Digital Wallet API Documentation

metin, ekran görüntüsü, yazılım, web sayfası içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

Figure 1: Swagger UI

**Overview**

This document describes the implementation of a backend Wallet API for a digital payment company using Java and Spring Boot. The service supports operations for both Customers and Employees, ensuring secure access through JWT-based authentication.

**Business Rules**

* **Deposits**:
  + >1000 → PENDING, only affects usable balance
  + ≤1000 → APPROVED, and affects balance + usable balance
* **Withdraws**:
  + Checks activeForShopping and activeForWithdraw settings (If the destination type is PAYMENT then the operation is payment. If the destination type is IBAN, the operation type is withdrawal.). If shopping is not active, payment is not allowed. If withdrawal is not active, withdrawal is not allowed.
  + >1000 → PENDING, only affects usable balance
  + ≤1000 → APPROVED, affects both balances
* **Approval Flow**:

Updates wallet balance accordingly based on transaction type and new status. If the operation is denied then the usable balance is returned back. If the operation is approved, the balance is updated accordingly.

**Features**

* JWT-based authentication & authorization
* Role-based access control: CUSTOMER, EMPLOYEE
* Create and list wallets
* Deposit and withdraw money
* Approve or deny transactions (employee only)
* Enforced business logic for wallet settings and transaction limits
* In-memory H2 database
* RESTful API with Swagger (optional)
* Testing API via Postman

**Technology Stack**

* Java 21
* Spring Boot 3.2.5
* Spring Security with JWT (JSON Web Token)
* Spring Data JPA
* H2 In-Memory Database
* Lombok
* Maven
* JUnit 5 (Integration + Unit Tests)
* Swagger / OpenAPI for documentation

**Project Structure**

com.ing.digital.wallet

├── controller

├── dto

├── model

├── repository

├── security

├── service

├── util

└── DigitalWalletApplication.java

**Entities**

* **Customer**: name, surname, TCKN, username, password, role
* **Wallet**: customer\_id, wallet\_name, currency, activeForShopping, activeForWithdraw, balance, usable\_balance
* **Transaction**: wallet\_id, amount, type (DEPOSIT/WITHDRAW), opposite\_party\_type (IBAN/PAYMENT), opposite\_party, status (PENDING/APPROVED/DENIED)

**Access Console & Swagger**

* **H2 Console**: <http://localhost:8080/h2-console>
  + JDBC URL: jdbc:h2:mem:walletdb
* **Swagger UI (if enabled)**: <http://localhost:8080/swagger-ui/index.html>

**Authentication & Authorization**

| **Role** | **Permissions** |
| --- | --- |
| CUSTOMER | Can manage their own wallets and transactions |
| EMPLOYEE | Can manage **all customers’** wallets & approve transactions |

**JWT Authentication**

* Login endpoint returns a JWT token.
* Token must be included in the Authorization header for protected endpoints:

Authorization: Bearer <token>

**API Endpoints**

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

Figure 2: Swagger UI Rest Services

metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

Figure 3: Service Definitions in Postman

1. **Authentication**

POST /api/auth/login — Login and receive JWT token

**Request Body**

{

"username": "john",

"password": "password"

}

POST /api/auth/employee/create — The endpoint to create employee (Authorization not required – It can be secured in another way through a secure channel)

POST /api/customers/create — The endpoint to create customer

1. **Wallet**

POST /api/wallets — Create wallet (EMPLOYEE or CUSTOMER)

GET /api/wallets/{customerId} — List wallets of a customer (EMPLOYEE or CUSTOMER)

| **Endpoint** | **Method** | **Auth Required** | **Role** |
| --- | --- | --- | --- |
| /api/wallets/create | POST | Yes | Both |
| /api/wallets/{customerId} | GET | Yes | Both |
| /api/wallets/{customerId}/currency/{currency} | GET | Yes | Both |

1. **Transactions: Deposit / Withdraw**

POST /api/transactions/deposit — Deposit into wallet

POST /api/transactions/withdraw — Withdraw from wallet

GET /api/transactions/{walletId} — List transactions of a wallet

PATCH /api/transactions/approve — Approve or deny a transaction (EMPLOYEE only)

| **Endpoint** | **Method** | **Auth Required** | **Role** |
| --- | --- | --- | --- |
| /api/transactions/deposit | POST | Yes | Both |
| /api/transactions/withdraw | POST | Yes | Both |
| /api/wallets/{walletId}/transactions | GET | Yes | Both |

1. **Approve Transaction**

| **Endpoint** | **Method** | **Auth Required** | **Role** |
| --- | --- | --- | --- |
| /api/transactions/approve | POST | Yes | EMPLOYEE |

1. **Database Structure**

**Customer**

| **Field** | **Type** |
| --- | --- |
| id | Long |
| name | String |
| surname | String |
| tckn | String |
| role | Enum (CUSTOMER, EMPLOYEE) |

**Wallet**

| **Field** | **Type** |
| --- | --- |
| id | Long |
| customer\_id | Long |
| wallet\_name | String |
| currency | Enum (TRY, USD, EUR) |
| activeForShopping | Boolean |
| activeForWithdraw | Boolean |
| balance | BigDecimal |
| usableBalance | BigDecimal |

**Transaction**

| **Field** | **Type** |
| --- | --- |
| id | Long |
| wallet\_id | Long |
| amount | BigDecimal |
| type | Enum (DEPOSIT, WITHDRAW) |
| oppositePartyType | Enum (IBAN, PAYMENT) |
| oppositeParty | String |
| status | Enum (PENDING, APPROVED, DENIED) |

1. **Business Rules**

* **Create Wallet**: Currency must be one of TRY, USD, EUR.
* **Deposit**:
  + Amount ≤ 1000: APPROVED, reflected in balance + usable balance.
  + Amount > 1000: PENDING, reflected only in balance.
* **Withdraw**:
  + Wallet must be active for shopping or withdraw.
  + Amount ≤ 1000: APPROVED, reflected in balance and usable balance.
  + Amount > 1000: PENDING, deducted only from usable balance.
* **Approve Transaction**:
  + Changes status and updates wallet balances accordingly.

1. **Testing**

* Integration tests using @SpringBootTest
* Sample test scenario:
  1. Create employee (unauthenticated)
  2. Login → Get token
  3. Create customer (with token)
  4. Create wallet and deposit/withdrawal/approval
  5. List wallets and assert balances

The implementation has been realized simply without detailed logging. If there is an unexpected situation, the eror is thrown and handled as generating json content with ErrorResponseDto class and returned in response body:

<public class ErrorResponseDto {  
 private LocalDateTime timestamp;  
 private int status;  
 private String error;  
 private List<String> messages;  
 private String path;  
}

**Integration Testing Strategy – WalletApiIntegrationTest**

This integration test class verifies the main use cases of the digital wallet system using an in-memory H2 database. The testing approach simulates a real-world flow where an EMPLOYEE interacts with the system and performs operations on behalf of a CUSTOMER. (The other cases can be tested via API)

**Test Flow Overview**

1. **Employee Creation (Without Authentication)**  
   The test begins by creating an employee using the endpoint that does not require authentication. This simulates an initial registration or bootstrap phase.
2. **Authentication & JWT Token Generation**  
   The created employee logs into the system using their credentials (username & password).  
   A JWT token is returned, which includes:
   * Username, password
   * role (either CUSTOMER or EMPLOYEE)
   * 1-hour expiration
3. **Customer Creation (Using Employee Token)**

Using the obtained JWT token (with EMPLOYEE role), the employee creates a customer account.  
Authorization is handled by passing the token as a Bearer token in the HTTP headers.

1. **Wallet Creation for Customer**

The employee (with proper permissions) creates a new wallet for the previously registered customer.

1. **Wallet Transactions (Deposit / Withdraw)**

Finally, the employee performs transaction operations such as depositing or withdrawing funds from the customer’s wallet.

**Notes**

* All endpoints are tested end-to-end over HTTP using TestRestTemplate.
* JWT token is passed in the Authorization header using the Bearer scheme.
* Since an in-memory H2 database is used, each test runs in a clean and isolated environment.
* The test ensures:
  + Role-based access control
  + Valid token generation and usage
  + Full coverage of entity creation and relationship usage (Employee → Customer → Wallet)
* No external DB required, uses H2 memory DB
* Can be adapted to use PostgreSQL or MySQL by updating application.properties
* JWT token has 1-hour expiration

**Example Users (on Startup)**

| **Username** | **Password** | **Role** |
| --- | --- | --- |
| john | 12345 | CUSTOMER |
| admin | admin | EMPLOYEE |

*(Can be customized in data.sql or app startup logic.)*

1. **Future Improvements**

* Add pagination to wallet/transaction listings
* Add auditing (createdAt, updatedAt)
* Support multi-currency conversion
* Dockerize for deployment

1. **Contact**

**Email**: huseyin.ersoy@gmail.com  
**GitHub**: <https://github.com/HuseyinErsoyOrdu/ing-wallet2>

1. **Some H2 Database Test Data:**

metin, ekran görüntüsü, yazılım, bilgisayar simgesi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

Figure 4: Customer/Employee Data in H2 database

metin, ekran görüntüsü, yazılım, bilgisayar simgesi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

Figure 5: Wallet Data in H2 database

metin, ekran görüntüsü, ekran, görüntüleme, yazılım içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

Figure 6: Transaction Data in H2 database

**TEST Deployment**

Since there is no admin access in my PC, I could not use the deployment method docker related. The Cloud try version is time limited…

I have used the way of compiling runnable jar file to run the program:

The bat file: **runDıgıtalWallti.bat**

REM ./mvnw clean package (creates the jar file on “Target” directory in project root)

REM **use java 21**

**java -jar digital-wallet-1.0.jar --server.port=8080**

After running the program you can use the links (use server port):

<http://localhost:8080/swagger-ui/index.html>

<http://localhost:8080/h2-console>

metin, ekran görüntüsü, sayı, numara, ekran, görüntüleme içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

Figure 7: H2 database connection credentials (Test purpose)