CMP9134M Advanced Software Engineering

Group Assingment

SAAD SAMEER - 28709550@STUDENTS.LINCOLN.AC.UK
GULRAIZ ASMAT – 28733504@STUDENTS.LINCOLN.AC.UK
MUHAMMAD WASIF – 28726627@STUDENTS.LINCOLN.AC.UK
VARSHA VARSHA - 28926634@STUDENTS.LINCOLN.AC.UK

Table of Contents

Objective:

The objective of this assignment is to design and implement a comprehensive banking software system that integrates a user-friendly interface with a highly secure and robust backend. This system will be developed using Agile methodologies, ensuring iterative progress through collaborative and adaptive planning, continuous improvement, and flexible responses to changes. The primary aim is to facilitate a variety of banking operations, including account management, financial transactions, security implementations, and account services, with an emphasis on secure data handling, compliance with financial regulations, and providing an optimal user experience.

This software system will cater to both personal and business banking needs, allowing users to create and edit accounts, deposit and withdraw funds, transfer money between accounts, and view detailed account information. it will offer functionality to request and manage banking services such as checkbooks and debit/credit cards, as well as setup and manage recurring payments. Overall, the development of this banking application seeks not only to meet the technical requirements of modern banking systems but also to enhance customer satisfaction through efficient, reliable, and secure banking solutions.

Methodology Employed:

Our team adopted the Agile methodology which helped in iterative development of this project. The key aspects to ensure comprehensive development and keen functionality are following:

Agile Methodology

Utilized Agile principles for iterative development, allowing for flexible responses to change and continuous improvement throughout the project lifecycle.

User Stories

Define clear and concise narratives that describe the functionality from the perspective of the end user, aiding in understanding specific needs and functionalities.

Interactive Design

Focus on creating intuitive and engaging user interfaces that enhance user experience and facilitate easy navigation through various banking operations.

Pair Programming

Implement collaborative coding sessions where two programmers work together at one workstation, simultaneously improving code quality and team knowledge sharing.

Testing

Comprehensive testing phases involving unit tests, integration tests, and user acceptance tests to ensure the application's reliability, security, and performance.

Tools:

We have used the following tools in the development process and documentation of the project:

- Python Language:
- GitHub: code was managed and version controlled using the GitHub.
- Stremlit: Used for the development of the robust and user-friendly web interface.
- Jira:
- Draw.io:

Roles and Responsibilities:

This section outlines the specific roles and responsibilities assigned to each team member, ensuring clarity and accountability throughout the development of the banking software system. Saad and Gulraiz, as Backend Developers, were tasked with creating and documenting the server-side logic necessary to support all banking operations. Wasif and Versha, as Frontend Developers, focused on designing and implementing the user interface and user experience. they were responsible for conducting thorough testing of the frontend components to guarantee functional and visual coherence across different devices and platforms. This collaborative effort was essential for delivering a robust and user-friendly banking application.

Saad (Technical Project Manager)

User Stories:

- Create Project Roadmap and how to kick-off the Project.
- Deposit cash with receipt generation.
- Withdraw cash with real-time balance update.
- Transfer Funds

Report Documentation:

Saad was responsible for creating comprehensive documentation covering the backend processes involved in account management and transaction handling, including security implementations.

Gulraiz (Full-Stack Developer)

User Stories:

- Create Registration Process
- Login Flow.
- Authentication & Hash

Report Documentation:

Gulraiz handled the documentation for backend processes related to funds transfers, account services management, and data protection measures, ensuring all details were accurately captured and easily understandable.

Versha (Full-Stack Developer)

User Stories:

- Manage Services (Checkbooks and Cards)
- Manage Transfer Funds
- Ensure data encryption for sensitive information.

Testing:

Versha was responsible for testing the frontend components related to funds transfer, account services, and security features, focusing on user experience and interaction consistency.

Wasif (Full-Stack Developer)

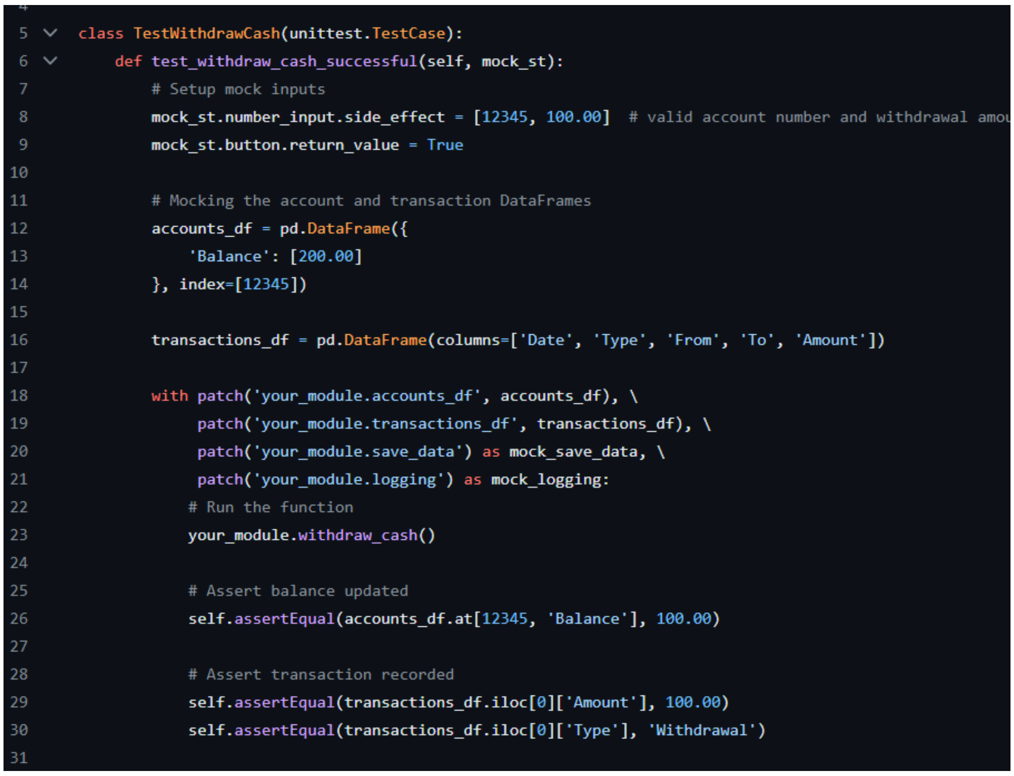
User Stories:

- Edit account details (user information, account type).
- Close an account.
- View detailed account information, including current balance and recent transactions.

Testing:

Whole Team conducted thorough testing on the respective Sprints interfaces for

Unit Testing: To Automate testing, we used 'unittest' Python Framework with 'unittest.mock' module to handle the User Interface components and data Frames. The code can be found for unit test in Git under *fund_transfer_28709550* branch.



```python
class TestWithdrawCash(unittest.TestCase):
    def test_withdraw_cash_successful(self, mock_st):
        # Setup mock inputs
        mock_st.number_input.side_effect = [12345, 100.00]  # valid account number and withdrawal amou
        mock_st.button.return_value = True

        # Mocking the account and transaction DataFrames
        accounts_df = pd.DataFrame({
            'Balance': [200.00]
        }, index=[12345])

        transactions_df = pd.DataFrame(columns=['Date', 'Type', 'From', 'To', 'Amount'])

        with patch('your_module.accounts_df', accounts_df), \
             patch('your_module.transactions_df', transactions_df), \
             patch('your_module.save_data') as mock_save_data, \
             patch('your_module.logging') as mock_logging:
            # Run the function
            your_module.withdraw_cash()

            # Assert balance updated
            self.assertEqual(accounts_df.at[12345, 'Balance'], 100.00)

            # Assert transaction recorded
            self.assertEqual(transactions_df.iloc[0]['Amount'], 100.00)
            self.assertEqual(transactions_df.iloc[0]['Type'], 'Withdrawal')
```

Figure 1 Unit Test

The backend Team worked with coordination of Frontend Team. The tasks were planned in a sprint. Backend team reviewed each other code, respectively Frontend team followed the same approach with testing and code review.

User Stories

Registration

User Story

As a new user, I want to register for an online banking account so that I can perform banking transactions online.

Acceptance Criteria

The user provides necessary information such as username and password.

The system validates the provided information for format and uniqueness.

Upon successful registration, the user's account is created in the system.

The user receives a confirmation that the account has been successfully registered.

Use Case Name: Register for Online Banking

Actor: New User

Preconditions: The user has navigated to the registration page of the banking application.

Basic Flow:

The user fills out the registration form with all required information.

The system validates the provided information for correctness and uniqueness.

Upon successful validation, the system creates a new user account and stores the information securely.

The user receives a confirmation message of successful registration.

Postconditions: User's account is created and ready for login.

Exception Paths: Information fails validation (e.g., username already exists), and the user is prompted to correct the data.

Login

User Story

As a bank customer, I want to log into my account using my username and password so that I can access my banking information securely.

Acceptance Criteria

The user must enter a valid username and password.

The system verifies the credentials against the stored data.

If credentials are correct, access is granted to the user's banking dashboard.

If credentials are incorrect, an error message is displayed.

Use Case Name: Register for Online Banking

- **Actor:** New User
- **Preconditions:** The user has navigated to the registration page of the banking application.
- **Basic Flow:**
    1. The user fills out the registration form with all required information.
    2. The system validates the provided information for correctness and uniqueness.

3. Upon successful validation, the system creates a new user account and stores the information securely.
4. The user receives a confirmation message of successful registration.

- **Postconditions:** User's account is created and ready for login.
- **Exception Paths:** Information fails validation (e.g., username already exists), and the user is prompted to correct the data.

Create New Account

User Story

As a bank customer, I want to create a new bank account so that I can manage my finances through the Advanced Banking Application.

Acceptance Criteria

- The user must provide their name, select an account type (Personal or Business), and specify an initial balance.
- The system validates the input to ensure all fields are correctly filled out.
- Upon successful validation, the system creates the account and stores the information in the database.
- The user receives a confirmation that the account has been created.

Use Case: Create a New Bank Account

Actor: Bank Customer

Preconditions: The user is logged in and has navigated to the "Create Account" section.

Basic Flow:

- The user selects "Create Account" from the options.
- The user enters their name in the name field.
- The user selects the account type (Personal or Business).
- The user enters the desired initial balance.
- The user submits the form.
- The system validates the provided information.
- If the information is valid, the system creates the account and confirms account creation to the user.
- The user receives a confirmation message and is redirected to the dashboard or account management page.

Edit Account Details

User Story

As a bank customer, I want to be able to edit my existing account details so that I can keep my banking information up-to-date and accurate.

Acceptance Criteria

- The user must enter a valid account number to access the editing interface.
- The system validates the account number and retrieves the associated account details.
- The user can update fields such as name, account type, or other relevant details.
- Changes are confirmed and saved upon user submission.
- The user receives a confirmation message that the account details have been updated successfully.

Use Case Name: Edit Bank Account Details

Actor: Bank Customer

Preconditions: The user is logged in and has selected the "Edit Account Details" option.

Basic Flow:

- The user enters the account number in the designated field.
- The system validates the account number and retrieves the account details.
- The user edits their account details in the provided fields.
- The user submits the updated information.
- The system verifies and saves the updates.
- The user receives a confirmation of successful update.
- **Postconditions:** The account details are updated in the system.
- Exception Paths:
    - If the account number is invalid, an error message is displayed.
    - If the update fails, the user is informed and asked to try again.

Close Account

User Story

As a bank customer, I want to be able to close my bank account through the online application so that I can terminate my services conveniently without needing to visit a branch.

Acceptance Criteria

- The user must enter a valid account number to initiate the account closure.
- The system verifies the account number and retrieves the associated account details.
- The user is asked to confirm the closure to prevent accidental closures.
- Upon confirmation, the account is closed, and all associated services are terminated.
- The user receives a confirmation message that the account has been successfully closed.

Use Case Name: Close Bank Account

- **Actor:** Bank Customer
- **Preconditions:** The user is logged in and has navigated to the "Close Account" option.
- **Basic Flow:**
    1. The user enters their account number in the provided field.
    2. The system verifies the account number and retrieves the relevant account.
    3. The system prompts the user to confirm their intention to close the account.
    4. Upon confirmation, the system processes the closure, terminating any related services and finalizing all pending transactions.
    5. The system confirms account closure to the user.
- **Postconditions:** The bank account is permanently closed.
- **Exception Paths:**
    - If the account number is invalid, an error message is displayed.
    - If the user cancels the closure process, no changes are made.

Deposit Cash

User Story

As a bank customer, I want to be able to deposit cash into my bank account through the application so that I can increase my balance easily without needing to visit the bank physically.

Acceptance Criteria

- The user must enter the amount of cash they wish to deposit.
- The system validates the entered amount to ensure it's a permissible value.

- Upon validation, the user is prompted to confirm the deposit to prevent accidental transactions.
- Once confirmed, the system updates the account balance with the deposited amount.
- The user receives a receipt (either on-screen or emailed) confirming the transaction details.

Use Case Name: Deposit Cash into Account

- **Actor:** Bank Customer
- **Preconditions:** The user is logged in and has selected the option to deposit cash.
- **Basic Flow:**
    1. The user selects the deposit option from the application menu.
    2. The user enters the amount of cash they want to deposit.
    3. The system verifies the amount and asks for confirmation.
    4. The user confirms the transaction.
    5. The system updates the user's account balance and issues a transaction receipt.
    6. The user views the updated balance and receives the receipt.
- **Postconditions:** The user's account balance is updated with the deposited amount.
- **Exception Paths:**
    - If the entered amount is invalid or outside permissible limits, an error message is displayed.
    - If the user cancels the transaction before confirming, no changes are made to the account.

Withdraw Cash

User Story

As a bank customer, I want to be able to withdraw cash from my account using the application so that I can manage my cash flow efficiently without having to visit a branch.

Acceptance Criteria

- The user must enter a valid account number and the amount they wish to withdraw.
- The system validates the account number and checks the account balance to ensure sufficient funds are available.
- The user is asked to confirm the withdrawal to prevent accidental transactions.
- Upon confirmation, the system processes the withdrawal, updates the account balance, and logs the transaction.
- The user receives a confirmation message and a digital receipt detailing the transaction.

Use Case Name: Withdraw Cash from Bank Account

- **Actor:** Bank Customer
- **Preconditions:** The user is logged in and has selected the "Withdraw Cash" option.
- **Basic Flow:**
    1. The user enters their account number and the amount they want to withdraw.
    2. The system validates the account number and checks for sufficient funds.
    3. The system prompts the user to confirm the withdrawal amount.
    4. The user confirms the transaction.
    5. The system updates the account balance, records the transaction, and issues a receipt.
    6. The user receives a transaction confirmation and views the updated balance.
- **Postconditions:** The account balance is updated to reflect the withdrawal.
- **Exception Paths:**
    - If the account number is invalid or the account has insufficient funds, an error message is displayed.
    - If the user cancels the transaction, no changes are made to the account.

Transfer Funds

User Story

As a bank customer, I want to be able to transfer funds between my accounts or to other accounts within the bank, so that I can manage my finances conveniently through the application.

Acceptance Criteria

- The user must enter valid account numbers for both the "from" and "to" accounts.
- The user must specify the amount to be transferred.
- The system validates the account numbers and checks that the "from" account has sufficient funds.
- The user is asked to confirm the transfer to prevent accidental transactions.
- Upon confirmation, the system processes the transfer, debiting the "from" account and crediting the "to" account.

- The user receives a confirmation message and details of the transaction.

Use Case Name: Transfer Funds Between Accounts

- **Actor:** Bank Customer
- **Preconditions:** The user is logged in and has navigated to the "Transfer Funds" option.
- **Basic Flow:**
  1. The user enters the account number to transfer funds from.
  2. The user enters the account number to transfer funds to.
  3. The user inputs the amount of money to transfer.
  4. The system validates the account numbers and checks the available balance in the "from" account.
  5. The system prompts the user to confirm the transaction.
  6. Upon confirmation, the system completes the transfer, updates both account balances, and logs the transaction.
  7. The user receives a confirmation of the successful transfer.
- **Postconditions:** The funds are transferred successfully, and both accounts reflect the new balances.
- **Exception Paths:**
  - If either account number is invalid, the system displays an error message.
  - If insufficient funds are available in the "from" account, the system notifies the user and cancels the transaction.
  - If the user cancels the transaction before confirming, no changes are made.

View Account Information

User Story

As a bank customer, I want to be able to view detailed information about my account so that I can monitor my balance, transactions, and other related details easily through the application.

Acceptance Criteria

- The user must enter a valid account number to retrieve account information.
- The system validates the account number and retrieves the information.
- The user is presented with details such as current balance, recent transactions, and account type.
- The information displayed is up-to-date and accurate.

Use Case Name: View Bank Account Information

- **Actor:** Bank Customer
- **Preconditions:** The user is logged in and has navigated to the "View Account Information" section.
- **Basic Flow:**
  1. The user enters their account number into the input field.
  2. The system validates the account number.
  3. Upon successful validation, the system retrieves and displays the account information.
  4. The user views the detailed account information, including current balance, recent transactions, and account type.
- **Postconditions:** The user has access to their account information.
- **Exception Paths:**
  - If the account number is invalid, an error message is displayed.
  - If there is a system error retrieving account details, the user is informed of the error.

Manage Services

User Story

As a bank customer, I want to be able to request and manage banking services such as checkbooks, debit cards, and credit cards through the application, so that I can handle these needs conveniently without visiting a bank.

Acceptance Criteria

- The user must enter a valid account number to request or manage services.
- The system validates the account number and retrieves the specific account details.
- The user selects the service they wish to manage (e.g., request a checkbook, debit card, or credit card).
- Depending on the selected service, the user completes any necessary steps to request or manage the service.
- The system processes the request and updates the account services accordingly.
- The user receives a confirmation message detailing the request and expected fulfillment time

Use Case Name: Manage Account Services

- **Actor:** Bank Customer
- **Preconditions:** The user is logged in and has navigated to the "Manage Services" section.
- **Basic Flow:**
    1. The user enters their account number.
    2. The system validates the account number.
    3. Upon successful validation, the user selects the service they need (checkbook, debit card, credit card).
    4. The user fills out any necessary information or forms related to the service request.
    5. The user submits the request.
    6. The system processes the request, logs the service order, and sends a confirmation to the user.
    7. The user receives a confirmation message with details about the service processing and delivery.
- **Postconditions:** The user's service request is processed, and they are informed of the status.
- **Exception Paths:**
    - If the account number is invalid, the user is notified and asked to correct it.
    - If there is an error processing the request, the user is informed and may be asked to try again later.

Setup and Manage Recurring Payments

User Story

As a bank customer, I want to be able to set up and manage recurring payments for my regular expenses through the application, so that I can automate my payments and ensure they are made on time without manual intervention each month.

Acceptance Criteria

- The user must enter a valid account number from which payments will be made.
- The user provides payment details including description, amount, frequency, and the next due date.
- The system validates the provided details and confirms the account has sufficient funds for the first transaction.
- Upon validation, the user confirms the setup of the recurring payment.
- The system schedules the payments according to the specified frequency and informs the user.
- The user receives a confirmation message that the recurring payment has been set up successfully.

Use Case Name: Setup Recurring Payments

- **Actor:** Bank Customer
- **Preconditions:** The user is logged in and has navigated to the "Setup Recurring Payments" section.
- **Basic Flow:**
    1. The user enters their account number and the details of the payment including description, amount, frequency, and next due date.
    2. The system validates the account number and payment details.
    3. The system displays a summary of the payment for user confirmation.
    4. The user reviews and confirms the payment setup.
    5. The system schedules the payment and sends a confirmation to the user.
    6. The user receives a confirmation message indicating the payment setup is complete.
- **Postconditions:** The recurring payment is scheduled, and the user can view and manage it if necessary.
- **Exception Paths:**
    - If the account number is invalid, an error message is displayed.
    - If there are insufficient funds for the initial payment, the user is informed.
    - If any detail is incorrect or missing, the user is prompted to correct it before proceeding.

Task Allocation

Registration Screen

1. **Frontend Development:**

- Create a streamlined and user-friendly registration form that includes fields for username, password, email address, and acceptance of terms and conditions.
- Implement client-side validation to ensure all form inputs adhere to expected formats and standards before they are submitted.

2. **Backend Development:**
   - Develop server-side logic to check user credentials against the stored data in the database.
   - Implement session management to confirm user registration and maintain user logged-in status appropriately without needing to re-enter credentials frequently.
3. **Testing:**
   - Conduct thorough testing of the registration process in various scenarios to verify its reliability and functional integrity.

Login Screen

4. **Frontend Development:**
   - Design a clean and intuitive login page that includes fields for username and password.
   - Implement client-side validation to ensure that inputs meet the required format before submission.
5. **Backend Development:**
   - Develop server-side logic to check user credentials against the stored data in the database.
   - Implement secure session management to keep the user logged in during the session without re-authenticating.
6. **Testing:**
   - Test the login process under various conditions to ensure it works as expected.

Create New Account

7. **Frontend Development:**
   - Design the user interface for the account creation page.
   - Implement form controls for entering the name, selecting the account type, and specifying the initial balance.
   - Ensure the form is accessible and user-friendly.
8. **Backend Development:**
   - Set up database schema to store account details.
   - Implement backend logic to handle account creation, including validation of input data.
   - Ensure secure handling of sensitive information.
9. **Testing:**
   - Unit test backend logic to ensure account details are correctly processed and stored.
   - Integration test to ensure the frontend and backend work together seamlessly.
   - User acceptance testing to verify that the user experience meets the requirements.
10. **Security Measures:**
    - Implement checks to prevent SQL injection and other security threats.
    - Ensure data validation both client-side and server-side.

Edit Account Details

1. **Frontend Development:**
   - Design and implement the interface for entering the account number and editing details.
   - Implement input validation to ensure accurate account number entry.
   - Enable dynamic retrieval and display of existing account details for modification.
2. **Backend Development:**
   - Develop the logic to validate and retrieve account details from the database.
   - Implement the update functionality to modify account details in the database.
   - Ensure that all changes are logged for audit and rollback purposes.
3. **Testing:**
   - Unit testing for backend processes to ensure that they handle data correctly.
   - Integration testing to ensure that the frontend and backend are synchronized in handling data updates.
   - User acceptance testing to confirm that the feature meets user needs and is user-friendly.
4. **Security Measures:**
   - Ensure secure transmission of data to prevent interception or tampering.

- Implement proper authentication and authorization checks to prevent unauthorized access or changes.

Close Account

1. **Frontend Development:**
    - Design and implement the interface for entering the account number and confirming account closure.
    - Implement client-side validations to check for the validity of the account number.
    - Provide a clear, user-friendly process for confirming the account closure to avoid accidental closures.
2. **Backend Development:**
    - Develop the logic to validate the account number and retrieve account details.
    - Implement the functionality to close the account, including terminating all related services and handling any remaining balance or pending transactions.
    - Ensure all records are updated appropriately to reflect the closed status.
3. **Testing:**
    - Unit tests for backend processes to ensure correct account retrieval and closure logic.
    - Integration tests to verify the interaction between frontend inputs and backend processes.
    - User acceptance tests to ensure the closure process is clear and functions as intended.

Deposit Cash

1. **Frontend Development:**
    - Design and implement the cash deposit interface, including input fields for the transaction amount and confirmation buttons.
    - Ensure user interface elements are clearly labeled and accessible.
2. **Backend Development:**
    - Develop the logic for handling cash deposit transactions, including validating transaction amounts and updating account balances.
    - Implement secure handling of transaction data to protect user information and maintain integrity.
3. **Testing:**
    - Unit test backend transaction processes to ensure they correctly handle various deposit scenarios.
    - Conduct integration tests to check the seamless operation between the frontend and backend.
    - Perform user acceptance testing to validate the entire process from a user's perspective.

Withdraw Cash

1. **Frontend Development:**
    - Design and implement the interface for entering the withdrawal details (account number and amount).
    - Ensure the user interface is intuitive and accessible, with clear instructions for transaction confirmation.
2. **Backend Development:**
    - Develop the logic to validate the account details and check the balance for sufficient funds.
    - Implement the transaction processing logic to handle withdrawals, including updating the account balance and recording the transaction.
    - Ensure that all financial transactions are secure and comply with financial regulations.
3. **Testing:**
    - Unit tests for backend processes to ensure they correctly handle withdrawal requests.
    - Integration tests to verify that the frontend inputs correctly trigger backend processes.
    - User acceptance tests to ensure the withdrawal process is clear and meets user expectations.

Transfer Funds

1. **Frontend Development:**
    - Design and implement the user interface for the funds transfer, including input fields for both account numbers and the transfer amount.
    - Ensure the interface includes a clear and concise confirmation step.
2. **Backend Development:**
    - Develop logic to validate the account numbers and check the balance of the "from" account.
    - Implement the transaction logic to transfer funds, update account balances securely, and log all transactions.
    - Ensure the transaction process adheres to banking regulations and security standards.

3. **Testing:**
    - Unit tests to ensure that validation, balance checks, and fund transfers are handled correctly.
    - Integration tests to confirm that the frontend and backend work together seamlessly.
    - User acceptance testing to verify that the funds transfer process is intuitive and error-free.

View Account Information

1. **Frontend Development:**
    - Design and implement the interface for entering the account number and viewing account details.
    - Ensure the interface is user-friendly and accessible, with clear labels and readable formats.
2. **Backend Development:**
    - Develop the logic to validate the account number and retrieve account details from the database.
    - Ensure the information is fetched efficiently and securely, maintaining data integrity and privacy.
3. **Testing:**
    - Unit tests for validating input and database queries to ensure they function as expected.
    - Integration tests to ensure that the frontend and backend systems work together seamlessly.
    - User acceptance testing to verify that the account information is displayed correctly and meets user needs.

Manage Services

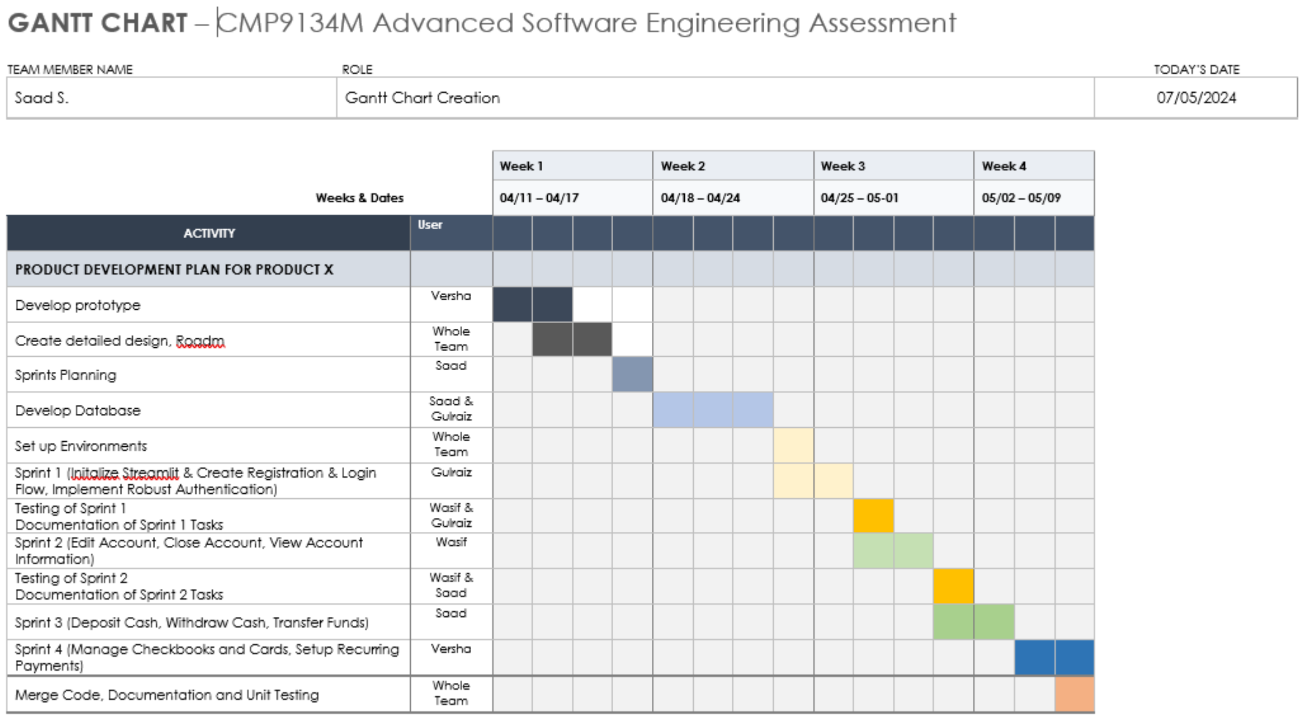
1. **Frontend Development:**
    - Design and implement the interface for entering account numbers and selecting services.
    - Ensure the forms for requesting each service are intuitive and guide the user through the process.
    - Implement validation feedback for user input to enhance user experience.
2. **Backend Development:**
    - Develop logic to validate account numbers and retrieve associated account details.
    - Implement the functionality to handle different service requests, including database updates and transaction logs.
    - Ensure that service requests are processed in a secure and efficient manner.
3. **Testing:**
    - Unit tests to ensure that validation and service request handling are functioning correctly.
    - Integration tests to confirm that the frontend and backend work seamlessly together.
    - User acceptance testing to ensure that the service management process meets customer needs and is error-free.

Setup and Manage Recurring Payments

1. **Frontend Development:**
    - Design and implement the user interface for setting up recurring payments, including fields for account number, payment description, amount, frequency, and next due date.
    - Ensure the interface guides the user through the setup process and provides clear options for confirming the details.
2. **Backend Development:**
    - Develop the logic to validate account details and payment information.
    - Implement the functionality to schedule recurring payments in the banking system.
    - Ensure secure handling of transaction data and schedule these transactions according to the user-defined frequency.
3. **Testing:**
    - Unit tests to ensure that the payment scheduling logic is accurate and handles edge cases, such as leap years for annual payments.
    - Integration tests to confirm that the frontend inputs correctly trigger backend processes and result in scheduled payments.
    - User acceptance testing to ensure the feature meets user expectations and is user-friendly.

Project Planning

Gantt Chart



GANTT CHART – CMP9134M Advanced Software Engineering Assessment

TEAM MEMBER NAME	ROLE	TODAY'S DATE
Saad S.	Gantt Chart Creation	07/05/2024

		Week 1				Week 2				Week 3				Week 4		
Weeks & Dates		04/11 – 04/17				04/18 – 04/24				04/25 – 05-01				05/02 – 05/09		
ACTIVITY	User															
PRODUCT DEVELOPMENT PLAN FOR PRODUCT X																
Develop prototype	Versha	■	■													
Create detailed design, Roadm	Whole Team		■	■												
Sprints Planning	Saad				■											
Develop Database	Saad & Gulraiz					■	■	■								
Set up Environments	Whole Team								■							
Sprint 1 (Initalize Streamlit & Create Registration & Login Flow, Implement Robust Authentication)	Gulraiz								■	■						
Testing of Sprint 1 Documentation of Sprint 1 Tasks	Wasif & Gulraiz										■					
Sprint 2 (Edit Account, Close Account, View Account Information)	Wasif										■	■				
Testing of Sprint 2 Documentation of Sprint 2 Tasks	Wasif & Saad												■			
Sprint 3 (Deposit Cash, Withdraw Cash, Transfer Funds)	Saad												■	■		
Sprint 4 (Manage Checkbooks and Cards, Setup Recurring Payments)	Versha														■	■
Merge Code, Documentation and Unit Testing	Whole Team															■

The above diagram represents the project timeline while development of the project. We opted for the agile methodology for the flexibility andthe iterative development of the project. We mainly divided this project into four parts i.e. account management, financial transactions management, account services and security and compliance. Thus, we have planned the milestones and completed within the assigned time.

System design:

The component diagram below represents the banking software workingvirtually.

- User interface: it a central point of interaction with the users.
- System: the users are guided to various services such as account management, financial transaction, account services and security andcompliances using the user interface.
- Database: the user data generated from the above level is stored inthe database.
- Backup system: it ensures the data integrity.
- Encryption services: it is used by security and compliances to secrethe sensitive data of the user such as account number, account password,etc.

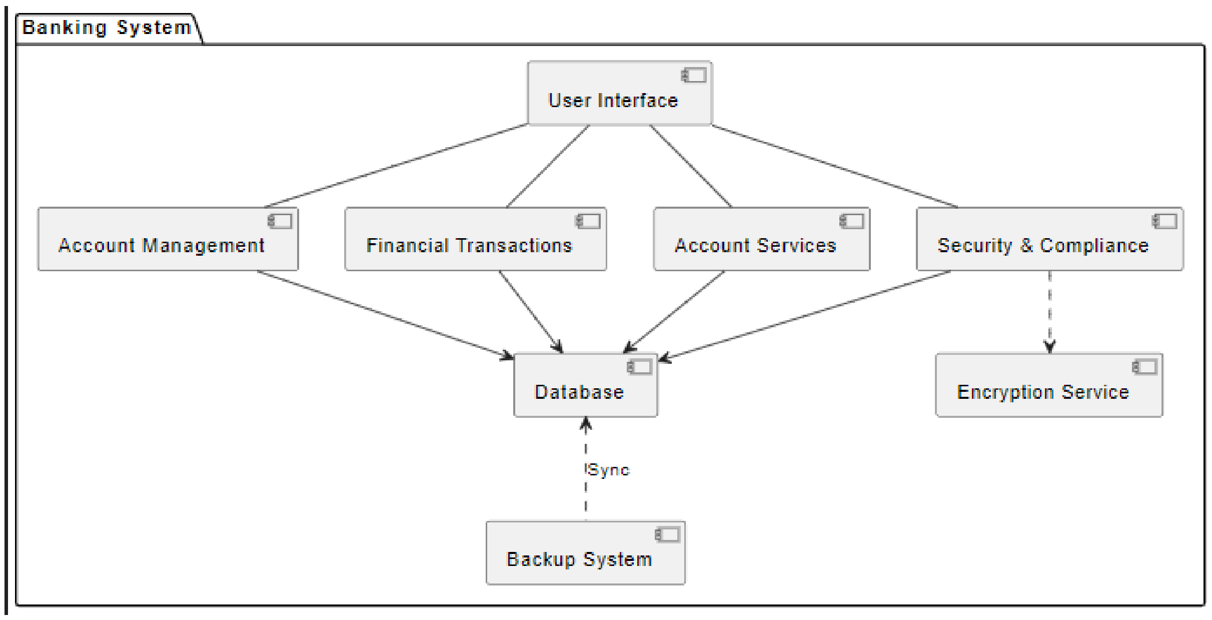
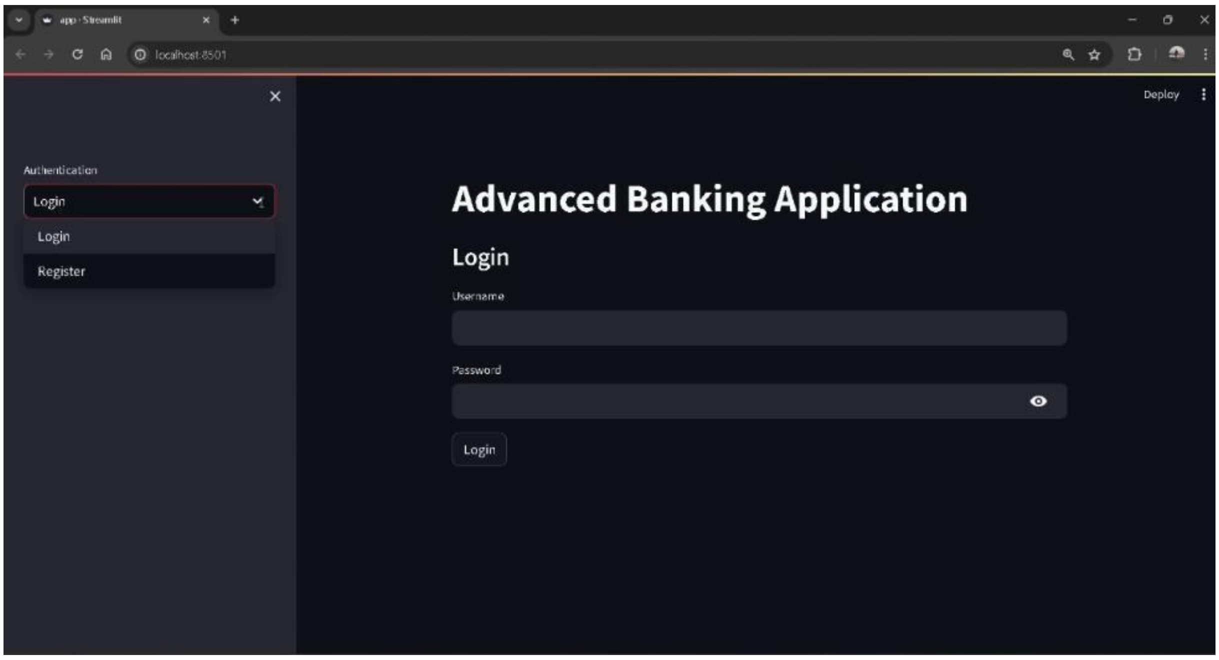


Figure 2: System diagram

Graphic user interface design [GUI]:

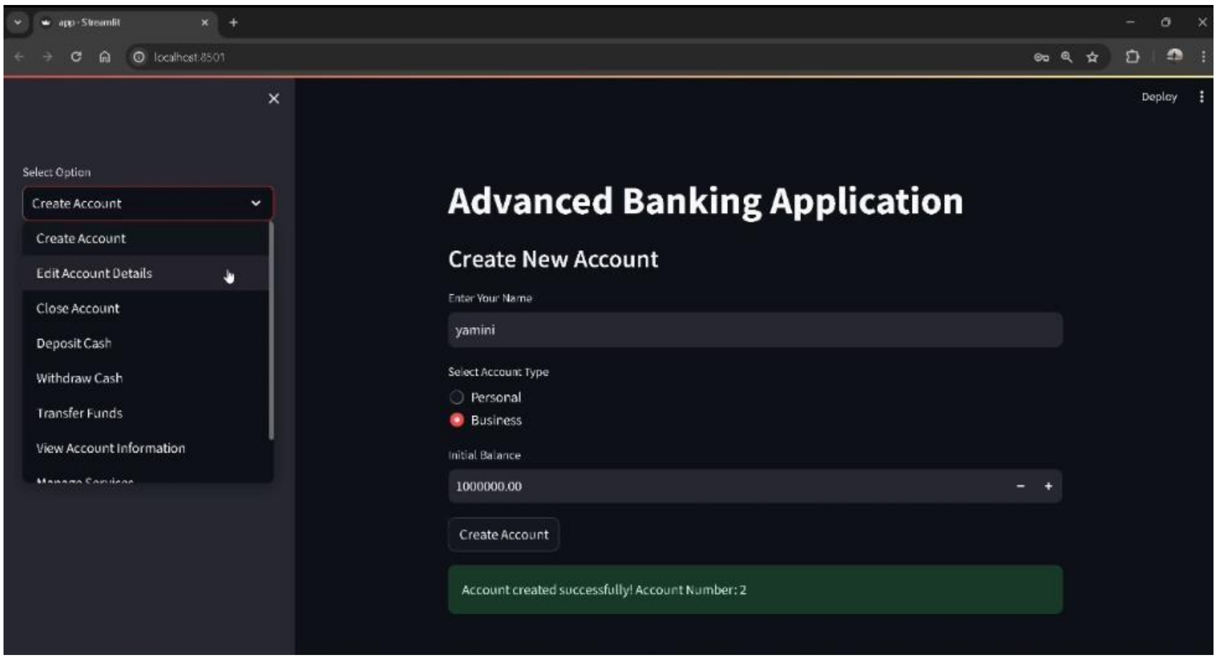
The GUI is divided into part i.e. main page where user can login using their own credentials or register if they are new and the following page which consist of various functionalities such as creating, editing, deletingthe account, withdraw or deposit the cash and many more.

This GUI is designed using the streamlit module which help in designing the robust and user friendly interface without any complication.



Authentication: Login

Figure 4Main page



Select Option: Create Account

Enter Your Name: yamini

Select Account Type: Business

Initial Balance: 1000000.00

Account created successfully! Account Number: 2

Figure 3Following page

Thus, the GUI is so self-explanatory and easy for the users to access theirdata.

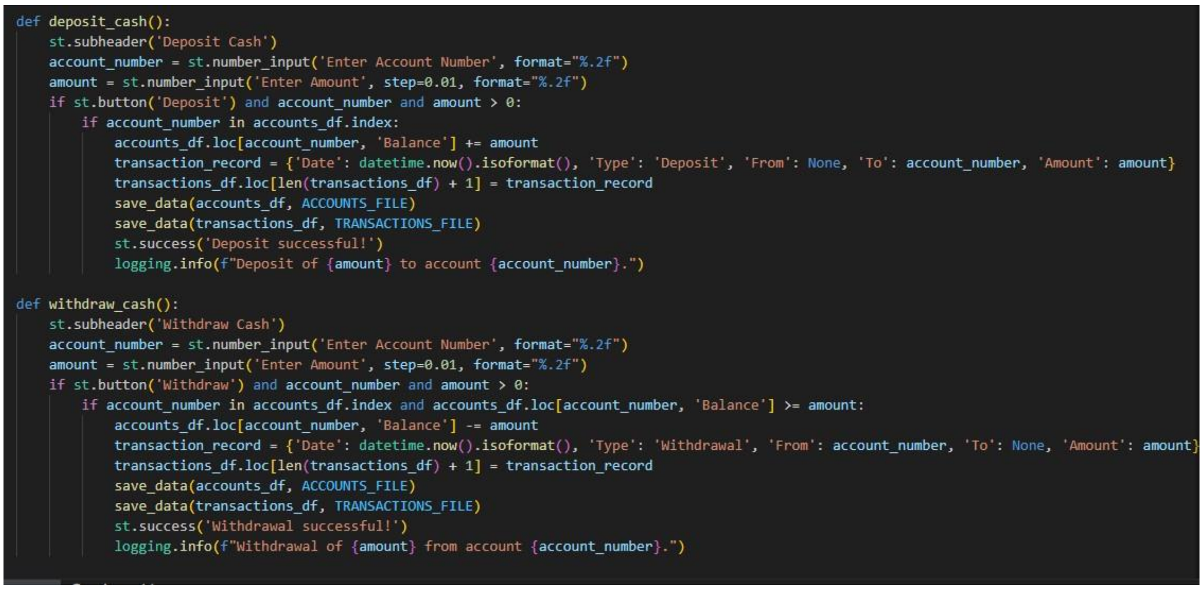
Version control using git:

We have used github for the version control using the git.

Pair programming approach:

- Planning: Before the coding, OOP principles were planned and discussed. The class, methods and hierarchies were decided.
- Coding: implemented the planned in agile methodology. It was a team approach by switching role constantly to stay actively engaged.
- Reviewing: discussed and reviewed the code to improve the quality of code as well as fix the bugs.
- Testing: performed unit testing on each section of the code for the improvement of the code.

Implementation:



```python
def deposit_cash():
    st.subheader('Deposit Cash')
    account_number = st.number_input('Enter Account Number', format="%.2f")
    amount = st.number_input('Enter Amount', step=0.01, format="%.2f")
    if st.button('Deposit') and account_number and amount > 0:
        if account_number in accounts_df.index:
            accounts_df.loc[account_number, 'Balance'] += amount
            transaction_record = {'Date': datetime.now().isoformat(), 'Type': 'Deposit', 'From': None, 'To': account_number, 'Amount': amount}
            transactions_df.loc[len(transactions_df) + 1] = transaction_record
            save_data(accounts_df, ACCOUNTS_FILE)
            save_data(transactions_df, TRANSACTIONS_FILE)
            st.success('Deposit successful!')
            logging.info(f"Deposit of {amount} to account {account_number}.")

def withdraw_cash():
    st.subheader('Withdraw Cash')
    account_number = st.number_input('Enter Account Number', format="%.2f")
    amount = st.number_input('Enter Amount', step=0.01, format="%.2f")
    if st.button('Withdraw') and account_number and amount > 0:
        if account_number in accounts_df.index and accounts_df.loc[account_number, 'Balance'] >= amount:
            accounts_df.loc[account_number, 'Balance'] -= amount
            transaction_record = {'Date': datetime.now().isoformat(), 'Type': 'Withdrawal', 'From': account_number, 'To': None, 'Amount': amount}
            transactions_df.loc[len(transactions_df) + 1] = transaction_record
            save_data(accounts_df, ACCOUNTS_FILE)
            save_data(transactions_df, TRANSACTIONS_FILE)
            st.success('Withdrawal successful!')
            logging.info(f"Withdrawal of {amount} from account {account_number}.")
```

Figure 5 Implementation

The above diagram shows the implementation of some functionality of the banking system. It shows the operation of the cash withdrawal and deposition.

Critical evaluation:

For the development of the banking system, there are many software tools which have played a vital role in development and implementation.

A] Advances in Software Processes

- Methodology: Agile methodology was adopted as it can work with the dynamic environment. Agile methodology specially the scrum, was chosen due to iterative nature and flexibility.
- Agile processes: the implementation of the agile process facilitated constant inspection, adaption and Improvement. Improved the product quality by addressing the issues.

B] Software engineering techniques

- Project management: JIRA is used to track and monitor the progress, tasks, and backlogs of the project.
- Prototype design: SKETCH tool is used to design and conceptualize the user interface. SKETCH is used to design the system design and workflow of the project.
- Version control: Code were managed and version controlled using the github.

C] How advanced Software Systems and Software Engineering have changed how we interact as a society with your system considering the following:

- Social impact: Online and mobile banking have made this financial services accessible virtually from any part of the world. The banking system has democratized the banking system accessible to all.

- Ethical impact: here, this project addresses the ethical concern by implementing the security and compliance feature by two factor authentication and data encryption.

Entrepreneurial impact: the development of the banking app can get into entrepreneurial activity by providing start-ups and small business for managing finance effectively.

Works Cited

Al-Saqqa, S., Sawalha, S. and AbdelNabi, H., 2020. Agile software development: Methodologies and trends. *International Journal of Interactive Mobile Technologies*, *14*(11).

Abdelghany, A.S., Darwish, N.R. and Hefni, H.A., 2019. An agile methodology for ontology development. *International Journal of Intelligent Engineering and Systems*, *12*(2), pp.170-181.

Dingsøyr, T., Falessi, D. and Power, K., 2019. Agile development at scale: the next frontier. *Ieee Software*, *36*(2), pp.30-38.

Saeed, S., Jhanjhi, N.Z., Naqvi, M. and Humayun, M., 2019. Analysis of software development methodologies. *International Journal of Computing and Digital Systems*, *8*(5), pp.446-460.

Ogundipe, D.O., Odejide, O.A. and Edunjobi, T.E., 2024. Agile methodologies in digital banking: Theoretical underpinnings and implications for custom satisfaction. *Open Access Research Journal of Science and Technology*, *10*(02), pp.021-030.

Rabbani, M.R. and Khan, S., 2020. Agility and fintech is the future of Islamic finance: A study from Islamic banks in Bahrain. *International Journal of Scientific and Technology Research*, *9*(3), pp.6955-6957.

Abdullahi, M.S., Shehu, U.R. and Usman, B.M., 2019. Impact of information communication technology on organizational productivity in the Nigeria banking industry: empirical evidence. *Noble International Journal of Business and Management Research*, *3*(1), pp.1-9.