VIETNAM GENERAL CONFEDERATION OF LABOR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

Tran Duc Huy – 523H0033

Nguyen Duc Anh – 523H0002

Phan Van Duong – 523H0017

MIDTERM REPORT

iBanking Application – Tuition Payment Subsystem

Ho Chi Minh City, November 2025

VIETNAM GENERAL CONFEDERATION OF LABOR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

Tran Duc Huy – 523H0033

Nguyen Duc Anh – 523H0002

Phan Van Duong – 523H0017

MIDTERM REPORT

iBanking Application – Tuition Payment Subsystem

Advised by

PhD. Duong Huu Phuc

Ho Chi Minh City, November 2025

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to **PhD. Duong Huu Phuc**, our instructor and mentor, for his valuable guidance and support throughout the mid-term report of our project on the *iBanking Application – Tuition Payment Subsystem*. He has been very helpful and patient in providing us with constructive feedback and suggestions to improve our work. We have learned a lot from his expertise in microservices and software engineering.

Ho Chi Minh City, *November 2025*

Authors

Tran Duc Huy

Nguyen Duc Anh

Phan Van Duong

DECLARATION OF AUTHORSHIP

We hereby declare that this is our own project and is guided by **PhD. Duong Huu Phuc**. The content, research, and results contained herein are original and have not been published in any form before. The data in the tables for analysis, comments, and evaluation are collected by the main author from different sources, which are clearly stated in the reference section.

In addition, the project also uses some comments, assessments as well as data of other authors, other organizations with citations and annotated sources.

If something wrong happens, we'll take full responsibility for the content of our project. Ton Duc Thang University is not related to the infringing rights, the copyrights that we give during the implementation process (if any).

Ho Chi Minh City, ***November 2025***

Authors

Tran Duc Huy

Nguyen Duc Anh

Phan Van Duong

CONTENTS

ABBREVIATIONS

API	Application Programming Interface
ERD	Entity Relationship Diagram
JWT	JSON Web Token
OTP	One-Time Password
REST	Representational State Transfer
SQL	Structured Query Language
SMTP	Simple Mail Transfer Protocol

Chapter 1 INTRODUCTION

1.1 Project Overview

This project implements a tuition payment subsystem using microservices architecture with FastAPI (Python 3.11), MySQL 8.0, and Docker containerization. The system consists of 6 independent services communicating through an API Gateway.

1.2 System Components

- **API Gateway** (Port 8000): Request routing and JWT validation
- **Auth Service** (Port 8001): User authentication
- **Customer Service** (Port 8006): Profile and balance management
- **Tuition Service** (Port 8002): Tuition records and payment eligibility
- **Payment Service** (Port 8003): Transaction processing
- **OTP Service** (Port 8004): OTP generation and verification

1.3 Key Features

- Sequential tuition payment (oldest unpaid first)
- OTP-based two-factor authentication
- Email notifications (OTP codes and invoices)
- Transaction history tracking

Chapter 2 SYSTEM ANALYSIS AND DESIGN

2.1 Use Case Diagram

The Use Case Diagram illustrates interactions between Customer, OTP Service, and Mail Service actors with system functionalities.

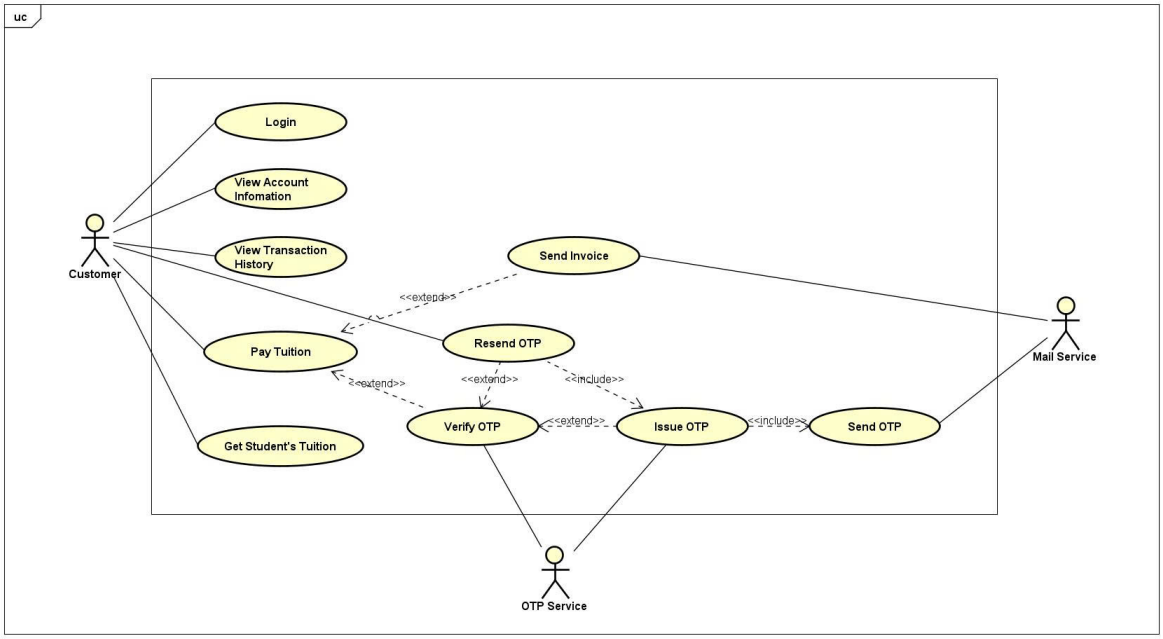

Figure 2.1: Use Case Diagram

2.1.1 Use Case Descriptions

Use Case	**Description**
Login	Customer authenticates with username/password
View Account Information	Display customer profile and balance
Get Student's Tuition	Retrieve tuition records with `canPay` flag
Pay Tuition	Create payment transaction (extends Issue OTP and Verify OTP)
Issue OTP	Generate 6-digit OTP (includes Send OTP to email)
Verify OTP	Validate OTP code for payment authorization
Resend OTP	Request new OTP if expired
View Transaction History	Display completed payment transactions
Send Invoice	Email payment confirmation (Mail Service)

Table 2.1: Use Case Summary

2.2 Entity Relationship Diagram

The ERD shows 4 tables across 4 separate databases for microservices independence.

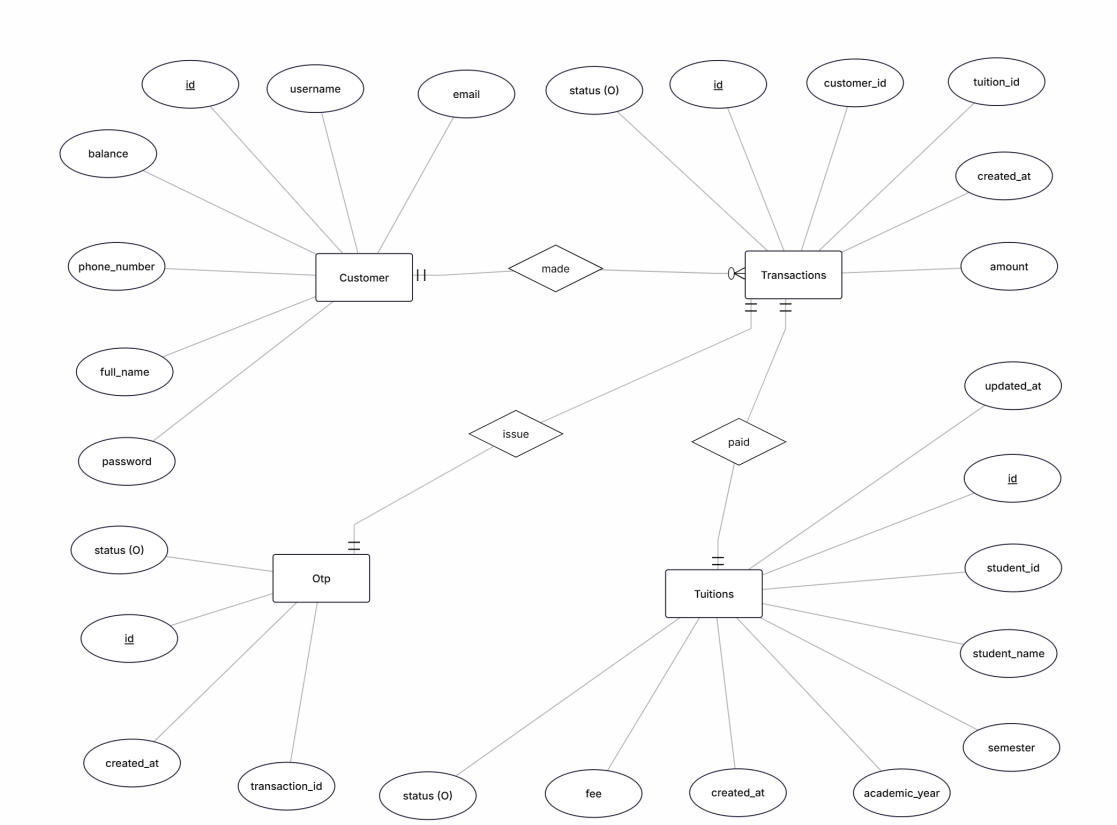

Figure 2.2: Entity Relationship Diagram

2.2.1 Table Relationships

- **customers**: Stores user accounts with username, email, password, balance, fullname, phonenumber
- **tuitions**: Links to customers via `student_id` (logical relationship)
- **transactions**: References `customer_id` and `tuition_id`
- **otp**: Associates with transactions via `transaction_id` (unique)

Note: Foreign keys are not physically enforced as tables exist in different databases.

Chapter 3 DATABASE IMPLEMENTATION

3.1 Physical Database Architecture

The system uses 4 separate MySQL 8.0 databases deployed in Docker containers.

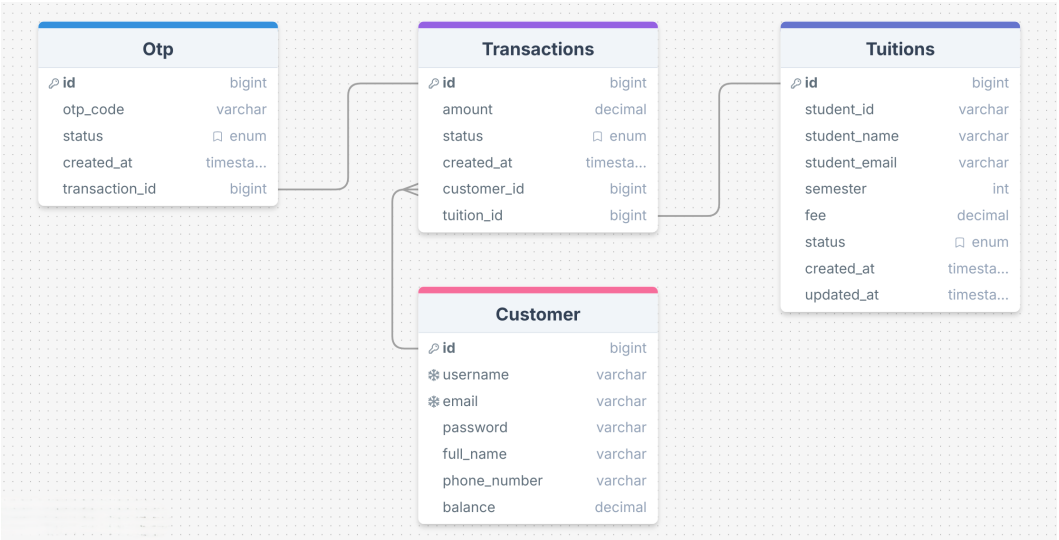

Figure 3.1: Physical Database Deployment Architecture

Database	Container	Port
customer_db	customer-db	3307:3306
tuition_db	tuition-db	3308:3306
payment_db	payment-db	3309:3306
otp_db	otp-db	3310:3306

Table 3.1: Database Container Mapping

3.2 Physical Database Implementation

All databases are implemented in MySQL 8.0 and initialized via `init.sql` scripts during Docker container startup.

3.2.1 customer_db Database

Listing 3.1: customer-service/init.sql

```
CREATE DATABASE IF NOT EXISTS customer_db;
USE customer_db;

CREATE TABLE IF NOT EXISTS customers (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    full_name VARCHAR(100) NOT NULL,
    phone_number VARCHAR(20) NOT NULL,
    balance DECIMAL(15,2) DEFAULT 0,
    INDEX idx_username (username),
    INDEX idx_email (email)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

3.2.2 customer_db Database

Listing 3.2: customer-service/init.sql

```sql
CREATE DATABASE IF NOT EXISTS customer_db;
USE customer_db;

CREATE TABLE IF NOT EXISTS customers (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    full_name VARCHAR(100) NOT NULL,
    phone_number VARCHAR(20) NOT NULL,
    balance DECIMAL(15,2) DEFAULT 0,
    INDEX idx_username (username),
    INDEX idx_email (email)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Sample data
INSERT INTO customers (username, email, password, full_name,
   phone_number, balance)
VALUES
('user123', 'user@example.com', 'password123', 'Nguyen␣Van␣User
   ', '0901234567', 10000000),
('admin', 'admin@example.com', 'admin123', 'Administrator', '
   0904567890', 50000000);
```

3.2.3 tuition_db Database

Listing 3.3: tuition-service/init.sql

```sql
CREATE DATABASE IF NOT EXISTS tuition_db CHARACTER SET utf8mb4
    COLLATE utf8mb4_unicode_ci;
USE tuition_db;

CREATE TABLE IF NOT EXISTS tuitions (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    student_id VARCHAR(20) NOT NULL,
    student_name VARCHAR(100) NOT NULL,
    student_email VARCHAR(100) NOT NULL,
    semester INT NOT NULL,
    academic_year VARCHAR(20) NOT NULL,
    fee DECIMAL(15,2) NOT NULL,
    status ENUM('unpaid', 'paid') DEFAULT 'unpaid',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
        CURRENT_TIMESTAMP,
    INDEX idx_student_id (student_id),
    INDEX idx_student_year_semester (student_id, academic_year,
        semester)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Sample data
INSERT INTO tuitions (student_id, student_name, student_email,
    semester, academic_year, fee, status)
VALUES
('52000123', 'Nguyen␣Van␣A', '52000123@student.tdtu.edu.vn', 2,
    '2023-2024', 0, 'paid'),
('52000123', 'Nguyen␣Van␣A', '52000123@student.tdtu.edu.vn', 1,
    '2024-2025', 5000000, 'unpaid'),
('52000123', 'Nguyen␣Van␣A', '52000123@student.tdtu.edu.vn', 2,
    '2024-2025', 5000000, 'unpaid');
```

3.2.4 payment_db Database

Listing 3.4: payment-service/init.sql

```sql
CREATE DATABASE IF NOT EXISTS payment_db;
USE payment_db;

CREATE TABLE IF NOT EXISTS transactions (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    customer_id BIGINT NOT NULL,
    tuition_id BIGINT NOT NULL,
    amount DECIMAL(15,2) NOT NULL,
    status ENUM('pending', 'completed','cancelled') DEFAULT '
       pending' NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    INDEX idx_customer_id (customer_id),
    INDEX idx_tuition_id (tuition_id),
    INDEX idx_customer_status (customer_id, status)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

3.2.5 otp_db Database

Listing 3.5: otp-service/init.sql

```sql
CREATE DATABASE IF NOT EXISTS otp_db;
USE otp_db;

CREATE TABLE IF NOT EXISTS otp (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    otp_code VARCHAR(6) NOT NULL,
    transaction_id BIGINT UNIQUE NOT NULL,
    status ENUM('active', 'used', 'expired') DEFAULT 'active'
       NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    INDEX idx_otp_code (otp_code),
    INDEX idx_transaction_id (transaction_id),
    INDEX idx_otp_code_status (otp_code, status)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Chapter 4 API DOCUMENTATION

4.1 API Endpoints Overview

The iBanking TDTU system uses a microservices architecture with an API Gateway as the central access point. All requests from the frontend should go through the API Gateway (port 8000) to automatically handle authentication and routing.

4.1.1 API Gateway Endpoints (Port 8000)

Table 4.1: Public API Endpoints - Recommended for Frontend

Method	**Endpoint**	**Description**	**Auth**
POST	/api/auth/login	User login	No
POST	/api/auth/logout	User logout	No
GET	/api/auth/me	Get current user information	Yes
PUT	/api/auth/profile	Update user profile	Yes
GET	/api/customers/me	Get customer information	Yes
PUT	/api/customers/update-profile	Update customer profile	Yes
POST	/api/students/search	Search student tuition fees	Yes
POST	/api/transactions/init	Create transaction + Send OTP	Yes
POST	/api/transactions/confirm	Confirm payment with OTP	Yes
GET	/api/transactions/history	Transaction history	Yes

4.2 Authentication API

4.2.1 POST /api/auth/login

Description: API for logging into the system with username and password.

Request Body:

```
{
  "username": "user123",
  "password": "password123"
}
```

Response (Success - 200 OK):

```
{
  "user": {
    "id": 1,
    "username": "user123",
    "email": "phanvanduong1223456@gmail.com",
    "balance": 10000000
  }
}
```

Response Headers:

```
Set-Cookie: access_token=eyJhbGciOiJIUzI1NiIs...;
            HttpOnly; Secure; SameSite=Strict;
            Max-Age=86400; Path=/
```

Business Logic:

1. Receive username and password from client
2. Call Customer Service `/api/customers/search` (Internal API) to verify credentials
3. Generate JWT token containing `user_id`, `username`, `email`
4. Set JWT token in HttpOnly cookie with 24-hour expiration
5. Return user information (token not included in response body)

Error Responses:

- **401 Unauthorized:** Invalid username or password
- **500 Internal Server Error:** Connection error with Customer Service

4.2.2 POST /api/auth/logout

Description: Log out of the system by deleting the JWT cookie.

Response (Success - 200 OK):

```
{
  "message": "Logged out successfully"
}
```

4.3 Customer Profile API

4.3.1 GET /api/customers/me

Description: Get detailed information of the current customer.

Headers Required:

```
Cookie: access_token=<JWT_TOKEN>
```

Response (Success - 200 OK):

```
{
  "id": 1,
  "username": "user123",
  "email": "phanvanduong1223456@gmail.com",
  "full_name": "Nguyen Van User",
  "phone_number": "0901234567",
  "balance": 5000000.00
}
```

4.3.2 PUT /api/customers/update-profile

Description: Update customer profile information.

Request Body (all fields are optional):

```
{
  "username": "newusername",
  "email": "newemail@example.com",
  "full_name": "Nguyen Van B",
  "phone_number": "0987654321",
  "current_password": "password123",
  "new_password": "newpassword123"
}
```

Response (Success - 200 OK):

```
{
  "success": true,
  "message": "Profile updated successfully",
  "user": {
    "id": 1,
    "username": "newusername",
    "email": "newemail@example.com",
    "full_name": "Nguyen Van B",
    "phone_number": "0987654321",
    "balance": 5000000.00
  }
}
```

Business Logic:

1. Check username/email uniqueness if changed
2. Require `current_password` if changing password
3. Validate `new_password` length >= 6 characters
4. Hash new password before saving to database

4.4 Tuition Search API

4.4.1 POST /api/students/search

Description: Search all tuition fees of a student by student code.

Request Body:

```
{
  "student_code": "52000123"
}
```

Note: API Gateway automatically converts `student_code` to `student_id` before forwarding the request to Tuition Service.

Response (Success - 200 OK):


```
{
  "student": {
    "student_id": "52000123",
    "student_name": "Nguyen Van A",
    "student_email": "52000123@student.tdtu.edu.vn"
  },
  "all_tuitions": [
    {
      "id": 1,
      "student_id": "52000123",
      "semester": 2,
      "academic_year": "2023-2024",
      "fee": 0,
      "status": "paid",
      "canPay": false
    },
    {
      "id": 2,
      "student_id": "52000123",
      "semester": 1,
      "academic_year": "2024-2025",
      "fee": 5000000.00,
      "status": "unpaid",
      "canPay": true
    },
    {
      "id": 3,
      "student_id": "52000123",
      "semester": 2,
      "academic_year": "2024-2025",
      "fee": 5000000.00,
      "status": "unpaid",
      "canPay": false
    }
  ]
}
```

Business Logic - canPay Flag:

- Tuition fees are sorted by: `academic_year ASC`, `semester ASC`
- Only the **oldest unpaid** tuition fee has `canPay = true`
- Students must pay tuition fees in order, cannot skip old fees
- In the example above, only semester 1 2024-2025 tuition (id=2) can be paid

4.5 Payment Transaction APIs

4.5.1 POST /api/transactions/init

Description: Create a payment transaction and send OTP code via email.

Request Body:

```
{
  "student_code": "52000123"
}
```

Response (Success - 200 OK):

```
{
  "success": true,
  "transaction_id": 1,
  "tuition_info": {
    "id": 2,
    "semester": 1,
    "academic_year": "2024-2025",
    "amount": 5000000.00
  },
  "message": "OTP has been sent via email. Please check your
     inbox.",
  "expires_in_minutes": 5
}
```

Business Logic Flow:

1. API Gateway receives `student_code`, converts to `student_id`
2. Gateway forwards request to OTP Service `/api/otp/issue`
3. OTP Service calls Payment Service `/api/transactions/create` (Internal)
4. Payment Service calls Tuition Service `/get-payable` to get payable tuition
5. Create transaction with status `pending` in database
6. OTP Service generates 6-digit random OTP code
7. Send email containing OTP code to customer
8. Return `transaction_id` and tuition information

Email Template:

```
Subject: [iBanking TDTU] OTP Verification Code for Tuition
   Payment

Your OTP code: 123456

Tuition: Semester 1 2024-2025
Amount: 5,000,000 VND

OTP code is valid for 5 minutes.
```

4.5.2 POST /api/transactions/confirm

Description: Confirm payment using OTP code received from email.

Request Body:

```
{
  "otp_code": "123456",
  "student_id": "52000123"
}
```

Response (Success - 200 OK):

```
{
  "success": true,
  "message": "Payment successful",
  "transaction": {
    "id": 1,
    "customer_id": 1,
    "tuition_id": 2,
    "amount": 5000000.00,
    "status": "completed",
    "created_at": "2025-11-09T10:30:00"
  },
  "new_balance": 5000000.00
}
```

Response (Error - Invalid OTP):

```
{
  "detail": "OTP is invalid or expired"
}
```

Response (Error - Insufficient Balance):

```
{
  "detail": "Insufficient balance. Current balance: 1,000,000 VND, Required: 5,000,000 VND"
}
```

Business Logic Flow (11 steps):

1. Get `customer_id` from JWT token (via X-Customer-ID header)
2. Call OTP Service `/api/otp/verify` (Internal) to verify OTP
3. OTP Service returns `transaction_id` if OTP is valid
4. Lock transaction using `SELECT FOR UPDATE` (prevent race condition)
5. Verify transaction belongs to customer and status = "pending"
6. Call Customer Service to get current balance
7. Check if `balance >= amount`
8. Call Customer Service `/deduct-balance` to deduct money
9. Call Tuition Service `/{tuition_id}/mark-paid` to mark as paid
10. Update transaction status = "completed"
11. Send invoice email (fire-and-forget, no wait for response)

Invoice Email Template:

```
Subject: [iBanking TDTU] Tuition Payment Invoice

Transaction ID: TXN00000001
Tuition: Semester 1 2024-2025
Amount: 5,000,000 VND
Remaining balance: 5,000,000 VND
Time: 2025-11-09 10:30:00

Thank you for using our service!
```

4.5.3 GET /api/transactions/history

Description: Get customer's transaction history.

Request: No body (customer_id automatically retrieved from JWT)

Response (Success - 200 OK):

```
{
  "transactions": [
    {
      "id": 1,
      "customer_id": 1,
      "tuition_id": 2,
      "amount": 5000000.00,
      "status": "completed",
      "created_at": "2025-11-09T10:30:00"
    },
    {
      "id": 2,
      "customer_id": 1,
      "tuition_id": 3,
      "amount": 5000000.00,
      "status": "pending",
      "created_at": "2025-11-09T11:00:00"
    }
  ]
}
```

4.6 Authentication & Authorization

4.6.1 JWT Token Structure

JWT token is generated with the following structure:

```
{
  "user_id": 1,
  "username": "user123",
  "email": "phanvanduong1223456@gmail.com",
  "exp": 1699520000
}
```

Token Properties:

- **Storage:** HttpOnly cookie with key `access_token`
- **Expiry:** 24 hours (86400 seconds)
- **Algorithm:** HS256
- **Secret Key:** Retrieved from environment variable `JWT_SECRET_KEY`

4.6.2 Header Requirements

Via API Gateway (Port 8000):

- Only requires Cookie with JWT token
- Does NOT require X-Customer-ID (Gateway auto-injects)
- Does NOT require X-API-Key

Direct Service Access (Ports 8001-8006):

- Requires X-Customer-ID for public endpoints
- Requires X-API-Key for internal endpoints
- X-API-Key value: `sk_live_51KxYz9H8mN2pQ3rS4tU5vW6xY7zA8bC9dE0fG1hI2jK3lM4nO5pQ6r`

4.7 Internal APIs

Internal APIs are used for communication between microservices. These APIs require X-API-Key header and should not be called directly from frontend.

4.7.1 Payment Service → Tuition Service

POST /get-payable - Get payable tuition fee

```
Request: { "student_id": "52000123" }

Response: {
  "success": true,
  "tuition": {
    "id": 2,
    "student_id": "52000123",
    "semester": 1,
    "academic_year": "2024-2025",
    "fee": 5000000.00,
    "status": "unpaid"
  }
}
```

4.7.2 Payment Service → Customer Service

POST /api/customers/deduct-balance - Deduct balance

```
Request: {
  "customer_id": 1,
  "amount": 5000000.00,
  "transaction_code": "TXN00000001"
}

Response: {
  "success": true,
  "new_balance": 5000000.00,
  "old_balance": 10000000.00
}
```

4.7.3 OTP Service → Payment Service

POST /api/transactions/create - Create transaction

```
Request: {
  "customer_id": 1,
  "student_id": "52000123"
}

Response: {
  "id": 1,
  "customer_id": 1,
  "tuition_id": 2,
  "amount": 5000000.00,
  "status": "pending",
  "created_at": "2025-11-09T10:25:00"
}
```

4.8 Complete Payment Flow Example

Below is a complete example of how to perform tuition payment from frontend using JavaScript:

```
// 1. Login
const loginResponse = await fetch('http://localhost:8000/api/
   auth/login', {
  method: 'POST',
  credentials: 'include',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    username: 'user123',
    password: 'password123'
  })
});
const user = await loginResponse.json();
console.log('Logged in:', user);
// 2. Search tuitions
const searchResponse = await fetch('http://localhost:8000/api/
   students/search', {
  method: 'POST',
```

```
  credentials: 'include',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ student_code: '52000123' })
});
const tuitions = await searchResponse.json();
console.log('Tuitions:', tuitions);
// 3. Initialize payment (creates transaction + sends OTP)
const initResponse = await fetch('http://localhost:8000/api/
    transactions/init', {
  method: 'POST',
  credentials: 'include',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ student_code: '52000123' })
});
const otpResult = await initResponse.json();
console.log('OTP sent:', otpResult);

// 4. Confirm payment with OTP (user enters OTP from email)
const confirmResponse = await fetch('http://localhost:8000/api/
    transactions/confirm', {
  method: 'POST',
  credentials: 'include',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    otp_code: '123456',
    student_id: '52000123'
  })
});
const paymentResult = await confirmResponse.json();
console.log('Payment result:', paymentResult);

// 5. View transaction history
const historyResponse = await fetch('http://localhost:8000/api/
    transactions/history', {
  method: 'GET',
  credentials: 'include'
});
const history = await historyResponse.json();
console.log('Transaction history:', history);
```

4.9 HTTP Status Codes

Table 4.2: HTTP Status Codes Used in System

Code	**Meaning**	**When**
200	OK	Request successful
400	Bad Request	Validation error, business logic error
401	Unauthorized	Token invalid/expired, missing auth
404	Not Found	Resource does not exist
500	Internal Server Error	Server error
502	Bad Gateway	Backend service unavailable
503	Service Unavailable	Service temporarily unavailable

Chapter 5 TESTING AND USER INTERFACE

5.1 Postman API Testing

The system includes a comprehensive Postman collection for testing all API endpoints.

5.1.1 Test Coverage Summary

Test Category	**Test Cases**
Authentication Tests	2
Customer Service Tests	2
Tuition Service Tests	2
Payment Service Tests	3
OTP Service Tests	2
Error Handling Tests	5
Total	**16**

Table 5.1: Postman Test Summary

5.2 User Interface

The system provides a web-based interface with 6 pages for complete user interaction.

5.2.1 Login Page

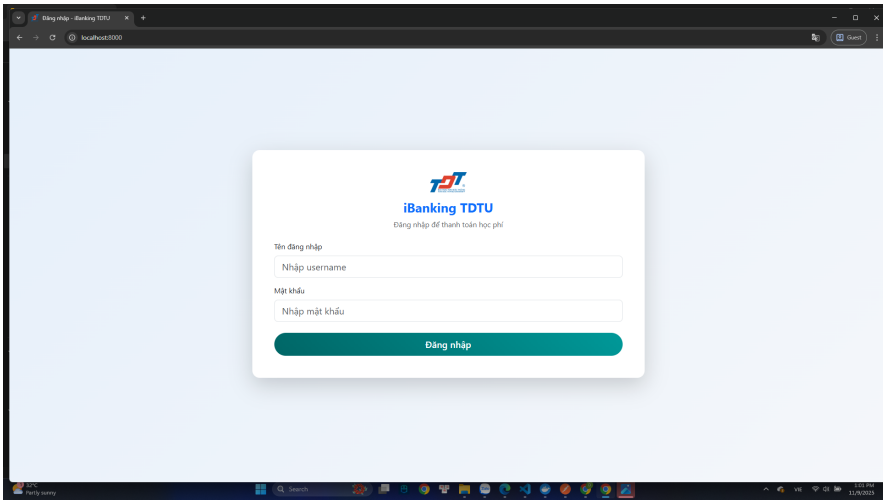

Figure 5.1: Login Page

Users authenticate with username and password. JWT token stored in HTTP-only cookie.

5.2.2 Profile Page

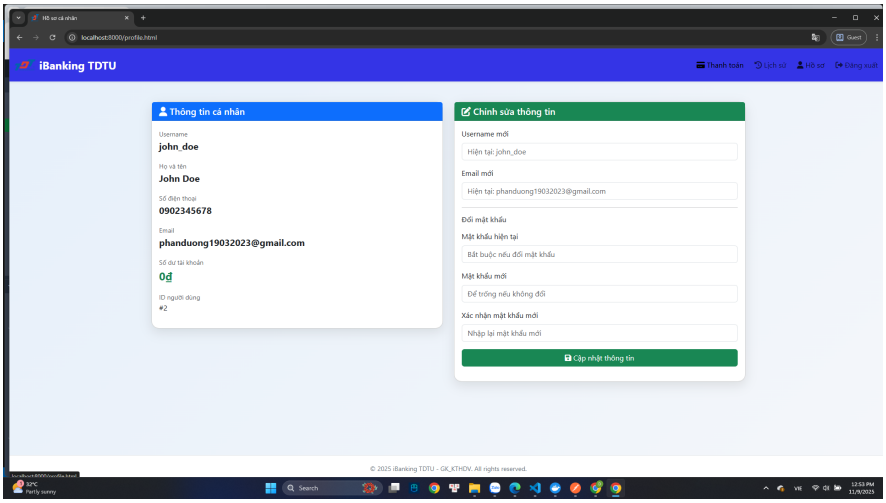

Figure 5.2: Customer Profile Page

Displays customer information (full name, email, phone number) and current account balance.

5.2.3 Payment Page

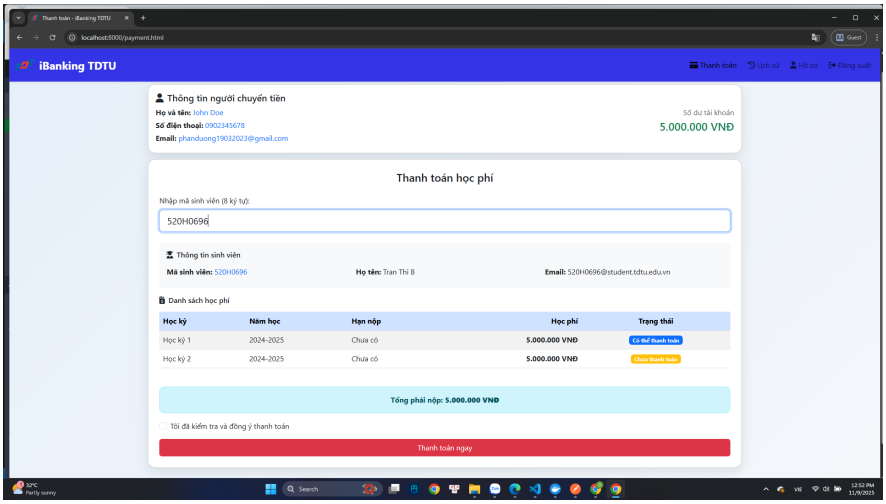

Figure 5.3: Tuition Payment Page

Lists all tuition records. Only tuitions with `canPay=true` have enabled "Pay" button.

5.2.4 OTP Verification Page

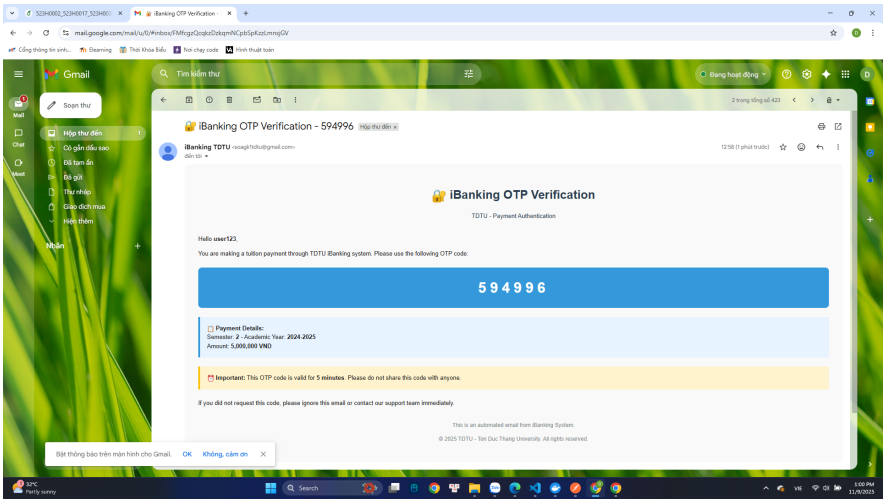

Figure 5.4: OTP Mail

Customer received

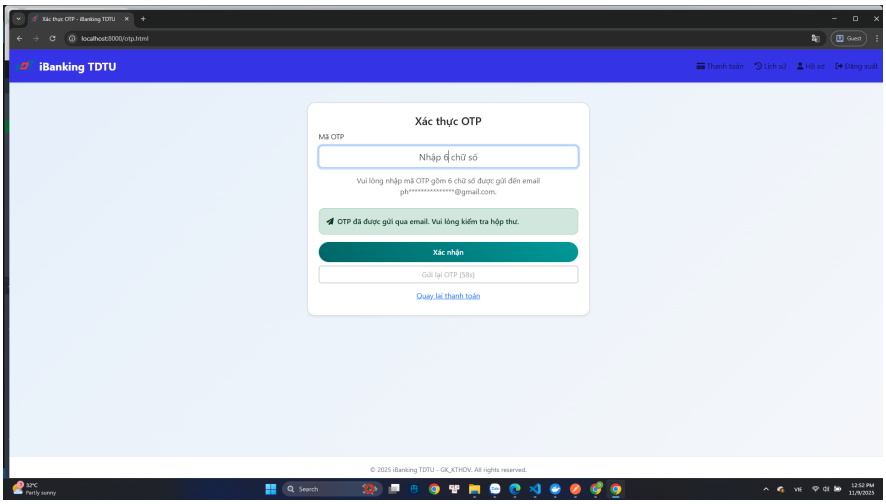

Figure 5.5: OTP Verification Page

Students enter 6-digit OTP received via email. Code expires in 5 minutes.

5.2.5 Success Page

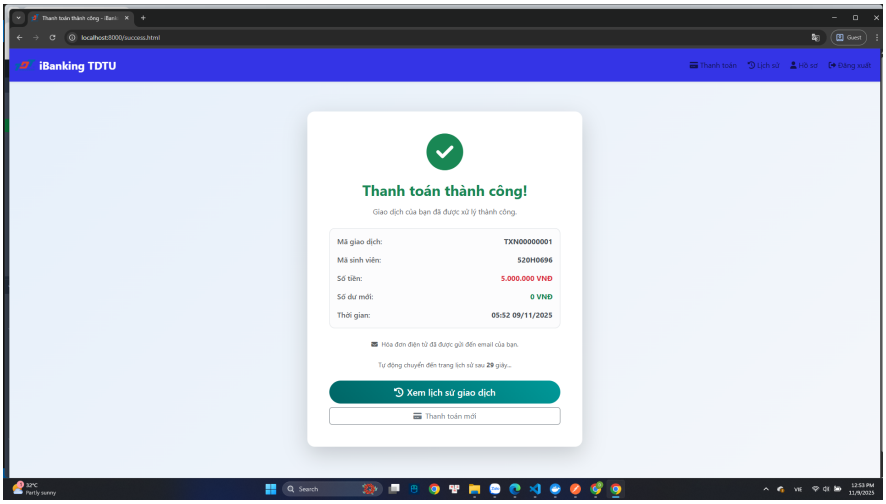

Figure 5.6: Payment Success Page

Confirmation page showing transaction details. Invoice sent to student email.

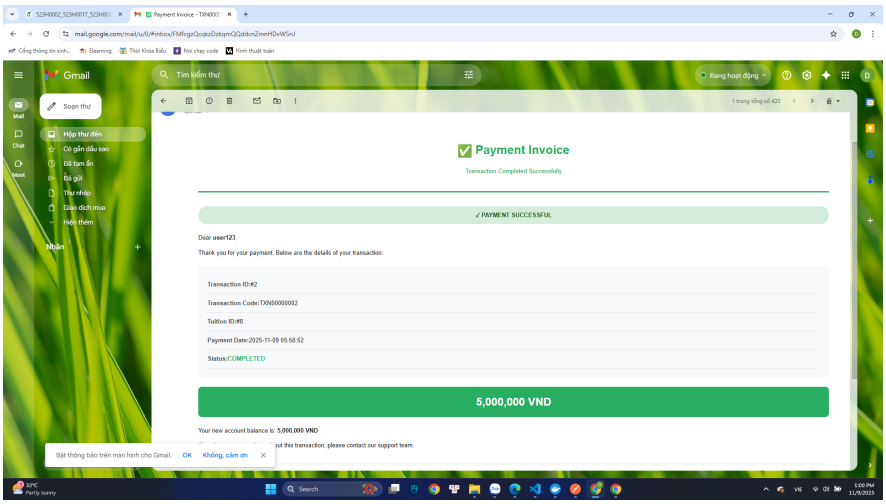

Figure 5.7: Mail receipt

5.2.6 Transaction History Page

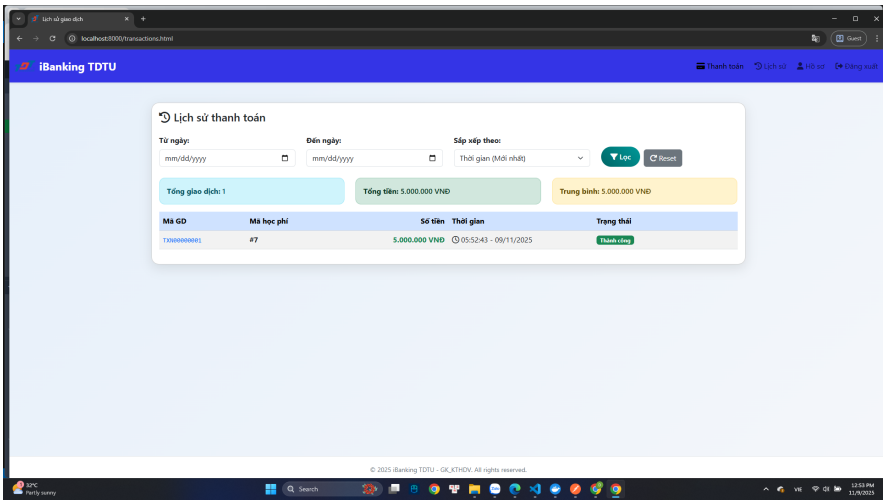

Figure 5.8: Transaction History Page

Table displaying all completed transactions with amount, status, and timestamp.

Chapter 6 CONCLUSION

6.1 Achievement Summary

This project successfully fulfilled all requirements:

1. **Use Case Diagram**: Visualized system functionalities with actors (Customer, OTP Service, Mail Service)
2. **ERD Diagram**: Designed 4-table database structure across 4 separate databases
3. **Physical Database**: Implemented all databases using MySQL 8.0 with proper schemas and indexes
4. **API Documentation**: Listed 11 API endpoints with clear Input/Output specifications
5. **Postman Collection**: Created 16 test cases covering all services
6. **API Implementation**: Implemented all functionalities with RESTful URLs (all endpoints operational)
7. **User Interface**: Developed 6-page web interface for complete payment workflow

6.2 Key Features

- Sequential payment logic (`canPay` flag)
- OTP-based two-factor authentication
- Race condition prevention (database row locking)
- Email integration (SMTP for OTP and invoices)
- Docker containerization (10 containers)

6.3 Technologies Used

- **Backend**: FastAPI (Python 3.11), SQLAlchemy ORM
- **Database**: MySQL 8.0 (4 databases)
- **Frontend**: Vanilla JavaScript, HTML5, CSS3, Bootstrap 5
- **Deployment**: Docker Compose 3.9
- **Authentication**: JWT with HTTP-only cookies

REFERENCES

1. FastAPI Documentation, https://fastapi.tiangolo.com/
2. SQLAlchemy Documentation, https://docs.sqlalchemy.org/
3. Docker Documentation, https://docs.docker.com/
4. MySQL 8.0 Reference Manual, https://dev.mysql.com/doc/
5. Bootstrap 5 Documentation, https://getbootstrap.com/