**CASH CARD APPLICATION**

1. **Deliverables:**
   1. Source code zipped.
   2. Apache JMeter file for performance test.
   3. Executable jar.
2. **Execution:**
   1. Jar file can be executed using command “java –jar cashCard-0.0.1-SNAPSHOT.jar”
   2. Exposed web service APIs are as follows :

|  |
| --- |
| **URL** : http://<host>:<port>/cashCard/add  **Request Method** : Post  **Sample Request** : {"cardNumber":"1234123412341234","pinNumber":"1234","accountBalance":122}  **Functionality :** Thisapi can be used to create a new card in the database. |
| **URL** : http://<host>:<port>/cashCard/addBalance  **Request Method** : Post  **Sample Request** : {"cardNumber":"1234123412341234","pinNumber":"1234","accountBalance":122}  **Functionality :** Thisapi can be used to add balance to a certain card. Card Number would be a mandatory attribute and will be used to identify the card from the database for adding balance. The response would be the JSON containing updated account balance |
| **URL** : http://<host>:<port>/cashCard/spendBalance  **Request Method** : Post  **Sample Request** : {"cardNumber":"1234123412341234","pinNumber":"1234","accountBalance":122}  **Functionality :** Thisapi can be used to spend balance from a certain card. Card Number would be a mandatory attribute and will be used to identify the card from the database for deducting balance. The JSON response would contain the updated account balance. |
| **URL** : http://<host>:<port>/cashCard/get/<cardNumber>  **Request Method** : Get  **Sample Request** : {"cardNumber":"1234123412341234","pinNumber":"1234","accountBalance":122}  **Functionality :** Thisapi can be used to view details for a card. Card Number would be a mandatory attribute and will be used to identify the card from the database for deducting balance. |

1. **Application Explanation:**
   1. The technology stack I have used in this application is :
      * Spring Boot with embedded tomcat container and embedded hsqldb database
      * Java
      * Mockito
      * Apache Jmeter
   2. Classes that I have created can be described as below :

|  |  |  |
| --- | --- | --- |
| Class or Interface | Functionality | Methods |
|  |  |  |
| CashCardResource | Exposes all the webservices and makes call to CashCardService for business logic implementation. It also gets a generic response object from the service which can be used for reading service response object and various response messages. We can populate the HttpStatus codes based on the message. | addCard()  getCard()  addBalance()  spendBalance()  updateCard() |
| CashCardService | This class handles all the business logic as well as interact with the repository for database operations | addCard()  getCard()  addBalance()  spendBalance()  updateCard() |
| CashCard | Domain object for Rest and Database mapping |  |
| CardResponse | Generic response object which can be used for various response handling scenarios |  |
| CashCardRepository | This interface extends JPARepository and handles all database interactions |  |
| CardConstants | This contains the application constants. |  |

* 1. Rest Resources can be directly accessed once the application is up or CashCard.jmx file which I created for testing can be used for individual or performance testing especially the simultaneous swipes at multiple retailers.
  2. I have covered unit testing mainly for CashCardService.java as others are either POJO’s or interfaces mostly.
  3. All the test cases are running successfully as well.
  4. I have tested addBalance and spendBalance services with a load of 100 threads and the final state of the card was correctly maintained after the execution. Some threads might fail due to lock acquisitions but in those cases transactions are getting rolled back and not leaving the card in vulnerable state.

1. **Improvements:**

ThoughI am still having a lot of thoughts about how this can be made perfect.

I am submitting the code in the interest of time. I will list down a few :

1. Though I have handled showing the Pin in all the responses , in a real life case, I would definitely be encrypting it.
2. Validations on card and pin formats dictating the size and allowed characters can be in place.
3. HTTP Codes can be handled in the Rest Resource more gracefully using the generic response object.