School of Computer Science and Software Engineering, Hohai University

Software Comprehensive Practice Report

Group leader's student ID	2219040109
Name	Li Jinying
Groupmates	Zhang Xin Lu Zeyuan
academic year /semester	Second semester of the 2024-2025 academic year
Major and Year	22 Dayu Computer
Instructor	Yu Lin

Table of contents

1 Overview

1.1 Project Background

With the development of information technology and the financial industry, banking operations have gradually achieved informatization and intelligent management. Traditional offline manual business processing methods are inefficient and prone to errors, failing to meet modern customers' demands for convenient, real-time, and efficient services. Therefore, developing a simple bank management system that leverages information systems to realize functions such as customer information management, bank card management, financial product purchase, and asset management is of significant practical importance.

This system is primarily designed for teaching banking business, student project training, and small-scale business simulations. Its functions are closely aligned with real banking business scenarios, with a focus on information management and basic operational processes. This facilitates students' mastery of database, front-end and back-end interaction, and system integration development capabilities.

1.2 Problem Definition

Currently, the following typical problems exist in some experimental teaching or small-scale system development scenarios:

- Manually managing customer and account information leads to disorganized data and is prone to errors.
- The management and purchase process for financial products lacks systematic support and is not standardized.
- Student experiments lack complete and systematic case studies, resulting in a disconnect between theory and practice.
- There is a lack of unified asset information management and status monitoring functions.

To address the above issues, a bank management system based on a B/S architecture was developed, providing functions such as customer information management, bank card management, financial product inquiry and purchase, and asset information inquiry and status change, with a good user experience and data reliability.

1.3 Feasibility study

- Technical feasibility

This system uses the mature Flask + MySQL technology stack, combined with HTML, CSS, and JavaScript to complete the front-end pages. The technology is simple to implement and easy to learn, making it suitable for students to develop and maintain.

- Economic feasibility

The system is developed based on open-source technology, requiring no additional hardware or software investment. It can be deployed in the existing laboratory environment, saving development costs.

- Operational feasibility

The system interface is simple and intuitive, with clear functional modules and a reasonable user operation process, making it easy for non-professional users to get started quickly, and students can also deploy and test it independently.

The system is feasible to operate. It is deployed in a local environment, relies on a MySQL database to support business data, and runs stably, meeting the needs of teaching and functional verification.

1.4 Development Environment

Tools Category	Specific configuration
operating system	Windows 10
database	MySQL 8.0
Backend development	Python 3.9 + Flask framework + PyMySQL library
Front-end development	HTML5 + CSS3 + JavaScript
Run tools	PyCharm , MySQL Workbench, and browsers (Edge/Chrome)
Testing tools	Postman, curl command line, browser developer tools

2 Requirements Analysis

2.1 introduction

2.1.1 Purpose of writing

This document aims to provide a detailed description of the overall requirements and module design of the bank management system, clarifying the system's functional scope, business processes, module division, and technical implementation approach, thus providing a complete technical basis for system development, testing, and subsequent maintenance. Through this design, we ensure that all functional modules of the system work collaboratively to meet the core business needs of bank customer information management, bank card management, financial product management, and asset management, while enhancing the system's security, stability, and scalability.

2.1.2 Project Overview

This project involves the design and implementation of a bank management system. Primarily designed for bank customers and system administrators, the system provides comprehensive functional modules for customer information management, bank card management, financial product management (covering wealth management products, insurance, funds, etc.), and customer asset management. The system adopts a modular design approach with clear division of labor, facilitating development and maintenance. Through this system, customers can self-register, log in, query information, and manage their bank cards. It supports dynamic management of financial products and real-time display of customer assets, ensuring efficient, secure, and standardized operation of business processes.

2.1.3 Abbreviations and Terms

Terms/Abbreviations	illustrate
client	(consumers) in the banking system.
bank card	Customer's linked debit or credit card in the system

wealth management products	Various wealth management and investment products offered by banks
Insurance products	Various insurance services sold by banks
Fund products	Fund investment products offered by banks
assets	Customer's holdings of financial products in the system
Module A	Customer and bank card management module
Module B	Financial Product Management Module
Module C	Purchase and Asset Management Module

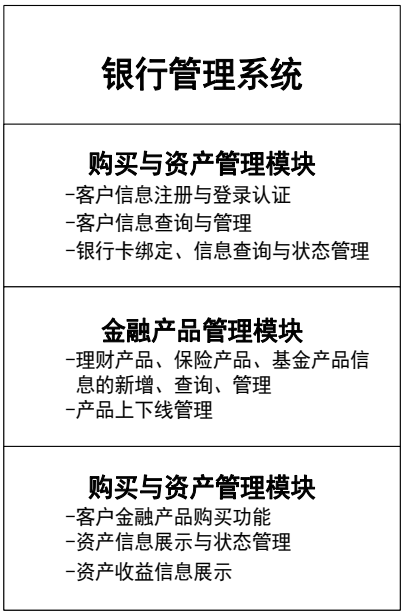
2.2 Project Overview

2.2.1 Construction goals

This project aims to design and implement a fully functional, clearly structured, and user-friendly bank management system. The system employs a modular design, dividing into multiple functional modules such as customer and bank card management, financial product management, and purchase and asset management, to meet the needs of daily bank operations and customer self-service. Specific development goals include:

- Enables unified management of customer information, supporting functions such as registration, login, and information query;
- It supports multi-card management, allowing customers to bind multiple debit or credit cards, thus enhancing account flexibility;
- Financial product information management function, supporting dynamic management of wealth management products, insurance products and fund products;
- Provides secure and reliable customer asset purchase and management functions, ensuring the accuracy and timeliness of asset information;
- The system's overall design boasts excellent scalability, security, and stability, facilitating subsequent function upgrades and maintenance.

2.2.2 Overall module diagram



2.2.3 User Role

The system is primarily designed for the following user roles [9]:

- Bank customers:

Register and log in to your personal account through the system;

Link and manage bank card information;

To check your own account information and asset information;

Purchase wealth management products, insurance products, and fund products;

- System Administrator:

Maintain system data and manage financial product information;

Overall operational status of the monitoring system;

Manage customer account status and information security.

2.3 System functional requirements

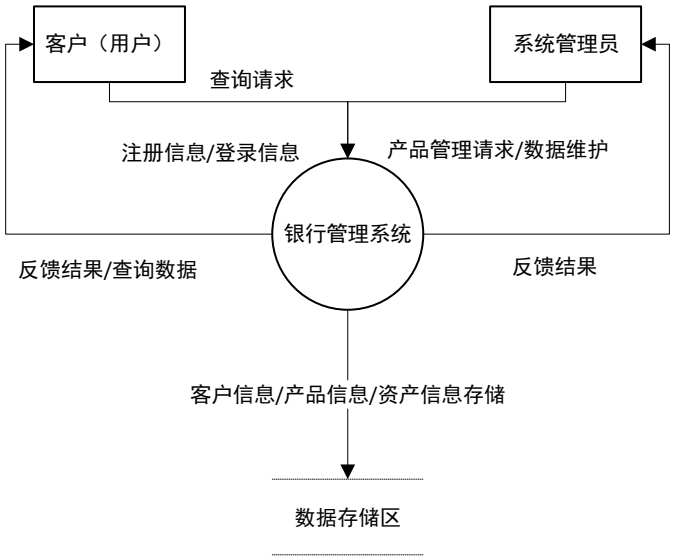
2.3.1 General Introduction

Functional module table:

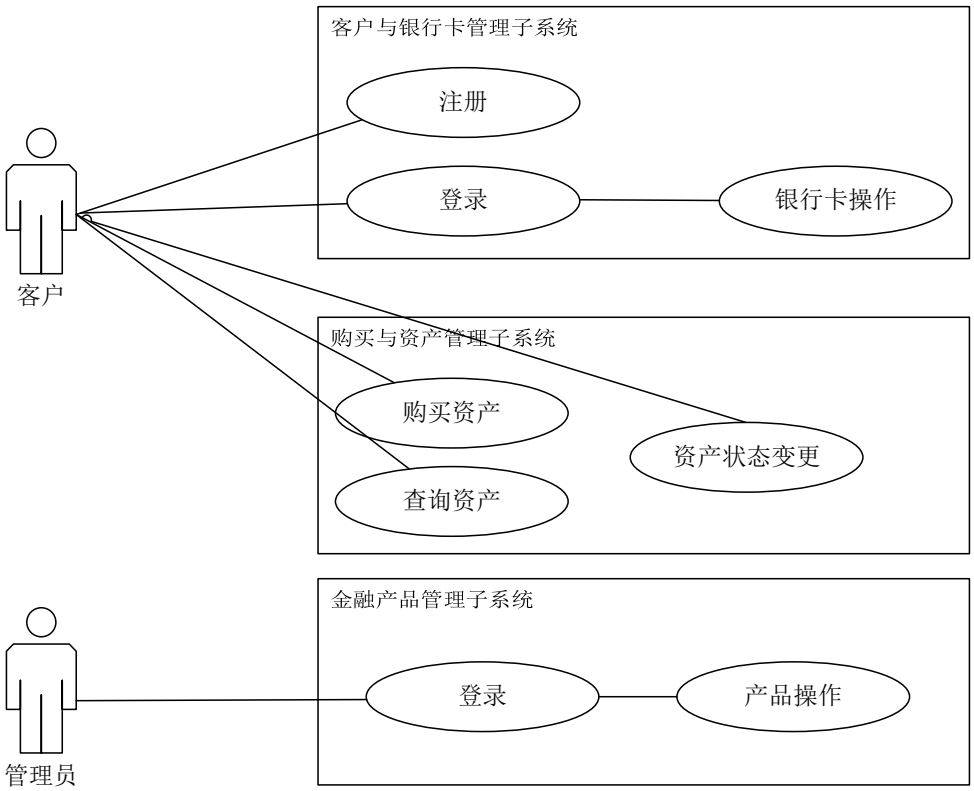
Functional modules	Sub-function	Regular users	administrator
Customer and bank card management	Information registration and authentication	★	★

Customer and bank card management	Information Inquiry and Management	★	★
Customer and bank card management	Bank card binding and management	★	★
Financial product management	Product Information Management		★
Financial product management	Product launch and decommissioning		★
Purchase and Asset Management	Purchase of financial products	★	★
Purchase and Asset Management	Asset Information Display	★	★

Top-level data flow diagram:



Use case diagrams [1][3]:



2.3.2 Data dictionary

field name	Field meaning explanation
c_id	A customer's unique identification number is used to identify the customer.
c_name	Customer's real name
c_id_card	Customer's ID number, 18 digits
c_phone	Customer's mobile phone number, 11 digits
c_mail	Customer's email address (optional), used for receiving bills, etc.
c_password	Customer login passwords must be stored in encrypted form.
b_id	The unique number of the bank card
b_c_id	Customer ID, associated customer information
b_number	Bank card number, maximum 19 digits
b_type	Bank card types, such as debit cards, credit cards, etc.
b_balance	Bank card account balance, in yuan
as_id	Unique ID of Customer Asset
as_c_id	Customer ID, associated customer information
as_pif_id	Product ID and associated financial product information

as_type	The asset-related product types are: 1- Wealth Management, 2- Insurance, 3- Funds
as_quantity	Quantity of this product purchased by the customer
as_status	Current status of the asset, such as "normal" or "frozen".
as_income	The client's current accumulated earnings on this asset are in yuan.
as_purchase_time	When did the customer purchase the asset?
pi_id	The unique identifier of a financial product, uniformly identified within the system.
pi_name	The product name is used for display.
pi_amount	The purchase amount or minimum investment amount for the product is in yuan.
pi_period	Product shelf life, in years.
pi_rate	Annualized rate of return, expressed as a percentage
pi_type	Product types: 1-Wealth management products, 2-Insurance products, 3-Fund products
pi_state	Product status: 1 - Available for sale, 2 - Discontinued
p_description	Extended description of wealth management products
i_target	The insurance is applicable to the following groups: 1-Children, 2-Adults, 3-Seniors.
i_project	Text description of insurance coverage items
f_category	Fund category codes, such as 1-mixed fund, 2-equity fund.
f_risk_level	Fund risk levels: 1 - High risk, 2 - Medium risk, 3 - Low risk
f_manager	Name of the fund's management institution or person in charge (maximum 50 characters)

2.3.3 Customer and Bank Card Management Module

2.3.3.1 General Description

This module primarily implements customer information management and bank card information management functions within the banking system, providing operations such as customer registration, login authentication, account information inquiry, and bank card binding, querying, and status management. Through this module, the system can ensure the integrity and accuracy of basic customer information , support the management of multiple bank cards, and provide user identity and bank card information support for subsequent financial transactions.

Function Name	enter	Processing	Output
Customer	Name, ID number, mobile	Verify the	Registration

registration	phone number, email address, password	uniqueness of the information → Encrypt and store the password → Write the password to the customer information table	success/failure message
Customer Login	Account (ID card number/mobile phone number/bank card number), password	Account information verification → Login status confirmation	Login result; if successful, permission information will be returned.
Customer Information Inquiry	Customer Number	Login status verification → Query customer information	Return customer basic information
Bank card binding	Customer ID, Bank Card Type	Verify customer existence → Generate card number → Write to bank card information table	Binding result feedback
Bank card information inquiry	Customer Number	Login status verification → Query all bank card information	Return to bank card list information
Bank card status management	Customer ID, Bank Card Number, New Status (Normal/Frozen/Cancelled)	Verify permissions and status validity → Update bank card status	Status change results

2.3.3.2 Customer information registration and login authentication

Customers can register accounts independently through the system. Registration information includes name, ID number, mobile phone number, email address, and login password. The system verifies the uniqueness of registration information to prevent duplicate accounts. Login supports ID number, mobile phone number, or bank card number as the account name, combined with a password for identity authentication. If login authentication fails more than a limited number of times, the system can lock the account to ensure security.

2.3.3.3 Customer Information Inquiry and Management

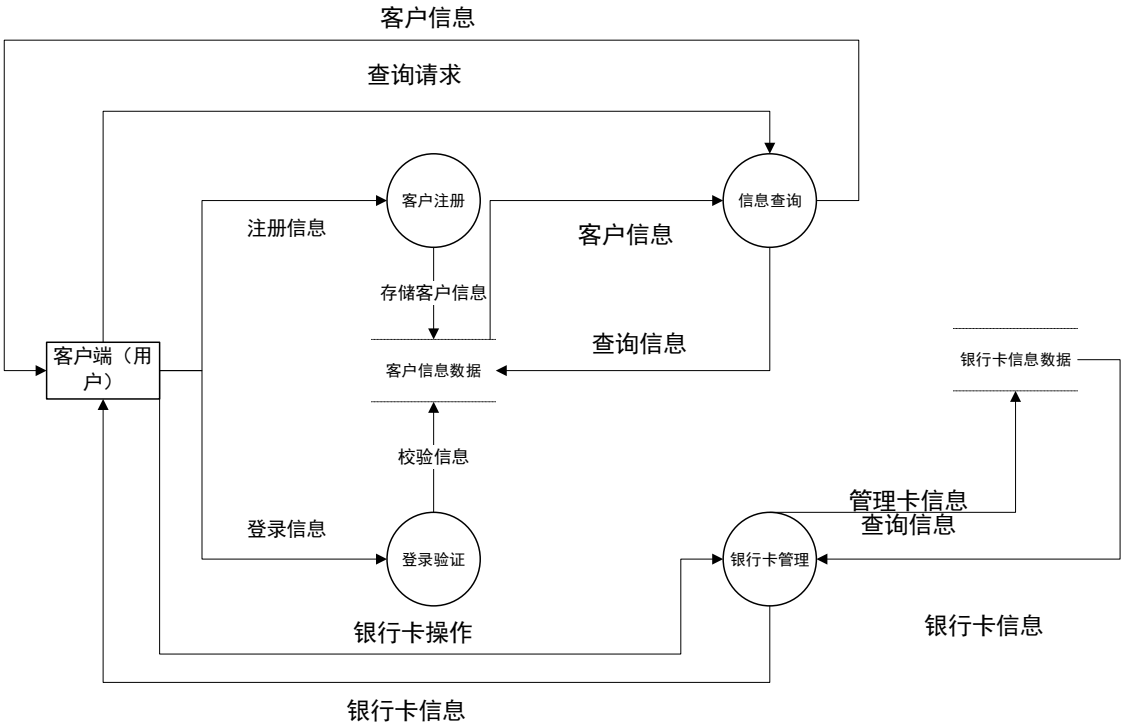
After successful login, customers can independently query their account information, including name, contact information, and account status. The system provides an

information display interface to ensure information accuracy and visualization. An information modification interface is reserved within the system (only accessible to administrators or in subsequent extended versions).

2.3.3.4 Bank card binding, information inquiry and status management

After successful login, customers can bind multiple bank cards, supporting both debit and credit cards. Card numbers are automatically generated by the system to ensure uniqueness. Customers can view information for all their bound bank cards. The system supports bank card status management, including normal, frozen, and cancelled statuses; certain operations are restricted under special statuses.

2.3.3.5 Data Flow Diagram



2.3.4 Financial Product Management Module

2.3.4.1 General Description

This module primarily implements financial product management functions, including adding, querying, and deleting products, as well as modifying product information. The module aims to ensure that administrators can smoothly complete the financial product management process and accurately manage and view various information for all their products .

Function Name	enter	Processing	Output
New	Product name,	Check if product name is	Add success / failure

Products	product type , product attributes	duplicated → Save product information → Generate product ID → Record	notifications and generate product ID.
Product Inquiry	Product ID | Product Name	Query all attribute information of this product , including the number of buyers, amount, category, etc.	Return product attribute information
Product deletion	Product ID | Product Name	Verify product information and user permissions → Confirm deletion → Delete product record	Deletion success / failure message
Product Information Modification	Product ID | Product Name	Modify the corresponding attributes according to different types of attribute requirements.	for success / failure of modification, and output the new product attribute information.

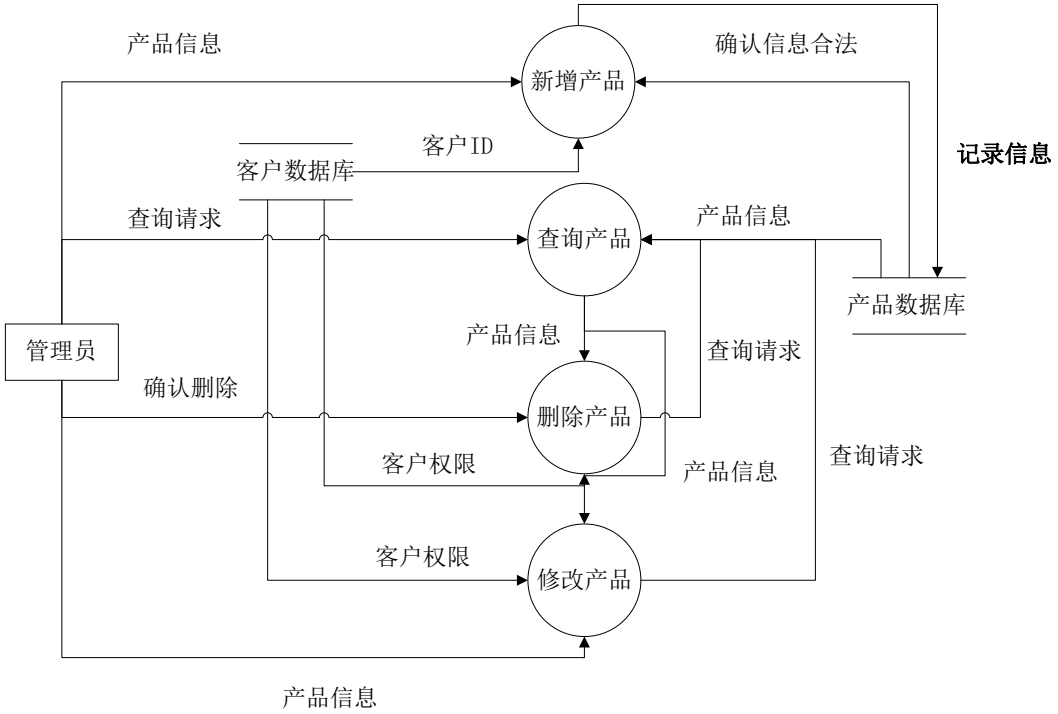
2.3.4.2 Information management of wealth management products, insurance products, and fund products

It supports the entry of information for various financial products, including product number, name, description, amount, term, risk level, and applicable population. Product information supports management operations such as querying, modifying, and deleting. Insurance products and fund products can be configured with different attributes and filtering conditions to meet diverse business needs.

2.3.4.3 Product production and decommissioning management

System administrators can control the online and offline status of products. Online products are visible to customers, who can choose to purchase them based on their needs. Offline products are no longer available to customers, but historical purchase records are retained to ensure data integrity.

2.3.4.4 Data Flow Diagram



2.3.5 Purchase and Asset Management Module

2.3.5.1 General Description

This module primarily enables customers to purchase wealth management products, insurance, and funds within the financial management system, as well as manage and query their asset status. The module aims to ensure customers can smoothly complete the purchase process for financial products and accurately manage and view various information about their held assets.

Function Name	enter	Processing	Output
Product Purchase	Customer ID, Product ID, Product Type , Purchase Quantity	Verify customer identity → Verify product availability → Verify account status → Save purchase history	Purchase success / failure notification , asset record generation
Asset Inquiry	Customer ID	Query all asset records for this customer , including status , earnings , quantity , and time.	Return to asset list
Asset status update	Customer ID, Product ID, New Status	Update the status of the corresponding assets (e.g., frozen , restored) .	Status update results

Profit Calculation	Customer ID	Obtain the return calculation results for each asset.	Revenue Display (categorized by product type)

2.3.5.2 Customer financial product purchase function

After logging in, customers can browse the available financial products and select suitable products to purchase. Before purchasing, the system verifies the account status, bank card status, and fund availability. The system records information for each purchase, including product category, quantity, time, and estimated return.

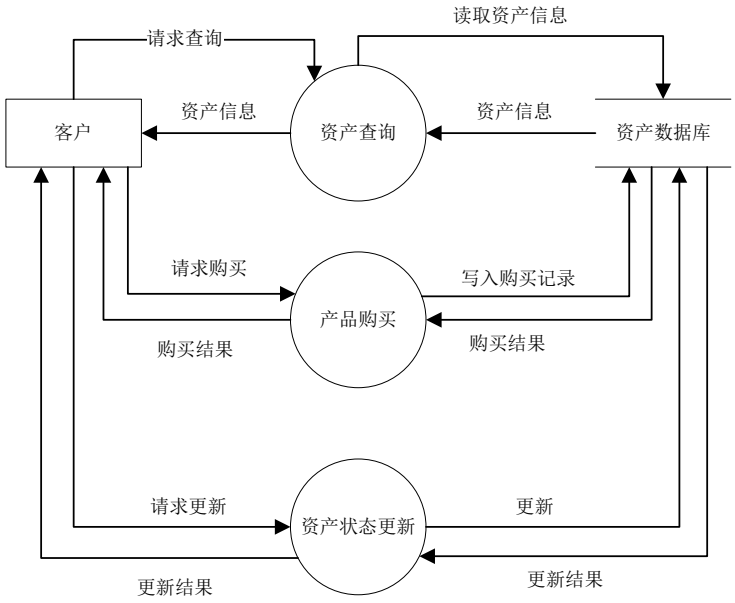
2.3.5.3 Asset Information Display and Status Management

Customers can inquire about their asset information at any time, including product category, quantity, status, and purchase date. Asset status supports management logic such as "normal" and "frozen." When assets are frozen, related operations are restricted to ensure business security.

2.3.5.4 Asset return information display

The system dynamically calculates asset returns based on product attributes and purchase information. Customers can view their asset returns in real time through the interface. The system provides detailed and summary displays of returns, making it easy for customers to understand their investment returns.

2.3.5.5 Data Flow Diagram



2.4 Non-functional requirements of the system

In addition to functional requirements, bank management systems must also meet the following non-functional requirements:

Stability and Reliability

The system should operate stably for a long time to ensure that each functional module continues to provide services under normal conditions. Critical operations should have fault tolerance mechanisms to avoid system unavailability due to failures.

- Security

All sensitive information (such as user passwords, ID card numbers, and bank card information) must be stored and transmitted in encrypted form. The system should use methods such as identity authentication, access control, and data verification to prevent unauthorized access and data leakage.

- Performance requirements

The system should have a fast response time, with registration, login, information query, and purchase operations completed within a reasonable time. It should also support concurrent operations by multiple users to ensure smooth system operation during peak periods.

- Scalability and Maintainability

The system modules are clearly divided, have good scalability, facilitate function upgrades and business expansion, and have a reasonable code structure, making it easy to maintain and optimize in the future.

- Ease of use

The system interface is simple and the operation process is user-friendly, allowing users to get started quickly and providing a good overall user experience, thus reducing the difficulty of use.

- Data accuracy and consistency

During data storage, transmission, and processing, the system must ensure data integrity and accuracy to prevent information loss and data inconsistency.

3 Preliminary Design

3.1 introduction

3.1.1 Purpose of writing

This preliminary design report aims to further clarify the overall system structure and functional division of each module based on the requirements analysis, providing clear technical guidance for subsequent detailed design and system implementation. Through standardized descriptions of key aspects such as system architecture, module interfaces, and database structure, this report ensures the development team has a consistent understanding of the system's core functions and module responsibilities, reducing communication costs and implementation deviations during development. Furthermore, this report provides a traceable design basis for project implementation, testing, maintenance, and iterative upgrades.

3.1.2 Abbreviations and Terms

Abbreviations, terms	explain
client	(consumers) in the banking system.
bank card	Customer's linked debit or credit card in the system
wealth management products	Various wealth management and investment products offered by banks
Insurance products	Various insurance services sold by banks
Fund products	Fund investment products offered by banks
assets	Customer's holdings of financial products in the system
Module A	Customer and Bank Card Management Module
Module B	Financial Product Management Module

Module C	Purchase and Asset Management Module

3.2 System Architecture

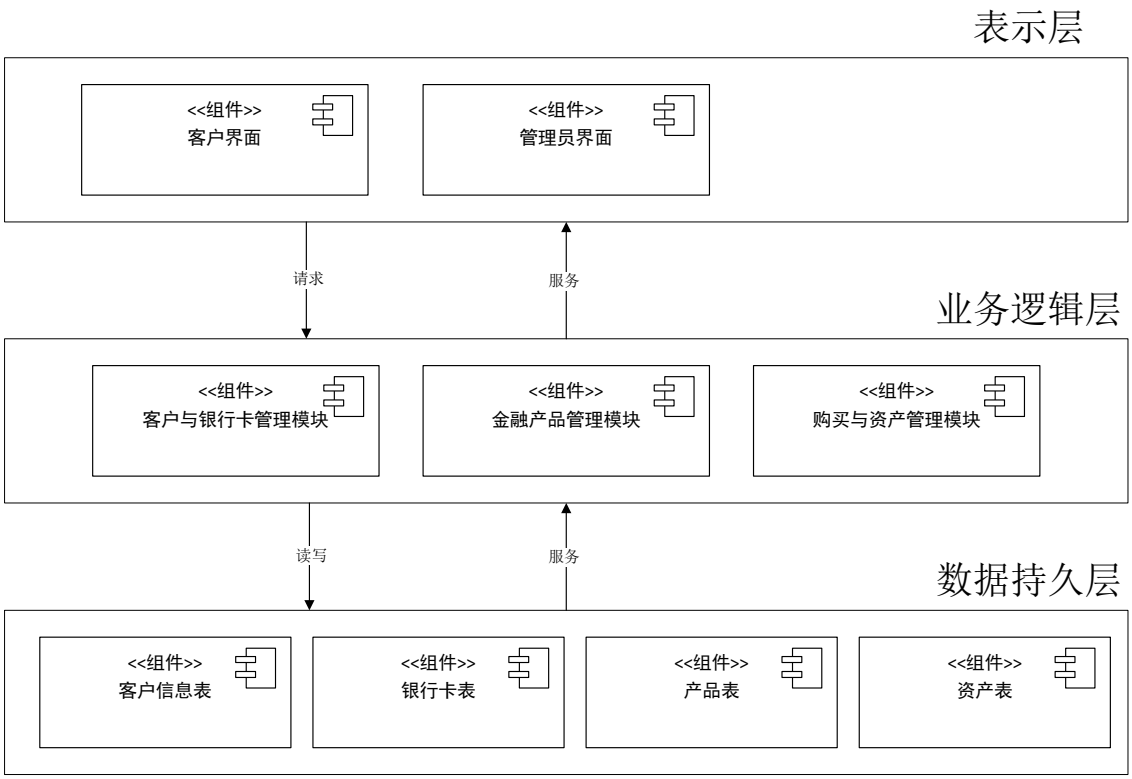
3.2.1 Overall architecture description

To achieve the goals of clear functionality, module decoupling, and ease of expansion and maintenance, this financial management system adopts a layered architecture design, dividing the system into three main layers: the presentation layer (user interaction layer), the business logic layer (service processing layer), and the data persistence layer (database operation layer). The overall system architecture is clear, and the modules interact and call functions through well-defined interfaces, avoiding direct coupling between modules and improving the system's scalability and maintainability.

- In the presentation layer, the system provides different operation entry points for bank customers and administrators respectively. Customers can use this layer to register and log in, check bank cards and assets, purchase products, and freeze assets; administrators can perform management operations such as adding, modifying, and delisting financial products. The presentation layer passes all input requests to the business logic layer, which is responsible for handling the core functions.
- The business logic layer is the core of the system, and it is divided into three independent business modules according to their functions: customer and bank card management module, financial product management module, and purchase and asset management module.
- The data persistence layer is responsible for reading and writing all structured data in the system, including customer information tables, bank card tables, financial product tables, and asset record tables. Each business module interacts with the database through a Data Access Object (DAO), ensuring data consistency, security, and persistence.

Through a hierarchical and complementary system architecture design, this financial management system achieves good functional integration while maintaining the independence of its modules, meeting the dual needs of internal bank management and customer service.

The following is the overall architecture diagram [1][2]:



3.2.2 Application Architecture

The application architecture of this system is modularly designed around the core functions of banking operations, and is mainly divided into three key functional modules:

- The customer and bank card management module is responsible for customer information registration, login, information modification, bank card binding, and status management. This module maintains the many-to-many relationship between customers and bank cards, ensuring data consistency and information security.
- The financial product management module provides a management interface for bank administrators, supporting CRUD operations on wealth management, insurance, and fund products. This module is a prerequisite for customers to purchase products; other modules require this module to retrieve product information.
- The Purchase and Asset Management module provides clients with an entry point to purchase financial products, processes the purchase process, and generates asset records. This module is also responsible for maintaining and querying asset holding information, status, and returns. It interacts with the Financial Products module via an interface to obtain product validity and detailed information.

Modules collaborate through standard function interfaces or data access interfaces, avoiding direct coupling. For example, the purchasing module needs to call the product verification interface in the product management module; the bank card management module calls the identity verification interface in the customer module.

3.2.3 Technical Architecture

This financial management system adopts a modular development approach, primarily using Java to write the backend logic and MySQL as the data storage solution, supporting transaction processing and multi-table join queries. The system does not include a front-end graphical interface; module functionality is verified and demonstrated mainly through command-line interaction or backend simulation calls.

Inter-module communication uses function calls, with operation results uniformly encapsulated in structure return values. The data access layer employs the DAO (Data Access Object) design pattern to encapsulate underlying database operations, improving system maintainability. The entire system is deployed on a local computer environment, using a simple compilation and execution process for module debugging and integration.

The project development process utilizes version control tools (Git) for team collaboration, and ER diagram modeling tools (Visio) are used for auxiliary modeling during the database structure design phase . The overall system architecture is simple and clear, and the technology stack is suitable for student project development and demonstration.

3.3 Functional design

3.3.1 Customer and bank card management module

3.3.1.1 Function Overview

The Customer and Bank Card Management module is a fundamental customer-facing functional module within the bank management system. It primarily handles the unified management of customer account information and the binding and maintenance of multiple bank cards. This module supports online customer registration, login authentication, information inquiry, bank card binding, information viewing, and status control, ensuring the integrity, uniqueness, and security of customer identity information and bank card data within the system. Through this module, the system accurately records basic customer information, including name, ID number, contact information, email address, and encrypted passwords. It supports customers in independently applying to bind debit or credit cards and allows real-time monitoring of the status, card number, and account balance of all bank cards. This module also features bank card status management, allowing customers to proactively freeze or cancel bank cards under specific circumstances, enhancing the system's control over the lifecycle of bank card usage.

3.3.1.2 Business Rules

This module follows these business rules during its implementation：

- Uniqueness of identity information: When customers register, the system must ensure the uniqueness of ID card number, mobile phone number, and email address to prevent duplicate accounts;
- Login security mechanism: Login authentication requires verification of the correctness of the account (ID card number/mobile phone number/bank card number) and password. Passwords are stored in encrypted form and must not be disclosed in plaintext.
- One-to-many relationship management: The system supports binding multiple bank cards under a customer's name, with each bank card information recorded independently and the card number unique;
- Bank card category differentiation: Bank card types are clearly distinguished into debit cards and credit cards, and business logic is adjusted according to the differences in type;
- Status control mechanism: Bank card status is divided into three categories: normal, frozen, and canceled. Bank cards in the frozen or canceled status are prohibited from participating in transactions and binding operations.
- Data consistency assurance: The addition, modification, and status update operations of customer information and bank card information must be atomic to ensure that data is not lost or conflicted;
- Operation permission restrictions: All information inquiries and bank card management operations can only be executed after the customer has successfully logged in, to ensure account security;

3.3.1.3 Key Implementation Points (Submodule Division)

Customer Information Management submodule

is responsible for registering, storing, verifying the uniqueness of, and querying customer information.

Input interface:

- registerCustomer (name, idCard , phone, email, password)
- getCustomerInfo (customerId)

Internal dependency interface: checkAccountExists (idCard , phone, email)

3.3.1.3.1 Login verification module

It provides account password verification and login authentication functions, and supports three login methods: ID card number, mobile phone number or bank card number.

Input interface: login(account, password)

Internal dependency interface: validateAccount (account , password)

3.3.1.3.2 Bank card management module

It is responsible for binding bank cards, querying information and managing status, and supports the differentiation between debit cards and credit cards.

Input interface:

- bindBankCard (customerId , cardType)
- queryBankCards (customerId)
- updateCardStatus (customerId , cardId , newStatus)

Internal dependency interfaces:

- generateCardNumber ()

- checkCustomerActive (customerId)

3.3.1.4 Key Implementation Points (Interface Design)

This section provides a detailed design of the interface functions defined in the three sub-modules mentioned above , covering their functional descriptions , parameter definitions , return value formats, and calling dependencies . All interface functions adopt explicit input/output specifications to ensure low coupling and high cohesion between modules .

3.3.1.4.1 registerCustomer(name, idCard, phone, email, password)

Functions: Customer registration, verification of information uniqueness, encrypted password storage, and writing of customer information.

Return value: Operation result (reason for success/failure)

3.3.1.4.2 login(account, password)

Function: Customer login authentication, verify account and password, and return login result.

Return value: Login result (reason for success/failure)

3.3.1.4.3 getCustomerInfo(customerId)

Function: Query basic customer information and return account information data.

Return value: Account information data

3.3.1.4.4 bindBankCard(customerId, cardType)

Function: Bind a new bank card, generate a unique card number and store the information.

Return value: Card binding result (reason for success/failure)

3.3.1.4.5 queryBankCards(customerId)

Function: Query all bank card information linked to a customer.

Return value: Bank card information

3.3.1.4.6 updateCardStatus(customerId, cardId, newStatus)

Function: Modify bank card status, supports freezing and cancellation.

Return value: Modification result (reason for success/failure)

3.3.1.5 Key Implementation Points (Internal Methods)

3.3.1.5.1 checkAccountExists(idCard, phone, email)

Function: Verify the uniqueness of customer registration information to prevent duplicate registrations.

Return value: Whether it is unique

3.3.1.5.2 validateLogin(account, password)

Function: Verifies the validity of login account and password, with encapsulated internal logic.

Return value: Whether it is valid

3.3.1.5.3 generateCardNumber()

Function: The system automatically generates a unique bank card number to ensure that the card number is not duplicated.

3.3.1.5.4 checkCustomerActive(customerId)

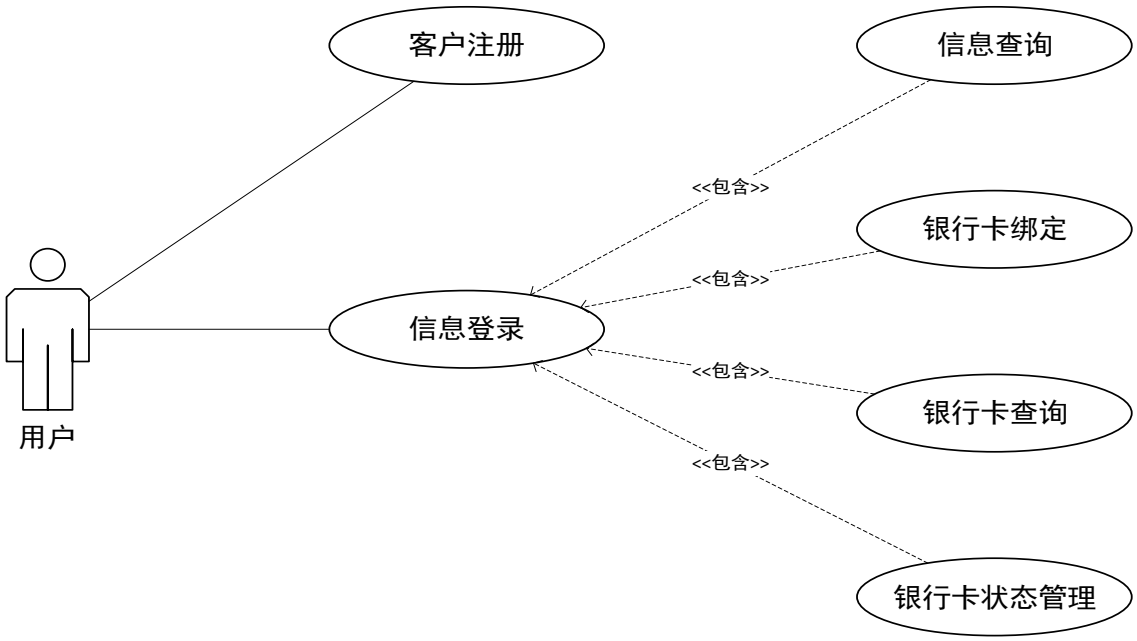
Function: Verify the status of customer accounts to ensure that the accounts are in a normal state.

Return value: Whether the system is in a normal state.

3.3.1.5.5 saveCardData()

Function: Internally saves bank card information to the database.

3.3.1.6 Use Case Diagram



3.3.2 Financial Product Management Module

3.3.2.1 Function Overview

, directly accessible to bank administrators. It primarily handles the bank's operations for adding, deleting, modifying , and querying financial products, as well as the unified management of existing products . Customers can use this module to request the

addition of products . Upon successful verification , the system generates a product addition record and stores it in the database . Users with sufficient permissions can delete product information stored in the database . Customers can also query detailed information about their existing products at any time , including product category , amount , and term .

3.3.2.2 Business Rules

This module follows these business rules during its implementation :

- Identity verification mechanism : All financial product management operations must be performed after the user has successfully logged in . The system must verify the user's identity to ensure data security .
- Product operability verification : During deletion operations , the system must verify the existence of the target product and the user's permissions to ensure system security.
- Unique Product Record Generation : Each product addition operation generates a unique product record , automatically generating the product ID , and automatically recording the customer ID, addition time, and product information .
- Product information is available: Any user can query detailed product information by product ID or product name .

3.3.2.3 Key Implementation Points (Submodule Division)

3.3.2.3.1 Product Add Module

This module is responsible for receiving and processing administrator requests to add financial products . It first verifies the user's identity and product information , and then generates and writes the product record .

Input interface :

addProduct (customerId , productName , productType , money)

Output interface :

Pending

Dependency interfaces :

- Calling the product module getProduct (productId , productName)
- Calling the user module verifyCustomer (customerId)

3.3.2.3.2 Product Inquiry Module

This submodule supports comprehensive query functionality for all users regarding their existing products . Users can obtain detailed information on all products, including financial products , insurance, and funds .

Input interface :

getProduct (productId , productName)

Dependency interfaces :

It only depends on the asset database and has no cross-module dependencies.

3.3.2.3.3 Product Modification Module

This submodule allows users to actively manage asset status and modify product details such as amount , term , and description .

Input interface :

updateProduct (customerId , productId , productName , productType)

Dependency interfaces :

- Calling the product module getProduct (productId , productName)
- Calling the user module verifyCustomer (customerId)

3.3.2.3.4 Product deletion module

This module is responsible for receiving and processing administrator requests to delete financial products . It first verifies the user's identity and product information , and then deletes the product from the product database .

Input interface :

deleteProduct (customerId , productId , productName)

Dependency interfaces :

- Calling the product module getProduct (productId , productName)
- Calling the user module verifyCustomer (customerId)

3.3.2.4 Key Implementation Points (Interface Design)

This section provides a detailed design of the interface functions defined in the three sub-modules mentioned above , covering their functional descriptions , parameter definitions , return value formats, and calling dependencies . All interface functions adopt explicit input/output specifications to ensure low coupling and high cohesion between modules .

3.3.2.4.1 addProduct(customerId, productName, productType, money)

Function : Processes requests from bank customers to add financial products , and performs operations such as identity verification , product validation , and product record creation .

Input parameters :

customerId : String , customer ID

productName : String , product name

productType : String , product type (financial management / fund / insurance)

money: A floating-point number , representing the amount.

Return value :

Operation result (reason for success / failure)

Dependency interfaces :

getProduct (productId , productName) (from the product module)

verifyCustomer (customerId) (from the user module)

3.3.2.4.2 getProduct(productId, productName)

Function : Query and return all existing product records for users to view their current product details .

Input parameters :

productId : String , product ID

productName : String , product name

Return value :

Query results (reasons for success / failure)

The data structure is a list of all assets under a customer's name , including fields such as asset number , product name , type , quantity , status , earnings , and purchase time .

3.3.2.4.3 updateProduct(customerId, productId, productName, productType)

Function : Modify the basic information of the product .

Input parameters :

customerId : String , customer ID

productId : String , product ID

productName : String , product name

productType : String , product type (financial management / fund / insurance)

Return value :

Update result (success / failure)

Dependency interfaces :

getProduct (productId , productName) (from the product module)

verifyCustomer (customerId) (to ensure the user's action is legitimate)

3.3.2.4.4 deleteProduct(customerId, productId, productName)

Function : Processes deletion requests for bank customers' financial products , and completes operations such as identity verification , product verification , and deletion of product records .

Input parameters .

Input parameters :

customerId : String , customer ID

productId : String , product ID

productName : String , product name

Return value :

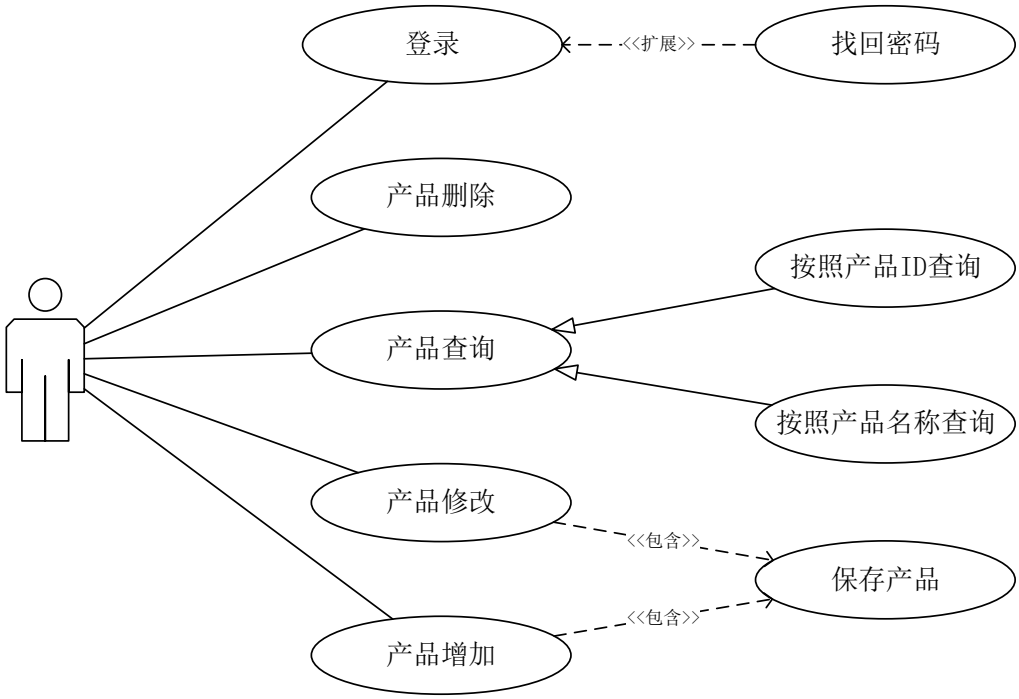
Update result (success / failure)

Dependency interfaces :

getProduct (productId , productName) (from the product module)

verifyCustomer (customerId) (to ensure the user's action is legitimate)

3.3.2.5 Use Case Diagram



3.3.3 Purchase and Asset Management Module

3.3.3.1 Function Overview

The Purchase and Asset Management module is a crucial customer-facing functional module within the financial management system. It primarily handles customer purchases of various financial products (including wealth management products, insurance, and funds) and the unified management of purchased assets. Customers can complete product purchase requests through this module, and the system generates asset records and stores them in the database upon successful verification. Customers can also query detailed information about their assets at any time, including asset status, quantity, returns, and purchase time. This module also provides asset status update functionality, supporting the setting of an asset to "frozen" or "unfrozen," thereby enhancing the system's control over the asset lifecycle. This module has an interface dependency with the Product Information Management module, requiring standard interface calls to retrieve product information for verification and display.

3.3.3.2 Business Rules

This module follows these business rules during its implementation :

- Identity verification mechanism: All purchase and asset management operations must be performed after the user has successfully logged in. The system must verify the user's identity to ensure data security.
- Product validity verification: During the purchase process, the system must query the existence and purchase status of the target product (e.g., not offline) through the interface, and verify key information such as its type, amount, and purchase limit.

- Unique asset record generation: Each purchase operation corresponds to a unique asset record, which must include customer ID, product ID, product type, purchase quantity, status (initially " available "), purchase time, etc.
- Asset status is controllable: Customers can actively freeze / unfreeze their assets, and assets under freeze status should be prohibited from any fund operations.
- The "Profit" field is read-only: Asset profits are calculated and generated by the system and cannot be modified by the customer; they can only be read and displayed.
- Data consistency requirements: Operations such as purchase records and status updates must be completed in an atomic manner to avoid data inconsistencies caused by partial submissions.

3.3.3.3 Key Implementation Points (Submodule Division)

3.3.3.3.1 Product purchase processing module

This module is responsible for receiving and processing customer purchase requests for financial products (wealth management , funds , insurance) . It first verifies the customer's identity and product information , and then generates and writes the asset records .

Input interface :

purchaseProduct (customerId , productId , productType , amount)

Dependency interfaces :

Calling the product module validateProductInfo (productId , productType)
Calling the user module verifyCustomer (customerId)

3.3.3.3.2 Customer Asset Inquiry Module

This submodule supports customers in comprehensively querying their asset holdings . Through this module, customers can obtain detailed information on all assets , including wealth management products , insurance, and funds , facilitating subsequent return analysis and financial decision-making .

Input interface :

getAssetListByCustomer (customerId)

Dependency interfaces :

It only depends on the asset database and has no cross-module dependencies.

3.3.3.3.3 Asset Status Management Module

This submodule allows customers to proactively manage asset status , such as setting assets to " frozen " to restrict their circulation , or restoring them to " available " to reactivate them . Status change functionality is a crucial component of asset security control , particularly suitable for sensitive periods (such as account anomalies or reported losses).

Input interface :

updateAssetStatus (customerId , assetId , newStatus)
Dependency interfaces:

Calling the user module verifyCustomer (customerId)

3.3.3.4 Key Implementation Points (Interface Design)

This section provides a detailed design of the interface functions defined in the three sub-modules mentioned above , covering their functional descriptions , parameter definitions , return value formats, and calling dependencies . All interface functions adopt explicit input/output specifications to ensure low coupling and high cohesion between modules .

3.3.3.4.1 purchaseProduct(customerId, productId, productType, amount)

Function : Processes customer purchase requests for financial products and performs operations such as identity verification , product validation , and asset record creation .

Input parameters :

customerId : String , customer ID

productId : String , product number

productType : String , product type (financial management / fund / insurance)

amount: A floating-point number , representing the purchase amount or number of units.

Return value :

Operation result (reason for success/failure)

Dependency interfaces :

validateProductInfo (productId , productType) (from the product module)

verifyCustomer (customerId) (from the user module)

3.3.3.4.2 getAssetListByCustomer(customerId)

Function: Queries and returns all asset records held by a customer, allowing the customer to view their personal asset details.

Input parameters:

customerId : String, customer ID

Return value:

Query results (reasons for success/failure)

The data structure is a list of all assets under a customer's name , including fields such as asset number , product name , type , quantity , status , earnings, and purchase time.

3.3.3.4.3 updateAssetStatus(customerId, assetId, newStatus)

Function: Modify the status of a customer's asset (such as frozen or unfrozen).

Input parameters:

customerId : String, customer ID

assetId : Integer, asset record number

newStatus : String (frozen/available)

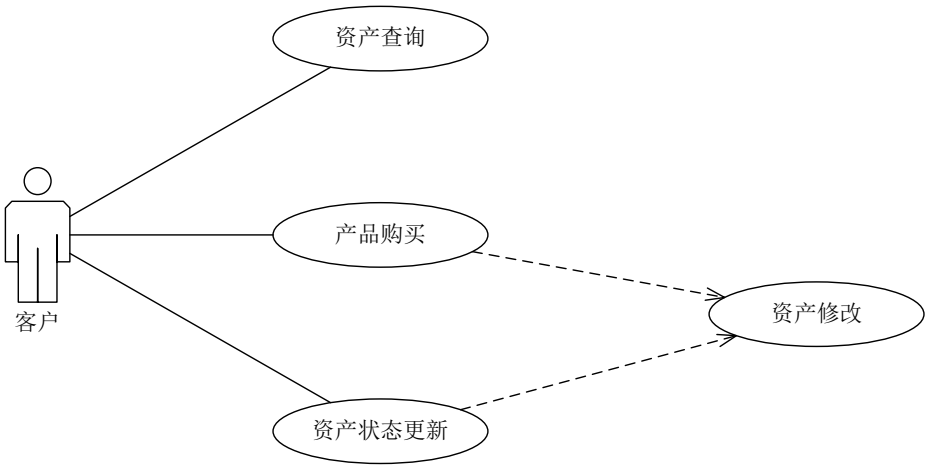
Return value:

Update result (success/failure)

Dependency interfaces:

verifyCustomer (customerId) (to ensure the customer's actions are legitimate)

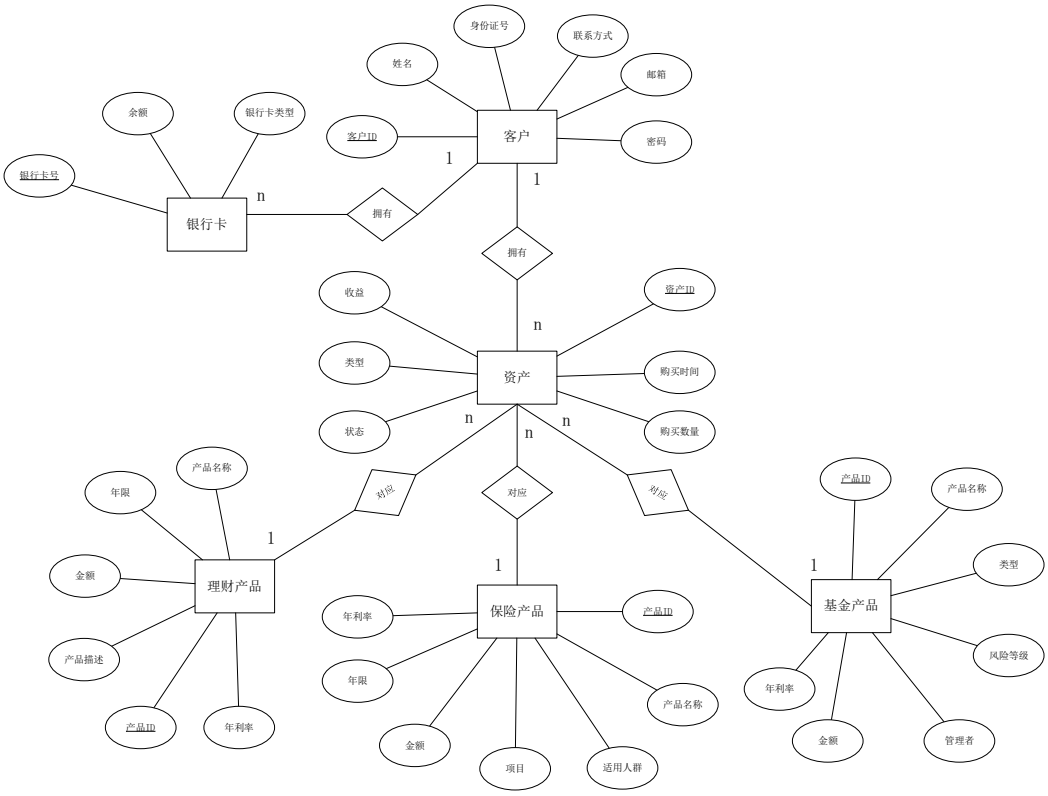
3.3.3.5 Use Case Diagram



3.4 Database design

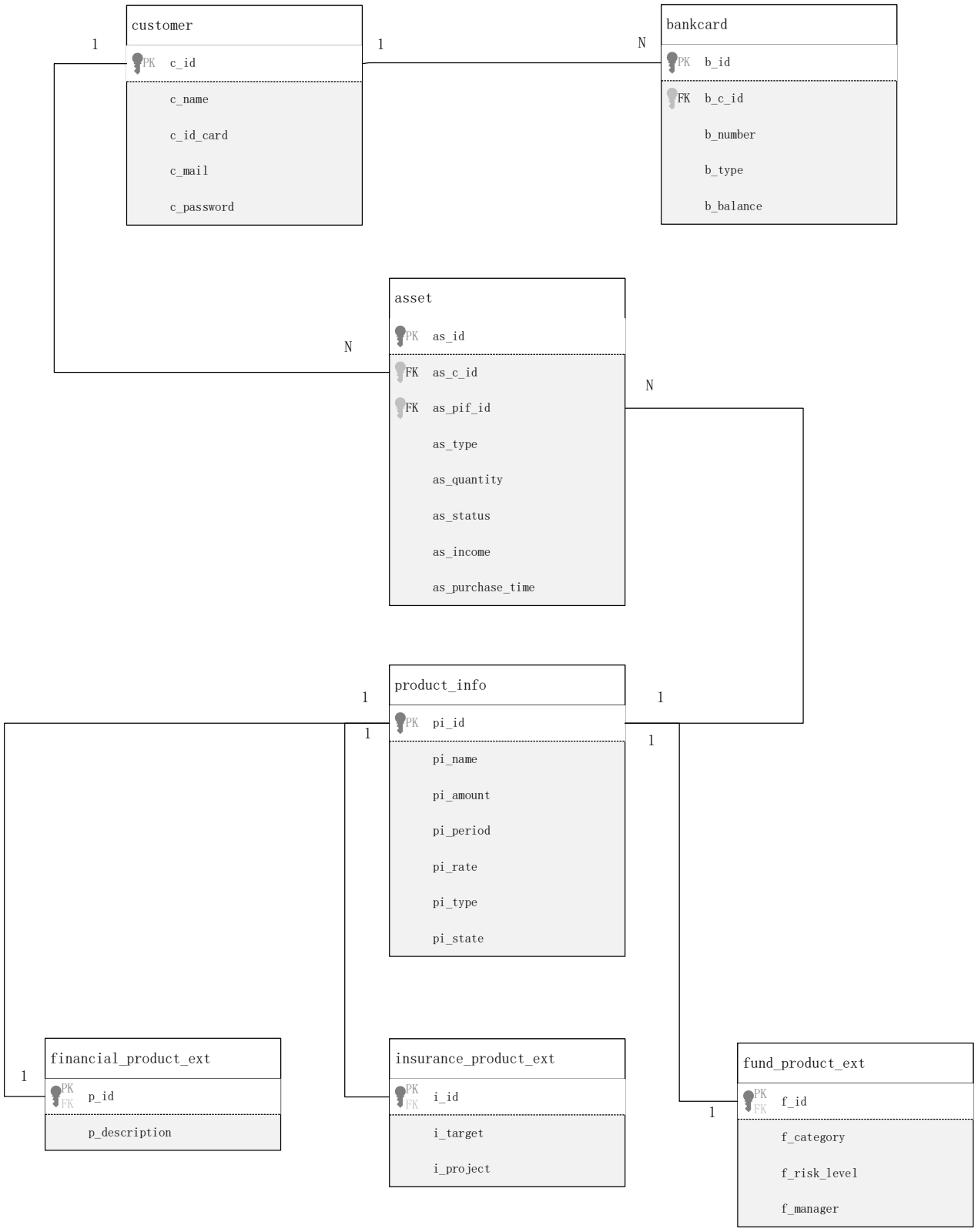
3.4.1 Database conceptual design

ER diagram:



3.4.2 Database logical design

The logical structure of the database is shown in the figure [4]:

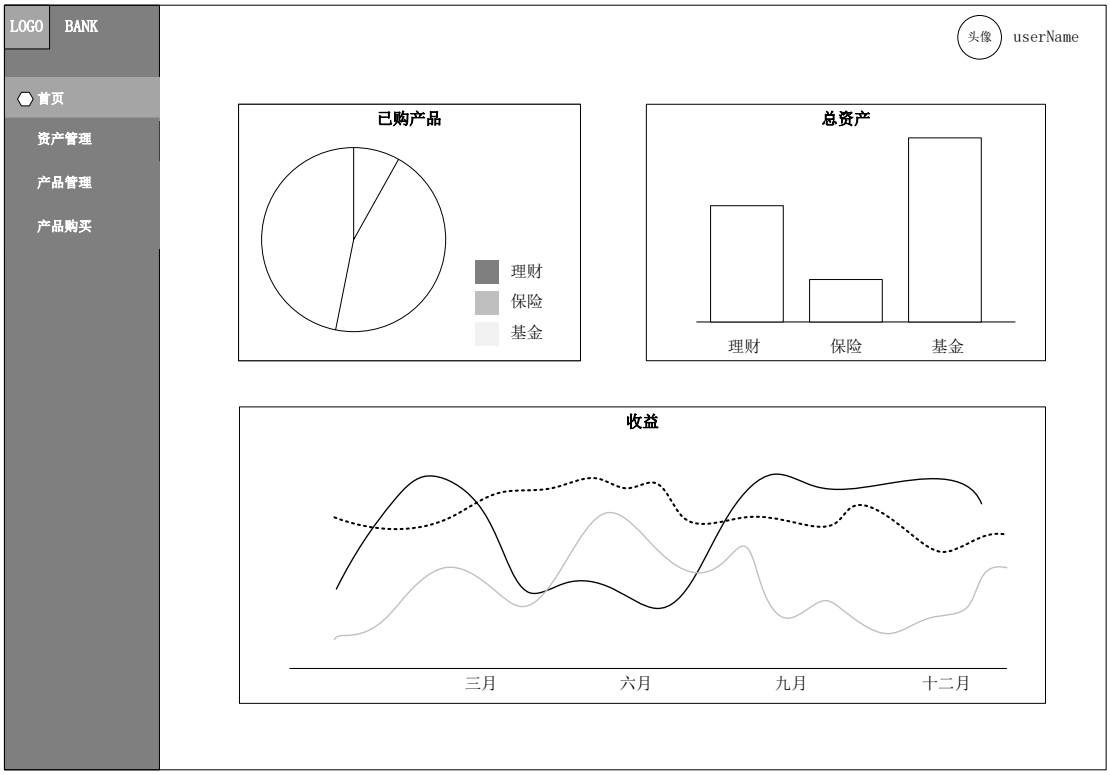


3.5 Interface Design

Log in:

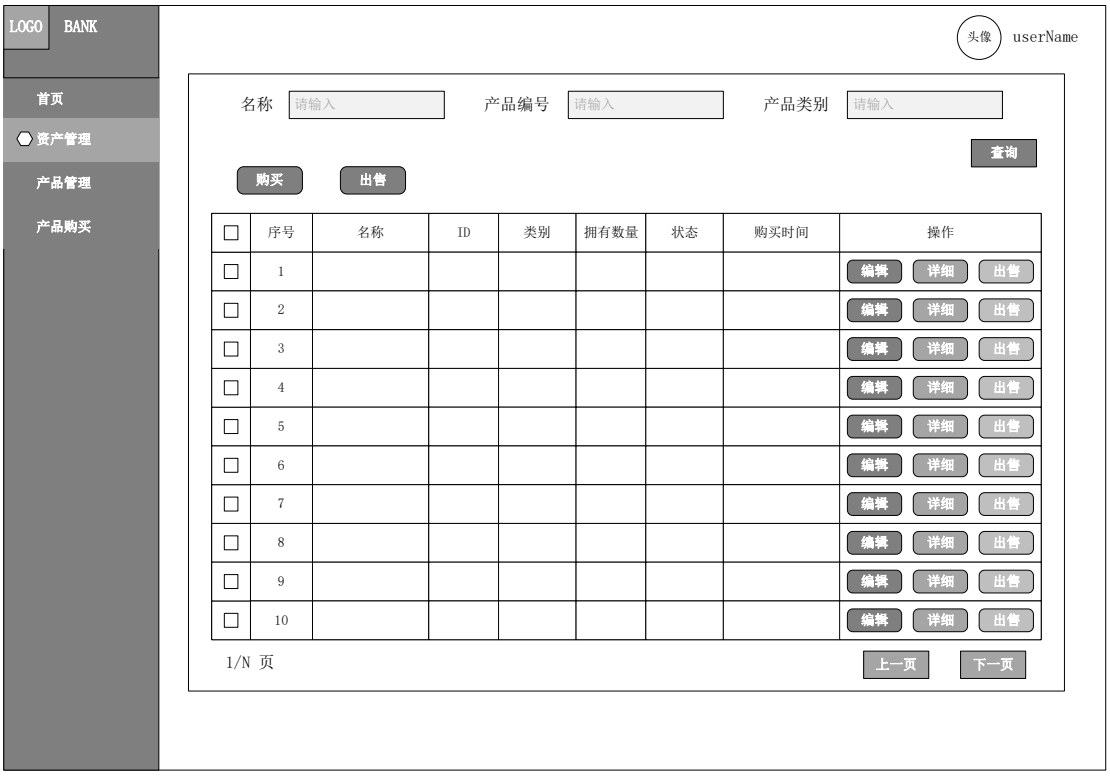
register:

front page:

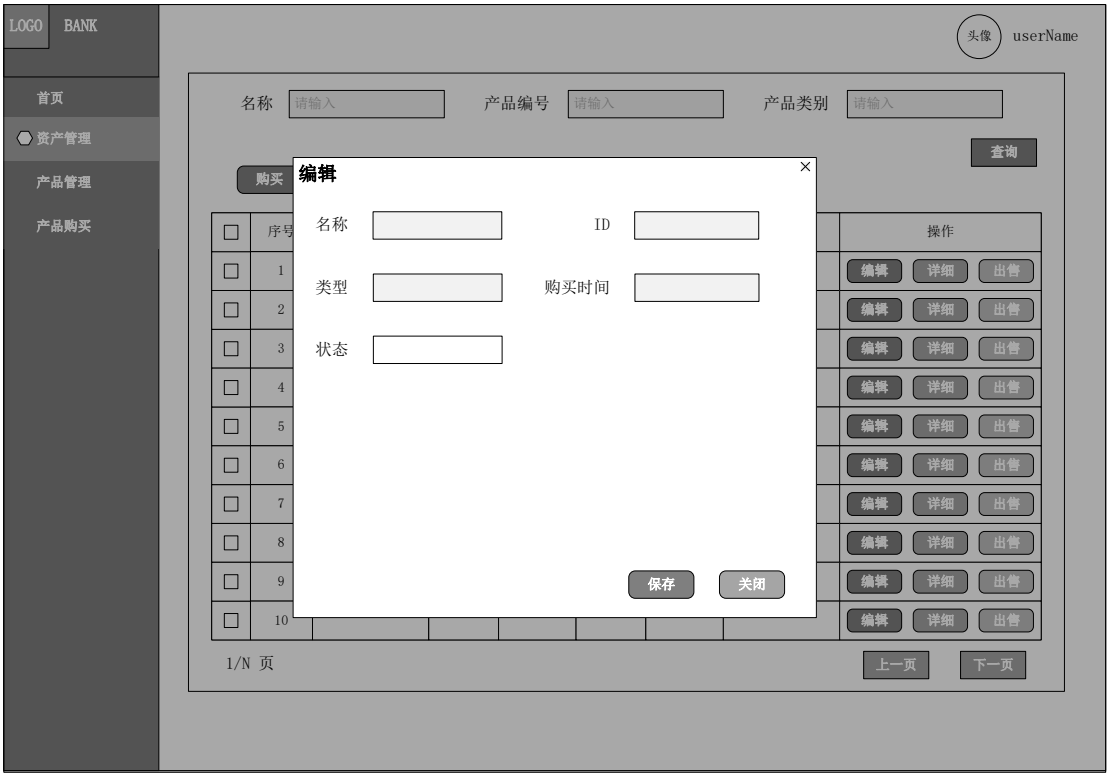


Personal Center:

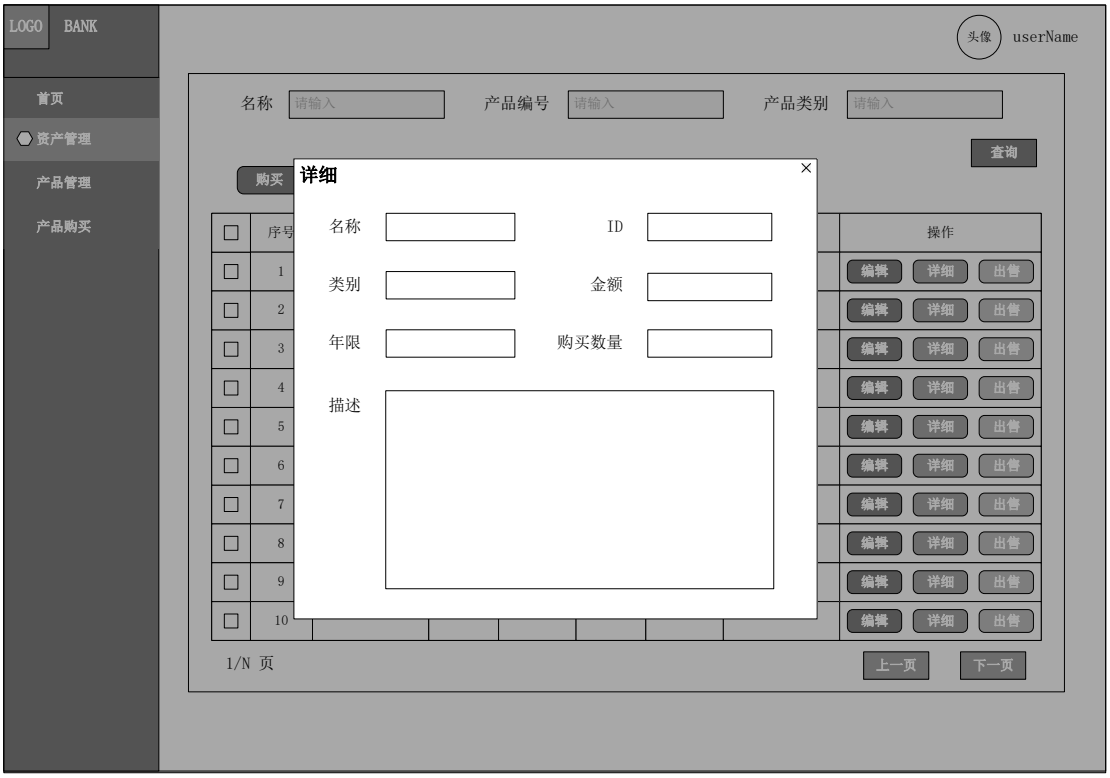
asset Management:



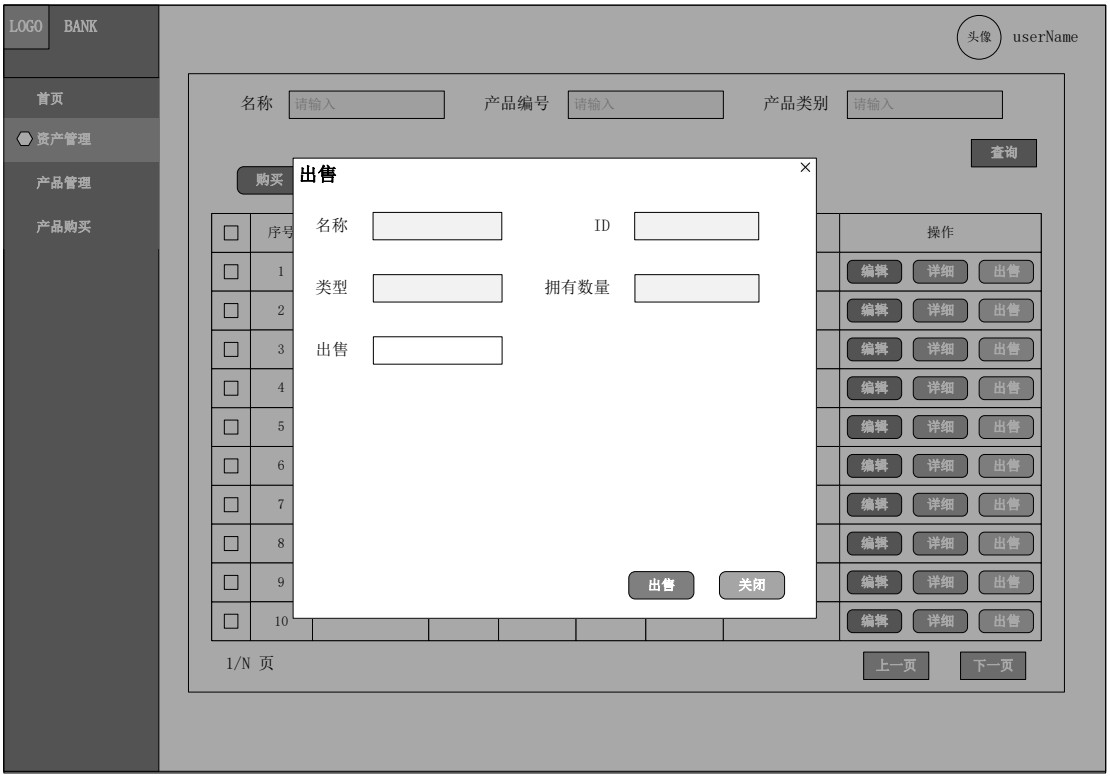
Asset Management - Editor:



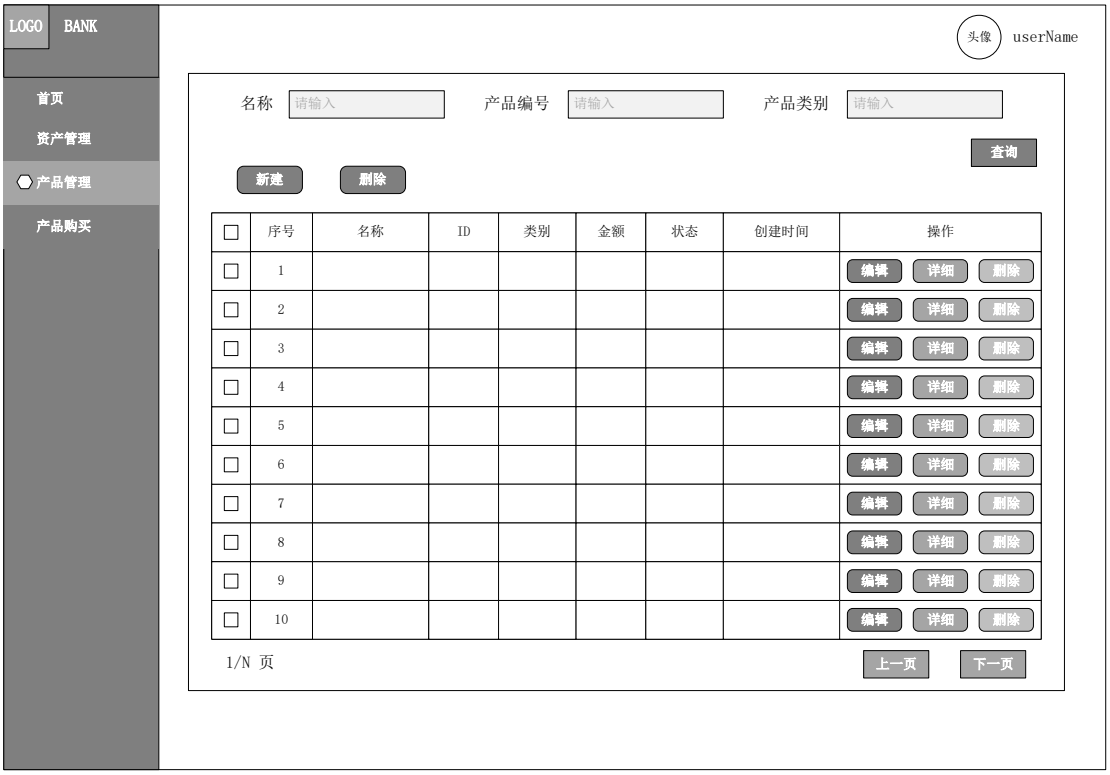
Asset Management - Details:



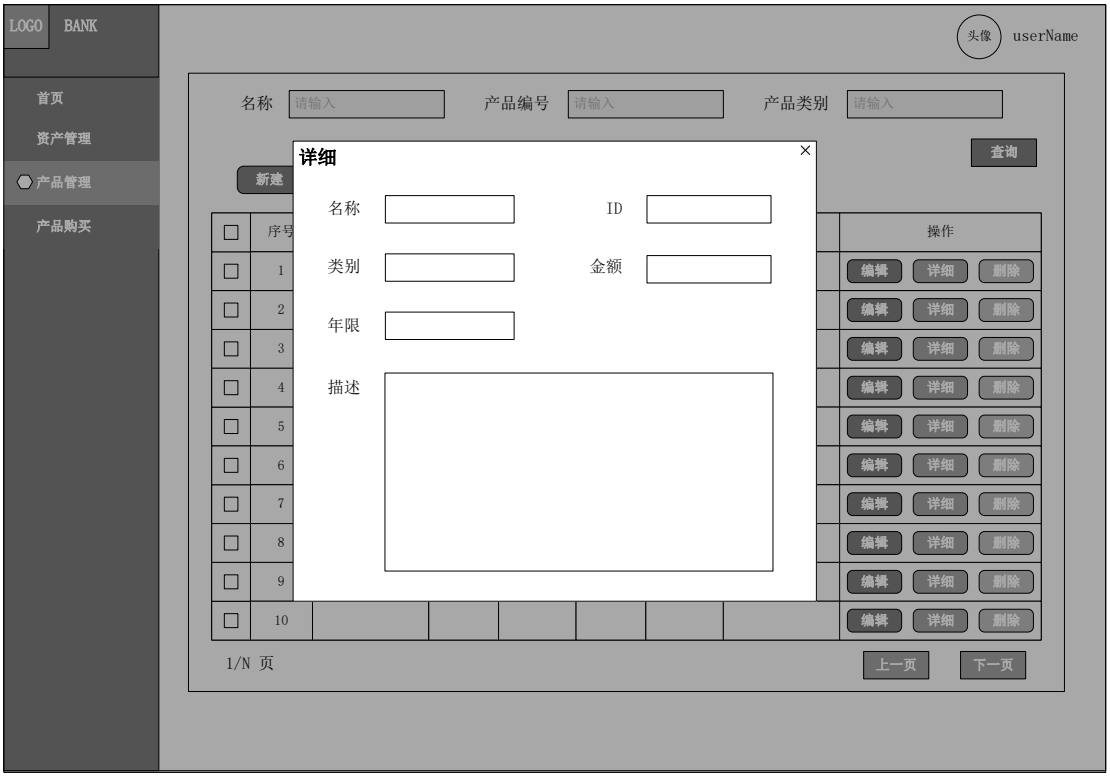
Asset Management - Sale:



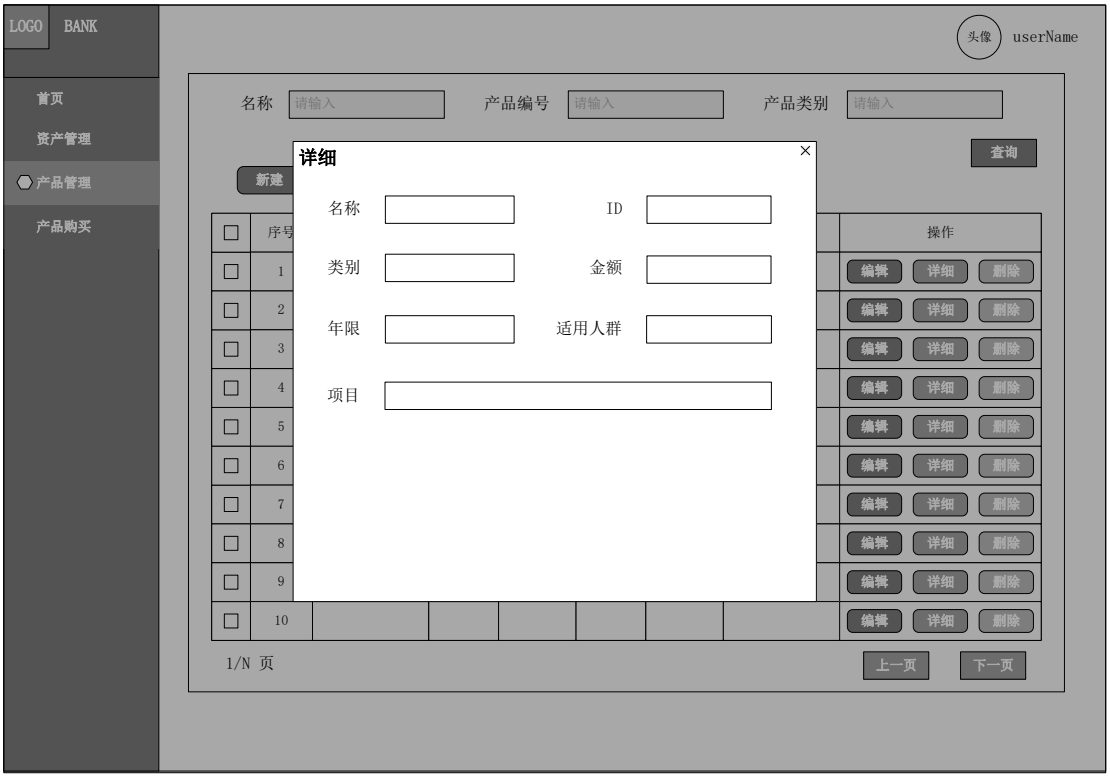
Product Management:



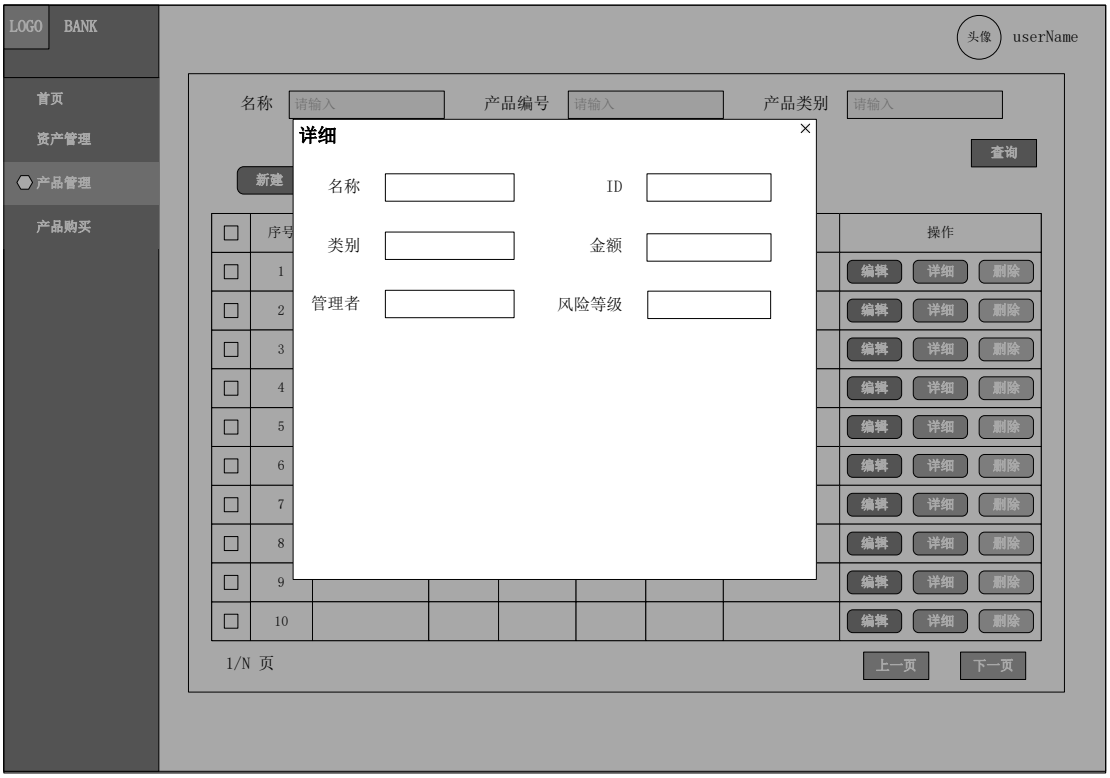
Product Management - Financial Details:



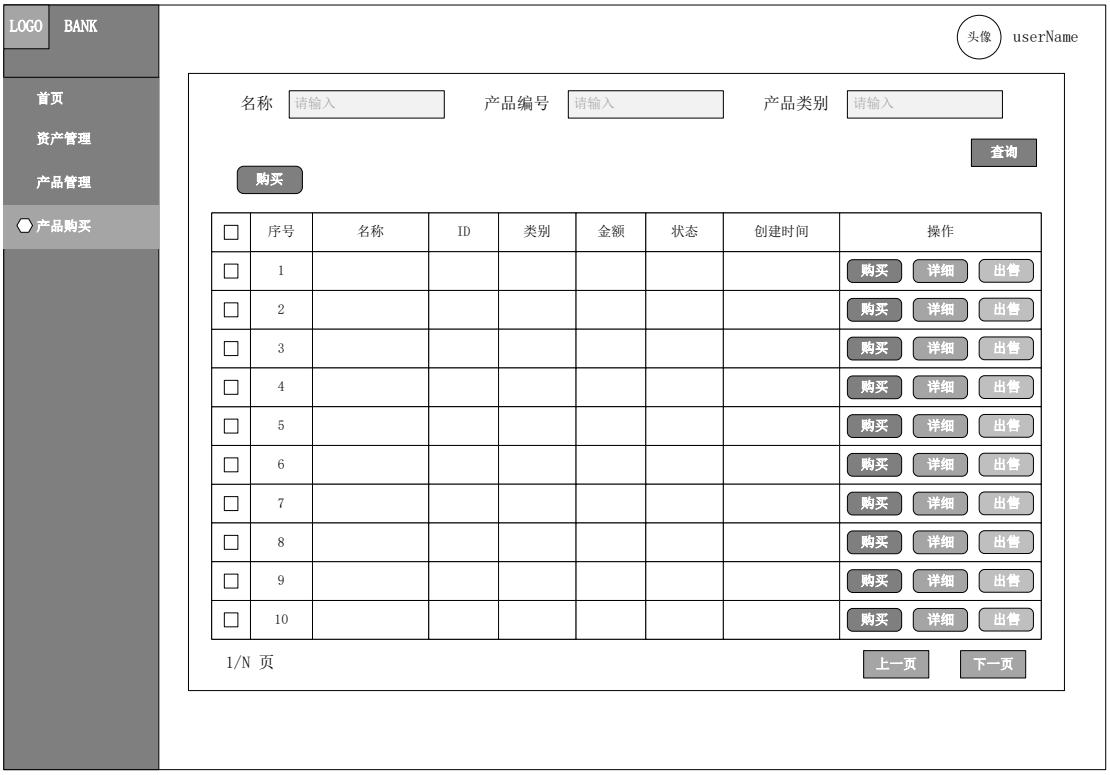
Product Management - Insurance Details:



Product Management - Fund Details:



Product Purchase:



4 Detailed design

4.1 introduction

4.1.1 Purpose of writing

This detailed design specification aims to systematically design and explain the server-side interfaces, database structure, and core functional processes of the bank management system developed in this project. This ensures that developers, testers, and maintenance personnel can fully understand the system's design concepts and implementation details. This specification also standardizes team development practices, clarifies module functions and interface specifications, and reduces communication costs during system development and maintenance.

This system is mainly designed for bank customers' account management, bank card information inquiry, asset purchase and sale, asset freezing and information display functions. It has good scalability and easy maintenance, and is suitable for small and medium-sized banks or financial project prototype verification. [8]

4.1.2 Abbreviations and Terms

Abbreviations/Terms	illustrate
system	Overall bank management system
Module A	Login and authentication module
Module B	Bank Card Information Management Module
Module C	Asset Management Module
user	System users refer to customers
Customer Number	A customer's unique identifier in the system
password	Verification information required to log in to the system
bank card	The bank cards held by customers include information such as card number, type, and balance.
product	Financial products available for purchase within the system
wealth management products	Financial products in the system
Insurance products	Insurance products in the system

Fund products	Fund products in the system
assets	Customer's asset records after purchase
Buy	Customers purchase products through the system
sell	The customer sells the assets he/she owns.
freeze	Lock the asset in its current state to prevent it from being sold.
Balance	Available funds in bank card account
state	Current status of assets or products
Information Form	The corresponding data storage structure in the database
interface	System-provided function access points

4.2 Database physical structure design

4.2.1 Customer Information Form

Stores basic information about customers in the banking system. [3][5]

The table is labeled: CUSTOMER

Table No.: A-01

Serial Number	field name	Field identifier	Type and length	Are null values allowed?	Units of measurement	primary key	index
1	Customer Number	c_id	N(8)	no		yes	1
2	Customer Name	c_name	VC(50)	no			2
3	ID number	c_id_card	C(18)	no			
4	Phone number	c_phone	C(11)	no			
5	Mail	c_mail	VC(100)	yes			
6	Login password	c_password	VC(100)	no			

Meaning and instructions for filling in each field:

Customer ID: A unique system identifier, an 8-digit code . The first 6 digits are fixed as 100010, and the last 2 digits increment .

Customer Name: Customer's real name, maximum 50 characters.

ID card number: Valid ID card number, 18 digits.

Mobile phone number: 11-digit mobile phone number.

Email: Customer email address (optional).

Login password: User password, stored in encrypted form.

4.2.2 Bank Card Information Form

Stores the bank card information linked to the customer.

The table identifier is: BANKCARD

Table No.: A-02

Serial Number	field name	Field identifier	Type and length	Are null values allowed?	Units of measurement	primary key	index
1	Bank card number	b_id	N(8)	no		yes	1
2	Customer ID	b_c_id	N(8)	no		Foreign Key	2
3	card number	b_number	C(19)	no			
4	Card type	b_type	C(10)	no			
5	Account Balance	b_balance	N(12,2)	no	Yuan		

Meaning and instructions for filling in each field:

Bank card number: A unique system identifier, an 8-digit code , with the first 6 digits fixed at 200010 and the last 2 digits incrementing automatically.

Customer ID: Related customer information table c_id .

Card number: Real bank card number, maximum 19 digits.

Card type: such as debit card, credit card, etc.

Account balance: Card balance, in yuan , supports two decimal places.

4.2.3 Asset Information Sheet

Record information on the various financial products and assets held by clients.

The table identifier is: ASSET

Table No.: A-03

Serial Number	field name	Field identifier	Type and length	Are null values allowed ?	Units of measurement	primary key	index
1	Asset Number	as_id	N(8)	no		yes	1
2	Customer ID	as_c_id	N(8)	no		Foreign Key	2
3	Product Information Number	as_pif_id	N(8)	no		Foreign Key	3
4	Product Type	as_type	N(1)	no			
5	Purchase Quantity	as_quantity	N(8)	no			
6	Asset Status	as_status	C(10)	no			
7	Profit amount	as_income	N(12,2)	no	Yuan		
8	Purchase time	as_purchase_time	DATETIME	no			

Meaning and instructions for filling in each field:

Asset Number: A unique identifier for an asset record, consisting of an 8-digit code . The first 6 digits indicate the category, and the last 2 digits are the serial number.

Customer ID: Related customer information table c_id .

Product Information Number: Associated with the primary key of the specific product information table.

Product types: 1-Wealth management, 2-Insurance, 3-Funds.

Purchase quantity: The quantity of products purchased.

Asset status: such as normal or frozen.

Earnings Amount: Cumulative earnings, in yuan.

Purchase time: The time when the asset was purchased.

4.2.4 General Product Information Sheet

The table identifier is: PRODUCT_INFO

Table No.: A-07

Note: Stores common information about financial products such as wealth management, insurance, and funds.

Serial Number	field name	Field identifier	Type and length	Are null values allowed?	Units of measurement	primary key	index
1	Product Number	pi_id	N(8)	no		yes	1
2	Product Name	pi_name	VC(50)	no			2
3	Amount	pi_amount	N(12,2)	no	Yuan		
4	Years	pi_period	N(2)	no	Year		
5	Annual interest rate	pi_rate	N(5,2)	no	%		
6	Product Type	pi_type	C(1)	no			
7	Product Status	p i_state	C (1)	no			3

Meaning and instructions for filling in each field:

Product ID: A unified primary key for financial products, consisting of an 8-digit numeric code . The first 6 digits are fixed at 400010, and the last 2 digits increment automatically.

Product Name: Product display name, maximum 50 characters.

Amount: The purchase amount or minimum investment amount for this product, in yuan.

Product lifespan: The product's effective period, expressed in years.

Annual interest rate: Annualized yield, expressed as a percentage, such as 5.25, which means 5.25%.

Product type: Used to distinguish product categories; 1-Wealth management products, 2-Insurance products, 3-Fund products.

Product Status: Whether the product is available for purchase. 1 - On Sale 2 - Discontinued

4.2.5 Extended Information Table for Wealth Management Products

Storage is only applicable to extended information related to financial products.

The table identifier is: FINANCIAL_PRODUCT_EXT

Table No.: A-05

Serial Number	field name	Field identifier	Type and length	Are null values allowed?	Units of measurement	primary key	index
1	Product Number	p_id	N(8)	no		yes	1
2	Product Description	p_description	VC(200)	yes			

Meaning and instructions for filling in each field:

Product ID: Corresponds to pi_id in PRODUCT_INFO , and must exist.

Product Description: An introduction to the financial product, up to 200 characters, optional.

4.2.6 Insurance Product Extended Information Table

Store additional information fields specific to insurance products.

The table identifier is: INSURANCE_PRODUCT_EXT

Table No.: A-06

Serial Number	field name	Field identifier	Type and length	Are null values allowed?	Units of measurement	primary key	index
1	Product Number	i_id	N(8)	no		yes	1
2	Target audience	i_target	N(1)	yes			
3	Insurance Projects	i_project	VC(100)	yes			

Meaning and instructions for filling in each field:

Product ID: Corresponds to pi_id in PRODUCT_INFO .

Target audience: Limited to children, adults, and seniors.

Insurance coverage: The coverage provided by this insurance product, described in text, up to 100 characters.

4.2.7 Extended Information Table of Fund Products

Record additional information fields specific to fund products.

The table identifier is: FUND_PRODUCT_EXT

Table No.: A-7

Serial Number	field name	Field identifier	Type and length	Are null values allowed?	Units of measurement	primary key	index
1	Product Number	f_id	N(8)	no		yes	1
2	category	f_category	N(1)	no			
3	Risk level	f_risk_level	N(1)	no			
4	Manager	f_manager	VC(50)	no			

Meaning and instructions for filling in each field:

Product ID: Corresponds to pi_id in PRODUCT_INFO .

Category: Fund category code, such as 1-mixed fund, 2-equity fund, etc.

Risk levels: 1 - High risk, 2 - Medium risk, 3 - Low risk.

Manager: The management institution or person in charge of the fund product, maximum 50 characters.

4.3 Functional Design Specification

4.3.1 System Module Division Overview

This bank management system is divided into the following three core functional modules:

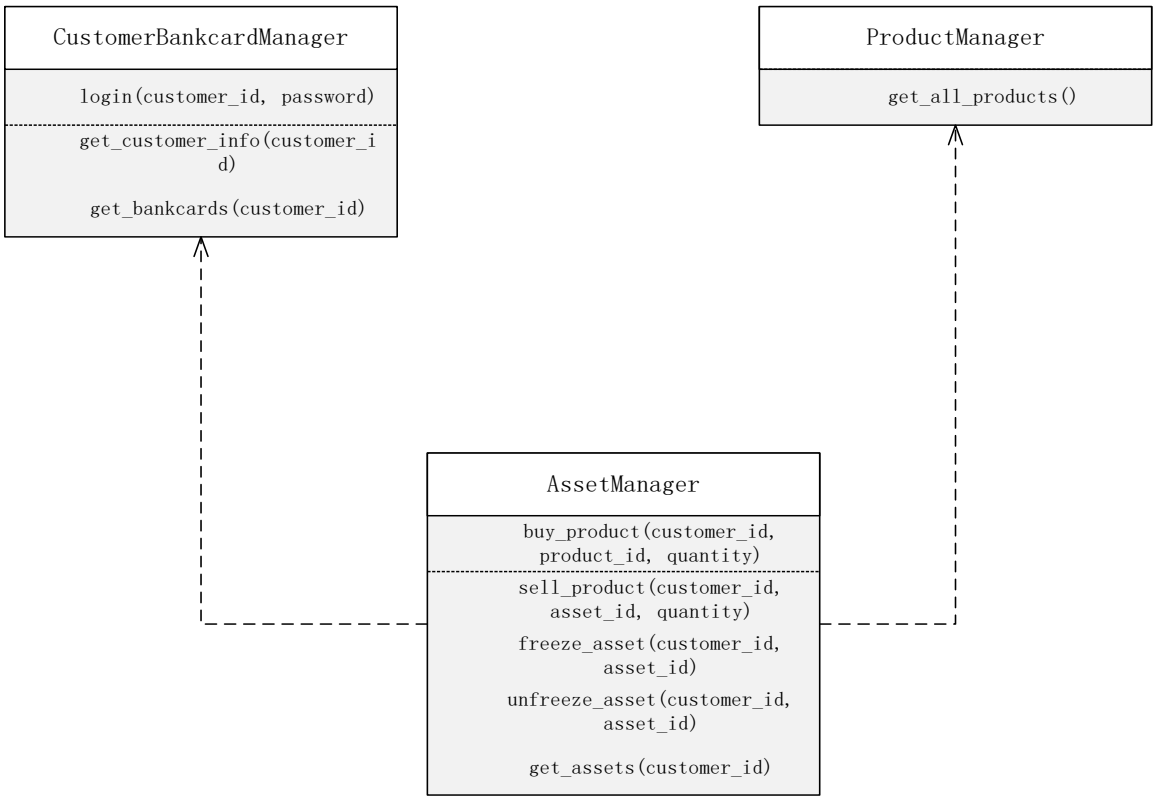
Customer and Bank Card Management Module (Module A)

Financial Product Management Module (Module B)

Purchase and Asset Management Module (Module C)

The modules are linked by primary and foreign keys, data is stored uniformly, and functions cooperate with each other. The specific design is as follows.

4.3.2 Class diagram



4.3.3 Customer and bank card management

4.3.3.1 Class Design

Class name:

CustomerBankcardManager

1.2 Main Responsibilities:

Responsible for user login and personal information inquiry

Responsible for the management and inquiry of bank card information

Method design:

name	Function	Input/Output
registerCustomer	Add new customer information, verify the uniqueness of mobile phone number and ID card, and store the initial password in encrypted form.	Input: Name, ID card number, mobile phone number, email address, password Output: Operation result
loginCustomer	Customer login verification, checking	Input: Customer ID, Password

	encrypted password	Output: Login Result
queryCustomerInfo	Query customer basic information by c_id	Input: Customer ID Output: Customer Information
bindBankCard	Add bank cards for customers and automatically verify customer existence.	Input: Customer ID, Bank Card Number, Card Type Output: Operation Result
queryBankCardInfo	Search bank card and balance information by customer number or card number.	Input: Customer ID or bank card number Output: Bank card information and balance

4.3.3.2 Module internal implementation description

This module serves as the foundational support for the system's user and bank card information, primarily implementing functions such as customer identity management, bank card binding, and account balance management. The system uses `c_id` to uniquely identify each customer, and all bank cards are associated with customer information via `b_c_id` , ensuring the accuracy of the one-to-many relationship. The `b_balance` field of the bank card balance directly reflects the account's financial status. When balance changes are involved, the system employs database transaction control to ensure data consistency and operational atomicity. Simultaneously, the system encrypts and stores customer login passwords to protect account security and prevent the leakage of sensitive information.

4.3.4 Financial Product Management

4.3.4.1 Class Design

Class name:

ProductManager

Main responsibilities:

- Managing and querying financial product information

Method design:

Method Name	Function	enter	Output
addProduct	Enter product information and initially discontinue	Product Fields	none

	sales.		
queryProduct	Search by number, type, or status	Product number, type, status	Eligible product information
manageProductStatus	Product listing and delisting management	Product number, target status	Was the operation successful?

4.3.4.2 Module internal implementation description

This module serves as the unified management module for the system's financial product information, maintaining data for multiple product categories such as wealth management, insurance, and funds. All product information is managed uniformly through `pi_id` , with basic information stored in the `PRODUCT_INFO` table, and specific category-specific extended information stored in their respective sub-tables. Sub-tables are linked to the main table via foreign keys, resulting in a clear structure and strong scalability. The system provides flexible product information query functions, supporting filtering by multiple conditions such as number, type, and status. The product status field `pi_state` strictly controls transaction permissions; users are prohibited from purchasing when a product is discontinued, ensuring transaction security. Adding and modifying product information is restricted to the operations of the backend administrator to prevent data tampering.

4.3.5 Purchase and Asset Management

4.3.5.1 Class Design

Class name:

AssetManager

Main responsibilities:

- Responsible for the purchase, sale, freezing, and unfreezing of assets.
- Managing customer asset information

Method Design

Interface Name	Function Description	Input parameters	Output
purchaseProduct ()	When a customer purchases a product, the balance and product status are verified, and an asset record is generated.	Customer ID, Product ID, Quantity Purchased	Was the purchase successful?

queryAssets ()	Query the list of all customer assets	Customer Number	Assets owned by the client
sellAsset ()	Sell assets, verify status and quantity, and return funds.	Customer ID, Asset ID, Quantity Sold	Was the sale successful?
freezeAsset ()	Assets frozen, sale prohibited	Customer ID, Asset ID	none

4.3.5.2 Module internal implementation description

This module serves as a unified management module for customer asset data, encompassing functions such as asset purchase, sale, freezing, and query. Asset records are linked to the customer table via `as_c_id` and to the product table via foreign keys using `as_pif_id` , ensuring information completeness and accuracy. During the purchase process, the system rigorously verifies bank card balances and product status to prevent abnormal transactions. All fund changes and asset generation operations are incorporated into database transaction control to prevent data inconsistencies. The asset freezing function is primarily used for risk control and anomaly handling scenarios; assets are frozen and cannot be sold, ensuring customer fund security.

4.3.6 Explanation of the relationship between modules

Module A provides customer and bank card data, which Module C relies on for identity verification and deductions. Module B provides product information, and Module C associates asset information based on product numbers to ensure data accuracy. Module C centrally manages customer assets, and all asset records depend on information from Modules A and B. The various modules of the system collaborate closely through foreign key relationships and parameter passing, resulting in clear overall logic, efficient data interaction, and stable system operation.

4.3.7 Summarize

This design strictly references relational tables, employing standard fields, standardized data types, and clear logical relationships to ensure complete system functionality, unified data structure, smooth module collaboration, and guarantees system stability, security, and scalability, meeting the business needs of the bank's management system.

4.4 Key Algorithm Design for Bank Management System

In this system design, several basic algorithms were designed and implemented based on practical functional requirements, primarily applied to modules such as login verification, account information management, and asset purchase and sale. The entire algorithm process emphasizes logical simplicity and intuitive operation, ensuring the normal operation of system functions while guaranteeing data accuracy and security. Through analysis of various common operational scenarios, the designed algorithms effectively meet the system's basic functional requirements.

4.4.1 Login authentication algorithm

Algorithm principle: Verify whether the customer number and password entered by the user match the database information to confirm the legitimacy of the login.

Input/output:

Input: customer_id (customer ID), password (user password)

Output: Login success or failure message

Implementation steps:

- Receive the customer ID and password entered from the front end.
- Query the CUSTOMER table to verify information.
- The login status will be returned based on the result.

pseudocode:

```
Connect to the database
and execute the SQL: SELECT * FROM CUSTOMER WHERE c_id = customer_id AND
c_password = password.
If the query results exist: Return {"status": "success", "message": "Login successful"}
Otherwise:
Return {"status": "fail", "message": "Incorrect customer ID or password"}
```

4.4.2 Bank card information query algorithm

Algorithm principle: Query bank card number, type, and balance based on customer ID.

Input/output:

Input: customer_id

Output: Bank card number, type, balance

Implementation steps :

- Enter the customer number.
- Query the BANKCARD table to retrieve bank card information.
- Return bank card data.

pseudocode:

The function queries bank card information (customer_id):
Connect to the database and execute the SQL: SELECT b_number , b_type , b_balance FROM BANKCARD WHERE b_c_id = customer_id.
If the query results exist: Return {"status": "success", "bankcard": relevant information}
Otherwise: Return {"status": "fail", "message": "bank card information not found"}

4.4.3 Product purchase algorithm

Algorithm principle: After verifying that the product is available for sale and that the balance is sufficient, the balance is deducted and an asset record is generated.

Input/output:

Input: customer_id , product_id , quantity

Output: Purchase results and new account balance

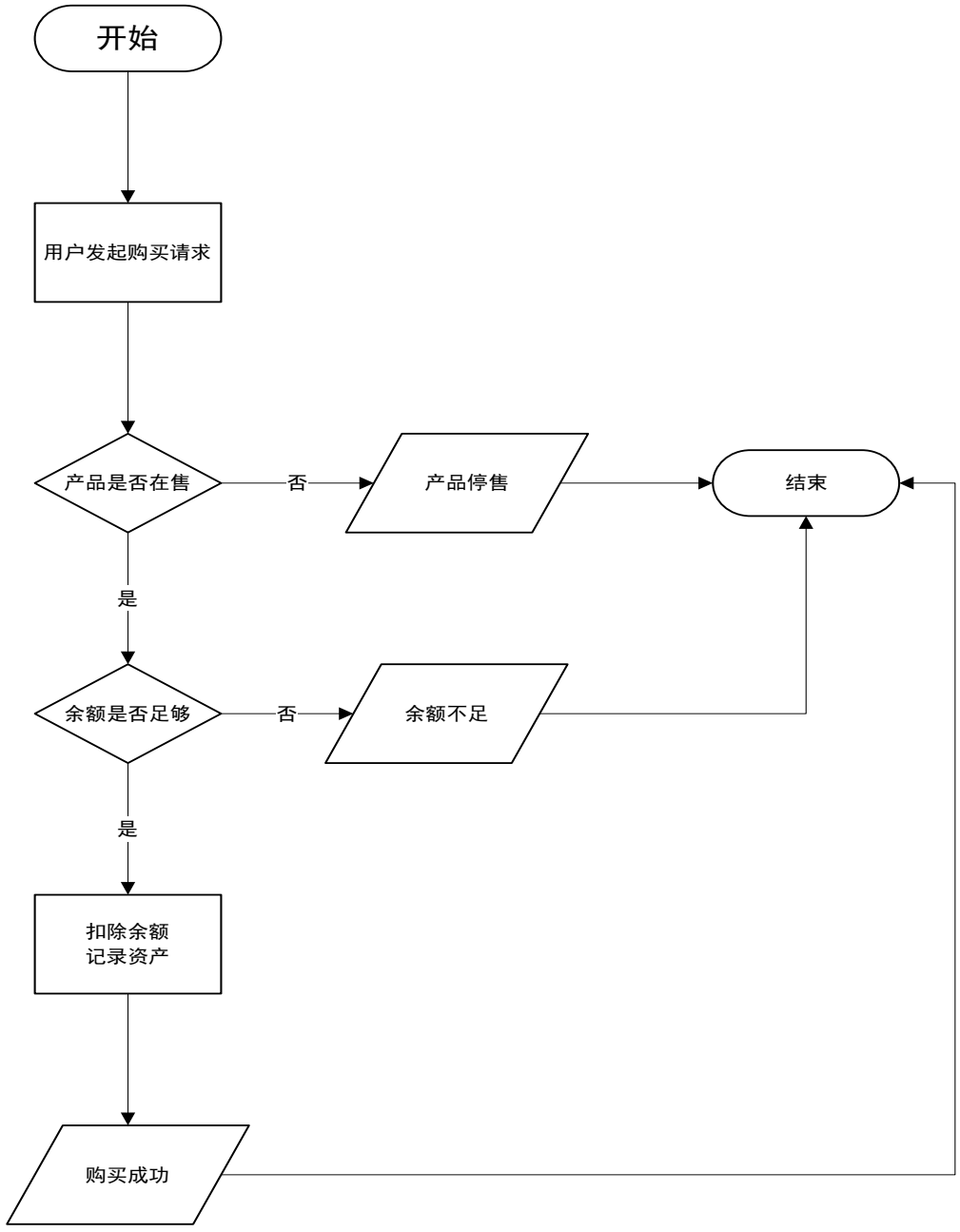
Implementation steps

- Check product prices and sales status.
- Check your bank card balance.
- Verify the sales status and balance.
- The balance is deducted, and new asset records are added.

pseudocode:

The function `purchase_product( customer_id , product_id , quantity)`
is used to: Query product information: `SELECT pi_amount , pi_state FROM PRODUCT_INFO WHERE pi_id = product_id`
Query bank card balance: `SELECT b_id , b_balance FROM BANKCARD WHERE b_c_id = customer_id`
If the product is available and the balance is sufficient: `total_price = unit price * quantity`
Deduct balance: `UPDATE BANKCARD SET b_balance = new balance WHERE b_id = bank card number`
Insert asset record: `INSERT INTO ASSET (as_c_id , as_pif_id , as_type , as_quantity , as_status , as_income , as_purchase_time)
VALUES (customer_id , product_id , product_type, quantity, "normal", 0, current time)`
Returns `{"status": "success", }` "message": "Purchase successful", "balance": New balance} Otherwise: Return {"status": "fail", "message": "Insufficient balance or product discontinued"}

Flowchart illustration:



4.4.4 Asset sale algorithm

Algorithm principle: If the assets are not frozen and there are sufficient assets, sell some or all of the assets and update the balance.

Input/output:

Input: customer_id , asset_id , quantity

Output: Sales results and new balance

Implementation steps

- Check asset status and quantity.
- The verification shows that the data is not frozen and the quantity is sufficient.
- Calculate earnings and update the balance.
- Update or delete asset records.

pseudocode:

The function `sell assets( customer_id , asset_id , quantity)`
is used to: Query asset information: `SELECT as_quantity , as_pif_id , as_status FROM ASSET WHERE as_id = asset_id AND as_c_id = customer_id`.
If the asset exists, is not frozen, and has sufficient quantity: Query product unit price: `SELECT pi_amount FROM PRODUCT_INFO WHERE pi_id = product_id`.
Query bank card balance: `SELECT b_id , b_balance FROM BANKCARD WHERE b_c_id = customer_id`.
`total_earnings = unit price * quantity`
Update balance: SQL: `UPDATE BANKCARD SET b_balance = new balance WHERE b_id = bank card number`
If remaining quantity = 0: SQL: `DELETE FROM ASSET WHERE as_id = asset_id`
Otherwise: SQL: `UPDATE ASSET SET as_quantity = remaining quantity WHERE as_id = asset_id`
Returns `{"status": "success", "message": "sale successful", "balance": new balance}`
Otherwise: Returns `{"status": "fail", "message": "sale failed, asset frozen or data abnormal"}`

4.4.5 Asset Freeze Algorithm

Algorithm principle: Set the asset status to "frozen" based on the asset number, prohibiting its sale.

Input/output:

Input: customer_id , asset_id

Output: Frozen results

Implementation steps

- Locate asset records.
- Update the asset status to "frozen".

pseudocode:

The function `Freeze Asset( customer_id , asset_id )` works as follows:
SQL: `UPDATE ASSET SET as_status = "Frozen" WHERE as_id = asset_id AND as_c_id =

customer_id`.
If the update is successful: it returns `{"status": "success", "message": "Asset frozen"}`.
Otherwise: it returns `{"status": "fail", "message": "Freezing failed, asset does not exist"}`.

4.4.6 Asset Information Query Algorithm

Algorithm principle: Jointly query asset and product information to return a detailed asset list.

Input/output:

Input: customer_id

Output: Asset details list

Implementation steps :

- Use ASSET and PRODUCT_INFO together to query asset details.
- The formatted result returns a list of assets.

pseudocode:

The function queries asset information (customer_id):
SQL: SELECT asset_information FROM ASSET JOIN PRODUCT_INFO ON as_pif_id = pi_id WHERE as_c_id = customer_id.
It returns {"status": "success", "assets": a list of assets}.

4.4.7 Summarize

The design and implementation of the core algorithms in this system ensures the stability and reliability of the bank management platform under high concurrency access and complex data interactions. Each algorithm not only meets functional requirements but also considers performance optimization and system scalability, possessing strong practical application value. The overall design scheme reflects a technical approach that prioritizes practical needs, efficiency, and security, ensuring the efficient and stable operation of the system.

5 Software Implementation

This bank management system adopted a front-end and back-end separation development approach. The back-end primarily used Python with the Flask framework for API development, while the front-end used Vue for page display, and MySQL was used for data management. Throughout the development process, the team strictly adhered to unified coding standards and a reasonable development workflow, ensuring the integrity of the system's functionality and the standardization of the program.

5.1 Development Environment and Reasons for Selection

To ensure development efficiency, reduce error rates, and facilitate later maintenance, we selected the following development environment based on the school's experimental requirements and actual technical needs:

1. Backend development environment

project	Technology Selection
programming language	Python 3.11
Development framework	Flask 3.0
Dependency Management	pip
Database connection library	pymysql 1.1.0
Database system	MySQL 8.0
Development tools	PyCharm 2024.1
Environmental Management	Anaconda

Reasons for choosing:

- Python has a concise and easy-to-learn syntax, making it suitable for students to get started quickly.

- The Flask framework has a simple structure, making it easy to build interfaces and suitable for the functional requirements of this system.
- pymysql library has good compatibility and is easy to use for operating MySQL databases;
- Using Anaconda virtual environments can effectively isolate project dependencies and avoid environment conflicts;
- PyCharm is feature-rich and easy to use for writing, debugging, and managing code.

2. Front-end development environment

project	Technology Selection
Front-end framework	Vue 3.5.13
UI component library	Element Plus
Visual charts	ECharts
Package management tools	pnpm 10.12.3
Operating environment	Node.js 22.16.0
Development tools	VSCode

Reasons for choosing:

- The Vue framework has a clear structure, supports component-based development, and is easy for team collaboration.
- Element Plus offers a rich selection of UI components to enhance the aesthetics of the interface;
- ECharts is a powerful charting tool that makes it easy to visualize data.
- Node.js and pnpm have good compatibility and efficient dependency management;
- VSCode is lightweight, supports a variety of plugins, and offers a good development experience.

3. Database Development Environment

project	Technology Selection
Database version	MySQL 8.0
Visual management tools	Navicat Premium

Reasons for choosing:

- MySQL, as a commonly used relational database, is known for its stability and high query efficiency.
- Navicat provides an intuitive visual interface, making table structure design and data maintenance easier;
- Use SQL scripts to quickly initialize the database and ensure data consistency.

5.2 Environment setup steps

The specific process for setting up a development environment for a team is as follows:

1. Install Python, Anaconda, and Node.js;

2. Create a virtual environment using Anaconda, activate it, and then install dependencies such as Flask and pymysql :

```
conda create -n bank_env python=3.11
conda activate bank_env
pip install flask pymysql
```

3. Install frontend dependencies:

```
npm install -g pnpm
pnpm install
```

4. Import the database SQL script (using Navicat or command line):

```
CREATE DATABASE bank;
USE bank; Execute SQL file content
```

5. Backend operation:

```
python app.py
```

6. Front-end execution:

```
pnpm run dev
```

After configuring the environment, access the front-end page through a browser to test the interface connectivity.

5.3 Coding Standards

To ensure consistency in team collaboration and code maintainability, the project has established the following coding standards:

Standard Category	Detailed Explanation
Variable naming	Lowercase + underscore, such as: customer_id
Function naming	Lowercase + underscore, such as: get_balance
Class naming	Large hump, such as: BankAccount
File naming	All lowercase letters plus underscores, such as: app.py

Database table name	All lowercase letters plus underscores, such as: bankcard, asset
Database fields	All lowercase letters plus underscores, such as: b_balance , as_quantity
Indentation Standards	Use 4 spaces for indentation and do not use tabs.
Commenting Guidelines	Key functions and complex logic must be commented in Chinese, explaining their function, parameters, and return value.

Example:

def freeze_asset (): Function: Freezes the specified asset. Input:
customer_id and asset_id
from the request body. Output: JSON response of the operation result.

5.4 Code Management

- The project code uses Git for version control, standardizing the team's development process.
- Use a remote repository for backups.
- Set up main (main branch) and dev (development branch);
- Each new feature is developed on a separate branch on dev.
- Periodically merge into dev, and merge into main once the functionality is stable;
- Submitted information should be concise and clear to facilitate version tracking;
- Regularly back up the database SQL file to ensure data security.

5.5 Summary of the rationality of the development environment

- The development environment chosen for this project takes into account practicality, ease of learning, and scalability:
- The backend uses Python + Flask, which reduces the learning curve and improves development efficiency;
- The front end uses the Vue ecosystem, resulting in a beautiful interface and a good interactive experience;
- The MySQL database has stable performance and meets the system's data storage requirements;
- A well-developed toolchain enhances the team's collaborative development capabilities.

In summary, the overall technical solution is well-suited to the actual needs of student projects, the environment is simple and easy to configure, and it facilitates subsequent system expansion and functional improvements for the team.

6 Software testing

6.1 introduction

6.1.1 Purpose of writing

This test report aims to systematically summarize the testing process and results of the bank management system, and to verify the correctness, stability and reliability of the functions of each module of the system. Through this test, we ensure that the system meets the design requirements before it is actually put into use, that the basic functions of the system are complete, that user operation is smooth, and that major defects occur after the system goes live. [6]

6.1.2 Project Background

This project is a bank management system developed using the Python Flask framework and integrated with a MySQL database. It implements core functions such as basic customer information management, bank card information management, wealth management product purchase, and asset inquiry and management. The system is designed for end-users, with a front-end and back-end separation architecture, and data interaction is completed via HTTP interfaces.

The initial development of the system has been completed, and it has now entered the system testing phase. The testing team writes and executes system tests based on the requirements specification and detailed design documents to ensure that the system meets the design and user requirements.

6.2 Test plan

6.2.1 Test target

- Covers the normal and abnormal conditions of each functional module of the system;
- Verify the correctness of the system's handling of boundary conditions, illegal inputs, and business logic;
- Assess system stability and data integrity;
- Discover potential system defects and security risks, and reduce system risks.

6.2.2 Test content

- This test mainly includes:
- Login function test
- Bank card information management test
- Asset purchase, sale, and freeze function testing
- Asset information query function test

- System interface stability and data accuracy verification

6.2.3 Test environment

Test Items	Configuration
operating system	Windows 10 Professional
database	MySQL 8.0
Backend framework	Flask + PyMySQL
Front-end tools	HTML, CSS, JavaScript
Programming Language	Python

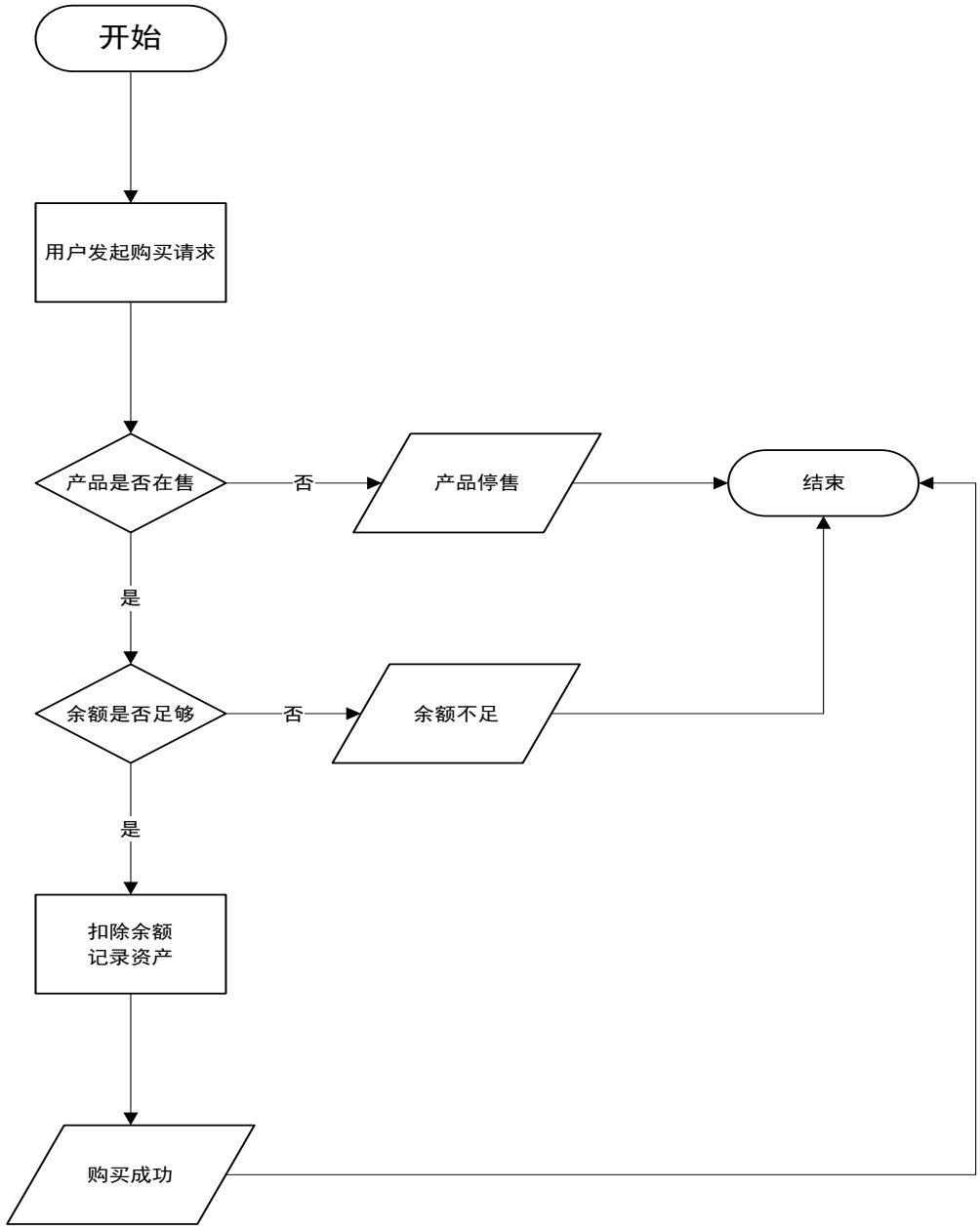
6.3 Testing Methods and Strategies

- A combination of white-box and black-box testing was used.
- White-box testing: Focus on testing the core business logic of the product purchase function, refer to the unit test documentation, and ensure that the internal implementation logic is correct.
- Black-box testing: Input typical and abnormal data on a per-function basis, observe the system response, and verify the integrity and fault tolerance of the functions.
- Randomly generate test data, covering boundary values, illegal inputs, extreme cases, etc.

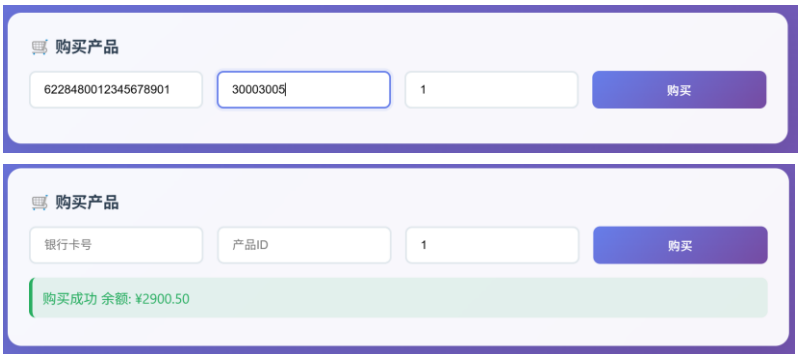
6.4 Test case design

6.4.1 White-box testing (taking product purchase function as an example)

Data flow diagram:



Construct white-box test cases based on data flow diagrams.

Module Name	Product purchase		
tester	L	Test type	Backend Functionality Testing
Test date	2025/6/26	Testing tools	none
Use Case ID	TIMS BP 01		
Use Case Description	Product available, sufficient balance, purchase successful.		
Prerequisites	(1) The system has products available for sale; (2) The user has sufficient balance; (3) Click to purchase.		
Expected results	The system deducts the balance, records the assets, and notifies the user of a successful purchase.		
Actual results	 Test successful		

Module Name	Product purchase		
tester	L	Test type	Backend Functionality Testing
Test date	2025/6/26	Testing tools	none
Use Case ID	TIMS BP 02		
Use Case Description	Product available, but insufficient balance, purchase failed.		
Prerequisites	(1) The system has products available for sale; (2) The user's balance is insufficient; (3) Click to purchase.		
Expected results	The system indicated insufficient balance; the purchase failed.		

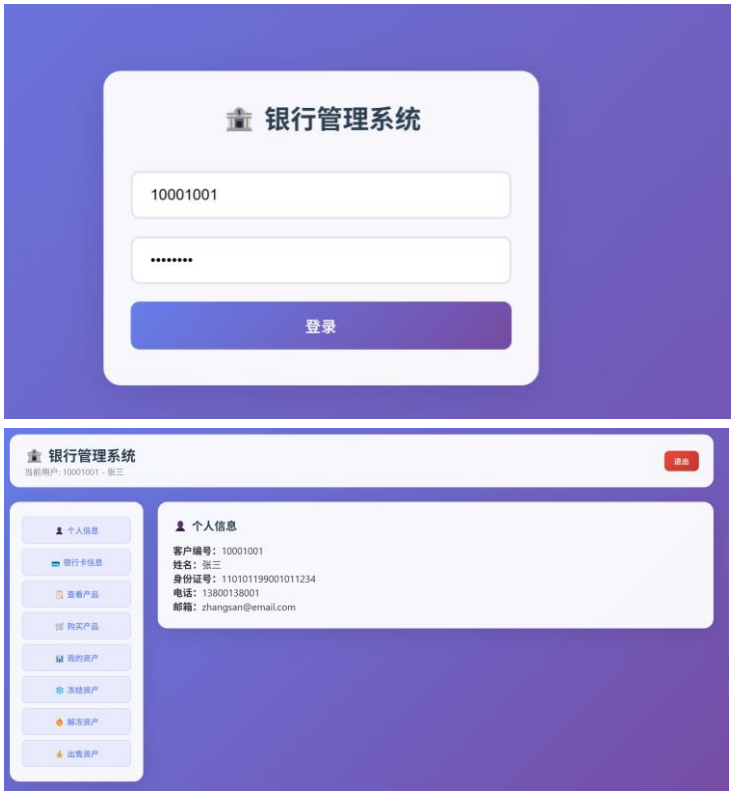
Actual results	 Test successful

Module Name	Product purchase		
tester	L	Test type	Backend Functionality Testing
Test date	2025/6/26	Testing tools	none
Use Case ID	TIMS BP 03		
Use Case Description	Product discontinued, purchase failed.		
Prerequisites	(1) The system indicates that this product is no longer available for sale; (2) Click to purchase.		
Expected results	The system indicated that the product was discontinued and the purchase failed.		
Actual results	 Test successful		

6.4.2 Black-box testing (functional interface testing)

6.4.2.1 Login function

Module Name	User Management - User Login
tester	L
Test type	Front-end functional testing
Testing tools	none

Use Case ID	BB-Login-01
Use Case Description	The test system logged in successfully.
Prerequisites	Enter your registered customer ID (10001001) and the correct password, then click Login.
Expecting results	Upon entering the homepage, personal information is displayed.
Actual results	
Remark	

6.4.2.2 Bank card information inquiry

Module Name	Bank Card Management - Bank Card Information Inquiry
tester	L
Test type	Front-end functional testing
Testing tools	none
Use Case ID	BB-Card-01
Use Case Description	Checking for existing bank card information
Prerequisites	Login successful, customer has bank card

Expecting results	Returns card number, card type, and account balance information.
Actual results	

6.4.2.3 Product Information Inquiry

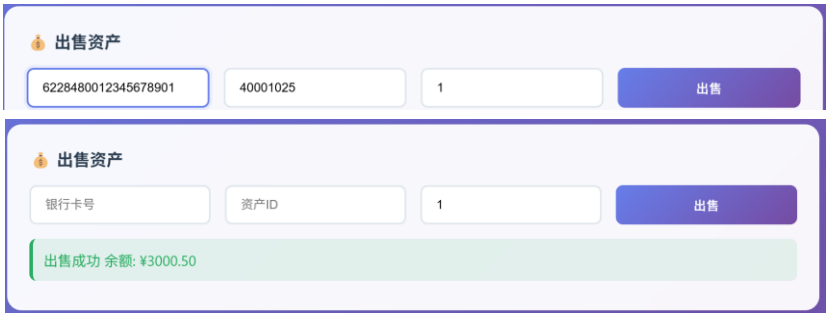
Module Name	Product Information - Product Information Inquiry
tester	L
Test type	Front-end functional testing
Testing tools	none
Use Case ID	BB-Product-01
Use Case Description	Query for existing product information
Prerequisites	Existing products
Expecting results	Display product details
Actual results	

6.4.2.4 Asset Information Inquiry

Module Name	Asset Management – Asset Information Inquiry
tester	L
Test type	Front-end functional testing
Testing tools	none
Use Case ID	BB-Asset-02
Use Case Description	Query asset information
Prerequisites	Enter the customer ID with asset information.
Expecting results	Return asset information
Actual results	

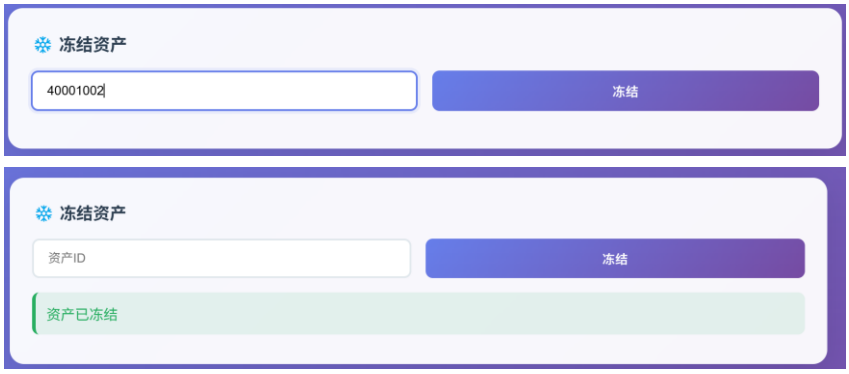
6.4.2.5 Sale of assets

Module Name	Asset Management – Selling Assets
tester	L
Test type	Front-end functional testing
Testing tools	none
Use Case ID	BB-Sell-01

Use Case Description	Normal sale of some assets
Prerequisites	Enter customer number, asset number, and quantity sold less than the quantity held.
Expecting results	Returning to success increases the balance and decreases assets.
Actual results	
Remark	

6.4.2.6 Frozen assets

Module Name	**Asset Management – Frozen Assets**
tester	L
Test type	Front-end functional testing
Testing tools	none
Use Case ID	BB-Freeze-01
Use Case Description	Normal asset freeze
Prerequisites	Enter customer number, asset number , and whether the asset is not frozen.
Expecting results	The system returns a success message, indicating that "Assets have been frozen."

Actual results	

6.5 Test Execution and Results

6.5.1 White-box test results

This white-box testing primarily targeted the critical code paths of the system's core functional modules, employing a logic coverage approach with a focus on statement coverage, decision coverage, and condition combination coverage. During the testing process, testers executed unit test cases for each functional module of the system, line by line, based on the test design outlined in the unit test documentation. Specifically, this included:

- Customer Information Management Module
- Bank card information processing module
- Product Information Management Module
- Asset Management and Trading Module

Test results show that all unit test cases passed normally, the system's internal code logic is clear, and the functional modules can execute correctly as designed under different input conditions, without issues such as infinite loops, missing logical branches, or abnormal crashes. Some tests verified the system's robustness through boundary values and special inputs, demonstrating that the system has effective protection against illegal inputs.

In summary, the system's key functions possess good logical correctness and robustness at the source code level, conform to design specifications and development standards, and are ready for stable online operation.

6.5.2 Black-box test results

Black-box testing primarily focuses on the user's perspective, employing a combination of functional testing, exception testing, and interface compatibility testing to comprehensively cover the functions of all system modules, ensuring stable operation

under various usage scenarios. This black-box testing primarily covered the following modules:

- Login function
- Bank card information inquiry
- Product Information Inquiry
- Asset sale function
- Asset Freeze Function
- Asset Information Inquiry
- System homepage interface status

During testing, various test cases were designed for the aforementioned functional modules, including normal processes, illegal inputs, boundary conditions, and abnormal operations, with a total of over 20 test cases executed . All black-box test cases were completed, and the actual test results were completely consistent with the expected results. The system functions stably, and no systemic defects or issues affecting normal use were found.

6.6 Test Conclusion

This test demonstrated that the bank management system is fully functional, operates smoothly, and interacts with data normally. All major functionalities were tested, and the results were consistent with expectations. The system exhibits good usability and stability, meeting the requirements for deployment.

suggestion:

- Further stress tests will be added to simulate high-concurrency scenarios.
- Strengthen security testing to prevent risks such as SQL injection and unauthorized operations.
- Optimize the front-end interface and prompts from a user experience perspective.
- The system can now enter the trial operation phase. We will continue to monitor the system's performance and improve its functionality.

7 Software maintenance

7.1 Maintenance Purpose

This software maintenance aims to improve the financial management system's performance in terms of functionality, stability, user experience, and future scalability. Through system testing and feedback analysis, potential operational issues were identified and fixed. Simultaneously, appropriate functional additions and structural optimizations were made based on user needs to ensure the system's long-term availability and maintainability in actual use.

7.2 Maintenance Scope

This maintenance work mainly targets the following modules:

Asset Management Module

- Fix the data matching error in the asset sale logic;
- Supplement front-end input validation and interactive prompts;
- Optimize the display and format of fields in the backend query interface.

Product Information Display Module

- Added display of product category and amount fields;
- Optimize the column order and style of the asset information list;
- Add field escaping and logical judgment to backend SQL queries to avoid errors.

Interface integration and data consistency

- A unified card number verification logic is implemented for asset purchase and sale operations;
- Enhance interface parameter validation and error feedback information to facilitate front-end prompts;
- Adjust the default values of some database fields and primary key constraints to improve data integrity.

7.3 Maintenance content

7.3.1 Functional maintenance

- Modify the /sell interface logic to resolve the issue of customers being unable to select a bank card when selling assets;
- Add card number input verification logic to ensure that users can only operate on assets and cards under their own name;
- Fixed an issue where incorrect field display was caused by non-standard SQL alias naming;

- Enhance the granularity of error messages by further subdividing "failure" feedback into clear categories such as "insufficient balance" and "asset freeze".

7.3.2 Interface data structure maintenance

- Standardize the names of backend interface fields (e.g., use English naming style for product_type_name).
- CASE in SQL queries to display the Chinese name of the product type improves the user-friendliness of the front-end display;
- The front-end asset information table has been updated to include product category and amount fields for a more comprehensive information display.

7.3.3 User interface maintenance

- Beautify the user interface
- Add prompts and error messages for input fields when users buy and sell products;
- Corrected table rendering order and format alignment issues;
- Add empty data and loading error messages to the front end to improve the user experience.

8 User Manual

8.1 Bank Management System

This chapter mainly provides a general overview of the features and functions of the bank management system .

8.1.1 Foreword

A bank management system is a specialized software system designed by a bank or financial institution to efficiently manage customer assets and financial operations. The system should be deployed on the bank's internal secure network or in an encrypted cloud environment. It provides core functions such as customer bank card information inquiry, financial product management, transaction processing, and personal asset management, serving as an integrated data processing platform for banking operations.

8.1.2 Software Features

Bank management system software has the following characteristics:

- using the Python language, it supports MySQL databases;
- The front-end uses HTML5+CSS3 to build a responsive interface, and integrates the Java Spring framework to implement a front-end and back-end separation architecture;
- The system uses customer accounts as the core carrier and establishes a three-tiered association model of "account-product-asset". Users can accurately search for basic product information and personal asset information , and independently update and manage asset information.

8.1.3 Software Functions

Bank management systems utilize information systems to realize functions such as customer information management, bank card management, financial product purchase, and asset management, which has significant practical implications.

This system is primarily designed for teaching banking business, student project training, and small-scale business simulations. Its functions are closely aligned with real banking business scenarios, with a focus on information management and basic operational processes. This facilitates students' mastery of database, front-end and back-end interaction, and system integration development capabilities .

8.1.3.1 System startup

Open the bank_frontend_ver4.html file to access the bank management system login page (Figure 7-1). After successful login, you will be taken to the main page of the bank management system .

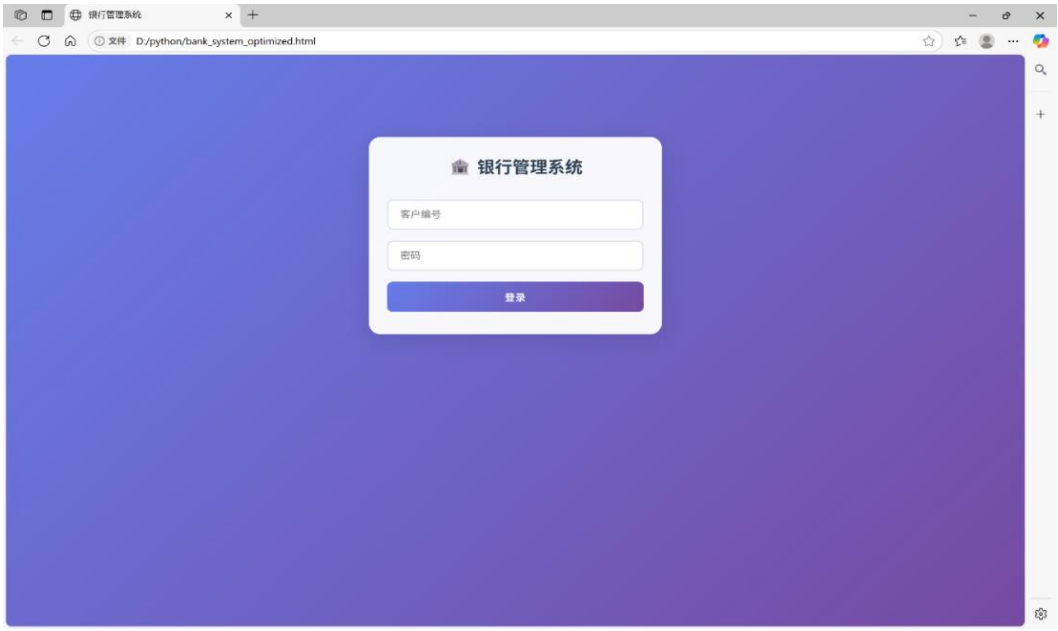


图8-1 System login page

8.1.3.2 System Homepage

the bank management system by entering the correct username and password .

7-2 , the user enters the main page :

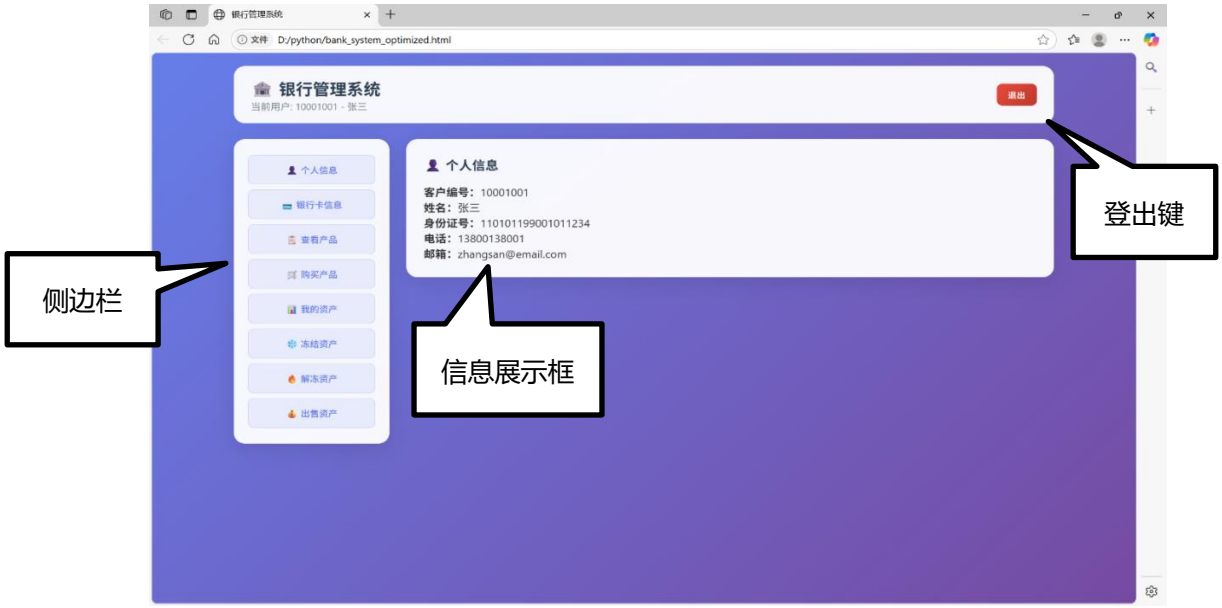


图8-2 Bank Management System Main Page

The system displays the customer's ID number, name, ID number, phone number, and email address . It consists of a sidebar , an information display box , and a logout button .

- [Sidebar]: Used to link personal information management, product purchase, and asset management ;
- [Information Display Box]: Used to display information on the current screen ;

- [Logout button]: Used to exit the current login and return to the login page ;

8.1.3.3 System operating platform

operating system: Windows 10

Database: MySQL 8.0

Development technologies and tools: Java， Flask， HTML5， CSS3， JavaScript

System operating tools : PyCharm， MySQL Workbench， browser （ Edge/Chrome ）

8.2 Personal Information Management

8.2.1 System Description

personal information management page allows users to query and browse their current personal information. After completing login authentication, users can query their personal bank card information ; clicking the query button will allow them to view their existing bank card information.

Key features:

- You can view your basic personal information;
- It can retrieve the bank card information currently held by the customer.

User interface diagram 7-3:

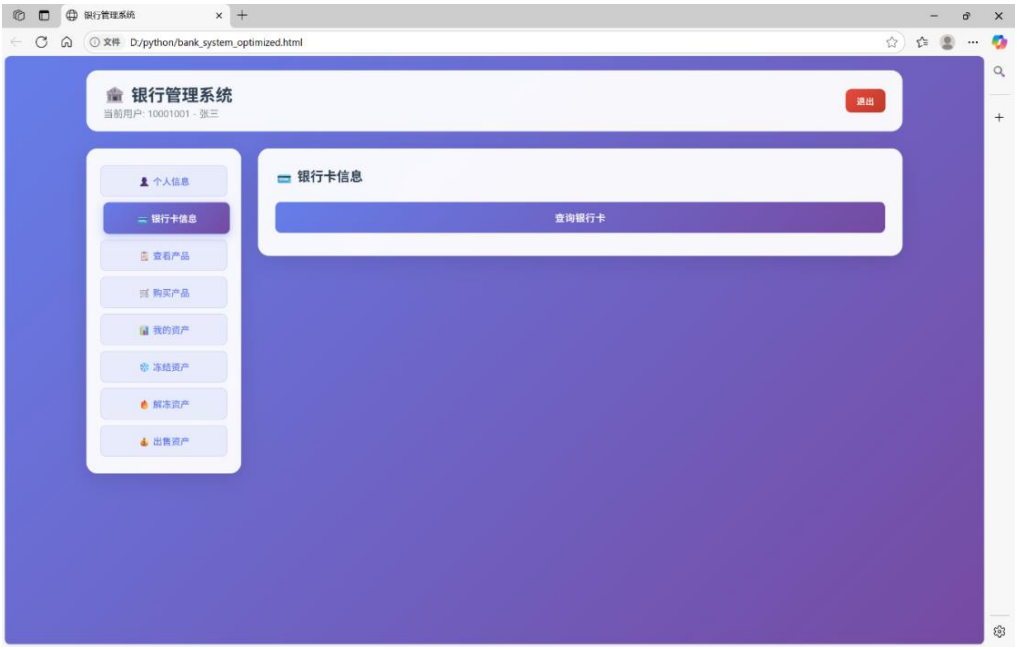


图8-3 Bank card information management

8.2.2 System Management Interface

Detailed instructions on function operation:

- Users can click on the bank card information in the sidebar to jump to the bank card information interface ;

- Clicking the "Query Bank Cards" button allows you to view the information of the bank cards you hold. The system lists all bank card numbers, card types, and balance information , as shown in Figure 7-4.

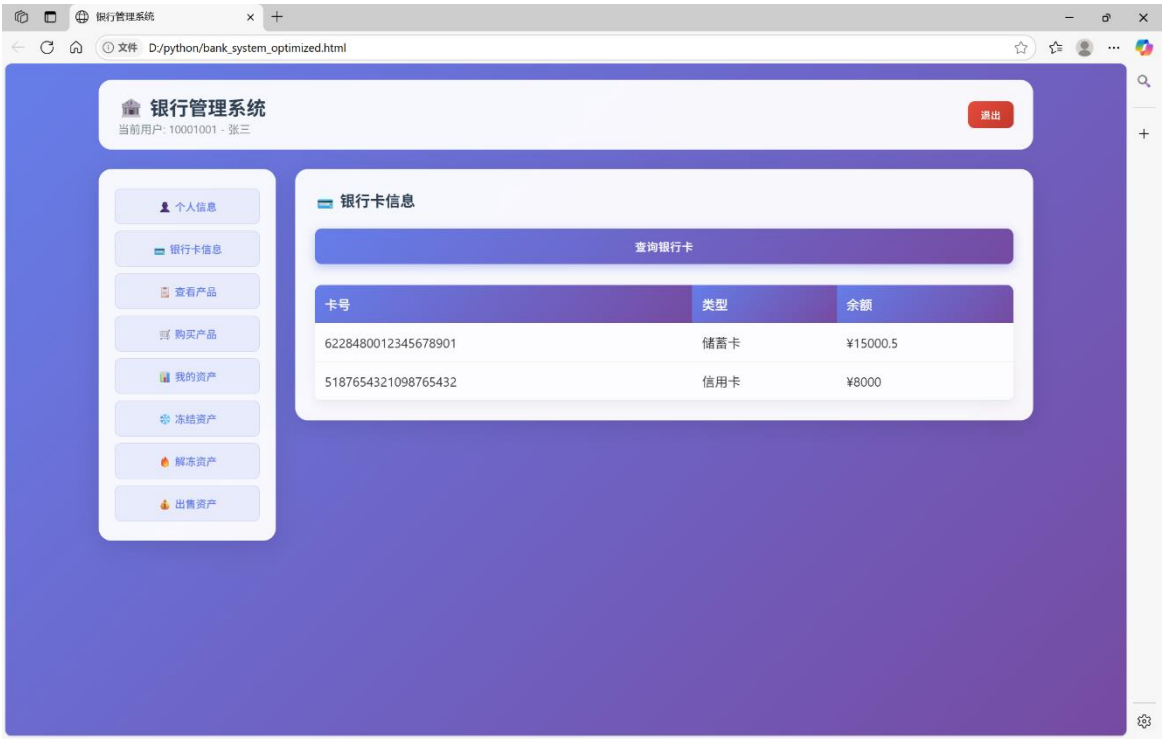


图8-4 View bank card information

8.3 Product purchase function

The product purchase function page provides users with basic functions for querying and purchasing existing products .

8.3.1 Product Information Inquiry and Purchase

Key features:

- Logged-in users can view information about existing products ;
- Logged-in users can purchase existing products.
- User interface diagram 7-5:

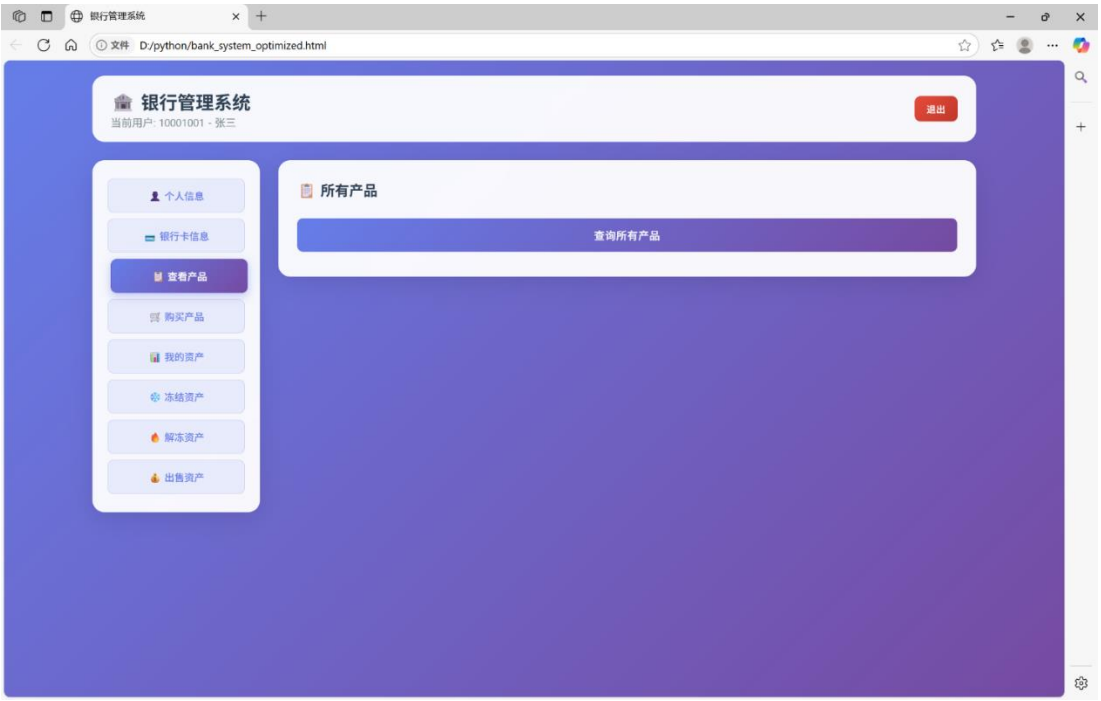


图8-5 Product Information Inquiry Page

Detailed instructions on function operation:

- Users can click "View Products" in the sidebar to jump to the query interface for all products ;
- Click the "Search All Products" button to view information on all wealth management, insurance, and fund products, including their number, name, amount, type, and status, as shown in Figure 7-6.

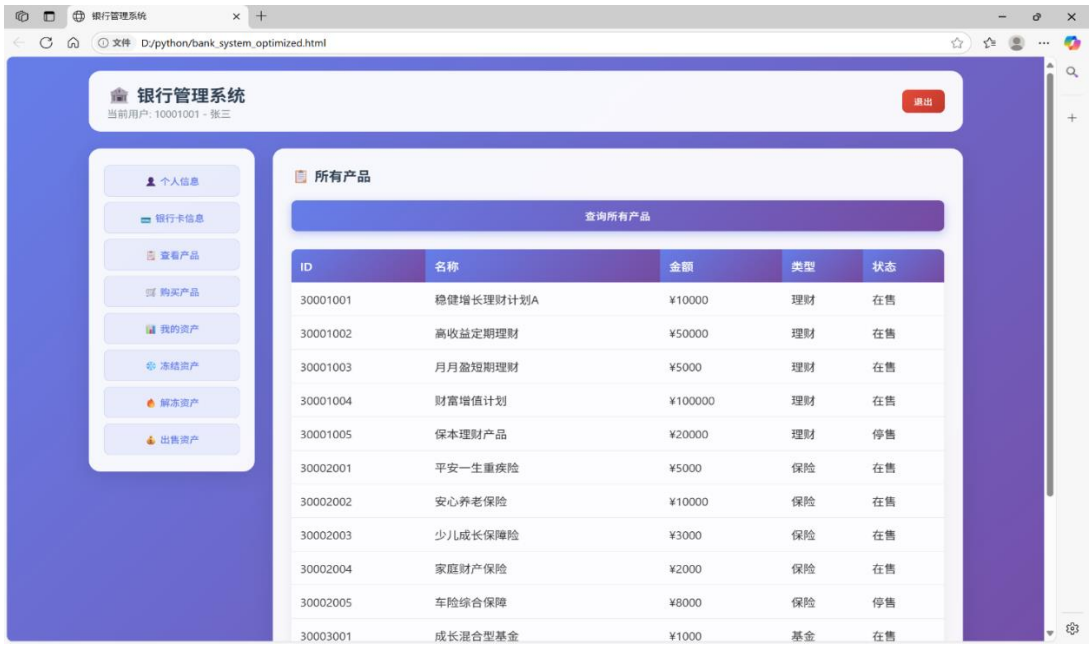


图8-6 All product information

- "Buy Products" in the sidebar to be redirected to the product purchase interface . They can then enter their bank card number, product ID , and quantity to complete the purchase, as shown in Figure 7-7.

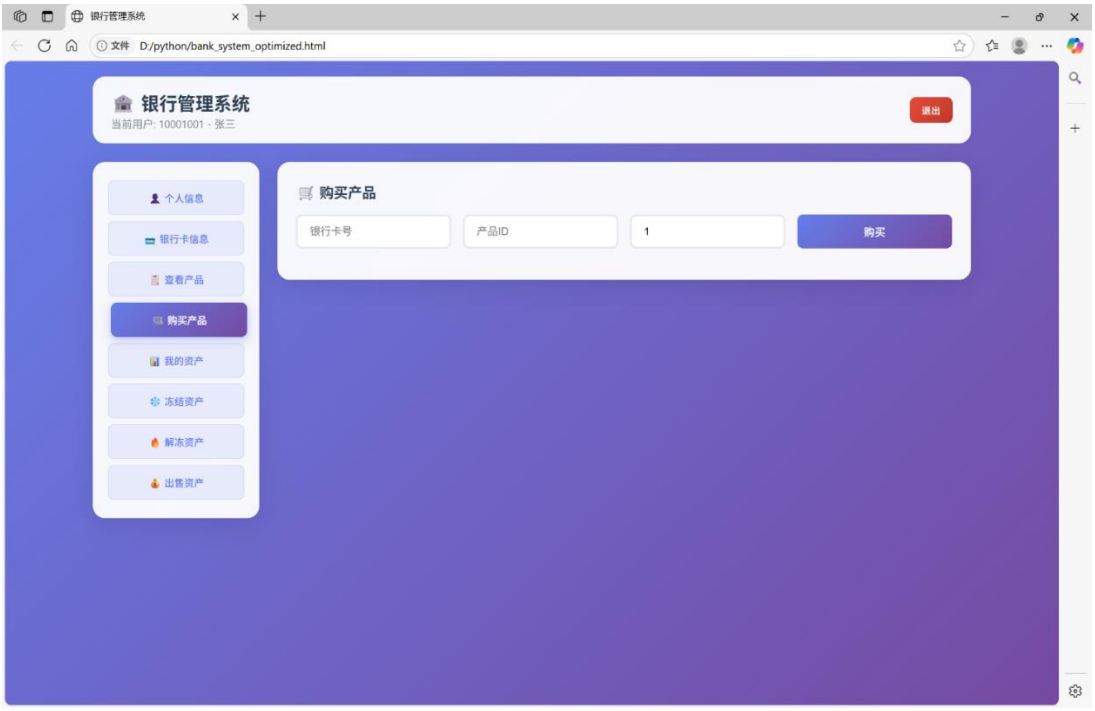


图8-7 Product purchase interface

8.4 Asset management function

The product purchase function page provides users with basic functions for querying and purchasing existing products .

8.4.1 Personal asset management

Key features:

- Logged-in users can view information about purchased products and sell existing products ;
- Logged-in users can freeze and unfreeze existing products.
- User interface diagram 7-8:

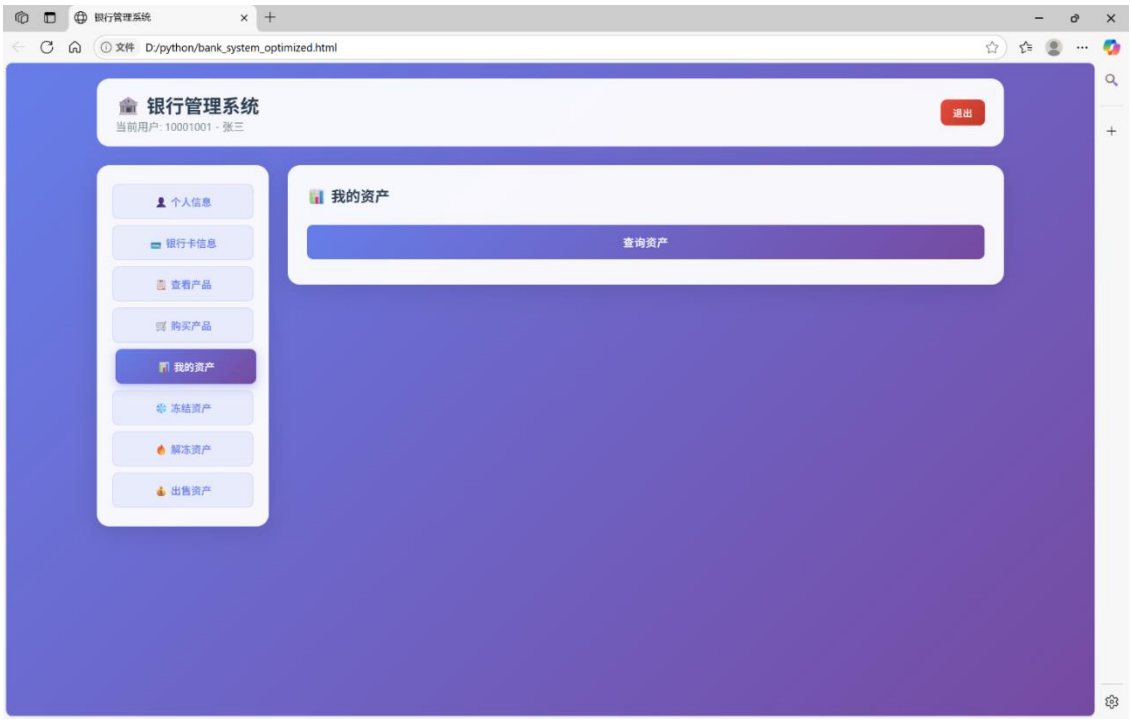


图8-8 Personal asset information inquiry page

Detailed instructions on function operation:

- Users can click on "My Assets" in the sidebar to jump to the interface for viewing all assets ;
- Click the "Query Assets" button to view all asset details: product name, amount, quantity, status, purchase time, returns, etc., as shown in Figure 7-9.

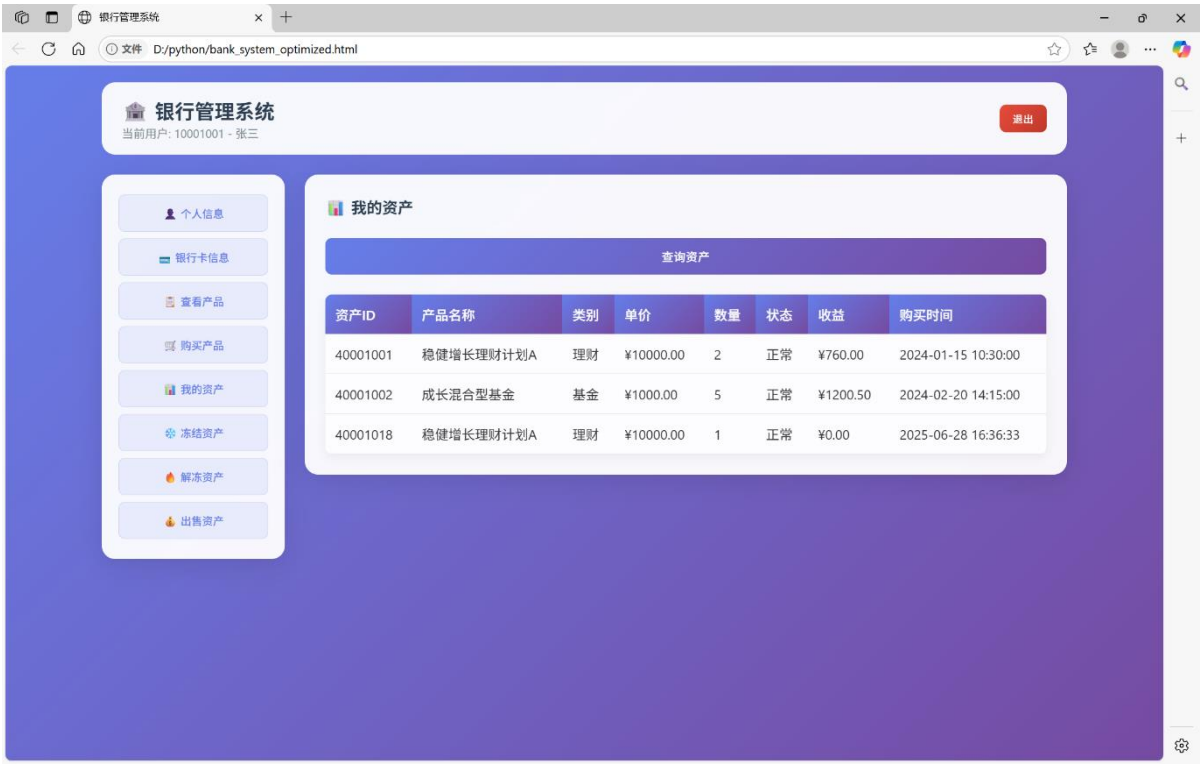


图8-9 My Asset Information

- "Freeze Assets" in the sidebar to access the frozen assets interface , enter the asset number, and click "Freeze," as shown in Figure 7-10.

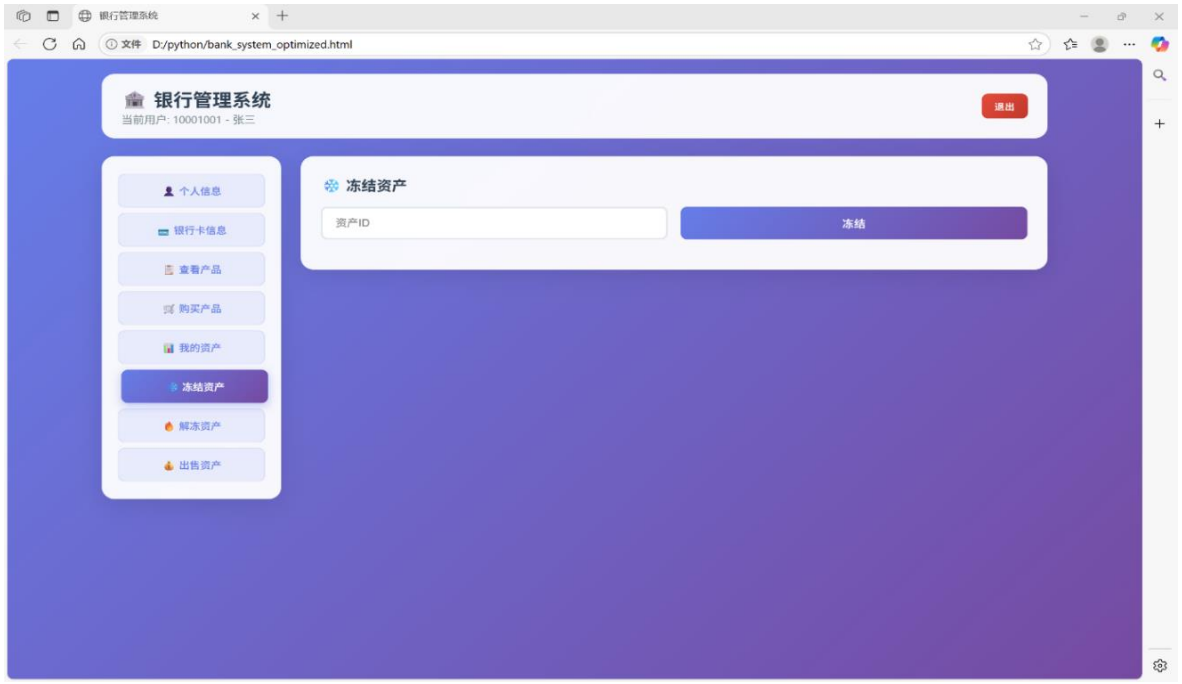


图8-10 Frozen Assets Interface

- Users can click on "Unfreeze Assets" in the sidebar to be redirected to the Unfreeze Assets interface. Enter the asset number and click "Unfreeze," as shown in Figure 7-11.

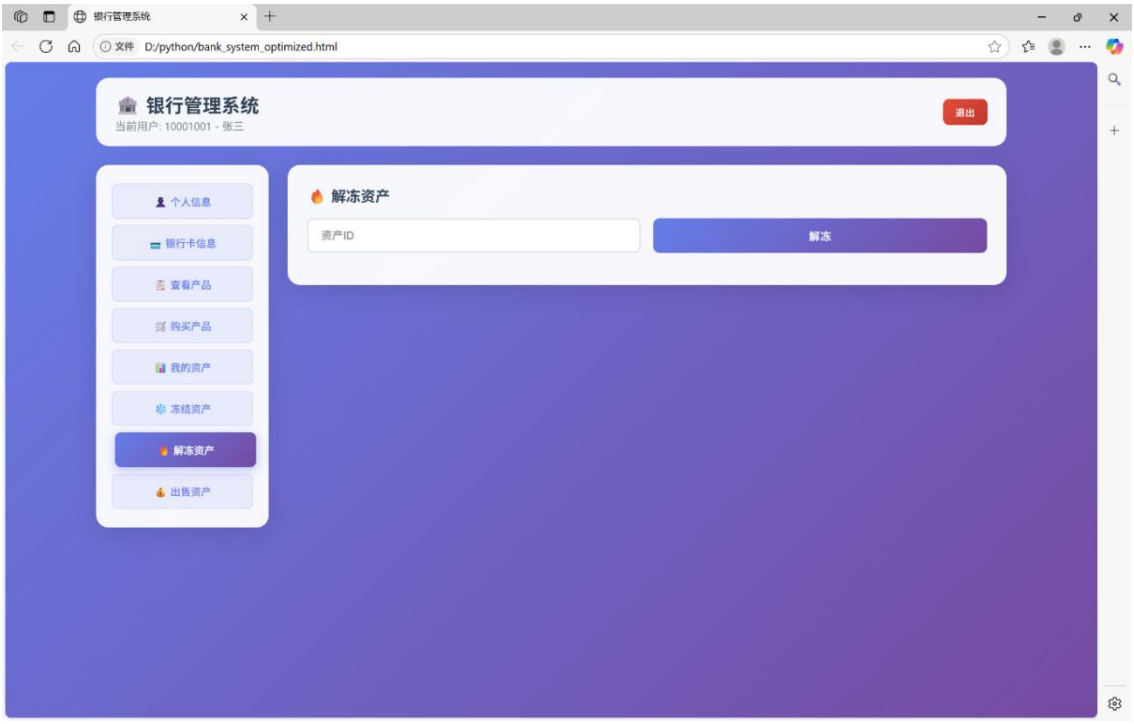


图8-11 Unfreeze Assets Interface

- "Sell Assets" in the sidebar to be redirected to the asset sale interface . They can then enter their bank card number, asset number, and quantity to sell, and click "Sell," as shown in Figure 7-12.

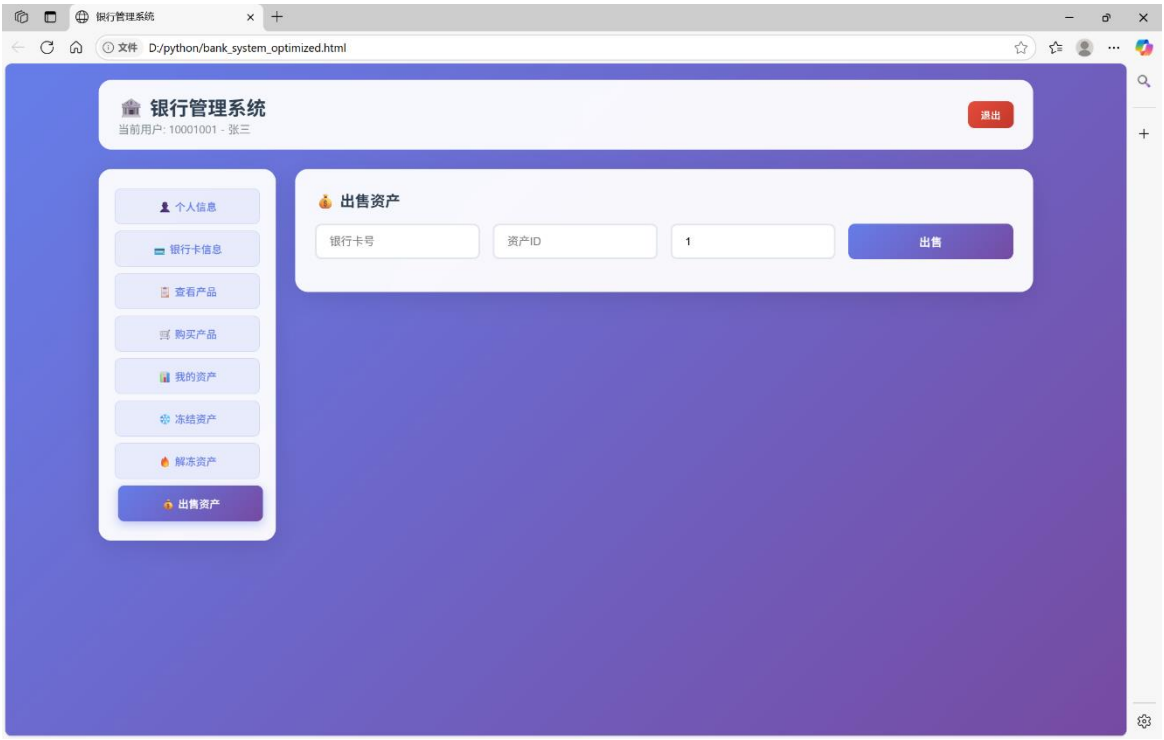


图8-12 Asset Sale Interface

9 Group division of labor and summary

9.1 Group division of labor

Student ID	Name	class	Division of labor
2 219040107	Li Jinying (Team Leader)	Dayu Computer	● Requirements analysis and conceptual design document for the purchasing and asset management module ● Database physical design and MySQL implementation ● Front-end HTML code writing, software testing, and software maintenance
2217020403	Zhang Xin	Dayu Computer	● Financial Product Management Module Requirements Analysis and Conceptual Design Document ● UI design, database logic design ● Front-end Vue code writing, software maintenance and user manual
2206050224	Lu Zeyuan	Dayu Computer	● Customer and Bank Card Management Module Requirements Analysis and Conceptual Design Document ● Detailed design, software implementation, and test report writing ● Writing backend Python code

9.2 Summarize

9.2.1 Li Jinying

In this comprehensive software practice, as the team leader of a three-person group, I made reasonable divisions of labor to ensure sufficient workload. My main responsibilities included the physical design of the database and the implementation of MySQL , the connection between the front-end and back-end, the writing of the front-end HTML , and the gradual improvement of the software's functions. I also wrote reports on the requirements analysis and high-level design of the purchasing and asset management module.

Our division of labor was quite reasonable, with everyone participating in the entire process of document writing, database development, and software implementation. After completing the

requirements analysis and high-level design documents, the three of us were responsible for the front-end Vue , the database MySQL , and the back-end Python implementation, respectively. Due to time constraints, we didn't properly understand and summarize the documents, which led to some incompatibility between the front-end and back-end implementations. The solution was to rewrite the HTML front-end, ultimately delivering a complete, working bank management system with certain functionalities.

In summary, through this experience, I gained a certain understanding of the entire software development process, learned how to create standardized documentation, write code, and implement databases, and also appreciated the significance of teamwork in development.

9.2.2 Zhang Xin

In this comprehensive software practice, I was mainly responsible for writing the front-end code using the Vue framework, designing and improving various functions, and also for the requirements analysis, high-level design document, and user manual for the product management module. I also designed UI diagrams to facilitate my later front-end design.

Although I have front-end design experience, my previous projects did not involve writing back-end code; instead, I created empty front-end frameworks. So, in this three-person team, we started working directly based on my previous experience, forgetting to consider the connection between the front-end and back-end functions and how to implement them. This resulted in a problem that took a whole day to solve. We gradually discovered and solved the problem, and eventually resolved the connection issue. However, the differences in functionality were too great, so our team opted to redesign a different front-end framework based on the back-end design, using a better solution.

Overall, this practical course made me realize the importance of framework design before writing code. Following the prescribed steps ensures that the software design stays on the right track. In solving the front-end and back-end connection problem, I also learned for the first time how to write the back-end and the front-end and back-end connection process, enriching my project experience.

9.2.3 Lu Zeyuan

In this comprehensive software practice, I was mainly responsible for writing the backend Python code, which involved functions such as login, customer information, bank card management, asset purchase and sale, and freezing and unfreezing. I also participated in writing the requirements analysis, detailed design, software implementation, and testing sections of the customer and bank card management modules in the documentation.

This was my first time designing and completing a project, and I was indeed inexperienced. At first, I didn't know where to start. Later, I referred to the PPT and reference documents provided by the teacher and tried to imitate them. It was like stumbling along step by step, but in the end, I still managed to complete the project successfully. There were also many problems with the connection between the backend Python and the frontend. Since it was completed by different members, some parts were not compatible. In the end, we completed it together in a group discussion.

Overall, this experiment not only solidified my programming skills but also honed my systematic thinking and teamwork abilities, yielding fruitful results. Moving forward, I will continue to strengthen my practical training and enhance my comprehensive development capabilities. It provided me with invaluable project experience, laying a solid foundation for my future studies and work. I will continue to strengthen my practical training and enhance my comprehensive development capabilities.

10 References

[1] UML Diagram Types and Examples

[2] UML System Analysis and Design Tutorial (2nd Edition)

[3] Database physical design example

[4] Database logical design examples

[5] Water quality database standard (SL325-2005)

[6] Yangtze River Commission xxx Development and Testing Report

[7] Yangtze River Commission xxx Operation Manual

[8] Detailed design of the Yangtze River Commission

[9] Yangtze River Commission xxx Requirements Specification

11 Appendix: Core Code

11.1 Database connection configuration

```
DB_CONFIG = {
'host': 'localhost', 'user': 'root', # MySQL username
'password': 'bank123', # MySQL password
'database': 'bank', # Database name
'port': 3306, 'charset': 'utf8mb4'}
```

11.2 Login Interface

```
@app.route("/login", methods=["POST"])
def login(): data = request.json
    customer_id = data.get (" customer_id ")
password = data.get ("password")

conn = get_conn ()
cursor = conn.cursor ()

# Validate customer ID and password
    cursor.execute ("SELECT c_name , c_password FROM CUSTOMER WHERE c_id
=%s", ( customer_id ,))
row = cursor.fetchone ()

if row is None: conn.close ()
return jsonify ({"status": "fail", "message": "Customer does not exist"})
       c_name , c_password = row
if password != c_password :
       conn.close ()
return jsonify ({"status": "fail", "message": "Incorrect password"})
       conn.close ()
# Returns the customer's name; the front-end displays it using
return jsonify ({"status": "success", "message": "Login successful", "
customer_id ": customer_id , " customer_name ": c_name })
```

11.3 Purchase Product Interface

```
@app.route("/buy", methods=["POST"])
def buy_product ():
data = request.json
    customer_id = data.get (" customer_id ")
    card_number = data.get (" card_number ")
    product_id = data.get (" product_id ")
```

```
quantity = data.get ("quantity", 1)

conn = get_conn ()
cursor = conn.cursor ()

# Query product information
    cursor.execute ("SELECT pi_amount , pi_state FROM PRODUCT_INFO WHERE
pi_id =%s", ( product_id ,))
product = cursor.fetchone ()

# Query bank card information
    cursor.execute ("SELECT b_id , b_balance FROM BANKCARD WHERE b_c_id =%s
AND b_number =%s", ( customer_id , card_number ))
bankcard = cursor.fetchone ()

if product and bankcard and product[1] == "1": #The product is on sale
       total_price = product[0] * quantity
if bankcard[1] >= total_price :
           new_balance = bankcard[1] - total_price
           cursor.execute ("UPDATE BANKCARD SET b_balance =%s WHERE b_id
=%s", ( new_balance , bankcard[0]))
           cursor.execute ("""
INSERT INTO ASSET ( as_c_id , as_pif_id , as_type , as_quantity , as_status ,
as_income , as_purchase_time )
VALUES (%s, %s, (SELECT pi_type FROM PRODUCT_INFO WHERE pi_id =%s), %s,
'normal', 0, NOW())
""", ( customer_id , product_id , product_id , quantity))
           conn.commit ()
           `conn.close ()`
returns `jsonify ({"status": "success", "message": "Purchase successful",
"balance": new_balance })`.
Otherwise, ` conn.close ()`
returns `jsonify ({"status": "fail", "message": "Insufficient balance"})`.

    conn.close ()
return jsonify ({"status": "fail", "message": "Product discontinued or
information error"})
```