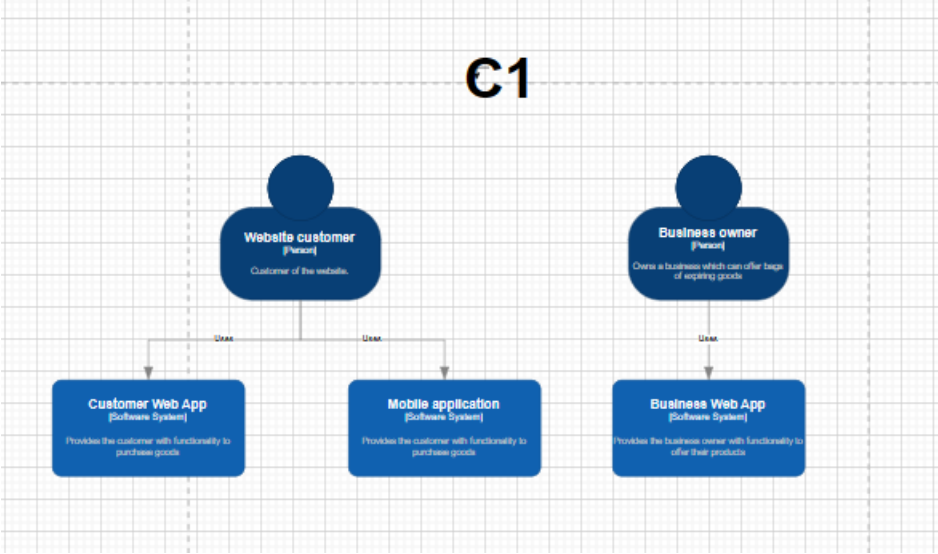
Architecture document

Introduction

This document is going to discuss design choices made for the architecture of the application, the process of making those choices. What is more, different iterations of the architecture will be included in order to display the improvement since the first draft.

Application Architecture

Per planning, this application would have 2 different types of users – customer and business owner. What is more, the required functionality for both users differentiates, which is the reason for the separate interfaces of the application. What is more, the customer side of the application is planned to have 2 interfaces utilizing the same functionality. The visualization of this can be seen in the image below.

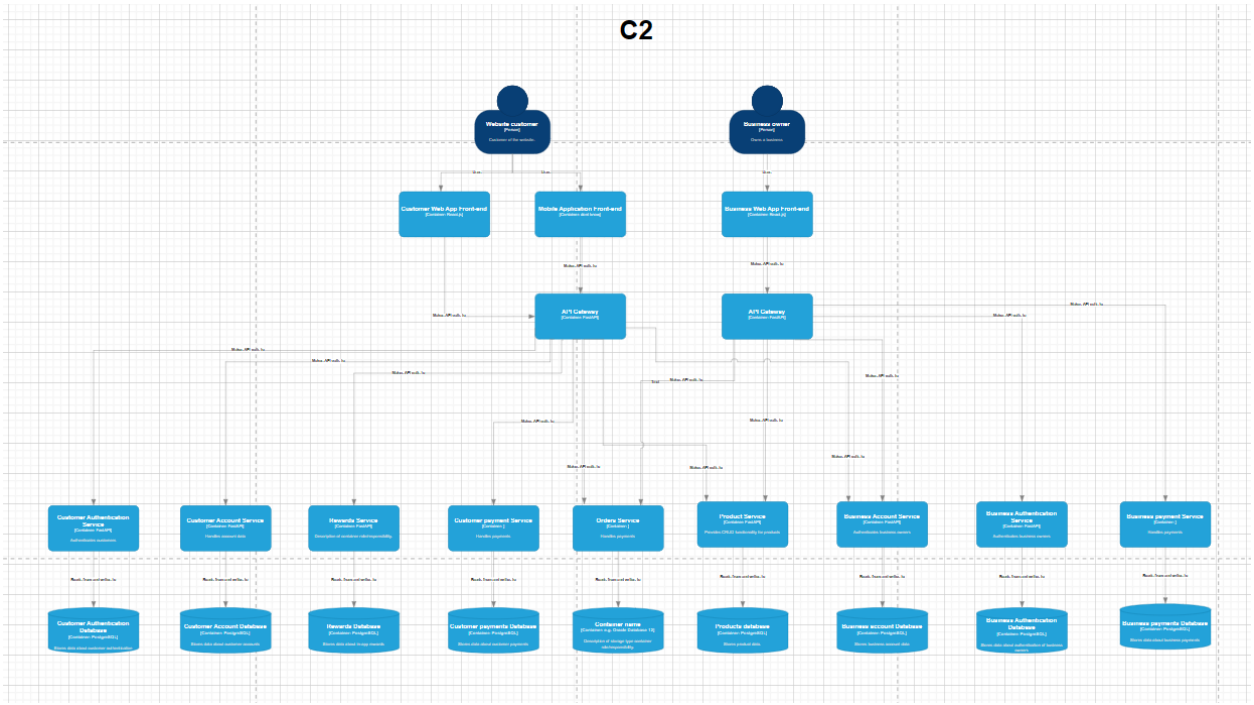


First draft of architecture

What is more, I designed this application to utilize microservice architecture. The reason for this is that I expect some functionalities would have a higher load than others, which can be mitigated by using microservices because separate components can be scaled differently. Furthermore, this type of design would allow my application to continue operating even if one of the services is not currently operational. In addition, customer and business side of the application would have an API gateway of their own for the same reason – different scaling requirements and ensuring high uptime of the application. The API gateway is going to communicate with each service through HTTP. In terms of services, taking all requirements into account, I designed the application to have 9 services in total. Each service would have its own database in order to apply the microservice architecture efficiently. The services are split by functionality and security reasons. For example, user data is split into 2 services in order to keep the sensitive user data like usernames and passwords separate from regular user data like account settings, preferences, etc.

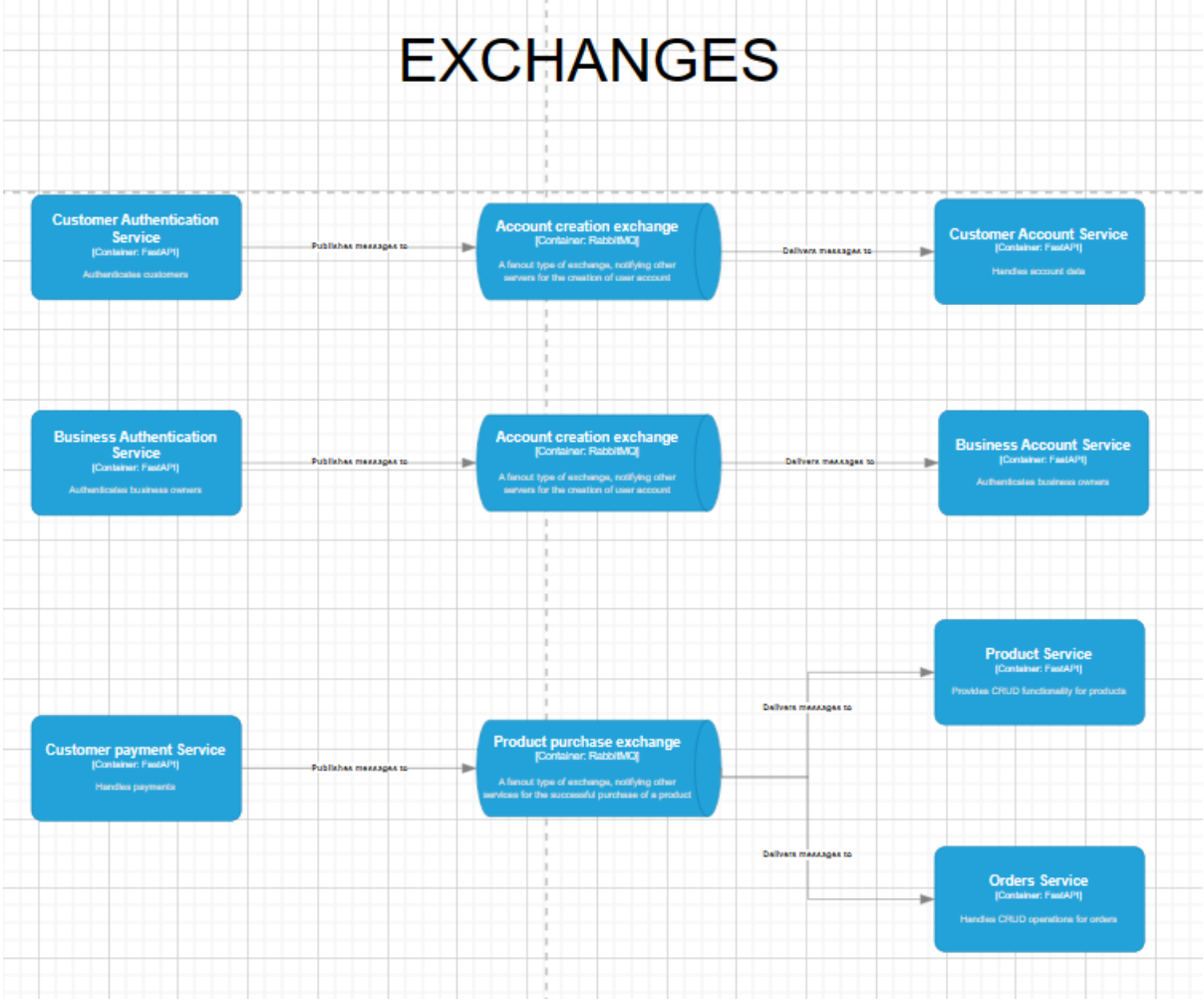
Feedback

After discussing this design with technical teachers, some flaws were identified. Firstly, since I planned to have the services communicate with each other over HTTP, I got advised to develop another way for them to communicate since that type of communication between services would slow the application down too much. What is more, one of the teachers noticed that I have designed my architecture with too many services for the timespan of this semester and suggested to narrow it down before implementation.

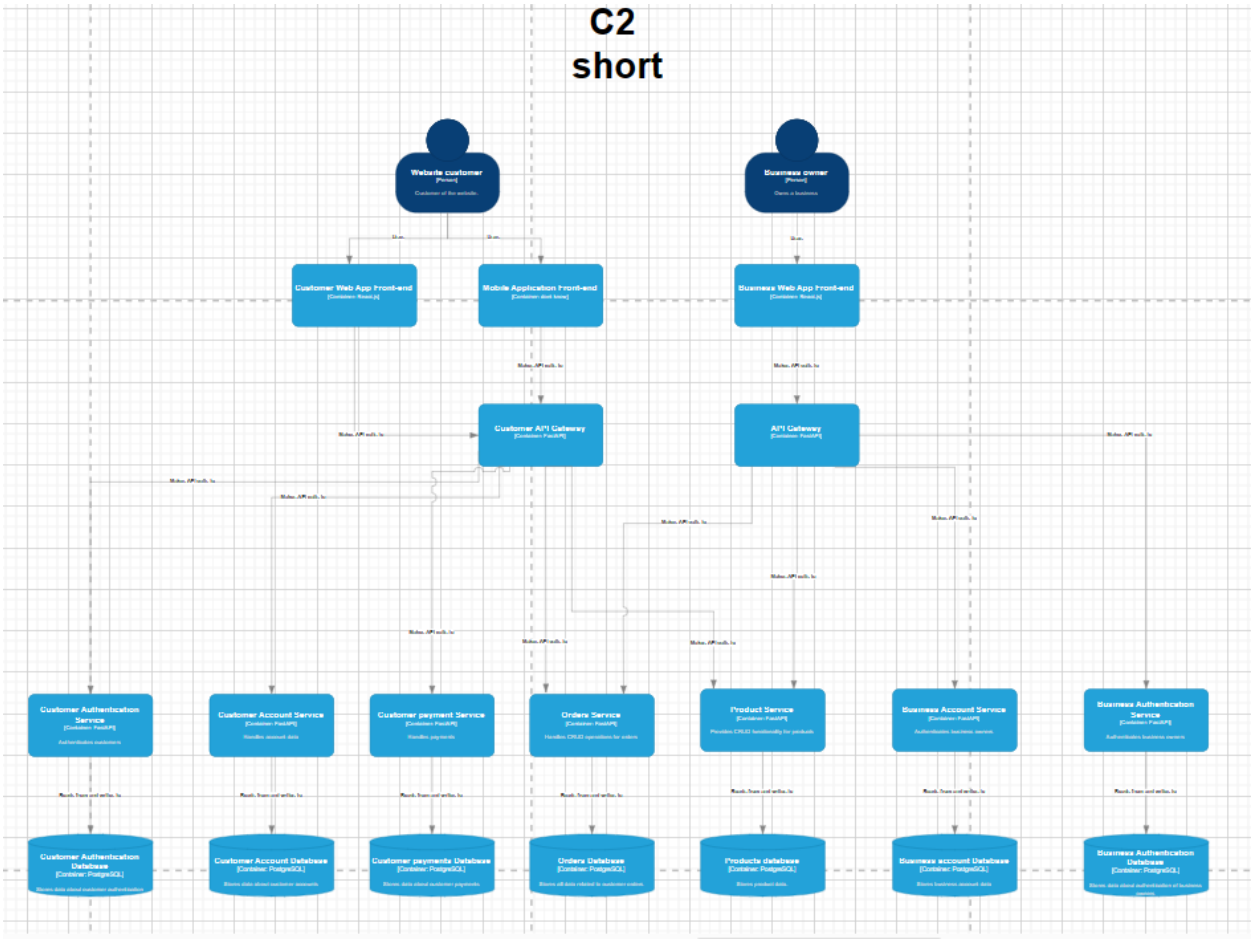


Second draft of architecture

Following the feedback of the last iteration, I began looking into solutions for the communication between different services. A solution that was suggested by one of the teachers was to make use of message brokers. The message brokers are going to run independently from the services and will ensure that the required information is delivered to the specified service, even if the service is currently occupied by other work using retry mechanisms. This way, when one service needs to provide information to another, it just posts it to a message channel which takes care of the transmission and the service can continue operating. In RabbitMQ, a widely used system for this purpose, pipelines that deliver that information are called “exchanges”. In order to support the functionality of the system, I designed a few “exchanges”, which are displayed in the diagram below:



Furthermore, some of the services needed to be removed from the design, in order to be able to complete it by the end of the semester. After a discussion with both teachers, we agreed that I would not be developing the functionality for the business account and authentication service since it would essentially be the same as the customer account and authentication. What is more, since the business side of the application is not going to be implemented for now, the products service is only going to be used for reading data.

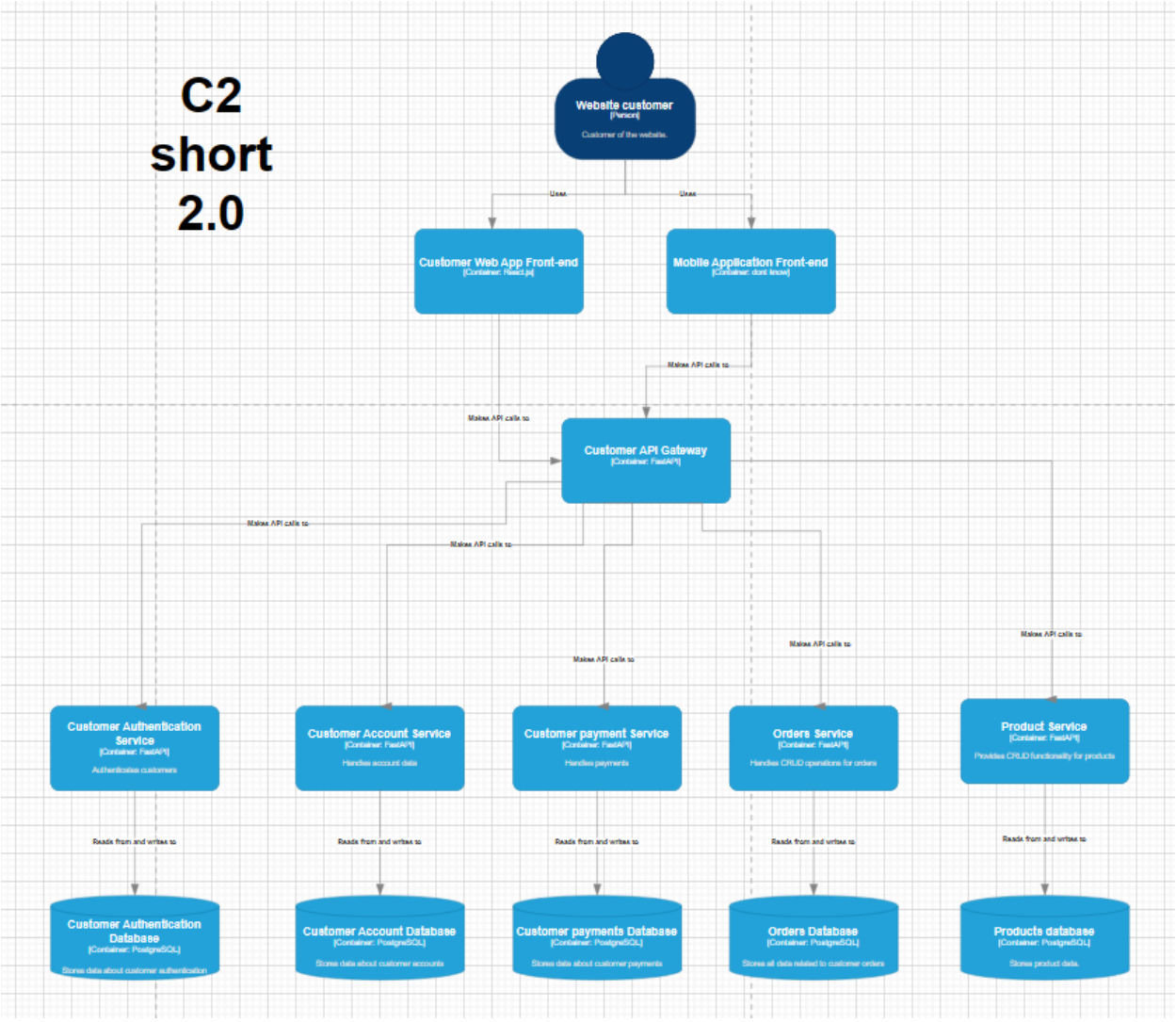


Feedback

After discussing this diagram with the teachers, we agreed that the business side of the application replicates the same functionality as the user side of it so I am not required to develop the business authentication service, business account service and business API gateway.

Final version of architecture

This is the final version of the architecture based on the given feedback. It consists of 5 services – authentication, account, payment, product and order service. All services would be accessed through 1 gateway. The front-end of the system is not required to be built.



Design choices

For the development of the microservice system, I would make use of Spring cloud and its associated packages:

- **SpringBoot** – development of RESTful services which communicate over HTTP.
- **Spring Cloud Gateway** – provides a simple way to route the APIs while providing monitoring capabilities and security functionalities.
- **Eureka** – Used for locating the other running services.
- **Spring Security** – generating and validating JWTs;

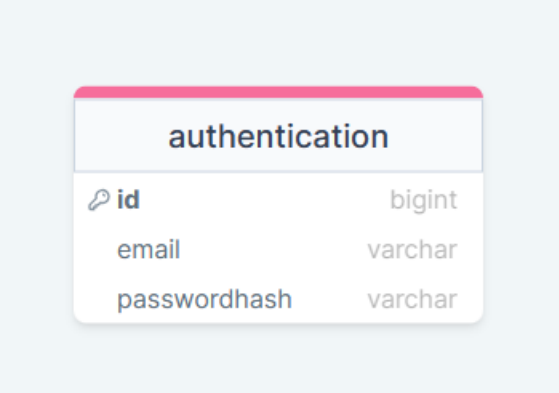
- **Spring JPA** – handles communication between a service and SQL-like databasae.
- **Spring Data MongoDB** – handles communication between a service and MongoDB database;
- **Spring Data Cassandra** – handles communication between a service and Apache Cassandra database;

Furthermore, for databases I selected to use:

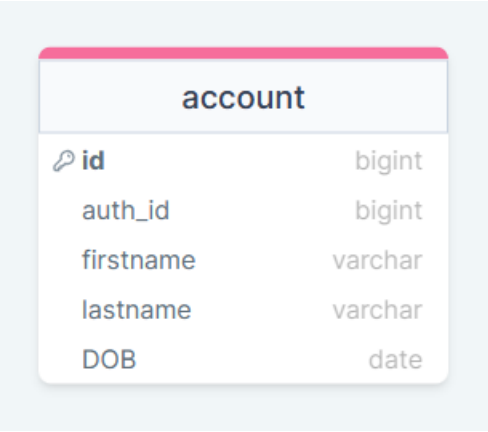
- **PostgreSQL** – Relational database. Will be used when the relationships between the data entities are required. Generally for read-heavy operations.
- **Apache Cassandra** – Non-relational database. Will be used in services where high volume of data is expected. Can handle both read-heavy and write-heavy operations at huge volumes due to its horizontal scaling across multiple nodes.

Database Architecture

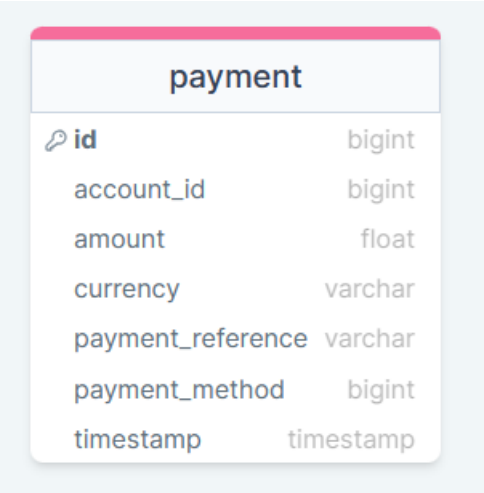
Authentication Service



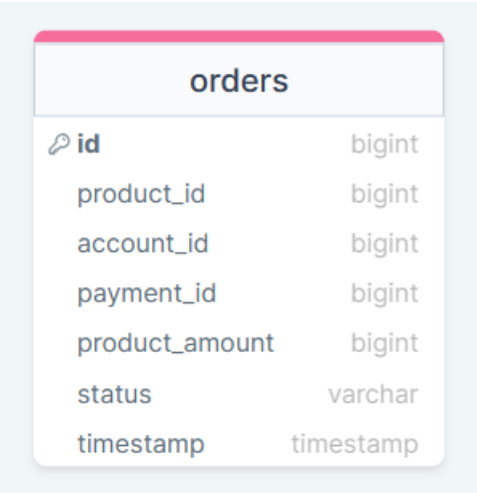
Account Service



Payment Service



Orders Service



Products Service