label.pack(side=tk.LEFT, padx=10)

    # Add navigation links
    nav_links = [
        ("Checking", "Information about Checking Accounts."),
        ("Savings & CDs", "Information about Savings Accounts and CDs."),
        ("Credit Cards", "Information about Credit Cards."),
        ("Home Loans", "Information about Home Loans."),
        ("Auto Loans", "Information about Auto Loans."),
        ("Investing", "Information about Investing Opportunities."),
        ("Security", "Information about Security and Fraud Protection.")
    ]

    for link, content in nav_links:
        nav_button = tk.Button(header_frame, text=link, font=("Arial", 12), bg="#0047ab", fg="white", borderwidth=0,
                               command=lambda l=link, c=content: show_info_page(l, c))
        nav_button.pack(side=tk.LEFT, padx=10)

    # 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")
```

```
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)

    password_label = tk.Label(login_frame, text="Password", font=("Arial", 14), bg="#d71a1a", fg="white")
    password_label.place(x=30, y=130)
    password_entry = tk.Entry(login_frame, show="*", font=("Arial", 14), width=20)
    password_entry.place(x=150, y=130)

    login_button = tk.Button(login_frame, text="Login", font=("Arial", 14), width=10, bg="#0047ab", fg="white", command=login)
    login_button.place(relx=0.5, y=200, anchor=tk.CENTER)

    # Top Right: Why Choose Us?
    benefits_frame = tk.Frame(root, bg="#f7f7f7", width=400, height=200, relief="solid", borderwidth=1)
    benefits_frame.place(x=510, y=120)

    benefits_label = tk.Label(benefits_frame, text="Why Choose Us?", font=("Arial", 18, "bold"), bg="#f7f7f7", fg="#0047ab")
    benefits_label.pack(anchor="w", padx=10, pady=10)

    benefits_text = """• Secure online banking.
• Manage your accounts with ease.
• 24/7 customer support.
• Comprehensive financial tools."""
    benefits_content = tk.Label(benefits_frame, text=benefits_text, font=("Arial", 14), bg="#f7f7f7", fg="black", justify="left")
    benefits_content.pack(anchor="w", padx=10)

    # Bottom Left: Latest Updates
    news_frame = tk.Frame(root, bg="#f7f7f7", width=400, height=200, relief="solid", borderwidth=1)
    news_frame.place(x=50, y=370)

    news_label = tk.Label(news_frame, text="Latest Updates", font=("Arial", 18, "bold"), bg="#f7f7f7", fg="#0047ab")
    news_label.pack(anchor="w", padx=10, pady=10)

    news_text = """• New mobile app features released.
• Updated terms of service for 2024.
• Holiday banking hours extended."""
```

```
    news_content = tk.Label(news_frame, text=news_text, font=("Arial", 14), bg="#f7f7f7", fg="black", justify="left")
    news_content.pack(anchor="w", padx=10)

    # Bottom Right: Testimonials
    testimonials_frame = tk.Frame(root, bg="#f7f7f7", width=400, height=200, relief="solid", borderwidth=1)
    testimonials_frame.place(x=450, y=370)

    testimonials_label = tk.Label(testimonials_frame, text="What Our Customers Say", font=("Arial", 18, "bold"), bg="#f7f7f7",
fg="#0047ab")
    testimonials_label.pack(anchor="w", padx=10, pady=10)

    testimonial_text = """• "Best online banking experience ever!" - Alex
• "Super secure and easy to use." - Jamie
• "I love the cashback rewards!" - Taylor"""
    testimonials_content = tk.Label(testimonials_frame, text=testimonial_text, font=("Arial", 14), bg="#f7f7f7", fg="black", justify="left")
    testimonials_content.pack(anchor="w", padx=10)

    # Footer Bar: Quick Access
    footer_frame = tk.Frame(root, bg="#0047ab", height=50)
    footer_frame.pack(fill=tk.X, side=tk.BOTTOM)

    quick_access_label = tk.Label(footer_frame, text="Quick Access", font=("Arial", 14, "bold"), bg="#0047ab", fg="white")
    quick_access_label.pack(side=tk.LEFT, padx=10)

    buttons = [
        ("Open New Account", "Information on how to open a new account."),
        ("Transfer Funds", "Instructions for transferring funds between accounts."),
        ("Contact Support", "Contact us at:\nPhone: 1-800-555-1234\nEmail: support@bankofpython.com")
    ]

    for text, content in buttons:
        button = tk.Button(footer_frame, text=text, font=("Arial", 12), bg="#0047ab", fg="white", command=lambda t=text, c=content:
show_info_page(t, c), width=20, borderwidth=0)
        button.pack(side=tk.LEFT, padx=10)
```

```
def login():
    username = username_entry.get()
    password = password_entry.get()

    # Clear previous error message
    error_label.config(text="")

    if username in hashed_credentials and hashed_credentials[username] is not None:
        if bcrypt.checkpw(password.encode(), hashed_credentials[username]):
            print("Password Correct")
            show_account_page(username)
        else:
            error_label.config(fg="white", text="Login Failed: Invalid username or password.")
            password_entry.delete(0, tk.END)
    else:
        error_label.config(fg="white", text="Login Failed: Invalid username or password.")
        password_entry.delete(0, tk.END)

# Initialize the main page
```

This is the entire code. Additionally, our code, documentation, and more can be found on our GITHUB.

Chapter 4

Discussion and Future Work

Upon presenting our work, there were no major complaints or apparent security flaws. The system provided a secure login mechanism that had encrypted usernames and passwords as promised, a user-friendly interface, and well organized and easily understood code. This demonstrates the feasibility of using Pythons libraries to make a user secured interface, whether it is for a bank terminal or some other project that needs introductory security. The implications of this project are significant as it serves as an easily replicable example of a secure banking terminal.

Despite its success there were some limitations to our project. The current data storage method which relies on a single file with different environmental variables is not scalable for larger datasets like what would be required for a full banking application. Additionally, if the env file is mishandled, uploaded to the wrong place, or deleted, all that data that would be encrypted in the actual

python code could be shared to the wrong people. The tkinter library, while good for introductory projects asking for simple shapes, is not good enough to make a complete application.

Future improvements would address those issues. We could transition to a more scalable and secure data solution like a full database. This would allow us to handle more data and get rid of the issue with the .env file that was mentioned. Additional safeguards could be added to if the env file system was kept, but a database is likely the way to go. Upgrading to a different GUI tool would also help to make it more appealing to the eye and allow for more complex features like a chat support bot for instance. PyQT is something that has been explored as a replacement for this. By implementing these enhancements, the project could evolve into a more comprehensive and scalable solution, paving the way for broader applications with more advanced features.

Chapter 5

Conclusion

The banking terminal successfully achieved all its objectives. It delivered a secure, user friendly, and functional banking terminal complete with some of the most apparent features. Key findings showed the effectiveness of Python as a multifunctional tool with near infinite expandability. We created a system with Bcrypt, DOTenv, and Tkinter that all together completed the project and allowed us to expand on our Python knowledge through an individual project. The bank's significance lies in its ability to successfully secure and encrypt information using straightforward tools and methodologies. It provides a solid foundation for other applications that seek to have a secure interface that can also host sensitive data. The successful execution of the project underscored the potential for Python-based solutions that address different real-world demands.

Bibliography

[1] Python Software Foundation. *python-dotenv.* https://pypi.org/project/python-dotenv/. Accessed November 22, 2024.

[2] W3Schools. *Python Tutorial.* https://www.w3schools.com/python/. Accessed November 22, 2024.

[3] Python Software Foundation. *bcrypt.* https://pypi.org/project/bcrypt/. Accessed November 22, 2024.

[4] Stack Overflow. *Hashing a password with bcrypt.* https://stackoverflow.com/questions/63387149/hashing-a-password-with-bcrypt. Accessed November 22, 2024.

[5] Python Software Foundation. *os — Miscellaneous operating system interfaces.* https://docs.python.org/3/library/os.html. Accessed November 22, 2024.

[6] Stack Overflow. *dotenv file is not loading environment variables.* https://stackoverflow.com/questions/42335016/dotenv-file-is-not-loading-environment-variables. Accessed November 22, 2024.

[7] Python Software Foundation. *tkinter — Python interface to Tcl/Tk.* https://docs.python.org/3/library/tkinter.html. Accessed November 22, 2024.

[8] GeeksforGeeks. *Python Tkinter Tutorial.* https://www.geeksforgeeks.org/python-tkinter-tutorial/. Accessed November 22, 2024.