Week 3: Spring Boot and Spring Data JPA

|  |  |
| --- | --- |
| **Complexity** | Medium |
| **Technology** | Java |
| **Framework** | Spring Boot |
| **Repository support** | Spring Data JPA |
| **Database** | MYSQL|Oracle |
| **API development Approach** | Code First approach |

Task estimation

|  |  |
| --- | --- |
| Task | Day |
| Task 1, Task2,Task 3 | Day 1(week3) |
| Task 4 | Day 2(week3) |
| Task 5 | Day 3(week3) |
| Task 6 | Day 4(week3) |
| Task 7 | Day 5(week3) |

Use case:

Create a payment wallet application The payment wallet application consists of below functionalities,

* creating a wallet account
* adding amount to the wallet account
* show account balance
* transfer fund

Consider the below entities for the use case. Add additional attributes or entities if required.

|  |  |
| --- | --- |
| User | Datatype |
| id | int(pk) |
| username | string |
| password | string |
| email id | string |

|  |  |
| --- | --- |
| Transaction | Datatype |
| id | int(pk) |
| wallet\_id | int(fk) |
| Status | string |
| Amount | double |
| Date | LocalDate |
| Time | LocalTime |

|  |  |
| --- | --- |
| Wallet | Datatype |
| id | int(pk) |
| user\_id | int(fk) |
| currency\_id | int(fk) |
| balance | double |

|  |  |
| --- | --- |
| Currency | Datatype |
| id | int(pk) |
| name | string |
| abbreviation | string |

* Task 1: Create a spring boot starter project and add the required dependency as per requirement to create a **REST based application**
* Task 2: Create a layered architecture. Create package with respect to different layers like, controller dao, service etc.
* Task 3: Create entity classes use the JPA annotation and establish the relationship among the entities.
* Task 4: User story 1:

Create payment wallet account:

1.Given the user details, the data entered should be validated before registering the user and creating an account for the same.

2. Given any error or exception causing condition, appropriate validation and exception handling should be available to ensure graceful handling of such scenarios.

Validate the payload data. Use Global exception handler.

3.Create endpoint adhering to REST best practice. Use HTTP status code and user understandable messages wherever required.

4.Make use of HATEOS wherever applicable

5.Do a static code analysis by integrating SonarLint to the IDE.

6. Write the test cases using Mockito

7.Test the rest endpoint using any of the Rest client like Postman, SoapUI etc

Note:

**Post creation of user ideally the wallet should have zero balance.**

The endpoint has the below format.

<http://host/basePath/v1/resource>

* http: protocol
* host: host is the domain name or IP address (IPv4) of the host that serves the API. Since it would be running in your local environment. The host would be localhost
* basePath: basePath is the URL prefix for all API paths, relative to the host root.
* v1: version of the api. Specify the version in the URI path
* resource: name of the resource for which the endpoint is created.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rest endpoint | HTTP Verb/Method | Request Format | Response Format | Http Status code |
| http://host/basePath/v1/users | **POST** | application/json | application/json | **201** |

Sample Request

POST /users

Accept: application/json

Content-Type: application/json

{

"username": "John",

"password": "\*\*\*\*\*\*",

"email id": [JohnRobert@gmail.com](mailto:JohnRobert@gmail.com)

}

}

Sample Response

Content-Type: application/json;

Http Status code: 201

{

"content": "Wallet account created successfully",

"\_links":{

"self":{

"href":<http://localhost:8080/users/1>223

}

}

Sample Response

Content-Type: application/json;

Http Status code: 400

{

Email id not in required format

}

Sample Request

POST /users

Accept: application/json

Content-Type: application/json

{

"username": "John",

"password": "\*\*\*\*\*\*",

"email id": [JohnRobertgmail.com](mailto:JohnRobert@gmail.com)

}

}

* Task5: User story 2:

**Adding amount to the payment wallet account**

1. Given the account holding user, the user is able to add desired amount to his/her respective wallet

2. Given the addition of amount is successful, the new account balance should be available to the user for utilization at his/her discretion.

3. Given amount addition request, the amount and account details are validated against user details.

4. Given any error or exception causing condition, appropriate validation and exception handling should be available to ensure graceful handling of such scenarios.

5.Make use of HATEOS wherever applicable

6. Write the test cases using Mockito

7.Test the rest endpoint using any of the Rest client like Postman, SoapUI etc

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rest endpoint | HTTP Verb/Method | Request Format | Response Format | Http Status code |
| http://host/basePath/v1/users/wallet | **PATCH** | application/json | application/json | **200** |

Sample Response

Content-Type: application/json;

Http Status code: 200

{

"content": "Balance added successfully",

"\_links":{

"self":{

href":"http://localhost:8080/users/1223/wallet"

}

}

}

Sample Request

PATCH /users/wallet

Accept: application/json

Content-Type: application/json

{

"userId”: 1223,

"balance”: 1000

}

}

* Task 6: User story 3:

**Show payment wallet account balance**

As a payment wallet account holder, I want to be able to view / see / check current available balance in my payment wallet account.

1. Given the account holding user, the user is able to see the available balance in his/her own account.

2. Given any error or exception causing condition, appropriate validation and exception handling should be available to ensure graceful handling of such scenarios.

3.Write the test cases using Mockito

4.Test the rest endpoint using any of the Rest client like Postman, SoapUI etc

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rest endpoint | HTTP Verb/Method | Request Format | Response Format | Http Status code |
| http://host/basePath/v1/users/{userId} | **GET** |  | application/json | **200** |

Sample Response

Content-Type: application/json;

Http Status code: 200

{

“userId”: 1223,

“balance”:1000,

“currency”: Rs

}

Sample Response 2

Content-Type: application/json;

Http Status code: 400

{

Bad request. User id cannot contain characters.

}

Sample Request 1

GET: /users/1223

Sample Request 2

GET: /users/abc123

* Task 7: User story 4:

**Transfer funds from one account to another**

As a payment wallet account holder, I want to be able to transfer funds from my account to another payment wallet account.

1. Given the account holding user, the user is able to transfer some or all of his/her funds from their own account to any other account registered on the payment wallet application.

2. Given the account holding user, the transfer amount, existing balance and the account to which the transfer is made should be subjected to appropriate validations.

3. Given any error or exception causing condition, appropriate validation and exception handling should be available to ensure graceful handling of such scenarios.

4. Given the developed code, respective test cases should be accompanied

5. Since the code is built using code first approach, generate the API documentation of the entire application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rest endpoint | HTTP Verb/Method | Request Format | Response Format | Http Status code |
| http://host/basePath/v1/users/wallet/transfer | **POST** | application/json | application/json | **200** |

Sample Response

Content-Type: application/json;

Http Status code: 200

{

Transfer done successfully. Use transfer id 767788 for further reference.

}

Sample Response 2

Content-Type: application/json;

Http Status code: 404

{

"content": "Insufficient balance. Kindly review your balance.",

"\_links":{

"self":{

href":"http://localhost:8080/users/1223/wallet"

}

Sample Request 1

POST: /users/wallet/transfer

{

“walletId” : 12334,

“toWalletId”:7888,

“balance”:100

}

Sample Request 2

POST: /users/wallet/transfer

{

“walletId” : 12334,

“toWalletId”:7888,

“balance”:10000

}