TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**PHAN ĐÌNH PHÚ – 52300051**

**NGÔ XUÂN QUANG – 52300055**

**NGUYỄN DUY QUÝ - 52300056**

**BÁO CÁO ĐỒ ÁN GIỮA KỲ**

**KIẾN TRÚC HƯỚNG DỊCH VỤ**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2025**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**PHAN ĐÌNH PHÚ – 52300051**

**NGÔ XUÂN QUANG – 52300055**

**NGUYỄN DUY QUÝ - 52300056**

**BÁO CÁO ĐỒ ÁN GIỮA KỲ**

**KIẾN TRÚC HƯỚNG DỊCH VỤ**

Người hướng dẫn

**ThS. Nguyễn Thị Hường**

Nhóm thực hiện

**Nhóm 06**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2025**

**LỜI CẢM ƠN**

Lời đầu tiên, chúng em xin gửi lời cảm ơn chân thành đến Trường Đại Học Tôn Đức Thắng và khoa Công Nghệ Thông Tin đã đưa môn học kiến trúc hướng dịch vô cùng chất lượng và bổ ích vào chương trình giảng dạy. Bọn em đã tiếp thu được rất nhiều kiến thức cần thiết phục vụ cho không những các môn học khác mà còn trên hành trình trở thành những kỹ sự phần mềm. Đặc biệt là lời cảm ơn sâu sắc tới giảng viên bộ môn, ThS. Nguyễn Thị Hường, người đã tận tình chỉ bảo và truyền đạt và những kiến thức trong sách vở, những kinh nghiệm quan trọng ngoài thực tiễn cho chúng em.

Những kiến thức quý giá của cô đã được chúng em áp dụng vào bài báo cáo lần này. Mặc dù chúng em đã cố gắng hết sức nhưng chắc chắn bài làm khó có thể tránh khỏi những thiếu sót và nhiều chỗ còn chưa chính xác, kính mong thầy xem xét và góp ý để bài của chúng em được hoàn thiện hơn.

Chúng em xin chân thành cảm ơn!

*TP. Hồ Chí Minh, ngày 01 tháng 10 năm 2025*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

*Phan Đình Phú*

*Ngô Xuân Quang*

*Nguyễn Duy Quý*

**CÔNG TRÌNH ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Chúng em xin cam đoan đây là công trình nghiên cứu của riêng chúng em và được sự hướng dẫn khoa học của ThS. Nguyễn Thị Hường. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong Dự án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào chúng em xin hoàn toàn chịu trách nhiệm về nội dung Dự án của mình**. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do chúng em gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 01 tháng 10 năm 2025*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

*Phan Đình Phú*

*Ngô Xuân Quang*

*Nguyễn Duy Quý*

**MỤC LỤC**

[CHƯƠNG 1. GIỚI THIỆU HỆ THỐNG 7](#_Toc210861253)

[1.1 Mô tả chung 7](#_Toc210861254)

[1.2 Giả định 7](#_Toc210861255)

[1.3 Thông tin Service 7](#_Toc210861256)

[1.4 Chi tiết Service 7](#_Toc210861257)

[1.4.1 UserService 7](#_Toc210861258)

[1.4.1.1 Chức năng 7](#_Toc210861259)

[1.4.1.2 API chính 7](#_Toc210861260)

[1.4.2 AccountService 8](#_Toc210861261)

[1.4.2.1 Chức năng 8](#_Toc210861262)

[1.4.2.2 API chính 8](#_Toc210861263)

[1.4.3 TuitionService 10](#_Toc210861264)

[1.4.3.1 Chức năng 10](#_Toc210861265)

[1.4.3.2 API chính 10](#_Toc210861266)

[1.4.4 OTPNotificationService 11](#_Toc210861267)

[1.4.4.1 Chức năng 11](#_Toc210861268)

[1.4.4.2 API chính 11](#_Toc210861269)

[1.4.5 Payment Service 13](#_Toc210861270)

[1.4.5.1 Chức năng 13](#_Toc210861271)

[1.4.5.2 API chính 13](#_Toc210861272)

[1.5 Công nghệ sử dụng 14](#_Toc210861273)

[CHƯƠNG 2. SƠ ĐỒ TỔNG QUAN 14](#_Toc210861274)

[1. Mô hình use case 14](#_Toc210861275)

[2. Đặc tả use case 14](#_Toc210861276)

[1. Use case đăng nhập 14](#_Toc210861277)

[2. Use case xem thông tin tài khoản và lịch sử giao dịch 16](#_Toc210861278)

[3. Use case xem số dư tài khoản 17](#_Toc210861279)

[4. Use case tra cứu thông tin học phí 18](#_Toc210861280)

[5. Use case thanh toán 19](#_Toc210861281)

[6. Use case nhập OTP 21](#_Toc210861282)

[7. Use case gửi lại mã OTP 22](#_Toc210861283)

[CHƯƠNG 3. THIẾT KẾ DỮ LIỆU 23](#_Toc210861284)

[1. Sơ đồ thực thể - mối quan hệ (ERD) và mô hình quan hệ 24](#_Toc210861285)

[2. Đặc tả cơ sở dữ liệu 24](#_Toc210861286)

[1. Bảng User – UserService 24](#_Toc210861287)

[1.1. Thuộc tính 25](#_Toc210861288)

[1.2 Đặc tả 25](#_Toc210861289)

[2. Account – AccountService 25](#_Toc210861290)

[2.1 Thuộc tính 25](#_Toc210861291)

[2.2 Đặt tả 25](#_Toc210861292)

[3. Bảng Transaction – AccountService 25](#_Toc210861293)

[3.1 Thuộc tính 25](#_Toc210861294)

[3.2 Đặc tả 26](#_Toc210861295)

[4. Bảng Student – TuitionService 26](#_Toc210861296)

[4.1Thuộc tính 26](#_Toc210861297)

[4.2 Đặc tả 26](#_Toc210861298)

[CHƯƠNG 4. TÀI LIỆU KIỂM THỬ 26](#_Toc210861299)

[1. User Service 26](#_Toc210861300)

[2. Account Service 27](#_Toc210861301)

[3. Tuition Service 28](#_Toc210861302)

**DANH MỤC BẢNG BIỂU**

[Bảng 1: Use case đăng nhập 15](#_Toc210860891)

[Bảng 2: Use case xem thông tin tài khoản và lịch sử giao dịch 16](#_Toc210860892)

[Bảng 3: Use case xem số dư tài khoản 17](#_Toc210860893)

[Bảng 4: Use case tra cứu thông tin sinh viên 18](#_Toc210860894)

[Bảng 17: Use case thanh toán 20](#_Toc210860895)

[Bảng 6: Use case nhập OTP 21](#_Toc210860896)

[Bảng 7: Use case gửi lại mã OTP 22](#_Toc210860897)

**DANH MỤC HÌNH VẼ**

[Hình 1: Sơ đồ Usecase tổng quát 13](#_Toc210860962)

[Hình 2: Sơ đồ ERD 23](#_Toc210860963)

[Hình 3: Mô hình quan hệ 23](#_Toc210860964)

# CHƯƠNG 1. GIỚI THIỆU HỆ THỐNG

## Mô tả chung

Phân hệ đóng học phí trong ứng dụng iBanking cho phép người dùng đăng nhập, nhập thông tin học phí, xác thực giao dịch thông qua OTP nhận từ email và thực hiện thanh toán học phí cho chính mình hoặc cho sinh viên khác. Hệ thống tập trung vào việc đảm bảo an toàn, tính nhất quán và xử lý đồng thời các giao dịch.

## Giả định

* Các thông tin về người dùng như họ tên, số điện thoại, email hay số dư ban đầu được thêm sẵn trong database. Hệ thống không cần chức năng đăng ký tài khoản mới.
* Dữ liệu học phí của sinh viên bao gồm họ tên, mã số sinh viên hay học phí cần thanh toán được thêm sẵn trong database. Mã số sinh viên là duy nhất.
* Hệ thống chỉ mô phỏng việc trừ tiền trong tài khoản người dùng và gạch nợ học phí. Không cần kết nối với hệ thống của ngân hàng thực tế hay cổng thanh toán quốc tế.

## Thông tin Service

* **UserService**: Quản lý đăng nhập và thông tin hồ sơ người dùng (họ tên, số điện thoại, email)
* **BalanceService**: Quản lý số dư khả dụng và lịch sử giao dịch.
* **PaymentService**: Điều phối giao dịch thanh toán.
* **TuitionService**: Quản lý thông tin sinh viên và học phí (MSSV, họ tên, số tiền học phí, trạng thái).
* **OTPNotificationService**: Tạo, xác thực OTP, và gửi email (OTP, xác nhận giao dịch).

## Chi tiết Service

### UserService

#### Chức năng

* Xác thực đăng nhập (username, password).
* Quản lý thông tin hồ sơ người dùng (họ tên, số điện thoại, email).

#### API chính

**POST /auth/login:** Xác thực đăng nhập

* Input: { username: string, password: string }.
* username: Không rỗng, tối đa 50 ký tự.
* password: Không rỗng, tối thiểu 8 ký tự.
* Output
* Thành công: {userId: number}.
* Lỗi: 401 Unauthorized: { error: "Invalid username or password", code: 401 }
* Lỗi: 400 Bad Request:{ "error": "Username or password cannot be empty", "code": 400 }.

**GET /users/{userId}**: Lấy thông tin hồ sơ người dùng.

* Input: userId
* userId: Số nguyên không âm.
* Output
* Thành công: {userId: number, fullName: string, phone: string, email: string}.
  + Lỗi: 404 Not Found { "error": "User not found", "code": 404 }
  + Lỗi: 400 Bad Request{ "error": "Invalid userId", "code": 400 }

### AccountService

#### Chức năng

* Quản lý số dư khả dụng.
* Quản lý lịch sử giao dịch.
* Xử lý khóa/mở khóa tài khoản để tránh xung đột số dư.

#### API chính

**GET /accounts/{userId}/balance: Lấy số dư.**

* Input: userId
* userId: Số nguyên không âm
* Output
* Thành công: {balance}
* Lỗi: 404 Not Found { "error": "User not found", "code": 404 }
* Lỗi: 400 Bad Request{ "error": "Invalid userId", "code": 400 }

**GET/accounts/{userId}/history: Lấy lịch sử giao dịch**

* Input: userId
* userId: Số nguyên không âm
* Output
* Thành công: [ { transactionId: string, type: string, description: string, amount: number, timestamp: string} ]
* Lỗi: 400 Bad Request: { "error": "Invalid userId", "code": 400 }
* Lỗi: 404 Not Found: { "error": "User not found", "code": 404 }

**POST /transactions: Lưu giao dịch.**

* Input: { userId: number, mssv: string, type: string, amount: number, description: string}
* userId, mssv: Không rỗng
* type: Không rỗng, phải là một trong các loại hợp lệ
* description: Tối đa 100 ký tự
* Output
* Thành công: { transactionId: string, timestamp: string, status: success}
* Lỗi: 400 Bad Request: { "error": "Invalid input data", "code": 400 }
* Lỗi: 404 Not Found: { "error": "User not found", "code": 404 }
* Ghi chú:
* Thuộc tính status được back-end set tự động dựa trên logic:
* pending: Trạng thái mặc định khi giao dịch được khởi tạo
* success: Khi OTP xác nhận đúng và số dư/học phí cập nhật thành công.
* failed: Khi OTP sai, số dư không đủ, hoặc lỗi khác
* transactionId trong Output sẽ được back-end sinh tự động bằng UUID và kiểm soát idempotency

**POST /accounts/{userId}/lock:** Đặt khóa Redis trên tài khoản để xử lý xung đột số dư.

* Input: { userId: number}
* userId không rỗng và tồn tại trong bảng User
* Output:
* Thành công{ locked: true, lockKey: string, expiry: number }
* Lỗi: 400 Bad Request: { "error": "Invalid userId or transactionId", "code": 400 }
* Lỗi: 404 Not Found: { "error": "User or transaction not found", "code": 404 }
* Lỗi: 409 Conflict: { "error": "Account already locked", "code": 409 }

**POST /accounts/{userId}/unlock**: Mở khóa Redis cho tài khoản.

* Input: { userId: number, lockKey: string}
* userId không rỗng, tồn tại trong bảng User và đang bị khóa
* lockKey: Không rỗng, khớp với key từ lock API
* Output:
* Thành công: { unlocked: true }
* Lỗi: 400 Bad Request:{"error": "Invalid userId or lockKey ", "code": 400 }
* Lỗi: 404 Not Found: { "error": "User or lock not found", "code": 404 }

**PUT /accounts/{userId}/balance: Cập nhật số dư**

* Input: {amount: number, transactionId: String, userId: number}
* transactionId và userId, amount không rỗng
* transactionId và userId phải tồn tại trong các bảng
* amount nếu là số âm thì phải bé hơn balance hiện tại
* Output:
* Thành công:{ userId: number, newBalance: number }.
* Lỗi: 404 Not Found: { "error": "User not found", "code": 404 }
* Lỗi: 400 Bad Request: { "error": "Invalid amount or insufficient balance", "code": 400 }
* Ghi chú:
* API này không expose qua REST, chỉ gọi nội bộ từ PaymentService.
* Sử dụng @Transactional để đảm bảo số dư được cập nhật đồng thời trong một giao dịch.

### TuitionService

#### Chức năng

* Quản lý thông tin sinh viên và học phí (MSSV, họ tên, số tiền học phí, trạng thái).
* Xử lý khóa/mở khóa MSSV để tránh thanh toán trùng lặp (sử dụng Redis lock).

#### API chính

**GET /students/{mssv}/tuition**: Lấy thông tin sinh viên và học phí.

* Input: { mssv: string }
* msssv: Không rỗng, tối đa 10 ký tự
* Output
* Thành công:{ mssv: string, fullName, tuitionFee: number, status: string }.
* Lỗi: 400 Bad Request: { "error": "Invalid MSSV", "code": 400 }
* Lỗi: 404 Not Found: { "error": "Student not found", "code": 404 }

**POST /students/{mssv}/lock:** Đặt khóa Redis trên MSSV để xử lý xung đột thanh toán trùng.

* Input: { mssv: string}
* mssv và userId không rỗng và tồn tại trong bảng Student
* Output:
* Thành công{ locked: true, lockKey: string, expiry: number }
* Lỗi: 400 Bad Request: { "error": "Invalid mssv", "code": 400 }
* Lỗi: 404 Not Found: { "error": "mssv not found", "code": 404 }
* Lỗi: 409 Conflict: { "error": "Tuition already locked", "code": 409 }

**POST /students/{mssv}/unlock:** Mở khóa Redis cho MSSV.

* Input: { mssv: string, lockKey: string}
* mssv không rỗng, tồn tại trong bảng Student và đang bị khóa
* lockKey: Không rỗng, khớp với key từ lock API
* Output:
* Thành công: { unlocked: true }
* Lỗi: 400 Bad Request:{"error": "Invalid mssv or lockKey", "code": 400 }
* Lỗi: 400 Bad Request: { "error": "MSSV mismatch", "code": 400 }

**PUT /students/{mssv}/status:** Cập nhật trạng thái học phí.

* Input: {transactionId: string, amount: number }.
* transactionId không rỗng
* amount âm và phải khớp với tuitionFee
* Output
* Thành công: {mssv: string, status: paid}
* Lỗi: 400 Bad Request: { "error": "Invalid amount or tuition already paid", "code": 400 }
* Lỗi: 404 Not Found: { "error": "Student not found", "code": 404 }
* Ghi chú:
* API này không expose qua REST, chỉ gọi nội bộ từ PaymentService.
* Sử dụng @Transactional để đảm bảo tránh xung đột trạng thái học phí.

### OTPNotificationService

#### Chức năng

* Tạo, xác thực OTP.
* Gửi email OTP và xác nhận giao dịch.

#### API chính

**POST /otp/generate: Tạo OTP.**

* Input: { userId: number, transactionId: String}.
* userId và transactionId không rỗng và tồn tại
* Output

Thành công:{ otpId: string, expiresAt: string}.

Lỗi: 400 Bad Request: { "error": "Invalid userId or transactionId", "code": 400 }

Lỗi: 404 Not Found: { "error": "User or transaction not found", "code": 404 }

Lỗi: 429 Too Many Requests: { "error": "Too many OTP requests for this transaction", "code": 429 }

* Ghi chú
* otpId trong Output sẽ được back-end sinh tự động bằng UUID và kiểm soát idempotency.
* Lưu trong Redis với cặp key-value otpId:{userId}:{transactionId} - value: otpCode: string), expiry 60 giây (1 phút).
* OTP (6 chữ số) được tạo ngẫu nhiên và kiểm tra tính duy nhất (nếu cặp key-value đã tồn tại, generate lại).
* Rate limiting: Max 5 OTP/giao dịch, lưu đếm trong Redis với key rate:otp:{transactionId}.

**POST /otp/verify**: Xác thực và kiểm tra expiry OTP.

* Input: { otpId: string, otpCode: number}.
* otpId không rỗng
* otpCode chính xác 6 số
* Output
* Thành công: { valid: true }.
* Lỗi: 400 Bad Request: { "error": "Invalid otpId or otpCode", "code": 400 }
* Lỗi: 410 Gone: { "error": "OTP expired", "code": 410 }
* Lỗi: 404 Not Found: { "error": "OTP not found", "code": 404 }
* Lỗi: 401 Unauthorized: { "error": "Invalid OTP", "code": 401 }
* Ghi chú
* Kiểm tra OTP và expiry trong Redis.
* Xóa OTP khỏi Redis sau khi verify thành công để tránh tái sử dụng.

**POST /notifications/email:** Gửi email (OTP hoặc xác nhận).

* Input: {userId: number, type: string, otpId: string, transactionId: string, amount: number, mssv: string}.
* userId: không rỗng
* type: Phải là “OTP” hoặc “CONFIRMATION”
* otpCode: chính xác 6 số và bắt buộc nếu type là “OTP”
* transactionId, mssv, amount: Không rỗng và bắt buộc nếu type là “CONFIRMATION”
* Output

Thành công: { "sent": true, "message": "Email sent successfully" }

Lỗi: 400 Bad Request: { "error": "Invalid input or missing required fields for type", "code": 400 }

Lỗi: 404 Not Found: { "error": "User or email not found", "code": 404 }

Lỗi: 429 Too Many Requests: { "error": "Too many email requests for this transaction", "code": 429 }

Lỗi: 500 Internal Server Error: { "error": "Failed to send email", "code": 500 }

* Template email
* OTP

<h2>iBanking OTP</h2>

<p>Your OTP is <strong>{otpCode}</strong>. It is valid for 1 minutes.</p>

<p>Please do not share this OTP with anyone.</p>

<p>Thank you for using iBanking!</p>

* CONFIRMATION

<h2>Transaction Confirmation</h2>

<p>You have successfully paid <strong>{amount} VND</strong> for MSSV <strong>{mssv}</strong>.</p>

<p>Transaction ID: <strong>{transactionId}</strong></p>

<p>Thank you for using iBanking!</p>

### Payment Service

#### Chức năng

* Điều phối giao dịch thanh toán học phí.
* Xử lý xung đột bằng cách yêu cầu khóa từ AccountService và TuitionService.

#### API chính

**POST /payments:** Khởi tạo giao dịch.

* Input: { userId: number, mssv: string , amount: number }.
* userId, mssv không rỗng và tồn tại trong database
* amount phải khớp với tuititonFree
* Output
* Thành công: { transactionId: string, otpId: string, expiresAt: string }.
* Lỗi: 400 Bad Request: { "error": "Invalid userId, mssv, or amount", "code": 400 }
* Lỗi: 404 Not Found: { "error": "User or student not found", "code": 404 }
* Lỗi: 409 Conflict: { "error": "Tuition already paid or locked", "code": 409 }
* Ghi chú
* Tự động gọi OTPService để generate OTP và trả otpId.
* Gửi event async đến OTPService để gửi email OTP.
* Gọi AccountService để lưu giao dịch (POST /api/transactions) với status "pending".
* Gọi TuitionService để kiểm tra tuitionFee và khóa trạng thái học phí (**POST /students/{mssv}/lock**).
* Gọi AccountService để khóa tài khoản (**POST /accounts/{userId}/lock).**
* Nếu một API/Service bị lỗi, tiến hành rollback các thay đổi trong DB và mở khóa tài khoản và học phí. (Sử dụng @Transactional)

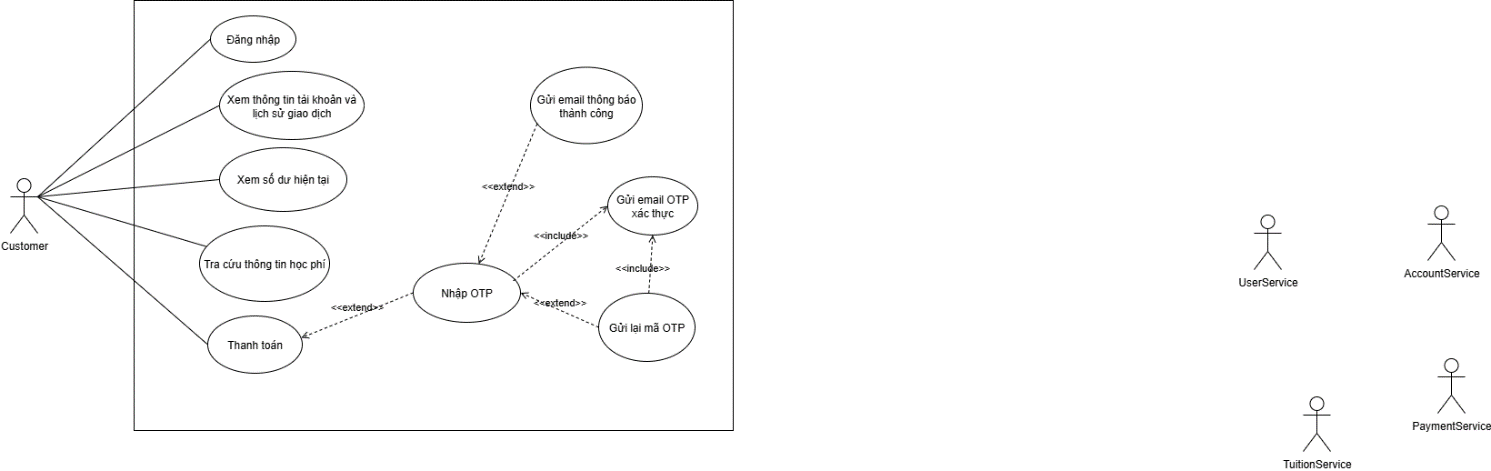
**POST /payments/confirm:** Xác nhận giao dịch sau OTP.

* Input: { transactionId: string, otpId: number, otpCode: string }.
* transactionId không rỗng và khớp với otpId
* otpId khớp với transactionId
* otpCode chính xác 6 chữ số
* Output
* Thành công: { status: "success", transactionId: string }.
* Lỗi: 400 Bad Request: { "error": "Invalid transactionId, otpId, or otpCode", "code": 400 }
* Lỗi: 401 Unauthorized: { "error": "Invalid OTP", "code": 401 }
* Lỗi: 410 Gone: { "error": "OTP expired", "code": 410 }
* Lỗi: 429 Too Many Requests: { "error": "Too many retry attempts", "code": 429 }
* Lỗi: 404 Not Found: { "error": "Transaction not found", "code": 404 }
* Ghi chú
* Tự động gọi OTPService để verify OTP (POST /otp/verify).
* Gọi AccountService để kiểm tra số dư (internal PUT /accounts/{userId}/balance).
* Nếu thành công, Gọi TuitionService để gạch nợ (internal PUT /students/{mssv}/tuition), và gửi email xác nhận (POST /api/v1/notifications/email).
* Nếu failed (Lỗi API/Service, vượt quá số lần nhập OTP), cập nhật status giao dịch thành "failed" và mở khóa (AccountService, TuitionService).

## Công nghệ sử dụng

* Cơ sở dữ liệu: MySQL
* Frontend: Android app, Jetpack Compose
* API: RestAPI
* Lock/Cache: Redis

# SƠ ĐỒ TỔNG QUAN

1. Mô hình use case

Hình 1: Sơ đồ Usecase tổng quát

1. Đặc tả use case
   1. Use case đăng nhập

|  |  |  |
| --- | --- | --- |
| **Tên Use Case** | Đăng nhập | |
| **Mã số Use Case** | UC\_01 | |
| **Tác nhân** | Khách hàng | |
| **Mô tả** | Người dùng điền username và password để đăng nhập hệ thống | |
| **Điều kiện tiên quyết** | Người dùng có tài khoản đăng nhập hợp lệ. | |
| **Sự kiện kích hoạt** | Người dùng chọn đăng nhập vào hệ thống | |
| **Luồng sự kiện** | Tác nhân | Hệ thống |
|  | 1. Người dùng truy cập vào ứng dụng.  2. Người dùng tiến hành nhập username và password và nhấn nút đăng nhập. | 1.1. Hệ thống hiển thị giao diện trang đăng nhập.  2.1. Hệ thống kiểm tra dữ liệu nhập  2.2. Hệ thống gọi đến API **POST/auth/login** của UserService để xác thực thông tin đăng nhập và trả về kết quả. |
| **Hậu điều kiện** | Người dùng đăng nhập thành công và truy cập giao diện chính hoặc nhận thông báo lỗi. | |
| **Ngoại lệ** | Sai username hoặc password → hệ thống hiển thị thông báo lỗi. Mất kết nối hoặc lỗi dịch vụ → thông báo “Không thể kết nối tới hệ thống”. | |

Bảng 1: Use case đăng nhập

* 1. Use case xem thông tin tài khoản và lịch sử giao dịch

|  |  |  |
| --- | --- | --- |
| **Tên Use Case** | Xem thông tin tài khoản và lịch sử giao dịch | |
| **Mã số Use Case** | UC\_02 | |
| **Tác nhân** | Khách hàng | |
| **Mô tả** | Khách hàng muốn xem thông tin tài khoản bao gồm họ tên, số điện thoại, email và lịch sử các giao dịch đã thực hiện. | |
| **Điều kiện tiên quyết** | Khách hàng đã đăng nhập thành công vào hệ thống. | |
| **Sự kiện kích hoạt** | Khách hàng bấm vào nút ‘Tài khoản của tôi” ở giao diện trang chủ | |
| **Luồng sự kiện** | Tác nhân | Hệ thống |
|  | 1. Khách hàng bấm vào nút “Tài khoản của tôi” ở giao diện trang chủ | 1.1 Hệ thống gọi đến API **GET /users/{userId}** của UserService để lấy thông tin hồ sơ người dùng.  1.2 Hệ thống gọi đến API **GET /accounts/{userId}/balance** củaAccountServiceđể lấy số dư.  1.3 Hệ thống gọi đến API **GET/accounts/{userId}/history** của AccountService để lấy lịch sử giao dịch.  1.4 Hệ thống hiển thị giao diện trang chứa thông tin tài khoản và lịch sử giao dịch. |
| **Hậu điều kiện** | Khách hàng xem được các thông tin liên quan và lịch sử giao dịch | |
| **Ngoại lệ** | Dịch vụ bị lỗi không cung cấp được dữ liệu. | |

Bảng 2: Use case xem thông tin tài khoản và lịch sử giao dịch

* 1. Use case xem số dư tài khoản

|  |  |  |
| --- | --- | --- |
| **Tên Use Case** | Xem số dư tài khoản | |
| **Mã số Use Case** | UC\_03 | |
| **Tác nhân** | Khách hàng | |
| **Mô tả** | Khách hàng muốn xem số dư của tài khoản hiện có. | |
| **Điều kiện tiên quyết** | Người dùng đã đăng nhập thành công vào hệ thống | |
| **Sự kiện kích hoạt** | Khách hàng bấm vào biểu tượng con mắt ở giao diện trang chủ để xem số dư tài khoản. | |
| **Luồng sự kiện** | Tác nhân | Hệ thống |
|  | 1. Khách hàng bấm vào biểu tượng con mắt ở giao diện trang chủ.  2. Người dùng bấm vào biểu tượng con mắt một lần nữa. | 1.1 Hệ thống gọi đến API **GET /accounts/{userId}/balance** của AccountService để lấy dữ liệu số dư.  1.2 Hệ thống hiển thị số dư lên giao diện.  2.1 Hệ thống ẩn thông tin số dư tài khoản. |
| **Hậu điều kiện** | Người dùng xem được thông tin số dư hiện tại của tài khoản | |
| **Ngoại lệ** | Dịch vụ bị lỗi không cung cấp được dữ liệu. | |

Bảng 3: Use case xem số dư tài khoản

* 1. Use case tra cứu thông tin học phí

|  |  |  |
| --- | --- | --- |
| **Tên Use Case** | Tra cứu thông tin học phí | |
| **Mã số Use Case** | UC\_04 | |
| **Tác nhân** | Khách hàng | |
| **Mô tả** | Người dùng muốn tra cứu thông tin học phí của sinh viên dựa trên mã số sinh viên | |
| **Điều kiện tiên quyết** | Người dùng đã đăng nhập thành công vào hệ thống | |
| **Sự kiện kích hoạt** | Người dùng nhập mã số sinh viên và nhấn Enter để tra cứu | |
| **Luồng sự kiện** | Tác nhân | Hệ thống |
|  | 1. Khách hàng nhập mã số sinh viên vào ô tra cứu và nhấn Enter | 1.1 Hệ thống hiển thị giao diện chứa ô tra cứu theo mã số sinh viên cho người dùng nhập  1.2 Hệ thống kiểm tra dữ liệu nhập từ người dùng  1.3 Hệ thống gọi API **GET /students/{mssv}/tuition** của TuitionService để lấy dữ liệu  1.4 Nếu tra cứu thành công, hệ thống hiển thị thông tin sinh viên kèm học phí và trạng thái học phí  1.5 Hệ thống bật nút “Tiếp tục” cho người dùng bấm nếu muốn tiến hành thanh toán.  1.6 Nếu thất bại, hệ thống hiển thị thông báo TOAST “Không tìm thấy thông tin sinh viên”. |
| **Hậu điều kiện** | Khách hàng tra cứu thành công thông tin sinh viên | |
| **Ngoại lệ** | 1. Không tìm thấy dữ liệu sinh viên tương ứng  2. Dịch vụ bị lỗi không cung cấp được dữ liệu | |

Bảng 4: Use case tra cứu thông tin sinh viên

* 1. Use case thanh toán

|  |  |  |
| --- | --- | --- |
| **Tên Use Case** | Thanh toán | |
| **Mã số Use Case** | UC\_05 | |
| **Tác nhân** | Khách hàng | |
| **Mô tả** | Khách hàng muốn tiến hành thanh toán sau khi đã tra cứu thông tin sinh viên thành công | |
| **Điều kiện tiên quyết** | 1. Khách hàng đã đăng nhập thành công vào hệ thống.  2. Khách hàng đã tra cứu thành công thông tin sinh viên tương ứng. | |
| **Sự kiện kích hoạt** | Tác nhân bấm nút “Tiếp tục” | |
| **Luồng sự kiện** | Tác nhân | Hệ thống |
|  | 1 Khách hàng bấm nút “Tiếp tục” | 1.1 Hệ thống gọi API **POST /payments** của PaymentServiceđểkhởi tạo giao dịch.  1.2 Nếu khởi tạo thành công, hệ thống hiển thị giao diện cho người dùng nhập OTP xác thực  1.3 Nếu khởi tạo thất bại, hệ thống hiển thị thông báo TOAST “Giao dịch thất bại” |
| **Hậu điều kiện** | Hệ thống khởi tạo thành công giao dịch và tiến hành gửi OTP xác thực về email người dùng | |
| **Ngoại lệ** | 1. Người dùng back về giao diện trang chủ trong quá trình thanh toán  2. Lỗi dịch vụ | |

Bảng 17: Use case thanh toán

* 1. Use case nhập OTP

|  |  |  |
| --- | --- | --- |
| **Tên Use Case** | Nhập OTP | |
| **Mã số Use Case** | UC\_06 | |
| **Tác nhân** | Khách hàng | |
| **Mô tả** | Khách hàng tiến hành nhập OTP xác thực được gửi về email để thanh toán học phí | |
| **Điều kiện tiên quyết** | 1. Người dùng đã đăng nhập thành công vào hệ thống  2. OTP đã được gửi về email người dùng  3. Giao diện nhập OTP đã được hiển thị | |
| **Sự kiện kích hoạt** | Người dùng nhập OTP và nhấn xác nhận | |
| **Luồng sự kiện** | Tác nhân | Hệ thống |
|  | 1. Người dùng nhập OTP và nhấn xác nhận | 1.1 Hệ thống gọi API **POST /otp/verify** của OTPNotificationService để xác thực và kiểm tra expiry OTP.  1.2 Nếu xác thực thành công, hệ thống gọi API **POST /notifications/email** của OTPNotificationService để gửi email xác nhận thanh toán thành công.  1.3 Nếu thất bại, hệ thống hiển thị thông báo TOAST “Sai mã OTP hoặc hết hạn vui lòng thử lại” |
| **Hậu điều kiện** | Khách hàng xác thực OTP thành công và hoàn thành giao dịch thanh toán học phí | |
| **Ngoại lệ** | 1. Người dùng back về giao diện trang chủ trong quá trình giao dịch  2. Lỗi dịch vụ  3. Người dùng nhập sai mã OTP hoặc OTP hết hạn | |

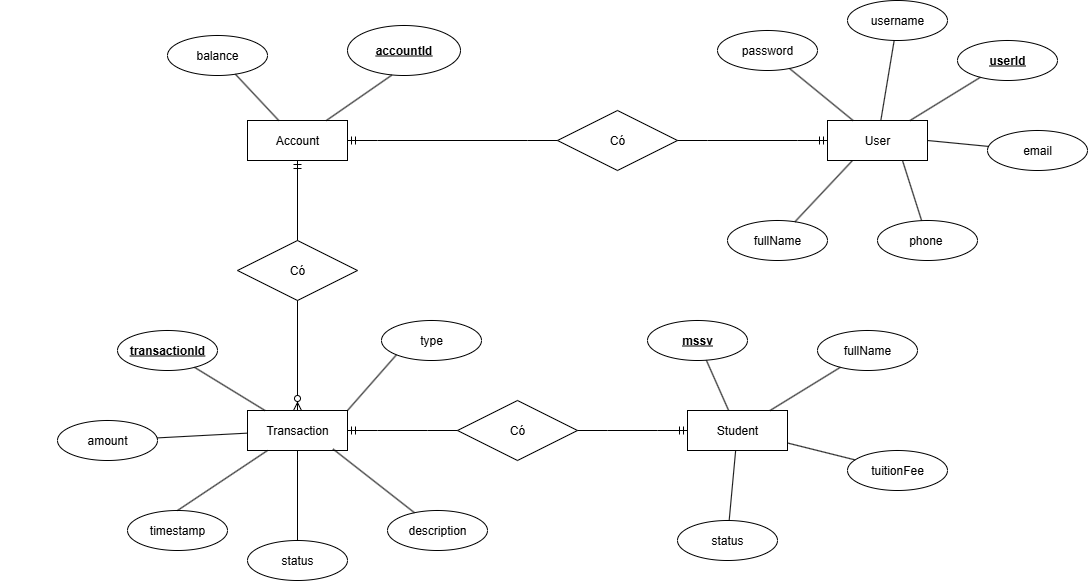
Bảng 6: Use case nhập OTP

* 1. Use case gửi lại mã OTP

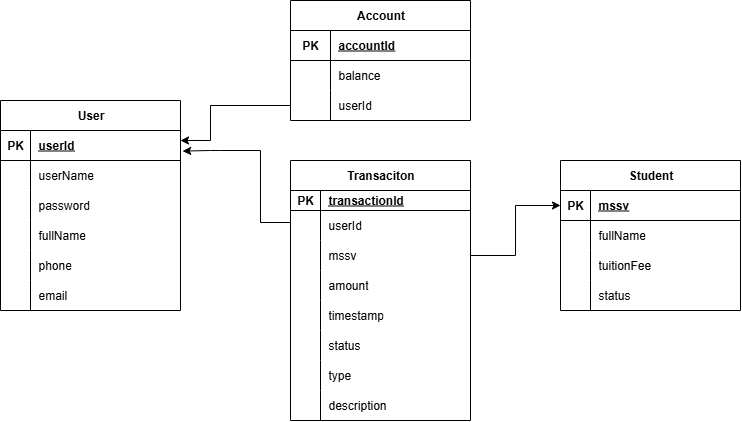
|  |  |  |
| --- | --- | --- |
| **Tên Use Case** | Gửi lại mã OTP | |
| **Mã số Use Case** | UC\_7 | |
| **Tác nhân** | Khách hàng | |
| **Mô tả** | Người dùng muốn hệ thống gửi lại mã OTP khi mã OTP hiện tại đã hết hạn | |
| **Điều kiện tiên quyết** | 1. Mã OTP hiện tại đã hết hạn  2. Người dùng đã đăng nhập thành công vào hệ thống  3. Số lần gửi lại mã OTP vẫn còn khả dụng (5 OTP – 1 giao dịch) | |
| **Sự kiện kích hoạt** | Người dùng nhấn vào nút gửi lại mã OTP | |
| **Luồng sự kiện** | Tác nhân | Hệ thống |
|  | 1. Tác nhân nhấn vào nút “Gửi lại mã” | 1.1 Hệ thống gọi API **POST /notifications/email:** của OTPNotificationService để gửi email OTP. |
| **Hậu điều kiện** | Người dùng nhận được mã OTP mới | |
| **Ngoại lệ** | 1. Người dùng nhấn nút “Gửi lại mã” khi mã OTP hiện tại chưa hết hạn  2. Dịch vụ lỗi | |

Bảng 7: Use case gửi lại mã OTP

# THIẾT KẾ DỮ LIỆU

1. Sơ đồ thực thể - mối quan hệ (ERD) và mô hình quan hệ

Hình 2: Sơ đồ ERD



Hình 3: Mô hình quan hệ

1. Đặc tả cơ sở dữ liệu
   1. Bảng User – UserService

1.1. Thuộc tính

* + **userID**: Khóa chính, trường số nguyên tự tăng, dùng để phân biệt từng người dùng trong hệ thống.
  + **username**: VARCHAR(50), UNIQUE và NOT NULL.
  + **password**: VARCHAR(255) NOT NULL
  + **fullName**: VARCHAR(100), NOT NULL
  + **email**: VARCHAR(100), UNIQUE và NOT NULL
  + **phone**: VARCHAR(20), NOT NULL

1.2 Đặc tả

Bảng User chứa thông tin về các người dùng có quyền truy cập hệ thống iBanking. Bảng này được sử dụng cho chức năng xác thực đăng nhập và quản lý hồ sơ người dùng.

* + **userID** đảm bảo mỗi người dùng có một bản ghi duy nhất trong bảng này
  + **username** và **email** là các trường unique để tránh trùng lặp tài khoản
  + **email** dùng để nhận thông báo và xác thực
  + **password** được mã hóa bằng hash hoặc thuật toán tương tự để bảo mật
  1. Account – AccountService

2.1 Thuộc tính

* **accountNumber** là khóa chính, định danh duy nhất cho mỗi tài khoản
* **userId**: Mã người dùng tham chiếu logic đến bảng Users thông qua API/event
* **balance**: DEMICAL(15.2), DEFAULT 0.00, NOT NULL và có ràng buộc CHECK (balance >= 0).

2.2 Đặt tả

Bảng Account chứa thông tin về số dư tài khoản của người dùng, bảng này được sử dụng cho chức năng quản lý số dư và xử lý giao dịch. Giữa bảng Account và User có mối quan hệ 1-1.

* + **accountNumber** là khóa chính và là số tài khoản ngân hàng của người dùng
  + **userId** là mã người dùng sở hữu tài khoản, dùng để liên kết tài khoản với chủ sở hữu.
  + **balance** là số dư hiện tại của tài khoản
  1. Bảng Transaction – AccountService

3.1 Thuộc tính

* + **transactionId**: khóa chính, VARCHAR(100), mã định danh duy nhất cho mỗi giao dịch
  + **userId**: Mã người dùng thực hiện giao dịch, tham chiếu đến bảng User thông qua API/event
  + **mssv**: Mã số sinh viên nhận thanh toán, tham chiếu đến bảng Student thông qua API/event
  + **amount**: DEMICAL(15,2) NOT NULL
  + **timestamp**: TIMESTAMP NOT NULL
  + **status**: Trạng thái giao dịch (PENDING, SUCCESS, FAILED)
  + **type:** Loại giao dịch (Thanh toán học phí, Nạp tiền,…)
  + **description:** VARCHAR(255)

3.2 Đặc tả

Bảng Transaction lưu trữ lịch sử giao dịch thanh toán học phí của người dùng

* + **transactionId** là mã giao dịch duy nhất, được backend sinh tự động bằng UUID. Dùng để idempotency.
  + **userId** giúp xác định người dùng nào đã thực hiện giao dịch
  + **mssv** liên kết giao dịch với sinh viên nhận thanh toán
  + **amount** là số tiền giao dịch
  + **timestamp** là thời điểm thực hiện giao dịch, mặc định là thời gian hiện tại (CURRENT\_TIMESTAMP).
  + **status** sẽ được backend set dựa trên logic thanh toán/OTP
  + **type** giúp người dùng phân loại giao dịch
  + **description** là mô tả thêm về giao dịch (tùy chọn)
  1. Bảng Student – TuitionService

4.1Thuộc tính

* + **mssv**: VARCHAR(20) là khóa chính
  + **fullName**: VARCHAR(100) NOT NULL
  + **tuitionFee**: DECIMAL(15,2) NOT NULL CHECK (tuitionFee > 0),
  + **status**: Trạng thái học phí (Chưa thanh toán, Đã thanh toán, Đang xử lý)

4.2 Đặc tả

Bảng Students chứa thông tin về các sinh viên và học phí cần thanh toán. Bảng này được sử dụng cho chức năng quản lý thông tin học phí và cập nhật trạng thái.

* **mssv** là khóa chính, đảm bảo mỗi sinh viên có một bản ghi duy nhất
* **tuitionFee** có constraint CHECK >= 0 để đảm bảo số tiền hợp lệ
* **status** có constraint CHECK để giới hạn các giá trị: 'Chưa thanh toán', 'Đã thanh toán', 'Đang xử lý'

# TÀI LIỆU KIỂM THỬ

1. User Service

**POST /auth/login**

|  |  |  |
| --- | --- | --- |
| Trường hợp | Input | Output |
| Thành công |  |  |

**GET /users/{userId}**

|  |  |  |
| --- | --- | --- |
| Trường hợp | Input | Output |
| Thành công | 1 |  |

2. Account Service

**GET /accounts/{userId}/balance**

|  |  |  |
| --- | --- | --- |
| Trường hợp | Input | Output |
| Thành công | 1 |  |
| Lỗi 404 Not Found | 2 |  |

**GET/accounts/{userId}/history**

|  |  |  |
| --- | --- | --- |
| Trường hợp | Input | Output |
| Thành công | 1 |  |

**POST /transactions**

|  |  |  |
| --- | --- | --- |
| Trường hợp | Input | Output |
| Thành công |  |  |
| Lỗi: 400 Bad Request |  |  |
| Lỗi 404 Not Found |  |  |

**POST /accounts/{userId}/lock**

|  |  |  |
| --- | --- | --- |
| Trường hợp | Input | Output |
| Thành công | 1 |  |
| Lỗi 404 Not Found | 2 |  |
| Lỗi 409 Conflict | 1 |  |

**POST /accounts/{userId}/unlock**

|  |  |  |
| --- | --- | --- |
| Trường hợp | Input | Output |
| Thành công |  |  |
| Lỗi: 400 Bad Request |  |  |
| Lỗi 404 Not Found |  |  |

3. **Tuition Service**

**GET /students/{mssv}/tuition**

|  |  |  |
| --- | --- | --- |
| Trường hợp | Input | Output |
| Thành công | 52300055 |  |

**POST /students/{mssv}/lock**

|  |  |  |
| --- | --- | --- |
| Trường hợp | Input | Output |
| Thành công | 52300055 |  |
| Lỗi 409 Conflict | 52300055 |  |

**POST /students/{mssv}/unlock**

|  |  |  |
| --- | --- | --- |
| Trường hợp | Input | Output |
| Thành công |  |  |
| Lỗi: 400 Bad Request |  |  |
| Lỗi: 400 Bad Request |  |  |

**PUT /students/{mssv}/status**

**3. OTPNotification Service**

**POST /otp/generate**

|  |  |  |
| --- | --- | --- |
| Trường hợp | Input | Output |
| Thành công |  |  |