MeaWallet

Senior Java Developer test

Estimated execution time: 8h

1 Description

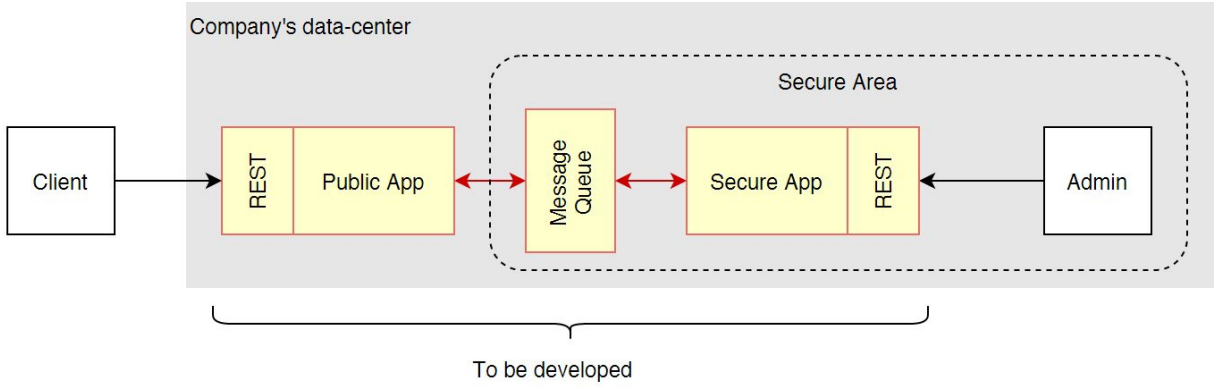
In today's world everything is based on the Internet and we are relying on security provided by service providers. To secure the data, and to limit access to core services, one of common approaches are to use Message Queues. Queues ensure single channel communication with strict data format flow, making it easier to monitor, administrate and secure communication between public and private parts.

Your task is to create a small set-up of such a solution, where you demonstrate your skills in creating web-services, interacting with Message Queues, managing the data, and showing best practices in code quality.

2 Details

Create an API which is able to provide client's information by card number. And which also provides the possibility to update or remove this information.

Solution has two applications: "Public App" and "Secure App". These two applications are communicating to each other by using the MQ server. Public App has a REST web-service, where a Client can connect and send requests through RESTful API. Additionally you can create a REST web-service on Secure App for managing all the data from Admin side.



2.1 Public App details

Public App should have a REST endpoint where the Client can pass a card or a list of cards in JSON format and receive expected response in JSON format back.
Each card has 3 fields - *cardNumber*, *firstName*, *lastName*. Format of a single card is shown in the example below.

Example:

```
{
  "cardNumber": "5309458502486118",
  "firstName": "Andrew",
  "lastName": "Lebanski"
}
```

Where input data validation should be applied:

- cardNumber - must be a valid card number
- firstName - Only latin characters, with size 2-30
- lastName - Only latin characters, with size 2-30

When *Public App* receives a request from the Client, it then forwards this request to *Secure App* through Message Queue and waits for response. Once a response is received, it returns data back to the requestor.

Public App REST API should provide following functionality:

- add one or multiple cards;
- get a list of added cards;
- update card;
- delete card.

Public App should return errors in format :

```
{
  "error" : {
      … specify here your own format
  }
}
```

***Note**: The Client for the REST API should not be developed. Use already existing applications or browsers' plugins for testing. Only Public App and Secure App should be created.*

2.2 Secure App details

Secure App listens for the incoming messages from MQ and once the request is received it handles it accordingly and replies with response back to MQ.
Data format for the messages between *Public App* and *Secure App* can be decided on your own.

3 Considerations

- ***Java 8***. Java 12 or newer would be a plus.
- ***Code styling and designing best practices***. Think about daily code reviews which you will have to pass (and later about the code you will have to review). Do your best.
- ***Test coverage***. Cover as much functionality as you can with unit tests.

- ***Make your code deliverable***. Please use Gradle (or Maven) in order to be able to build artifacts and to launch unit tests. Think of future integration of your project with Jenkins.
- ***Technology stack.*** If you have difficulties with determining appropriate technology stack, then use: Java 12, Spring Boot Web (Spring Boot Webflux would be a plus), in-memory broker (ActiveMQ, RabbitMQ, ...), in-memory database (H2, HSQLDB, …), JUnit, Gradle;
- ***Security.*** No authentication or authorization are required.
- ***Result***. Please send your sources and brief description on how to launch/test your app.

IMPORTANT: If you have no experience with JMS or think you will not be able to complete the task in time – feel free to use any communication protocol between *Public App* and *Secure App*.
IMPORTANT: If you can't finish the task due to time limitation, please send to us as much as you have, as the important part here is not the completed task, but the way you were doing it.

4 Extra points

You can earn extra points by showing your skills in functional and reactive programming or by developing additional functionality (choose any).

- Use Spring Boot Webflux or other reactive frameworks.
- Add data encryption between *Public App* and *Secure App* using asymmetric encryption (RSA keys can be exchanged between *Public App* and *Secure App*).
- Implement basic security for the REST API to authenticate clients.
- Use security to authorize client actions (do not show card data of others, allow update and deletion of only own added cards).
- Implement REST API on *Secure App* for administration purposes with methods allowing CRUD manipulations on all card data.