Fintech Microservice Application API Documentation

Comprehensive Guide to Endpoints, Authentication, and Usage

Generated Documentation

Fintech Microservice Application API Documentation

Table of Contents

Overview

This documentation provides comprehensive information about the Fintech Microservice Application API, including all available endpoints, request/response formats, authentication mechanisms, and usage examples.

The application consists of five microservices:

- **Auth Service**: User registration, authentication, and JWT token management
- **Accounts Service**: Account management and balance operations
- **Transfer Service**: Fund transfers with idempotency and external integrations
- **Ledger Service**: Double-entry bookkeeping and transaction recording
- **Consumer Service**: Event consumption for analytics and audit logging

Authentication Flow

The authentication flow in this application uses JWT (JSON Web Tokens) for secure API access:

1. **User Registration**

- Client sends registration request to `/auth/register`
- Server validates input and creates user account
- Password is securely hashed using bcryptjs
- Server responds with user details (without password)

2. **User Login**

- Client sends credentials to `/auth/login`
- Server validates credentials against stored hash
- If valid, server generates:
- Access Token (1 hour expiration by default)
- Refresh Token (24 hours expiration by default)
- Server responds with tokens and user information

3. **API Access**

- Client includes Access Token in Authorization header
- Format: `Authorization: Bearer <access_token>`
- Server validates token before processing request

4. **Token Refresh**

- When Access Token expires, client uses Refresh Token
- Client sends Refresh Token to `/auth/refresh`
- Server validates and issues new Access Token
- Original Refresh Token remains valid

JWT Token Management

Token Structure

Tokens are JWT (JSON Web Token) formatted with the following claims:

- `userId`: Unique identifier of the authenticated user
- `email`: Email address of the authenticated user
- `roles`: Array of roles assigned to the user (e.g., ['user'], ['admin'])
- `exp`: Expiration timestamp (automatically added by JWT library)

Environment Configuration

Token behavior can be customized through environment variables:

- `JWT_SECRET`: Secret key for signing access tokens
- `JWT_EXPIRES_IN`: Access token expiration time in seconds (default: 3600)
- `REFRESH_TOKEN_SECRET`: Secret key for signing refresh tokens
- `REFRESH_TOKEN_EXPIRES_IN`: Refresh token expiration time in seconds (default: 86400)

Security Considerations

- Tokens are signed using HS256 algorithm
- Access tokens have short lifespan (1 hour) to minimize security risk
- Refresh tokens have longer lifespan (24 hours) but are only used for token renewal
- Tokens should be transmitted over HTTPS only
- Store tokens securely on the client side (HttpOnly cookies recommended)

Service Endpoints

Auth Service (Port 3001)

Register a New User

Endpoint: `POST /auth/register`

Description: Creates a new user account

Request Body:

{

"email": "string",

"password": "string",

"roles": ["string"] // Optional, defaults to ["user"]

}

Example Request:

curl -X POST http://localhost:3001/auth/register \
-H "Content-Type: application/json" \
-d '{
"email": "user@example.com",
"password": "securePassword123",
"roles": ["user"]
}'
Success Response (201):
{
"user": {
"id": 1,
"email": "user@example.com",
"roles": ["user"]
}
}
Error Responses:
- 400: Missing email or password
- 409: User already exists
- 500: Internal server error

User Login

Endpoint: `POST /auth/login`
Description: Authenticates user and returns JWT tokens
Request Body:
{
"email": "string",
"password": "string"
}
Example Request:
curl -X POST http://localhost:3001/auth/login \
-H "Content-Type: application/json" \
-d '{
"email": "user@example.com",
"password": "securePassword123"
}'
Success Response (200):
{
"accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
"refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
"user": {
"id": 1,
"email": "user@example.com",
"roles": ["user"]
}
}

Error Responses:

- 400: Missing email or password
- 401: Invalid credentials
- 500: Internal server error

Refresh Access Token

Endpoint: `POST /auth/refresh`

Description: Renews access token using refresh token

Request Body:

```
{
"refreshToken": "string"
}
```

Example Request:

```
curl -X POST http://localhost:3001/auth/refresh \
-H "Content-Type: application/json" \
-d '{
"refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}'
```

Success Response (200):

```
{
"accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Error Responses:

- 400: Missing refresh token
- 401: Invalid refresh token
- 500: Internal server error

Health Check

Endpoint: `GET /health`

Description: Checks service health status

Example Request:

```
curl -X GET http://localhost:3001/health
```

Success Response (200):

```
{
"status": "UP",
"timestamp": "2023-01-01T00:00:00.000Z",
"services": {
"database": "OK"
}
}
```

Error Response (500):

```
{
"status": "DOWN",
"timestamp": "2023-01-01T00:00:00.000Z",
"services": {
"database": "ERROR"
```

},
"error": "Connection failed"
}

Accounts Service (Port 3002)

Get User Accounts

Endpoint: `GET /accounts`
Description: Retrieves all accounts belonging to the authenticated user
Authentication: Required (Bearer Token)
Example Request:
curl -X GET http://localhost:3002/accounts \
-H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
Success Response (200):
{
"accounts": [
{
"id": 1,
"account_number": "ACC001",
"account_type": "checking",
"balance": "1000.00",
"currency": "USD",
"status": "active",
"created_at": "2023-01-01T00:00:00.000Z"
}
]
}
Error Responses:
- 500: Internal server error

Get Account Balance

Endpoint: `GET /accounts/{id}/balance`
Description: Retrieves the current balance of a specific account
Authentication: Required (Bearer Token)
Path Parameters:
- `id`: Account ID (integer)

Example Request:
curl -X GET http://localhost:3002/accounts/1/balance \
-H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
Success Response (200):
{
"balance": "1000.00",
"currency": "USD"
}
Error Responses:
- 400: Invalid account ID

- 404: Account not found
- 500: Internal server error

Deposit Funds

Endpoint: `POST /accounts/{id}/deposit`
Description: Deposits funds into a specific account
Authentication: Required (Bearer Token)
Path Parameters:
- `id`: Account ID (integer)

Request Body:

```
{
"amount": "number",
"description": "string" // Optional
}
```

Example Request:

```
curl -X POST http://localhost:3002/accounts/1/deposit \
-H "Content-Type: application/json" \
-H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..." \
-d '{
"amount": 500.00,
"description": "Salary deposit"
}'
```

Success Response (201):

```
{
"message": "Deposit successful",
"transactionId": 1,
"newBalance": "1500.00",
"createdAt": "2023-01-01T00:00:00.000Z"
}
```

Error Responses:
- 400: Invalid account ID or amount
- 404: Account not found
- 500: Internal server error

Withdraw Funds

Endpoint: `POST /accounts/{id}/withdraw`
Description: Withdraws funds from a specific account
Authentication: Required (Bearer Token)
Path Parameters:
- `id`: Account ID (integer)

Request Body:

```
{
"amount": "number",
"description": "string" // Optional
}
```

Example Request:

curl -X POST http://localhost:3002/accounts/1/withdraw \
-H "Content-Type: application/json" \
-H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..." \
-d '{
"amount": 200.00,
"description": "ATM withdrawal"
}'
Success Response (201):
{
"message": "Withdrawal successful",
"transactionId": 2,
"newBalance": "1300.00",
"createdAt": "2023-01-01T00:00:00.000Z"
}
Error Responses:
- 400: Invalid account ID, amount, or insufficient funds
- 404: Account not found
- 500: Internal server error

Health Check
Endpoint: `GET /health`
Description: Checks service health status
Example Request:
curl -X GET http://localhost:3002/health
Success Response (200):
{
"status": "UP",
"timestamp": "2023-01-01T00:00:00.000Z",
"services": {
"database": "OK"
}
}
Error Response (500):
{
"status": "DOWN",
"timestamp": "2023-01-01T00:00:00.000Z",
"services": {
"database": "ERROR"
},
"error": "Connection failed"
}

Transfer Service (Port 3003)

Initiate Fund Transfer
Endpoint: `POST /transfer`

Description: Transfers funds between two accounts with idempotency support
Authentication: Required (Bearer Token)
Headers:
- `Idempotency-Key`: Unique identifier for the transfer request (required)

Request Body:
```
{
"fromAccountId": "integer",
"toAccountId": "integer",
"amount": "number"
}
```
Example Request:
```
curl -X POST http://localhost:3003/transfer \
-H "Content-Type: application/json" \
-H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..." \
-H "Idempotency-Key: unique-transfer-id-123" \
-d '{
"fromAccountId": 1,
"toAccountId": 2,
"amount": 100.00
}'
```
Success Response (201):
```
{
"message": "Transfer completed successfully",
"transferId": 1,
"fromTransactionId": 1,
"toTransactionId": 2
}
```
Already Processed Response (200):
```
{
"message": "Transfer already processed",
"transferId": 1,
"status": "completed"
}
```
Error Responses:
- 400: Invalid request data, insufficient funds, or fraud detected
- 404: Account not found
- 429: Rate limit exceeded
- 500: Internal server error

Get Transfer Status

Endpoint: `GET /transfer/{id}/status`
Description: Retrieves the status of a specific transfer
Authentication: Required (Bearer Token)
Path Parameters:
- `id`: Transfer ID (integer)

Example Request:

curl -X GET http://localhost:3003/transfer/1/status \
-H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
Success Response (200):
{
"status": "completed",
"createdAt": "2023-01-01T00:00:00.000Z",
"updatedAt": "2023-01-01T00:00:00.000Z"
}
Error Responses:
- 400: Invalid transfer ID
- 404: Transfer not found
- 500: Internal server error

Health Check
Endpoint: `GET /health`
Description: Checks service health status
Example Request:
curl -X GET http://localhost:3003/health
Success Response (200):
{
"status": "UP",
"timestamp": "2023-01-01T00:00:00.000Z",
"services": {
"database": "OK",
"redis": "OK",
"kafka": "OK"
}
}
Error Response (500):
{
"status": "DOWN",
"timestamp": "2023-01-01T00:00:00.000Z",
"services": {
"database": "ERROR",
"redis": "OK",
"kafka": "OK"
},
"error": "Database connection failed"
}

Ledger Service (Port 3004)

Get Ledger Entries
Endpoint: `GET /ledger/accounts/{accountId}`
Description: Retrieves ledger entries for a specific account with pagination
Authentication: Required (Bearer Token)

Path Parameters:
- `accountId`: Account ID (integer)

Query Parameters:
- `page`: Page number (default: 1)
- `size`: Number of entries per page (default: 10)

Example Request:

```
curl -X GET "http://localhost:3004/ledger/accounts/1?page=1&size=10" \
-H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
```

Success Response (200):


```
{
"entries": [
{
"id": 1,
"transaction_id": 1,
"account_id": 1,
"entry_type": "debit",
"amount": "100.00",
"balance_after": "900.00",
"description": "Transfer to account 2",
"created_at": "2023-01-01T00:00:00.000Z",
"transaction_type": "transfer",
"transaction_description": "Transfer to account 2"
}
],
"pagination": {
"page": 1,
"size": 10,
"totalCount": 5,
"totalPages": 1
}
}
```


Error Responses:
- 400: Invalid account ID
- 500: Internal server error

Get Account Transactions

Endpoint: `GET /ledger/accounts/{accountId}/transactions`

Description: Retrieves transactions for a specific account with pagination

Authentication: Required (Bearer Token)

Path Parameters:
- `accountId`: Account ID (integer)

Query Parameters:
- `page`: Page number (default: 1)
- `size`: Number of transactions per page (default: 10)

Example Request:

```
curl -X GET "http://localhost:3004/ledger/accounts/1/transactions?page=1&size=10" \
```

-H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
Success Response (200):
{
"transactions": [
{
"id": 1,
"account_id": 1,
"transaction_type": "transfer",
"amount": "100.00",
"description": "Transfer to account 2",
"reference_id": "unique-transfer-id-123",
"status": "completed",
"created_at": "2023-01-01T00:00:00.000Z"
}
],
"pagination": {
"page": 1,
"size": 10,
"totalCount": 3,
"totalPages": 1
}
}
Error Responses:
- 400: Invalid account ID
- 500: Internal server error

Health Check

Endpoint: `GET /health`
Description: Checks service health status
Example Request:
curl -X GET http://localhost:3004/health
Success Response (200):
{
"status": "UP",
"timestamp": "2023-01-01T00:00:00.000Z",
"services": {
"database": "OK"
}
}
Error Response (500):
{
"status": "DOWN",
"timestamp": "2023-01-01T00:00:00.000Z",
"services": {
"database": "ERROR"

},
"error": "Connection failed"
}

Data Models

User

{
"id": "integer",
"email": "string",
"roles": ["string"]
}

Account

{
"id": "integer",
"account_number": "string",
"account_type": "string",
"balance": "string",
"currency": "string",
"status": "string",
"created_at": "string (ISO 8601 datetime)"
}

Transaction

{
"id": "integer",
"account_id": "integer",
"transaction_type": "string",
"amount": "string",
"description": "string",
"reference_id": "string",
"status": "string",
"created_at": "string (ISO 8601 datetime)"
}

Ledger Entry

{
"id": "integer",
"transaction_id": "integer",
"account_id": "integer",
"entry_type": "string", // "debit" or "credit"
"amount": "string",
"balance_after": "string",
"description": "string",

"created_at": "string (ISO 8601 datetime)"
}

Transfer

{
"id": "integer",
"from_account_id": "integer",
"to_account_id": "integer",
"amount": "string",
"reference_id": "string",
"status": "string",
"otp_verified": "boolean",
"payment_processed": "boolean",
"created_at": "string (ISO 8601 datetime)",
"updated_at": "string (ISO 8601 datetime)"
}

Error Responses

All services follow a consistent error response format:

{
"error": "string"
}

Common HTTP status codes:

- **400 Bad Request**: Invalid input data or malformed requests
- **401 Unauthorized**: Missing or invalid authentication credentials
- **404 Not Found**: Requested resource does not exist
- **409 Conflict**: Resource already exists (registration conflicts)
- **429 Too Many Requests**: Rate limit exceeded
- **500 Internal Server Error**: Unexpected server error

Rate Limiting

The Transfer Service implements rate limiting to prevent abuse:

- Maximum 100 requests per 15 minutes per account
- Exceeding the limit returns a 429 Too Many Requests response
- Rate limiting is tracked per account ID making transfers

Configuration can be adjusted through environment variables:

- `RATE_LIMIT_WINDOW_MS`: Time window in milliseconds (default: 900000 = 15 minutes)
- `RATE_LIMIT_MAX_REQUESTS`: Maximum requests per window (default: 100)

Idempotency

The Transfer Service supports idempotent operations to safely retry requests:

- Clients must provide a unique `Idempotency-Key` header with each transfer request
- If a transfer with the same key is submitted again, the service returns the status of the original transfer
- This prevents duplicate transfers when clients retry failed requests

Best practices for idempotency keys:

- Use UUIDs or other globally unique identifiers
- Reuse the same key when retrying the same operation
- Generate new keys for distinct operations