Energy Management Application

Distributed Systems - Assignment 1 Documentation

Vlad Bartolomei
Group 30643
Guiding Teachers: Marcel Antal, Vasile Ofrim

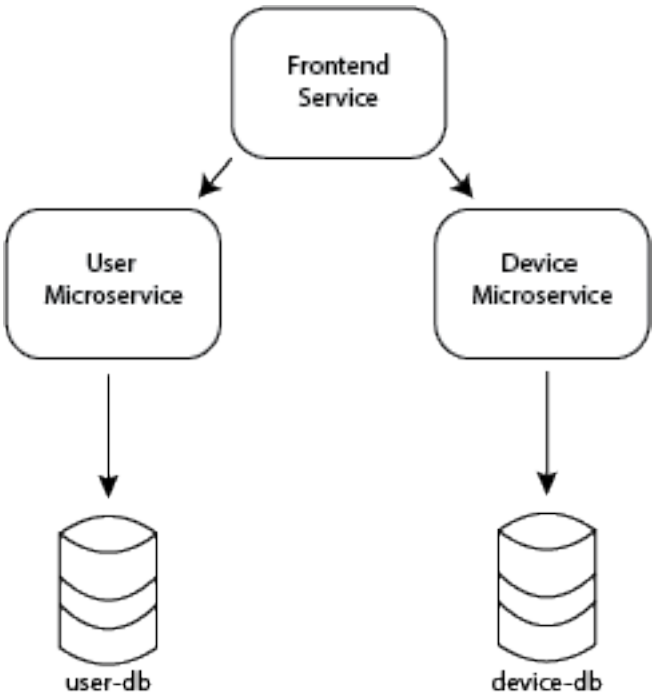
October 31, 2024

1 Conceptual Architecture of the Distributed System

The current assignment consists of a web application responsible with energy management of a household. As it was specified in the requirements, two[1] microservices, each one with its own database:

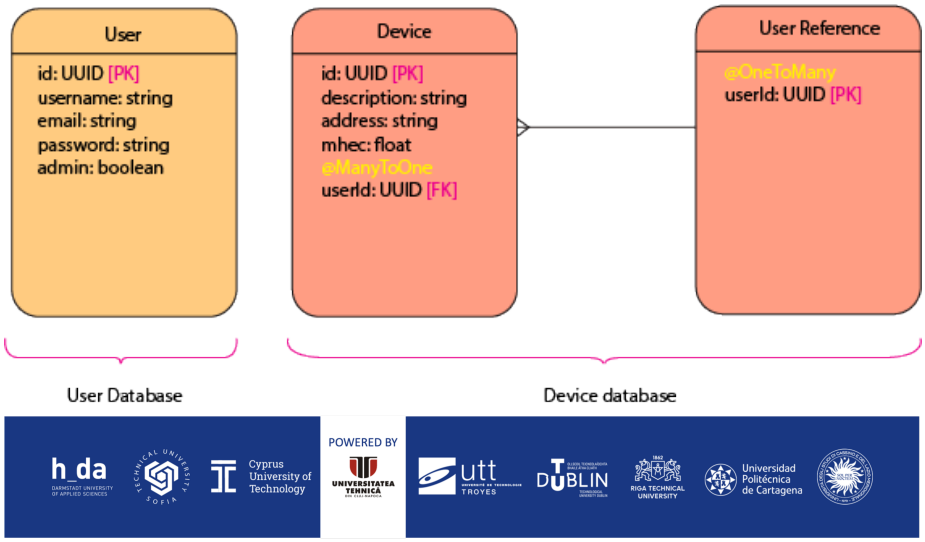
- UserService - managing the users;
- DeviceService - managing the devices of a household. Each users has a handful of devices, but a device can belong to one user only.

[1] Actually three, we'll talk about that when we'll introduce security



1.1 Database

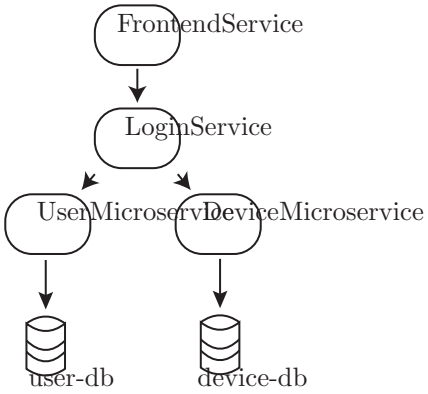
The platform can handle two sorts of users: *clients* and *admins*.



1.2 Security

Normally, no logged admin should be access a client's page nor the other way round. To fulfil that, Angular (our frontend framework) provides a tool called *Guards*. In essence, the authorization guard we defined takes the logged user and exposes its *admin* field. Guards come into play on routing modules. All routes will be guarded by the earlier-described guard and for each route will also be specified the allowed role. Matching of that role and the logged user role (is or isn't an admin) will determine if access to that route is allowed or not[2].

More broadly (and conceptually) speaking about security introduces to us a new microservice, present in our project and architecture. We're talking about Login Service, which acts as a proxy for the other two microservices. The disadvantage with this architecture is that all requests are bloated into this Login Service, an extensive project which re-routes the http requests. So, in fact, our project looks like this:



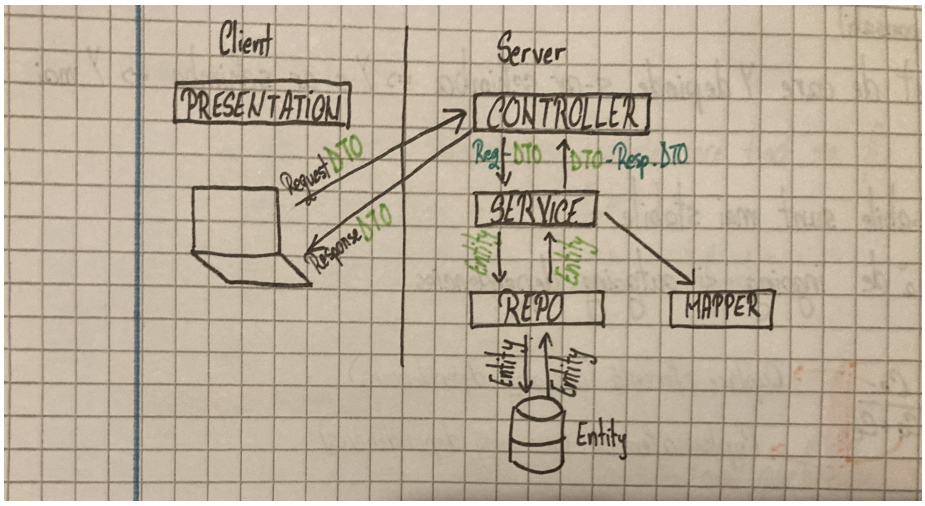
2 Deployment Diagram

By using HTTP requests, it's bound that the whole system is designed as a client-server application.

2.1 Controller-Service-Repository

Server-wise we decided to go with a broadly used design pattern: **Controller-Service-Repository**.

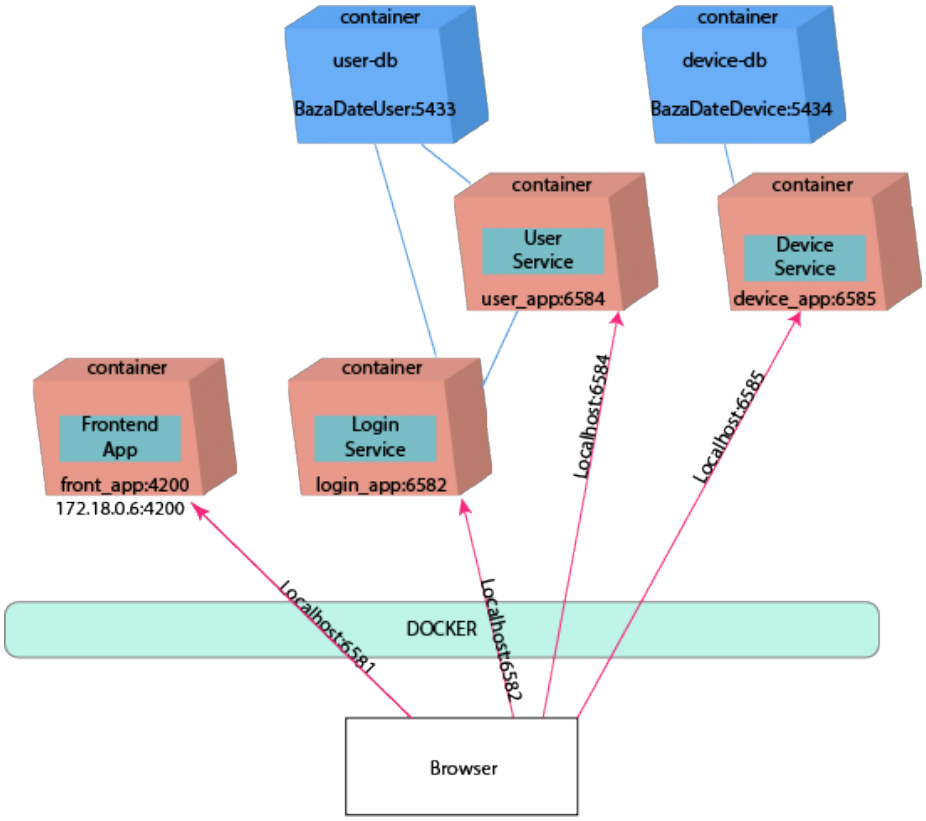
[2]Case in which the client is redirected to a Not Permitted page



To reduce traffic and hide some uninteresting details (when making a request, a client probably wouldn't want to send an UUID), we use Data Transfer Objects (DTOs). But at the closest-to-database layer, namely Repository, we can only work with Entities. So, for each Entity, a Mapper is needed to handle Entity-DTO conversion (or, in some cases, List of Entity - List of DTO).

Controller uses RESTful endpoints and it's annotated as @RestController.

2.2 Docker Deployment Diagram



3 Readme

Please refer to the project repository. There can be found the Readme.md document in discussion.