NHẬP MÔN CÔNG NGHỆ PHẦN MỀM

Project Proposal

Bộ môn Công nghệ phần mềm

Khoa Công nghệ thông tin

Đại học Khoa học tự nhiên TP HCM

Table of Contents

Project Proposal

General Information

The document aims to focus on the following topics:

- ✓ Create Project Proposal document
- ✓ Complete Project Proposal document with following contents:
    - Brief problem statement
    - Proposed solutions
    - Development plan
    - Human resources and cost
- ✓ Read and understand the Project Proposal document.

1 Member Evaluation Table

StudentID	Name	Contribution (%)	Sign
22127060	Phạm Quang Duy	25	Duy
22127088	Lê Hoàng Đạt	25	Đạt
22127270	Nguyễn Quang Minh	25	Minh
22127389	Nguyễn Phúc Thành	25	Thành

2 Brief Problem Statement

In today's rapidly advancing technological landscape, which permeates every facet of life, personal financial management, particularly savings account management, is also evolving. To meet the growing demands of customers for fast, secure, and convenient banking services, the banking sector must implement advanced technological solutions. As the software development lead, it is crucial to develop a modern savings account management system comprising two main modules: the back-office module for bank staff and the client module for customers.

2.1. Reasons for Developing a Savings Account Management System

1. **Enhanced Management Efficiency**: The system automates account opening, deposit, and withdrawal processes, minimizing errors, enhancing efficiency, and reducing manual intervention.
2. **Improved Security**: The system integrates advanced security technologies to safeguard customer financial information and prevent fraud, ensuring secure online transactions.
3. **Data Analysis and Reporting**: The system provides robust data analytics capabilities, allowing banks to gather detailed insights into customers' saving behaviors, thereby facilitating smarter business and marketing decisions.

2.2. Benefits of Using a Savings Account Management System

1. **Time and Cost Savings**: Automation reduces the time and labor needed to manage transactions, thereby saving costs for the bank.
2. **Scalability**: The system allows banks to easily scale operations without significant investments in physical infrastructure.
3. **Legal Compliance**: The system ensures that the bank adheres to legal regulations regarding financial management and transactions.
4. **Market Responsiveness**: The system enables banks to quickly respond to changes in the financial market, such as adjusting interest rates or terms, allowing for more competitive savings offerings.

2.3. Operating Environment

1. **User Interface**: Web browser supporting HTML5 and CSS3 to ensure compatibility across multiple platforms, from desktop to mobile.
2. **Database**: Using MySQL Database to manage data with high performance and reliability.
3. **Programming Language**: Python (Flask) for the server-side to leverage security and scalability features; JavaScript, HTML, and CSS for the frontend.

2.4. Design and Deployment Constraints

1. **User Interface**: Web browsers supporting HTML5 and CSS3 for compatibility across platforms, from desktops to mobile devices.
2. **Database**: Utilizing MySQL Database for high-performance and reliable data management.
3. **Programming Languages**: Python (Flask) for the server-side to leverage security and scalability; JavaScript, HTML, and CSS for the client-side interface.

2.5. Detailed Requirements

1. **Open Savings Accounts**: Allowing customers to open savings accounts with various terms.
2. **Deposit Money**: Customers can easily deposit money into their savings accounts.
3. **Withdraw Money**: Customers can withdraw money from their savings accounts according to bank regulations.
4. **Account Inquiry**: Both staff and customers can query detailed information about savings accounts.
5. **Reporting**: The system must provide periodic reports on operational status, revenue, and other key metrics.
6. **Updates**: The system allows for proactive interest rate adjustments to stay competitive in the market, while also providing flexibility for customers to choose suitable deposit terms.

3 Proposed solutions:

3.1 Software

3.1.1. List of software function:

Functional			
No.	Demand	Requirements	Regulations
1	As a bank employee, I want a feature that allows me to easily open savings accounts for customers.	Open a Savings Account	+ Savings Types: On-demand / 3-month / 6-month + Minimum Deposit: 100,000 VND
2	As a bank employee, I want to record the amount of money that customers deposit into their savings accounts.	Create Deposit Slip	+ Additional deposits are allowed if the savings type is ON-DEMAND. + Minimum additional deposit: 100,000 VND
3	As a bank employee, I want to record the amount of money that customers withdraw from their savings accounts.	Create Withdrawal Slip	+ Withdrawals are allowed if the account has been open for at least 15 days. + For FIXED-TERM savings: withdraw at maturity, withdraw the entire amount; interest depends on the type of savings account. + For ON-DEMAND savings:

			withdraw within the available balance; interest calculated for deposits of 1 month or more - 0.15%.
4	As a bank employee, I want to look up information about customers' savings accounts to assist them when needed.	Account Lookup	
5	As a bank manager, I want to check and summarize the savings accounts that have been opened or closed during the month, as well as the total deposits and withdrawals made in the month, to get an overview of the operational status.	Generate Monthly Report: + Daily sales report + Monthly account opening/closing report	
6	As a bank employee, I want to help customers easily adjust interest rates, deposit terms, or minimum deposit periods.	Change Regulations	+ Change the number of term types and minimum deposit amount + Change the minimum deposit period and interest rates for different term types

Non-functional		
No.	Demand	Requirements
1	As a bank employee, I want all customer information to be stored securely.	- All customer data must be stored securely to prevent unauthorized access. - Access to customer data should be restricted based on roles and permissions.
2	As a bank employee or manager, I want all data retrieval to be accurate to support customers as well as for storage, analysis, and reporting purposes.	- Data retrieval processes must be accurate and reliable to support customer inquiries promptly. - Historical data should be preserved for analysis and reporting purposes. - Real-time data availability for operational efficiency.
3	As a bank employee, I want the program to execute quickly, efficiently, and avoid errors.	- The system must execute operations quickly to minimize customer wait times. - Error handling mechanisms should be in place to prevent system failures. - Performance tuning to optimize response times and resource utilization.
4	As a bank employee, I want a user-friendly interface that is easy to interact with.	- The user interface should be intuitive and easy to navigate. - Ensure accessibility features for users with disabilities. - Provide clear instructions and guidance for all functionalities.

3.1.2. General software architecture:

1. Overview of the architecture:

– The system will be built by using a Three-tier architecture, which includes:
  + Presentation Layer
  + Business Logic Layer
  + Data Access Layer

2. Describe the layers:

a. Presentation Layer:

– **Objective**: Interact with end users, receive data from users and display results.
– **Components**:
  + User Interface (UI).
  + Interface module: login page, savings account opening page, deposit slip page, withdrawal slip page, account lookup page, monthly report page.
– **Technologies**: HTML, CSS, JavaScript.

b. Business Logic Layer:

– **Objective**: Handle business logic and perform operations related to the business: opening accounts, depositing money, withdrawing money, account lookup, and generating reports.
– **Components**:
  + Services: AccountService, TransactionService, ReportService.
  + Business logic processing: interest calculation, balance checking, customer information verification.
– **Technologies**: Python, Flask (Python).

c. Data Access Layer:

– **Objective**: Interact with the database to store and retrieve data.
– **Components**:
  + Entity Models.
– **Technologies**: MySQL, framework sqlalchemy.

3. Detail the functions according to each layer of architecture

a. Savings Account Opening Function:

- **Presentation Layer:** The user interface where users input savings account information.
- **Business Logic Layer:** 'AccountService' processes the logic for opening a new account and calculates related parameters.
- **Data Access Layer:** Stores the new savings account information in the database .

b. Deposit Slip Function:

- **Presentation Layer:** The user interface for inputting deposit transaction details.
- **Business Logic Layer:** 'TransactionService' processes the logic for depositing money, checks the balance, and updates the balance.
- **Data Access Layer:** Stores the deposit transaction in the database.

c. Withdrawal Slip Function:

- **Presentation Layer:** The user interface for inputting withdrawal transaction details.
- **Business Logic Layer:** 'TransactionService' processes the logic for withdrawing money, checks the balance, and updates the balance.
- **Data Access Layer:** Stores the withdrawal transaction in the database .

d. Account Lookup Function:

- **Presentation Layer:** The user interface where users input lookup criteria.
- **Business Logic Layer:** 'AccountService' processes the logic for looking up savings account information.
- **Data Access Layer:** Retrieves savings account information from the database.

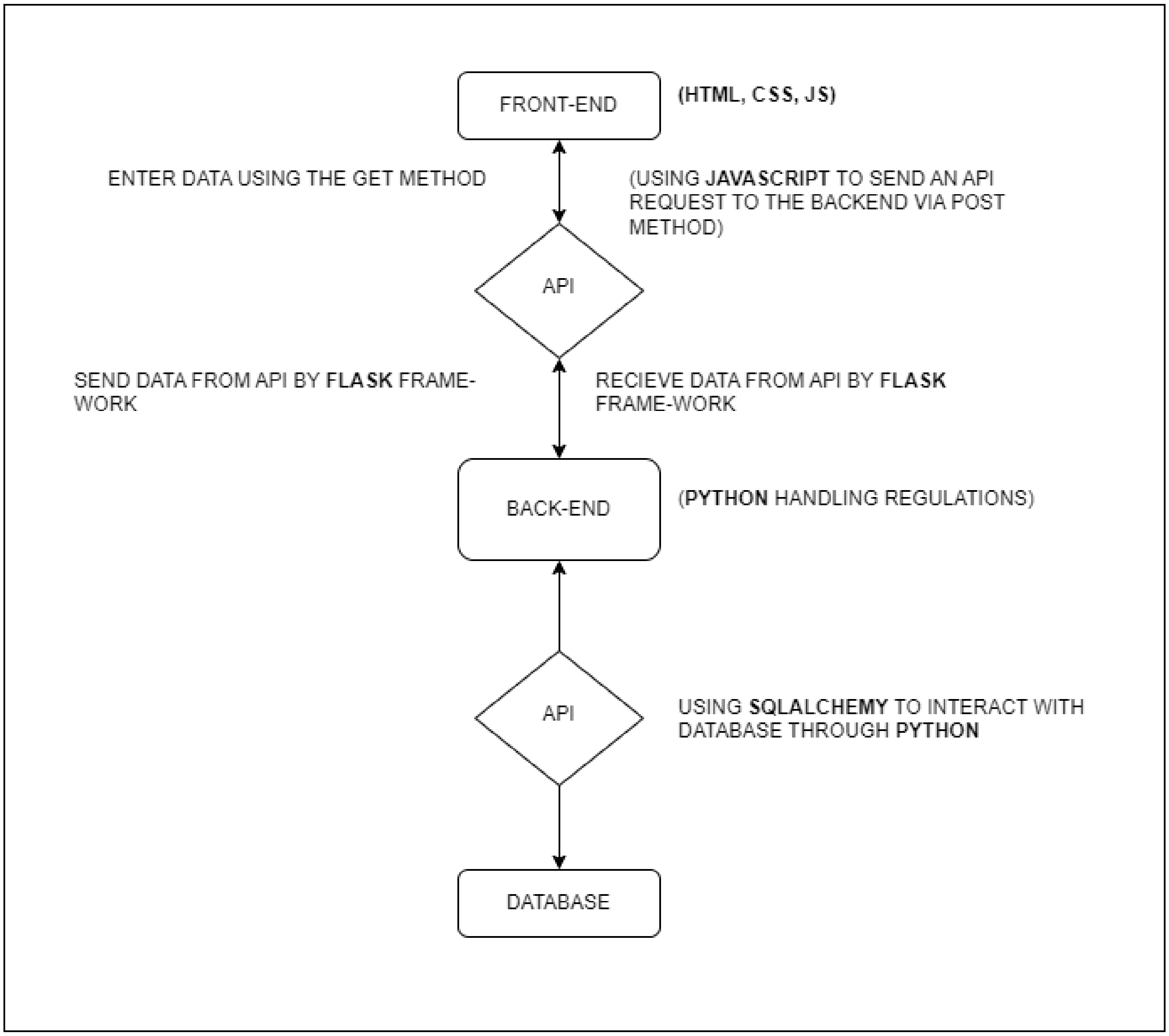
e. Monthly Report Generation Function:

- **Presentation Layer:** The user interface displaying the monthly report.
- **Business Logic Layer:** 'ReportService' processes the logic for aggregating and summarizing data.
- **Data Access Layer:** Retrieves the necessary data from the database.

f. Change Regulations Function:

- **Presentation Layer:** The user interface where users can choose to change regulations.
- **Business Logic Layer:** 'AccountService' processes the logic for changing regulations of that account.
- **Data Access Layer:** Retrieves savings account information from the database.

4. Overall architecture diagram:


FRONT-END
(HTML, CSS, JS)
ENTER DATA USING THE GET METHOD
(USING JAVASCRIPT TO SEND AN API REQUEST TO THE BACKEND VIA POST METHOD)
API
SEND DATA FROM API BY FLASK FRAME-WORK
RECIEVE DATA FROM API BY FLASK FRAME-WORK
BACK-END
(PYTHON HANDLING REGULATIONS)
API
USING SQLALCHEMY TO INTERACT WITH DATABASE THROUGH PYTHON
DATABASE

3.2 Hardware:

Criteria	Windows	macOS
Operating System	Windows 10 (latest version).	macOS Catalina (latest version) or macOS Big Sur
Processor	At least Intel Core i5 or equivalent. Examples include Core i5-8400, Core i5-10400.	At least M1
RAM	Minimum 8GB RAM. Consider 16GB RAM for multitasking and stable performance.	
Storage	Use an SSD for faster data access speed and ensure sufficient capacity to store both software and banking data. Recommended capacity starts from 256GB	
Network	Ensure stable network connectivity, including Wi-Fi and/or Ethernet, to facilitate remote data access and efficient data exchange between system components.	

4 Development plan

4.1 *Requirement Analysis*

a. Requirement Gathering:

- **Stakeholder Interviews:** Conduct interviews with bank employees, managers, and customers to understand their needs and expectations for the system.
- **Focus Groups:** Organize focus groups including representatives from different departments within the bank to discuss necessary features and priorities for the system.
- **Surveys:** Develop and distribute surveys to collect widespread opinions from the bank's customers on the functions they expect in the savings account management system.
- **JRD meeting** (Joint Requirement Development).

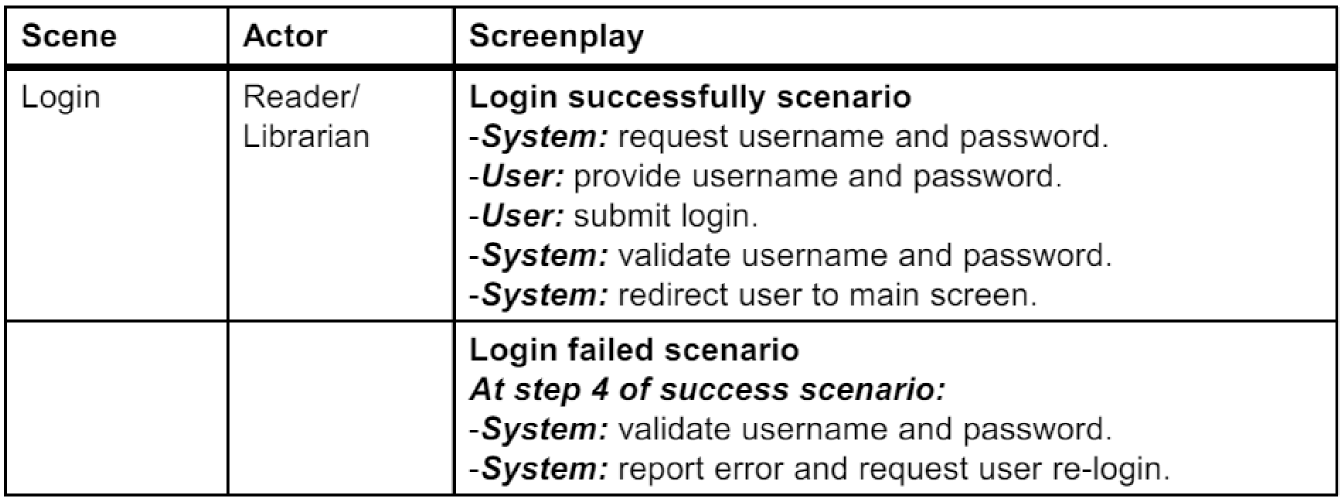
b. Requirement Identification:

- **Functional Requirements:** List all the functions that the system must have, including:
  + Opening new savings accounts.
  + Recording transactions of deposits and withdrawals.
  + Looking up savings account information.
  + Generating periodic reports on transaction activities.
  + Login and management of employee accounts.
  + Security and access control.
- **Non-Functional Requirements:**
  + **Performance**: The system must process transactions quickly and efficiently.
  + **Reliability**: The system must operate stably with minimal errors.
  + **Security**: Data encryption and user authentication to protect information.
  + **Scalability**: The system should be scalable to meet the growing needs of the bank.

c. Requirement Specification:

- Using use case Analysis to specified these functional and non-function requirement

– Example:



Scene	Actor	Screenplay
Login	Reader/ Librarian	**Login successfully scenario** -***System:*** request username and password. -***User:*** provide username and password. -***User:*** submit login. -***System:*** validate username and password. -***System:*** redirect user to main screen.
		Login failed scenario ***At step 4 of success scenario:*** -***System:*** validate username and password. -***System:*** report error and request user re-login.

d. Requirement Validation:

– Check out 4.4 for more clarify information

e. Detail timeline/plan for members:

Work	Member in charged	Timeline
Requirement gathering	Thành(play role as BA asking question), Minh, Duy, Đạt note down informations.	5/6/2024 (online meeting with mr.Khoa (customer)
Requirement identification	Thành	6/6/2024 -> 9/6/2024 (3 days)
Requirement Specification	Minh (Form 1,2), Duy (3,4) Đạt (5)	9/6/2024 ->12/6/2024
Requirement Validation	Thành(validate all requirement specification)	13/6/2024

4.2 Design software

a. Front-end (FE) Design:

- **Wireframe and Prototype:**
  - + **Wireframe:** Create basic wireframes for the main screens of the web application.
    - Key screens include:
      - Login/Register Page
      - Dashboard
      - Open Savings Account Page
      - Deposit Slip Page
      - Withdrawal Slip Page
      - Account Lookup Page
      - Monthly Report Page
      - Change Regulations Page
  - + **Prototype:** Develop a clickable prototype based on the wireframes to simulate user interaction with the interface.
- **Tools Used:**
  - + **Wireframe:** Figma.
  - + **Prototype:** Figma.

b. Back-end (BE) Design:

- **Technology Stack:**
  - + **Programming Language:** Python
  - + **Framework:** Flask
- **API Design:**
  - + **Main functions:**
    - *open_savings_account*
    - *create_deposit_slip*
    - *create_withdrawal_slip*
    - *lookup_account*
    - *change_regulations*
    - *generate_monthly_report*
  - + **API Connection:**
    - **API Approach:** RESTful API

- **API Usage:**
  - **Front-End (Client-Side):** Axios library in JavaScript to send requests.
  - **Back-End (Server-Side):** Flask framework in Python to handle requests and perform operations.

c. Database (DB) Design:

– **ER Diagram:**
  + Design ER Diagram:
    - Identify tables and relationships between them.
    - Use draw.io to create the ER diagram.

– **Key tables:**
  + *accounts*
  + *transactions*
  + *customers*
  + *reports*
  + *regulations*

– **Database Setup:**
  + **Database:** MySQL
  + **ORM:** SQLAlchemy to manage database connections.

d. Timeline Summary:

Task	Details	Assignment	Timeline
FE Design	Wireframe and Prototype using Figma	Minh	11 days (24/6-4/7)
BE Design	Setting up technology stack, API design, directory structure	Duy	11 days (24/6-4/7)
DB Design	Designing the database schema and creating the ER diagram using draw.io and MySQL setup	Đạt	11 days (24/6-4/7)
TOTAL			11 + 2 (reserve) days

4.3 Software implementation:

a. Front-End (FE) Development:

Step 1: Setting up the development environment:

- Ensure Node.js (latest version) and npm are installed.
- Set up React using Create React App (npx create-react-app project-name).
- IDE: Use Visual Studio Code (VSCode) for coding.

Step 2: Developing the User Interface:

1. **Login/Register Page:**
   - **Component:** Login and registration forms.
   - **Implementation:** Convert Figma designs into HTML/CSS.
   - **React Integration:** Develop React components for forms, validate inputs, and integrate with backend APIs for user authentication and registration.
2. **Dashboard:**
   - **Component:** Financial overview dashboard.
   - **Implementation:** Translate Figma dashboard layout into HTML/CSS.
   - **React Features:** Use React to build dynamic components, integrate with charting libraries for data visualization, and connect to backend APIs for real-time data updates.
3. **Open Savings Account Page:**
   - **Component:** Account creation form.
   - **Implementation:** Convert Figma designs into HTML/CSS.
   - **React Setup:** Develop React components for the form, handle form validation, and connect frontend to backend APIs for account creation.
4. **Deposit Slip Page:**
   - **Component:** Deposit transaction form.
   - **Implementation:** Translate Figma designs into HTML/CSS.
   - **React Implementation:** Develop React components for the form, validate input fields, and integrate with backend APIs for processing deposit transactions.
5. **Withdrawal Slip Page:**
   - **Component:** Withdrawal transaction form.
   - **Implementation:** Convert Figma designs into HTML/CSS.

- **React Integration:** Create React components for the form, implement input validation, and connect to backend APIs for handling withdrawal transactions.

6. **Account Lookup Page:**
   - **Component:** Account details view.
   - **Implementation:** Convert Figma designs into HTML/CSS.
   - **React Development:** Use React to build components for displaying account details dynamically, and integrate with backend APIs (Axios/Fetch) to fetch and render data.
7. **Monthly Report Page:**
   - **Component:** Financial report view.
   - **Implementation:** Translate Figma designs into HTML/CSS.
   - **React Features:** Develop React components to structure and present financial reports, integrate with data visualization libraries, and connect to backend APIs for fetching monthly financial data.
8. **Change Regulation Page:**
   - **Component:** Regulations management form.
   - **Implementation:** Convert Figma designs into HTML/CSS.
   - **React Implementation:** Develop React components for the form, handle user inputs, and connect to backend APIs (Axios/Fetch) to update banking regulations.

b. Back-end (BE) Development:

Step 1: Setting Up the Development Environment

- Ensure Python is installed (preferably Python 3.x).
- Install Flask (pip install Flask).
- Set up MySQL and SQLAlchemy for database operations (pip install Flask-SQLAlchemy).
- IDE: Use Visual Studio Code (VSCode) or any preferred IDE for coding.

Step 2: Developing RESTful APIs

1. **User Authentication and Registration:**
   - **Endpoints:**
     + POST /api/auth/register
     + POST /api/auth/login
   - **Functionality:**

    + Handles user registration and login processes.
    + Validates user credentials and returns authentication tokens.

2. **Dashboard Data**
   - **Endpoint:**
     + GET /api/dashboard
   - **Functionality:**
     + Retrieves data for the financial overview dashboard.
     + Provides real-time data updates for charts and financial metrics.
3. **Open Savings Account:**
   - **Endpoint:**
     + POST /api/accounts
   - **Functionality:**
     + Accepts requests to create a new savings account.
     + Validates input data and stores account details in the database.
4. **Deposit Transaction:**
   - **Endpoint:**
     + POST /api/deposits
   - **Functionality:**
     + Processes deposit transactions for user accounts.
     + Updates account balances and transaction records in the database.
5. **Withdrawal Transaction:**
   - **Endpoint:**
     + POST /api/withdrawals
   - **Functionality:**
     + Handles withdrawal transactions for user accounts.
     + Validates withdrawal amounts and updates account balances.
6. **Account Lookup:**
   - **Endpoint:**
     + GET /api/accounts/<account_id>
   - **Functionality:**
     + Retrieves account details and transaction history based on the account ID.
7. **Generate Monthly Report:**
   - **Endpoint:**

  + GET /api/reports/monthly
- **Functionality:**
  + Generates and provides financial reports for the current month.
  + Includes transaction summaries and account balances.

8. **Change regulations:**
   - **Endpoint:**
     + PUT /api/regulations
   - **Functionality:**
     + Allows updates to banking regulations stored in the database.
     + Validates regulatory changes and updates relevant data.

c. Integration and testing:

Step 1: Integration

- Ensure frontend components correctly call the respective backend APIs.
- Test API connections to ensure data is transmitted properly.

Step 2: Functionality Testing:

- Use tools like Postman for API testing.
- Test each API endpoint with valid and invalid inputs.
- Verify database updates and error handling.

d. Timeline Summary:

Task	Details	Assignment	Timeline
FE Development	Developing the UI, converting Figma designs to HTML/CSS, and building React components	Minh	16 days (7/7-22/7)
BE Development	Developing core APIs with Flask and SQLAlchemy	Duy & Đạt	16 days (7/7-22/7)
Integration and Testing	Connecting FE with BE APIs and performing comprehensive testing	Thành	6 days (25/7-30/7)
TOTAL			22 + 2 (reserve) days

4.4 *Software Testing Model*

a. Model Used for Software Testing:

– Our team utilizes the Verification and Validation (V&V) model. With this evaluation model, we can:
  + Ensure software quality
  + Minimize risks and critical software errors
  + Ensure the software fully complies with general standards
  + Facilitate easy modification and future development of the software

b. Implementation Plan for the V&V Model in Software Development:

– The Verification and Validation (V&V) model operates throughout the entire software development process, from the initial requirement gathering to the final product completion. The detailed plan for each stage is as follows:
  + **Requirements Analysis Phase**
    - **Verification**: Ensure all requirements are collected thoroughly, clearly, unambiguously, and with complete transparency. All member queries must be addressed clearly and promptly, ideally on the first day of receiving the requirements. Ensure the accuracy of the survey by gathering feedback from customers.
    - **Validation**: Review and analyze individual contributions from team members to ensure the analysis is accurate and meets the project requirements. This is achieved through team meetings and discussions.
  + **Design Phase**
    - **Verification**: Review tools and documents used for design to ensure they are appropriate and contain all necessary functions for both back-end and front-end design. Ensure the design documentation is complete, suitable, and the software will perform efficiently.
    - **Validation:** Ensure that the system design meets user requirements and customer expectations. The design can be sent to the instructor for preliminary review and validation of its suitability
  + **Implementation Phase**
    - **Verification**: Conduct source code testing at various levels such as Unit Testing and Integration Testing to ensure the source code adheres to the design and is

error-free. Ensure the implementation results align with the design, with no deviations that could lead to unexpected outcomes.

- **Validation**: Ensure that the integrated software components function correctly according to requirements. Methods such as System Testing and Alpha Testing are used to evaluate the overall system.

+ **Testing Phase**
  - **Verification**: Perform formal testing to verify that the software system complies with technical and design requirements. Activities include functional testing, performance testing, and security testing.
  - **Validation:** Ensure that the software system operates according to end-user requirements. This includes User Acceptance Testing (UAT), Beta Testing, and Usability Testing.
+ **Deployment and Maintenance Phase**
  - **Verification**: Ensure the software deployment in the real environment is error-free and that updates and fixes during maintenance do not disrupt existing functionality.
  - **Validation**: Ensure the software, when used in a real environment, fully meets user requirements. Feedback from users post-deployment will be utilized for software improvement.

c. Phân công trong việc triển khai kiểm thử phần mềm:

STT	Nội dung kiểm thử	Thời Gian	Người thực hiện kiểm tra
1	Requirement Identification	12/6	Lê Hoàng Đạt
2	Detailed Analysis	16/6	Nguyễn Phúc Thành
3	Front-End Design Review	3/7	Nguyễn Quang Minh
4	Back-End Design Review	3/7	Phạm Quang Duy
5	Database Design Review	3/7	Lê Hoàng Đạt
6	Front-End Completion Review	3/8	Nguyễn Quang Minh
7	Back-End Completion Review	3/8	Phạm Quang Duy

8	Database Completion Review	3/8	Lê Hoàng Đạt
9	Full Software Testing	15/8	Nguyễn Phúc Thành

4.5 Deployment and Maintenance

a. Software Deployment Steps:

– After completing the final review post-testing phase, proceed to the product deployment phase.

Step 1: Plan the deployment (including schedule, resources for deployment, and the deployment environment).

Step 2: Execute the deployment, install the software, run it in the real environment, and record any errors or unexpected behaviors.

b. Software Maintenance Steps:

– The maintenance phase begins after the software has been deployed and is operational. This phase aims to ensure the software continues to function effectively, fixes any arising errors, and meets user change requests. The maintenance process includes the following activities:

Step 1: Start fixing the errors recorded during the deployment process.

Step 2: Review the related functions that may be affected by the maintenance and repairs.

5 Human resources and cost

Details of human resources and associated cost

June																					July					
10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	1	2	3	4	5	6
Proposal: Nguyễn Phúc Thành																										
								Analysis: Phúc Thành, Hoàng Đạt, Quang Minh, Quang Duy																		
														FE Design: Nguyễn Quang Minh											reserve	
														BE Design: Phạm Quang Duy												
														DataBase: Lê Hoàng Đạt												

July																										
7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30			
FE Implement: Nguyễn Quang Minh																reserve										
BE: Phạm Quang Duy																										
DataBase: Lê Hoàng Đạt																										
																		Testing: Phúc Thành								