iBanking Tuition Payment System - API Documentation

Table of Contents

Overview

Base URL

```
http://localhost:3000/api
```

Request Headers

```json
{
  "Content-Type": "application/json",
  "Authorization": "Bearer {token}"  // Required for authenticated endpoints
}
```

Response Format

All responses are in JSON format with the following structure for success:

```json
```

```
{
  "data": {...},
  "message": "Success message"
}
```

And for errors:

```json
{
  "error": "Error message",
  "errors": []  // Optional validation errors array
}
```

Authentication

The API uses JWT (JSON Web Token) for authentication. After successful login, include the token in the Authorization header for all protected endpoints.

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Token expiration: **1 hour**

API Endpoints

Authentication APIs

1. User Login

Endpoint: `POST /auth/login`
Description: Authenticates user and returns JWT token
Authentication Required: No
Rate Limit: 5 requests per minute per IP

Input:

```json
```

```
{
  "username": "string",  // Required, 3-50 characters
  "password": "string"   // Required, minimum 6 characters
}
```

Output (Success - 200):

```json
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "id": 1,
    "username": "johndoe",
    "fullName": "John Doe",
    "email": "john.doe@email.com",
    "phone": "0901234567",
    "balance": 50000000.00
  }
}
```

Output (Error - 401):

```json
{
  "error": "Invalid credentials"
}
```

Output (Validation Error - 400):

```json
```

```
{
  "errors": [
    {
      "field": "username",
      "message": "Username is required"
    },
    {
      "field": "password",
      "message": "Password is required"
    }
  ]
}
```

User Management APIs

2. Get User Profile

Endpoint: `GET /user/profile`
Description: Retrieves authenticated user's profile information
Authentication Required: Yes
Rate Limit: 30 requests per minute per user

Input: None (user identified by JWT token)

Output (Success - 200):

```json
{
  "fullName": "John Doe",
  "phone": "0901234567",
  "email": "john.doe@email.com",
  "balance": 50000000.00
}
```

Output (Error - 404):

```json
```

```
{
  "error": "User not found"
}
```

Output (Error - 401):

```json
{
  "error": "Access token required"
}
```

Student Information APIs

3. Get Student Information

Endpoint: `GET /student/{studentId}`
Description: Retrieves student information including tuition details
Authentication Required: Yes
Rate Limit: 20 requests per minute per user

Input:

- **URL Parameter:**
  - `studentId` (string): Student ID (e.g., "51900001")

Output (Success - 200):

```json
{
  "studentId": "51900001",
  "studentName": "Nguyen Van A",
  "tuitionAmount": 15000000.00,
  "isPaid": false,
  "academicYear": "2024-2025",
  "semester": 1,
  "dueDate": "2025-01-31"
}
```

Output (Error - 404):

```json
{
  "error": "Student not found"
}
```

Payment Processing APIs

4. Initiate Payment

Endpoint: `POST /payment/initiate`
Description: Creates a new payment transaction
Authentication Required: Yes
Rate Limit: 10 requests per minute per user

Input:

```json
{
  "studentId": "string"  // Required, valid student ID
}
```

Note: The tuition amount is automatically retrieved from the database based on the student ID.

Output (Success - 200):

```json
{
  "transactionId": 123,
  "transactionCode": "TXN1703123456ABCD1234",
  "studentId": "51900001",
  "studentName": "Nguyen Van A",
  "amount": 15000000.00,
  "status": "pending"
}
```

Output (Error - 404):

```json
{
  "error": "Student not found"
}
```

Output (Error - 400):

```json
{
  "error": "Tuition already paid"
}
```

Output (Error - 400):

```json
{
  "error": "Insufficient balance"
}
```

---

5. Send OTP

Endpoint: `POST /payment/send-otp`
Description: Sends OTP code to user's registered email
Authentication Required: Yes
Rate Limit: 3 requests per 5 minutes per transaction

Input:

```json
{
  "transactionId": 123  // Required, numeric, valid transaction ID
}
```

Output (Success - 200):

```json
{
  "message": "OTP sent successfully",
  "otpSent": true,
  "expiresIn": 300  // Seconds (5 minutes)
}
```

Output (Error - 404):

```json
{
  "error": "Transaction not found or invalid"
}
```

Output (Error - 500):

```json
{
  "error": "Failed to send OTP email"
}
```

---

6. Verify OTP

Endpoint: `POST /payment/verify-otp`
Description: Verifies the OTP code entered by user
Authentication Required: Yes
Rate Limit: 5 attempts per OTP

Input:

```json
{
  "transactionId": 123,     // Required, numeric
  "otpCode": "123456"       // Required, 6 digits
}
```

Output (Success - 200):

```json
{
  "verified": true,
  "transactionStatus": "otp_verified",
  "message": "OTP verified successfully"
}
```

Output (Error - 400):

```json
{
  "error": "Invalid or expired OTP"
}
```

Output (Error - 400):

```json
{
  "error": "Invalid OTP code"
}
```

Output (Error - 400):

```json
{
  "error": "Too many failed attempts"
}
```

---

7. Confirm Payment

Endpoint: `POST /payment/confirm`

Description: Finalizes the payment transaction

Authentication Required: Yes

Rate Limit: 1 request per transaction

Input:

```json
{
  "transactionId": 123  // Required, numeric, must be OTP verified
}
```

Output (Success - 200):

```json
{
  "success": true,
  "message": "Payment completed successfully",
  "newBalance": 35000000.00,
  "receipt": {
    "transactionCode": "TXN1703123456ABCD1234",
    "studentId": "51900001",
    "amount": 15000000.00,
    "completedAt": "2025-01-15T10:30:45.000Z"
  }
}
```

Output (Error - 400):

```json
{
  "error": "Invalid transaction or not verified"
}
```

Output (Error - 400):

```json
```

```
{
  "error": "Insufficient balance"
}
```

Output (Error - 409):

```json
{
  "error": "Another transaction is in progress. Please try again."
}
```

Transaction History APIs

8. Get Transaction History

Endpoint: `GET /transactions/history`
Description: Retrieves user's transaction history
Authentication Required: Yes
Rate Limit: 20 requests per minute per user

Input (Query Parameters):

- `limit` (optional): Number of records to return (default: 10, max: 100)
- `offset` (optional): Number of records to skip (default: 0)

Example: `GET /transactions/history?limit=20&offset=0`

Output (Success - 200):

```json
```

```json
{
  "transactions": [
    {
      "transactionCode": "TXN1703123456ABCD1234",
      "studentId": "51900001",
      "studentName": "Nguyen Van A",
      "amount": 15000000.00,
      "status": "completed",
      "createdAt": "2025-01-15T10:25:30.000Z",
      "completedAt": "2025-01-15T10:30:45.000Z"
    },
    {
      "transactionCode": "TXN1703123400EFGH5678",
      "studentId": "51900002",
      "studentName": "Tran Thi B",
      "amount": 12000000.00,
      "status": "completed",
      "createdAt": "2025-01-14T14:20:15.000Z",
      "completedAt": "2025-01-14T14:25:30.000Z"
    }
  ],
  "total": 15,
  "limit": 10,
  "offset": 0
}
```

Error Codes

HTTP Status Codes

Code	Description	Usage
200	OK	Successful request
400	Bad Request	Invalid input or business logic error
401	Unauthorized	Missing or invalid authentication token
403	Forbidden	Valid token but insufficient permissions
404	Not Found	Resource not found
409	Conflict	Resource conflict (e.g., concurrent transactions)
429	Too Many Requests	Rate limit exceeded
500	Internal Server Error	Server error

Business Error Codes

Error Message	Description	Resolution
"Invalid credentials"	Username or password incorrect	Verify login details
"Student not found"	Student ID doesn't exist	Check student ID
"Tuition already paid"	Student has already paid tuition	No action needed
"Insufficient balance"	User balance less than tuition amount	Top up account
"Invalid or expired OTP"	OTP expired or doesn't exist	Request new OTP
"Too many failed attempts"	OTP verification failed 3+ times	Start new transaction
"Another transaction is in progress"	Concurrent transaction detected	Wait and retry

---

Data Models

User Model

```
json
```

```
{
  "id": "integer",
  "username": "string",
  "fullName": "string",
  "phone": "string",
  "email": "string",
  "balance": "decimal",
  "isActive": "boolean",
  "createdAt": "timestamp",
  "updatedAt": "timestamp"
}
```

Student Model

```json
{
  "id": "integer",
  "studentId": "string",
  "fullName": "string",
  "tuitionAmount": "decimal",
  "isPaid": "boolean",
  "academicYear": "string",
  "semester": "integer",
  "dueDate": "date"
}
```

Transaction Model

```json
```

```json
{
  "id": "integer",
  "transactionCode": "string",
  "payerId": "integer",
  "studentId": "string",
  "amount": "decimal",
  "status": "enum(pending, otp_sent, otp_verified, completed, failed, cancelled)",
  "createdAt": "timestamp",
  "completedAt": "timestamp"
}
```

OTP Model

json

```json
{
  "id": "integer",
  "transactionId": "integer",
  "otpCode": "string",
  "email": "string",
  "isUsed": "boolean",
  "attempts": "integer",
  "expiresAt": "timestamp",
  "createdAt": "timestamp"
}
```

API Flow Diagram

```
1. Login (/auth/login)
   ↓
2. Get User Profile (/user/profile)
   ↓
3. Get Student Info (/student/{studentId})
   ↓
4. Initiate Payment (/payment/initiate)
   ↓
5. Send OTP (/payment/send-otp)
   ↓
6. Verify OTP (/payment/verify-otp)
   ↓
7. Confirm Payment (/payment/confirm)
   ↓
8. View Transaction History (/transactions/history)
```

Security Considerations

1. **Authentication**: All payment-related endpoints require JWT authentication
2. **Rate Limiting**: Implemented to prevent abuse and brute force attacks
3. **OTP Security**:
   - 6-digit random codes
   - 5-minute expiration
   - Maximum 3 verification attempts
   - Single-use codes
4. **Transaction Locking**: Prevents concurrent transactions on same account/student
5. **Input Validation**: All inputs are validated and sanitized
6. **HTTPS**: Should be enforced in production
7. **SQL Injection Prevention**: Using parameterized queries

Testing with Postman

Collection Variables

```json
{
  "baseUrl": "http://localhost:3000/api",
  "token": "{{authToken}}"
}
```

Test Sequence

1. **Login**: Save token to collection variable
2. **Get Profile**: Verify user details
3. **Check Student**: Verify tuition amount
4. **Initiate Payment**: Save transactionId
5. **Send OTP**: Check email
6. **Verify OTP**: Enter received code
7. **Confirm Payment**: Complete transaction
8. **Check History**: Verify transaction record

Notes

- All monetary values are in VND (Vietnamese Dong)
- Dates are in ISO 8601 format
- Student IDs follow TDTU format (8 digits)
- Transaction codes are unique and follow pattern: TXN{timestamp}{random}
- Email notifications are sent for OTP and payment confirmation
- System supports only full tuition payment (no partial payments)