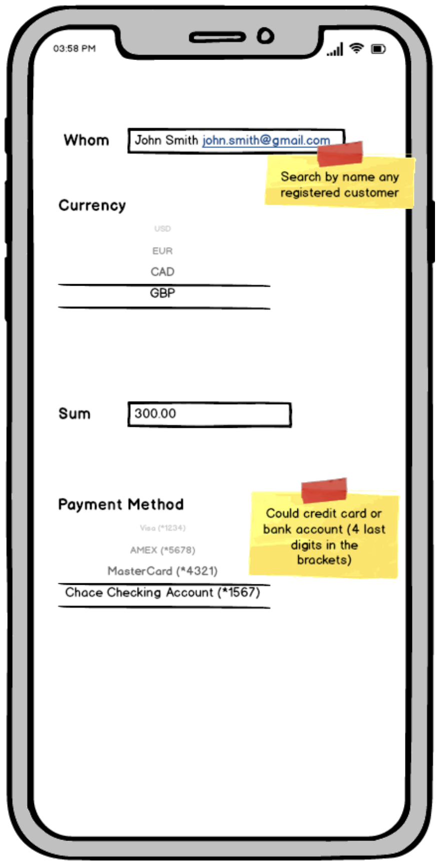
Payment System Craft Demonstration

General Context

You will design and implement a peer to peer distributed payment system that can be integrated to one of the Intuit products. The proof-of-concept integration will be done as a REST API using Maven, JAX-RS, Spring and Tomcat [or Spring Boot], any queue broker of your choice [for example, RabbitMQ, Kafka etc.] and database engine like MySQL. Although it is a proof-of-concept, we expected other teams will quickly see value in the integration and new requirements will be discovered. For this reason, the implementation should be built with a mind towards agile development and thin slices.

User Story

As a customer, I need to pay other Intuit customers for various services or products I buy from them.


03:58 PM
Search by name any registered customer
USD
EUR
CAD
Visa (*1234)
AMEX (*5678)
MasterCard (*4321)
Could credit card or bank account (4 last digits in the brackets)

Whom: John Smith john.smith@gmail.com

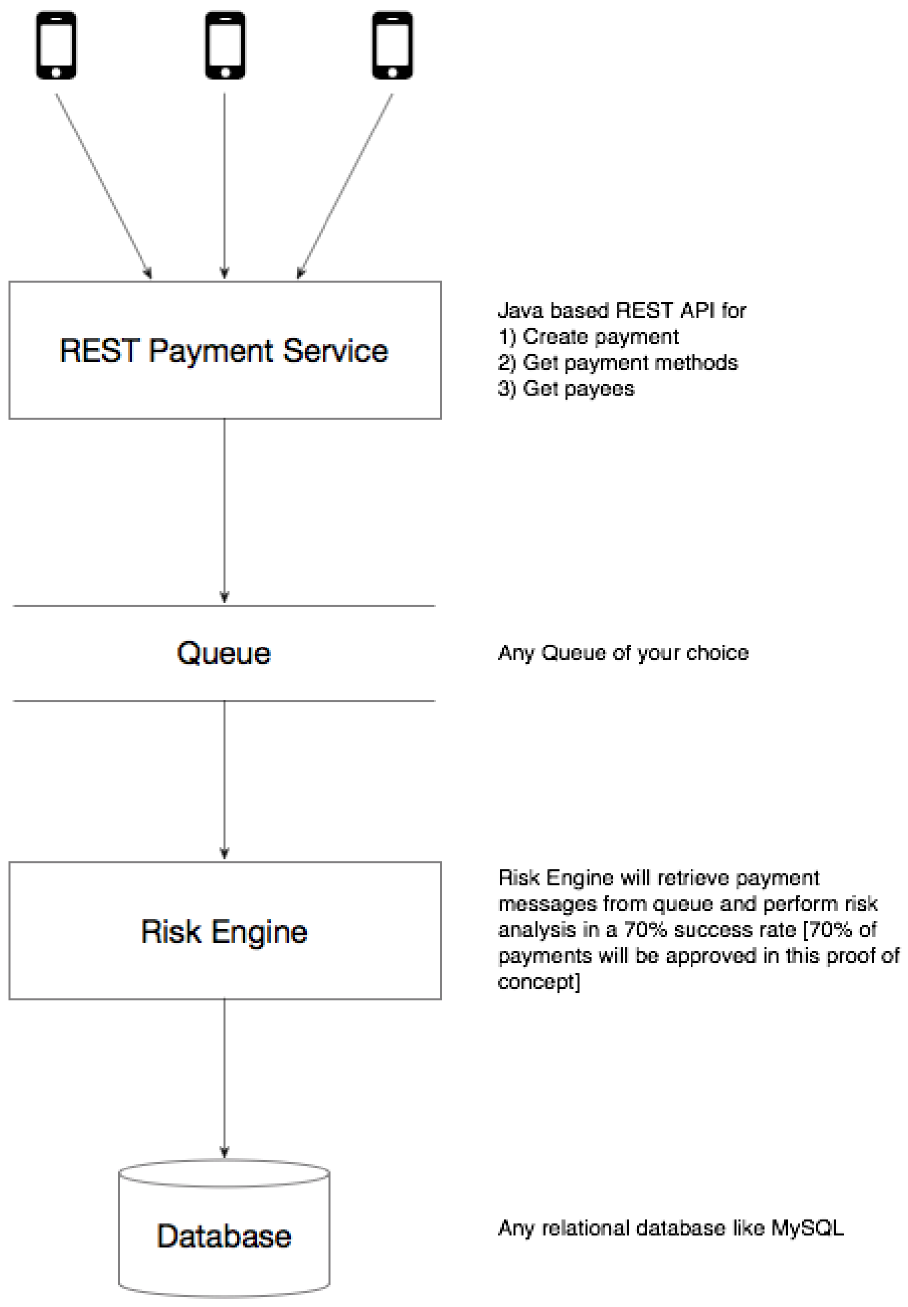
Currency: GBP

Sum: 300.00

Payment Method: Chace Checking Account (*1567)

System Design

Architecture



Java based REST API for
1) Create payment
2) Get payment methods
3) Get payees

Any Queue of your choice

Risk Engine will retrieve payment messages from queue and perform risk analysis in a 70% success rate [70% of payments will be approved in this proof of concept]

Any relational database like MySQL

Payment Entity JSON Example

```
{
  "amount": 70.5, // type: number, description: amount to pay
  "currency": "USD", // type: string, description: payment currency
  "userId": "e8af92bd-1910-421e-8de0-cb3dcf9bf44d", // type: string, description: Paying user unique identifier (GUID)
  "payeeId": "4c3e304e-ce79-4f53-bb26-4e198e6c780a", // type: string, description: Payee user unique identifier (GUID)
  "paymentMethodId": "8e28af1b-a3a0-43a9-96cc-57d66dd68294" // type: string, description: Payment Method unique identifier (GUID)
}
```

Assignment

[Nice-to-haves are **optional** to implement, **not mandatory**. We would like you to consider them from a **theoretical** perspective, as we will refer to them during the technical assessment]

Question 1

Design and implement REST API using Maven, JAX-RS, Spring and Tomcat [or Spring Boot]. Consider REST API design practices, response codes and bodies, error codes and bodies, naming conventions.

(Nice-to-have: Testability considerations)

Question 2

Write code in REST API that will publish new payments into any queue of your choice like Kafka, RabbitMQ or any other.

Question 3

Develop a proof of concept of the risk engine that will consume payment messages from queue and perform risk analysis. For the simplicity of the exercise allow only 70% of the payments to succeed. Assume that risk engine will be hosted in a cloud service.

(Nice-to-have: horizontal scalability requirements, build considerations such that refactoring or adding new functionality can be achieved with confidence).

Question 4

Answer theortically: define database structure and store payments along with their risk assessment in the database of your choice.

(Nice-to-have: implement your solution)