02267: Software Development of Web Services

Exam Project

Hubert Baumeister
huba@dtu.dk

Department of Applied Mathematics and Computer Science
Technical University of Denmark

January 2024

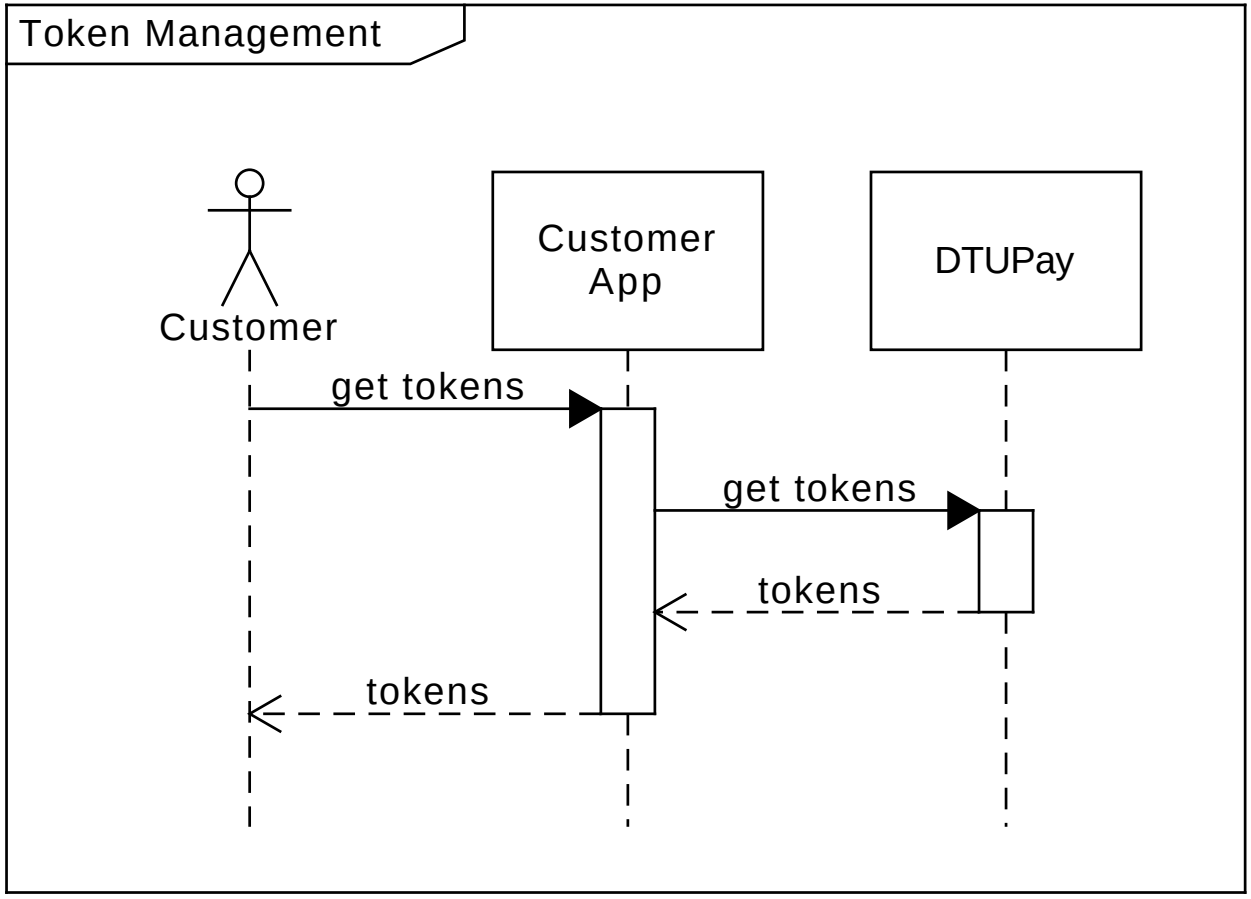
Problem: DTU Pay

- Mobile pay solution between customer and merchants
- Similar to Simple DTU Pay: Differences
  - Customer authorizes money transfer with token
  - More functionality
  - Use several communicating Microservices

Payment

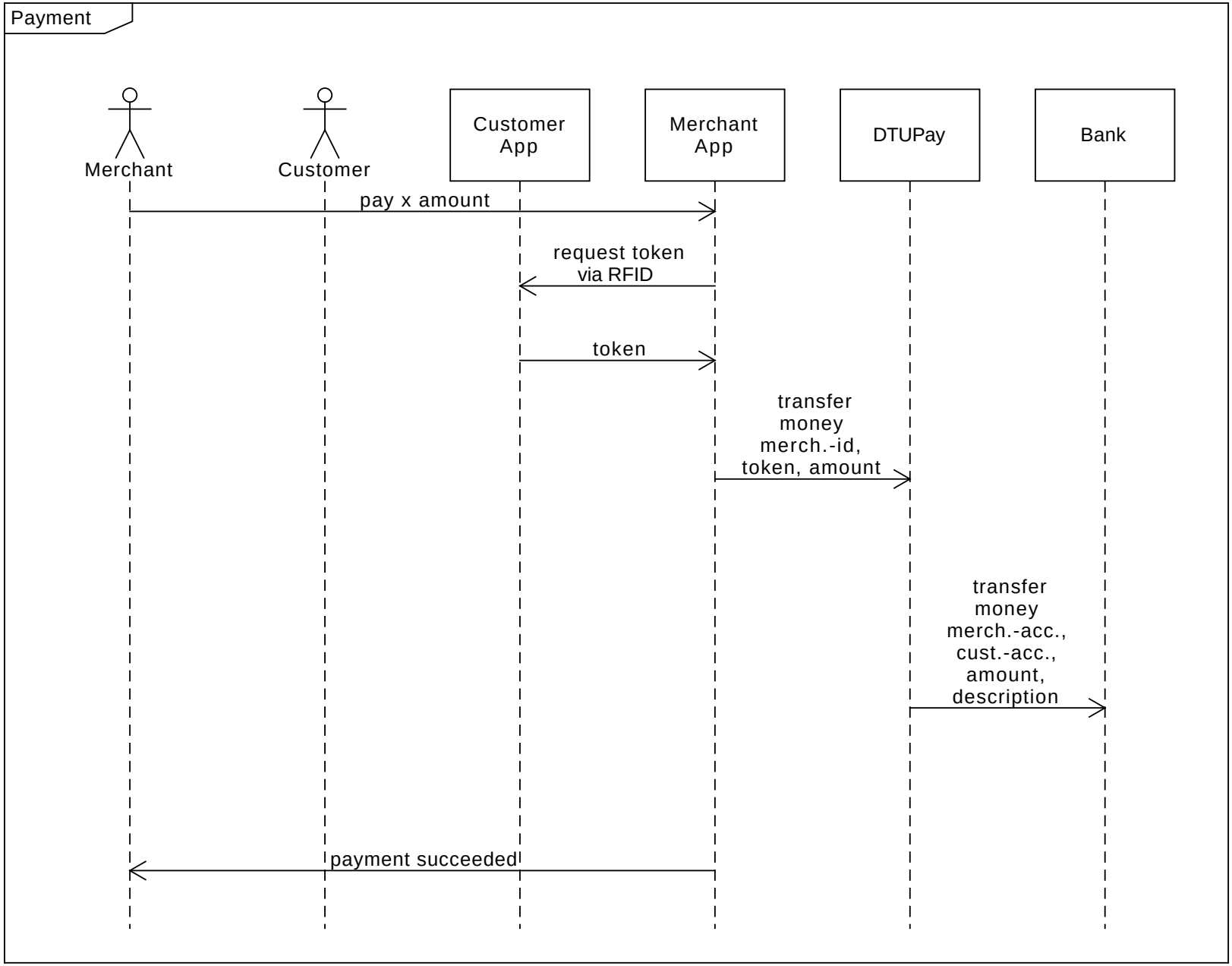
- Customer and merchant have each a bank account
- Customer and merchant are registered with DTU Pay
- Customer has up to 6 unqiue tokens from DTU Pay on his mobile phone
- Customer shows **one** token to the merchant (using RFID) to authorize the payment
- Merchant uses this token plus his id with DTU Pay plus the amount to pay to initiate the payment
- Privacy: The customer id is not known to the merchant nor the bank account number of the customer
  - Only DTU Pay can make the connection from the token to the customer's account
- DTU Pay initiates the money transfer between the customer and the merchant

Token Management

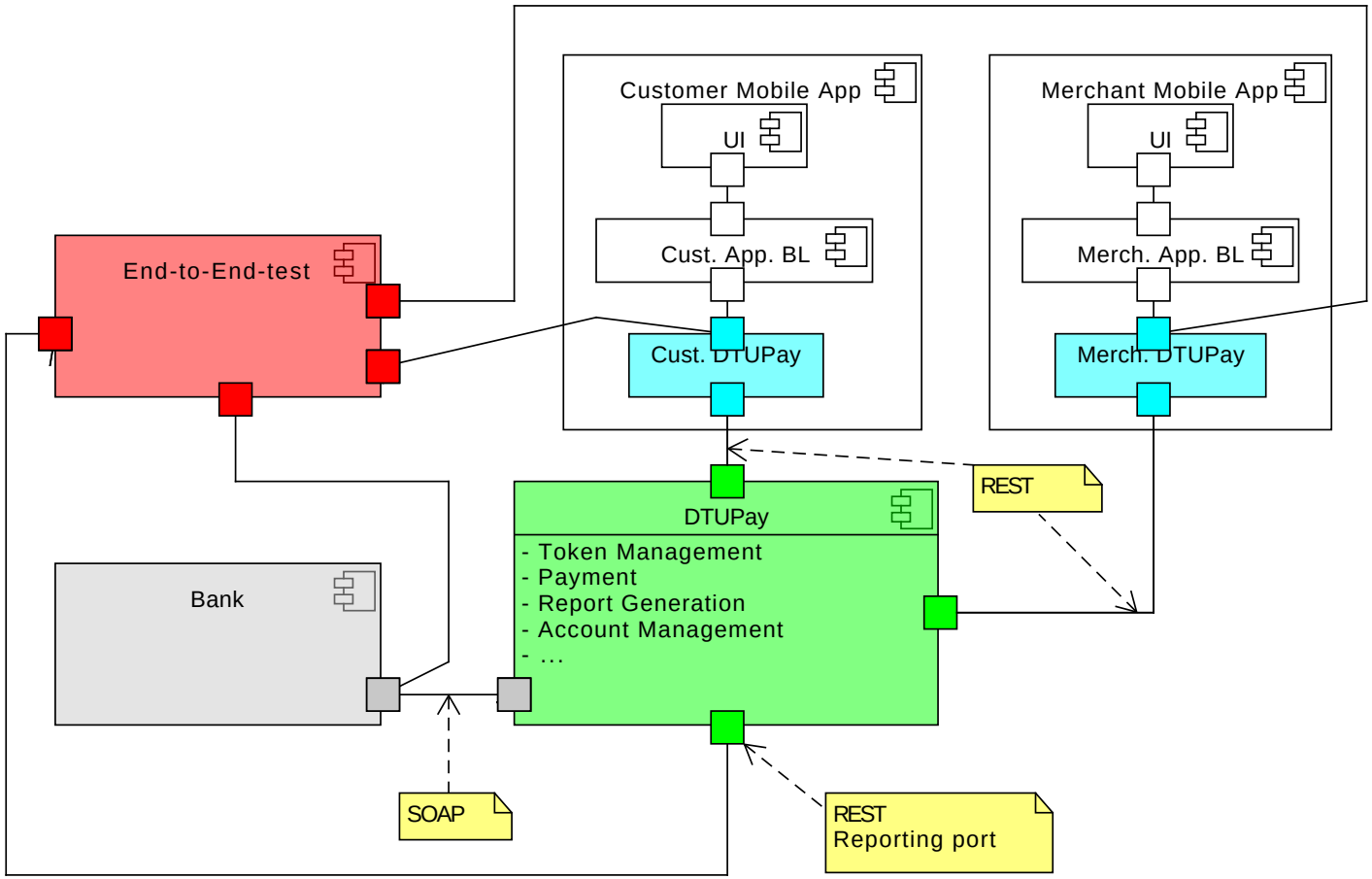


- Max 6 unused tokens on App

Payment Process



Architecture



- Green: Business logic belonging to DTU Pay
- Red: Components belonging to the end-to-end test
- Blue: APIs to DTU Pay running on the mobile phones

Functionality (success-, failure-, edge cases)

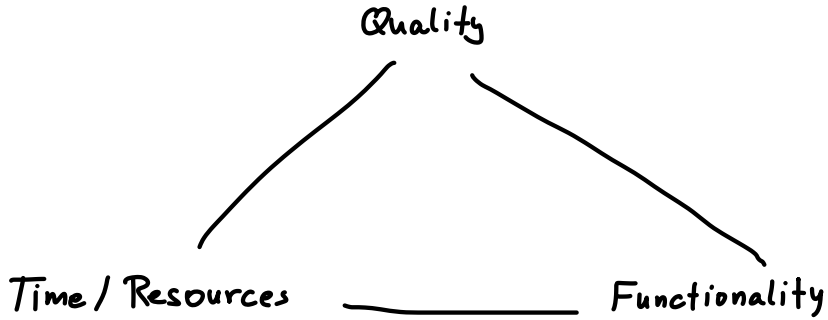
- Payment at merchant
- Token management
- Customer/Merchant Account Management
- Reporting
- Refund Payment
- ...

Priority

- Payment and token management

$\rightarrow$ ask me if in doubt

What is important (quality before functionality)



- Microservice architecture using docker container
- Good but **simple** design (KISS and YAGNI)
- Ideally *asynchronous communication* with *message queues* between microservices
- Communication to the outside: REST and SOAP (to the bank)
- Automated tests (end-to-end and service tests) with a high code coverage (*no documentation needed*)
- Business logic tests using Cucumber scenarios.
- Automated build, test, and deployment using Jenkins
- Repositories are needed but can use lists and maps instead of, e.g., SQL databases

Deliverables

- Source code with tests
- Build scripts (*I need to be able to build, deploy and run the tests locally* ***without using your Linux VM***)
  - You can assume that the computer I am using runs Linux and Ubuntu
- Reports (domain model, architecture, user manual, instruction manual)
- Swagger/OpenAPI files for the REST interfaces
- **Deadline: Friday next week (19.1.) at 17:00**
- Mark your contribution (in the report and the source code) (aka. *individualized* report/software)
- Make sure everybody did something of everything
  - *Everybody has to fulfill, and document that he has fulfilled, the learning objectives*

Tips

- Watch the remaining videos and *run and understand the example code*
- Iterate through user stories focusing on the payment scenario first
- Start with a *small* working system, add a small change to get a new working system
  - Evolutionary Design 2min video by Joshu Kerijevski (`https://youtu.be/r6I9M8FaGlU`)
  - Evolutionary Design Animated James Shore (`https://youtu.be/QJRAeoOHewo?t=2795`)
- For a user story:
  - Use *event storming* to discover the domain events and make a domain-driven design
    - Identifying aggregates, aggregate roots, and bounded contexts
    - Implement happy paths first, and then test for an implement failure scenarios
- Use Mob programming to get a common understanding of the problem and the solution