# DESIGN DOCUMENT FOR SPRING BOOT PROJECT

Project Description:

Create a microservices based project using Java as the primary language that represents a eCommerce application using Spring Boot. Note, no need to focus on a UI layer.

At its simplest the application created should provide an API that exposes functionality to allow users to:

* search for products
* make payments
* make an order
* leave a review on a product

Since this is a microservice based project it needs to support the following base features of microservices using Spring Boot:

* Lightweight
* Scalable
* Out of the box capabilities
* Security
* Intelligent Routing
* Monitoring & Manageability

Design for microservices layout

User database

Product Database

API ENTRY POINT

Cart database

orders

Product Review Database

Sales services

Third party credit card api.

Unique guest cart

User Carts

Show past orders

User Services

Authentication/Authorization

Products

Services

Product update

Product create

Product delete

stock

Products

Review

Get Review

Put Review

Delete Review

Get ReviewSearch

API DESIGN :

Product API:

/GETProduct?<search String>

Produces product json with multiple Products:

/productUpdate?<ProductId>

<jsonPayload>

/POSTProduct?<ProductId>

<jsonPayload>

Produces product json for update one product only.

/DELETEProduct?<ProductId>

Produces return code success failure .

/productUpdateStock?<ProductId>?<StockLevel>

Produces product json. return code success failure.

/productCreateReview?<ProductID>?<review>

Produces review

Sample of one Product Json

1. {
2. "product":{
3. "id": 1,
4. "name": "poker chip",
5. "description”: “a simple poker chip",
6. "stock":1,
7. "review":[
8. {
9. "id":1,
10. }
11. ]
12. }

Product review : Id:

1. "rating":5,
2. "userID":123,
3. "comments": "a good product"

User

User API:

/Authorize<authcode>?userid

Returns ReturnCode

/Register<user payload>

ReturnCode?authCode?userID

Admin functions:

/GETUser?<authcode>?userid

Returns user Object

/POSTUser?<user payload>{

Returns user Object

/PUTUser?<userPayload>

Returns user Object

/DELETEUser?<userPayload>

returnCode

USER JSON

1. {
2. "user":{
3. "id": "1",
4. "name": "poker chip",
5. "email":"sample@sample.com",
6. "authcode": 123,
7. "authTimeStamp":"15:50:10-10-2019"
8. }
10. }

Sales

/AddProductToCart?<ProductID>?<quantity>?<Userid>?<authcode>

Returns cart

/ClearCart<Userid>?authcode

Returns cart

/CheckoutCart<Userid>?authcode

/ShowPreviousOrders<Userid>?<authcode>

Returns Orders

CART JSON

1. {
2. "cart":{
3. "cartID": 1,
4. "userID": 2,
5. "products":[
6. {"productID": 1,
7. "quantity":2
8. }
9. ],
10. "delivery": 12.99,
11. "total price":123
12. }
13. }

ORDER JSON

1. "order":{
2. "orderid":1,
3. "paymentReference":"123456",
4. "delivered":**true**,
5. "products":[
6. {
7. "productID":1,
8. "quantity":2
9. }
10. ],
11. "total price":123
12. }
13. }

Johns FEEDBACK

Design for Microservices Layout  
1. What are you planning on using for the API Entry Point ?  
    a. What methods are exposed publically?  
2. Products Service:  
    a. No READ method?  
    b. Would/Could Reviews be a seperate microservice?  
3. User Service  
    a. I assume you have CRUD operations here?  
    b. Any thoughts on what you'll use Authentication/Authorization?  
4. Sales Service  
    a. Payment - you've mentioned Credit card api but what about other payment options  
        How about here, using a Messaging service, i.e. put on a Message Queue the payment needed, a payment type and other details required, then a seperare microservice can listen to the queue and mock payments  
5. While not really important to know, any ideas for DB choice for each service?

API Design  
1. Your resources really shouldnt have what their doing  
instead of productUpdate productCreate,  productSearchwhy not:  
GET product/  
PUT product/  
POST product/  
DELETE product/  
GET production/search/{productName}

2. Why is Stock capitalised?  
3. As mentioned before I think reviews could be a seperate service.  
4. I dont think we need a User login method, we should look for other tools to do that.  Maybe have a look at https://developer.okta.com/blog/2019/05/15/spring-boot-login-options for inspiration.  
5. same comment about users as made about products in point 1