DM848 Microservice

WebApplication using the microservice architecture

Lars Thomasen
<latho12@student.sdu.dk>

Table of Contents

1 Introduction **2**
1.2 Problem formulation 2
1.3 Running the code 3

2 Modelling **3**
2.1 Exposing the front-end 3
2.2 The services 5

3 Implementation **5**
3.1 Service registration 5
3.2 The proxy 5
3.3 Communication 5

4 Conclusion **5**

Chapter 1

Introduction

For this project a web application using the microservice architectural style has been created. This allows one to take the theories learned from the course and implement them into a real product, thus ensuring that the theory is well understood and that the know-how on how to use them in practice is there.

The microservice architectural style is a stryle to building a single application, consisting of a suite og small services, each running on its own process and instead communicates with the other services. This is a great contrast to the otherwise monolithic style that are well known, where the entire application often consists of a single executable.

This approach offers great flexibility, making it possible to quickly increase the processing power of pre-defined areas of the application, by simply increasing the number of that specific service. Likewise it increases the reliability, as the decline of one or more services wont result in loss of service, as long as atleast one or more services remain running.

1.2 Problem formulation

The specific problem formulation for this project are as following:

Goal: Create a web application using the microservice architecture, this application must have several different services, each serving its own purpose. The application must be highly scalable and take advantage of load balancing and have a well defined interface to the public.

Tools: In order to follow the ideas used widely in the community, this will be done using the Netflix Open Source[1] stack, which is a set of tools developed by netflix when creating microservice services.

[1] See **Common Runtime Services & Libraries** at https://netflix.github.io/

1.3 Running the code

The following dependencies are required in order to run the program:

1. Java 8
2. Maven

A batch script has been created, simply run `mvnpackage.sh` in order to compile all the services using maven, and then run `runServices.sh` in order to start the 5 services. Note that starting all the services and registering them with eureka may take a couple of minutes. The webapplication is available at localhost:9999.

A readme file is included in the project root folder if a manual approach is wanted. This allows for setting a given port by feeding it a parameter, and will have logging enabled. Alternatively, the application is deployed at `http://lthomasen.me:9999`.

Chapter 2

Modelling

This chapter will describe the design choices followed when modelling the application.

2.1 Exposing the front-end

One of the very first thing to define was how to expose all the services to the user. There er several approaches to this, some of these are described here, and the reason for going with one of these are also outlined.

1. The first and most simple approach would be to have a single monolithic style front-end, which handles all incomming and outgoing requests. This front-end will handle exposing the views and then communicate with all the services behind it. This allows for a static front-end which will always be available at a known location, much like any other approach.

2. Another method would be to split up the front-end into individual services, the best way to describe this would be to imagne the front-end web design split up into sections, and then each section being its own service.

3. And finally, the third option would be to use a proxy, which takes incoming requests and then delegates these requests, often in a round.robing style, to all the services which accepts the type of request given.

For this project option (3) was used. There are several advantages to this approach, and some of these are as following.

The first advantage being that having an extremely lightweight proxy delegate the requests allows for load balancing the web-server, this allows to spin up more than a single web-server such that no bottleneck would be present in generating views and sending them to the users.

The second advantage of this approach is that following the idea of splitting up the front-end into several services would still be possible, by modifying the proxy to simple look at the request type and send it to the correct service.

And finally this allows for having a single point of entry, such that all requests are served over the same port. This avoids any issues which would otherwise be present when working with Ajax-requests using JavaScript, which would be blocking any cross-site scripting.

2.2 The services

Chapter 3

Implementation

3.1 Service registration

Load balancing

3.2 The proxy

3.3 Communication

Chapter 4

Conclusion