System Requirements Specification Index

For

Digital Wallet Management System

Version 1.0

IIHT Pvt. Ltd.
IIHT Ltd, No: 15, 2nd Floor, Sri Lakshmi Complex, Off MG Road, Near SBI LHO,
Bangalore, Karnataka – 560001, India
fullstack@iiht.com

Digital Wallet Management System

System Requirements Specification

---

1. BUSINESS-REQUIREMENT:

1.1 PROBLEM STATEMENT:

Digital Wallet Management System is .Net Core web API 3.1 application integrated with MS SQL Server, where it refers to development of a Digital Wallet and Expense Management System that enables users to manage their finances and make payments seamlessly. With the rise of digital transactions, users need a secure and convenient way to manage their payments and expenses. This system will offer users a digital wallet to store funds, track spending, and facilitate easy payments, all in one place.

1.2 FOLLOWING IS THE REQUIREMENT SPECIFICATION:

	Digital Wallet Management System
Modules	
1	Wallet
2	Payment
3	Transaction
4	Security
Functionalities	
1	Get All Wallet Details
2	Create Wallet
3	Retrieve Wallet by id
4	Update Wallet
5	Delete Wallet
6	Scheduled Payment
7	Instant Payment
8	Create Transaction
9	Retrieve Transaction History
10	Get Transaction Detail
11	Two-Factor Authentication Setup
12	Security Audit Log

2. Assumptions, Dependencies, Risks / Constraints

2.1 Digital Wallet Constraints:

- If any Id does not exist then the operation should throw a custom exception.
- While fetching the any details by id, if id does not exist then the operation should throw a custom exception.

2.4 Common Constraints

- For all rest endpoints receiving @RequestBody, validation check must be done and must throw custom exception if data is invalid
- All the business validations must be implemented in model classes only.
- All the database operations must be implemented on entity object only
- Do not change, add, remove any existing methods in service layer
- In Repository interfaces, custom methods can be added as per requirements.
- All RestEndpoint methods and Exception Handlers must return data wrapped in **ResponseEntity**

3. Business Validations

3.1 Wallet Class Entities

- **WalletId** (int, primary key): Unique identifier for the wallet.
- **UserId** (int): Unique identifier for the user to whom the wallet belongs.
- **Balance** (decimal): The current balance of the wallet.
- **CreatedAt** (DateTime): The date and time when the wallet was created.
- **Name** (string): The name of the wallet (e.g., "Personal Wallet").

3.2 Payment Class Entities

- **ScheduleId** (int, primary key): Unique identifier for the scheduled payment.
- **WalletId** (int): Unique identifier of the wallet associated with the payment.
- **Amount** (decimal): The amount to be paid in the scheduled transaction.
- **RecipientId** (int): Unique identifier for the recipient of the payment.
- **Frequency** (string): The frequency of the payment (e.g., daily, weekly, monthly).
- **StartDate** (DateTime): The date the scheduled payment starts.
- **EndDate** (DateTime?, nullable): The optional end date for the scheduled payment, if applicable.

3.3 Transaction Class Entities

- **TransactionId** (int, primary key): Unique identifier for the transaction.
- **WalletId** (int): The wallet associated with the transaction.
- **Amount** (decimal): The amount of money involved in the transaction.
- **Type** (string): The type of transaction (e.g., payment, deposit).
- **Date** (DateTime): The date and time the transaction was initiated.
- **Status** (string): The status of the transaction (e.g., pending, completed, failed).

3.4 Security Audit Log Class Entities

- **LogId** (int, primary key): Unique identifier for the security audit log entry.
- **UserId** (int): The unique identifier for the user who triggered the event.
- **EventType** (string): The type of security event (e.g., login, password change).
- **Details** (string): A description or details about the event.
- **Timestamp** (DateTime): The date and time the security event occurred.
- **IpAddress** (string): The IP address from which the event was triggered.
- **DeviceInfo** (string): Information about the device used for the event (e.g., browser details).

3.5 Two Factor Authentication Request Class Entities

- **AuthId** (int, primary key): Unique identifier for the two-factor authentication request.
- **UserId** (int): Unique identifier for the user requesting two-factor authentication.
- **PhoneNumber** (string): The phone number for sending the verification code.
- **IsEnabled** (bool): Whether two-factor authentication is enabled for the user.
- **CreatedDate** (DateTime): The date and time when the two-factor authentication request was created.

3.6 Response Class Entities

- **Status** (string): Shows success/error message.
- **Message** (string): Shows description.

4. CONSIDERATIONS

- You can perform the following possible actions

Wallet,Payment,Transaction,Security

REST ENDPOINTS

Rest End-points to be exposed in the controller along with method details for the same to be created

5.1 WalletController

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>/api/wallet/get-all
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>-</td></tr><tr><td>Return</td><td>List<Wallet></td></tr></table></td><td>Get All Wallet Details</td></tr>
<tr><td>/api/wallet/create
<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>Wallet model</td></tr><tr><td>Return</td><td>Response Entity</td></tr></table></td><td>Create Wallet</td></tr>
<tr><td>/api/wallet/{walletId}
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Int walletId</td></tr><tr><td>Return</td><td>Wallet</td></tr></table></td><td>Retrieve Wallet by id</td></tr>
<tr><td>/api/wallet/update/{walletId}
<table><tr><td>Http Method</td><td>PUT</td></tr><tr><td>Parameter 1</td><td>Int walletId</td></tr><tr><td>Parameter 2</td><td>Wallet model</td></tr><tr><td>Return</td><td>Response Entity</td></tr></table></td><td>Update Wallet</td></tr>
<tr><td>/api/wallet/delete/{walletId}
<table><tr><td>Http Method</td><td>DELETE</td></tr><tr><td>Parameter 1</td><td>Int walletId</td></tr><tr><td>Return</td><td>Response Entity</td></tr></table></td><td>Delete Wallet</td></tr>
</table>

5.2 PaymentController

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>/api/payment/schedule?walletId={walletId}&amount={amount}&recipientId={recipientId}&frequency={frequency}&startDate={startDate}

<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>Int walletId</td></tr><tr><td>Parameter 2</td><td>decimal amount</td></tr><tr><td>Parameter 3</td><td>Int recipientId</td></tr><tr><td>Parameter 4</td><td>string frequency</td></tr><tr><td>Parameter 5</td><td>DateTime startDate</td></tr><tr><td>Return</td><td>Response Entity</td></tr></table></td><td>Scheduled Payment</td></tr>
<tr><td>/api/payment/instant?walletId={walletId}&amount={amount}&recipientId={ recipientId }
<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>Int walletId</td></tr><tr><td>Parameter 2</td><td>decimal amount</td></tr><tr><td>Parameter 3</td><td>Int recipientId</td></tr><tr><td>Return</td><td>Response Entity</td></tr></table></td><td>Instant Payment</td></tr>
</table>

5.3 TransactionController

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>/api/transaction/history?walletId={walletId}&startDate={startDate}&endDate={endDate}&transactionType={transactionType}

<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Int walletId</td></tr><tr><td>Parameter 2</td><td>Datetime startDate</td></tr><tr><td>Parameter 3</td><td>Datetime endDate</td></tr><tr><td>Parameter 4</td><td>string TransactionType</td></tr><tr><td>Return</td><td>List<Transaction></td></tr></table></td><td>Retrieve Transaction History</td></tr>
<tr><td>/api/transaction/create-transaction

<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>Transaction model</td></tr><tr><td>Return</td><td>Response Entity</td></tr></table></td><td>Create Transaction</td></tr>
<tr><td>/api/transaction/{transactionId}
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Int transactionId</td></tr><tr><td>Return</td><td>Transaction</td></tr></table></td><td>Get Transaction Details</td></tr>
</table>

5.4 SecurityController

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>/api/security/two-factor/setup?userId={userId}&phoneNumber={phoneNumber}

<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>Int userId</td></tr><tr><td>Parameter 2</td><td>String phoneNumber</td></tr><tr><td>Return</td><td>Response Entity</td></tr></table></td><td>Two-Factor Authentication Setup</td></tr>
<tr><td>/api/security/audit?userId={userId}

<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Int userId</td></tr><tr><td>Return</td><td>TwoFactorAuthenticationRequest</td></tr></table></td><td>Security Audit Log</td></tr>
</table>

6. TEMPLATE CODE STRUCTURE

6.1 Package: DigitalWalletManagementSystem

Resources

Names	Resource	Remarks	Status
Package Structure			
controller	WalletController, PaymentController, TransactionController, SecurityController	Controller class to expose all rest-endpoints for auction related activities.	Partially implemented
Startup.cs	Startup CS file	Contain all Services settings and SQL server Configuration.	Already Implemented
Properties	launchSettings.json file	All URL Setting for API	Already Implemented
	appsettings.json	Contain connection string for database	Already Implemented

6.2 Package: DigitalWalletManagementSystem.BusinessLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
Interface	IWalletService,IPaymentService,ITransactionService,ISecurityService interface	Inside all these interface files contains all business validation logic functions.	Already implemented
Service	WalletService,PaymentService,TransactionService,SecurityService CS file	Using this all class we are calling the Repository method and use it in the program and on the controller.	Partially implemented
Repository	IWalletRepository,IPaymentRepository,ITransactionRepository,ISecurityRepository (CS files and interfaces)	All these interfaces and class files contain all CRUD operation code for the database. Need to provide implementation for service related functionalities	Partially implemented
ViewModels	WalletViewModel, PaymentViewModel, TransactionViewModel, SecurityViewModel	Contain all view Domain entities for show and bind data. All the business validations must be implemented.	Partially implemented

6.3 Package: DigitalWalletManagementSystem.DataLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
DataLayer	DigitalWalletDBContext cs file	All database Connection,collection setting class	Already Implemented

6.4 Package: DigitalWalletManagementSystem.Entities

Resources

Names	Resource	Remarks	Status
Package Structure			
Entities	Wallet ,Payment,Transaction,Security (CS files)	All Entities/Domain attribute are used for pass the data in controller and status entity to return response Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. Generate the **Id** using the **IDENTITY** strategy	Partially implemented

7. EXECUTION STEPS TO FOLLOW

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**

2. **To open the command terminal the test takers need to go to the Application menu (Three horizontal lines at left top) Terminal → New Terminal.**

3. On command prompt, cd into your project folder (cd <Your-Project-folder>).

4. To connect SQL server from terminal:
   (DigitalWalletManagementSystem /sqlcmd -S localhost -U sa -P pass@word1)
   - To create database from terminal -
     1> Create Database DigitalWalletDb
     2> Go

5. Steps to Apply Migration(Code first approach):
   - Press Ctrl+C to get back to command prompt
   - Run following command to apply migration-
     (DigitalWalletManagementSystem /dotnet-ef database update)

6. To check whether migrations are applied from terminal:
   (DigitalWalletManagementSystem /sqlcmd -S localhost -U sa -P pass@word1)

   1> Use DigitalWalletDb
   2> Go
   1> Select * From __EFMigrationsHistory
   2> Go

7. To build your project use command:
   (DigitalWalletManagementSystem /dotnet build)

8. To launch your application, Run the following command to run the application:
   (DigitalWalletManagementSystem /dotnet run)

9. This editor Auto Saves the code.

10. To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN.

11. To test web-based applications on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.

    Note: The application will not run in the local browser

12. **To run the test cases in CMD, Run the following command to test the application: (DigitalWalletManagementSystem.Tests/dotnet test --logger "console;verbosity=detailed")**
    (You can run this command multiple times to identify the test case status,and refactor code to make maximum test cases passed before final submission)

13. **If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B - command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.**

14. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**

15. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**
