Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

**Computer Science**

**MSc in Web and Mobile Development**

**CCP6427: Cloud Engineering**

**Spring - 2021**

**Report**

**Coursework 1**

**Submission Deadline: 22/05/2022**

|  |  |
| --- | --- |
| **Professor** | **Student** |
| Simos Veloudis  sveloudis@york.citycollege.eu | Grigoris Papapostolou (WMY21034)  gpapapostolou@york.citycollege.eu |

**Declaration**: *I have completed and submitted this work by myself without assistance from or  communication with another person either external or fellow student. I understand that not working  on my own will be considered grounds for unfair means and will result in fail mark for this work  and might invoke disciplinary actions. It is at the instructor’s discretion to conduct an oral  examination, if necessary, which will result in the award of the final grade for that particular piece  of work.*

**Table of Contents**

**Chapter 1. System Architecture.................................................................... 3**

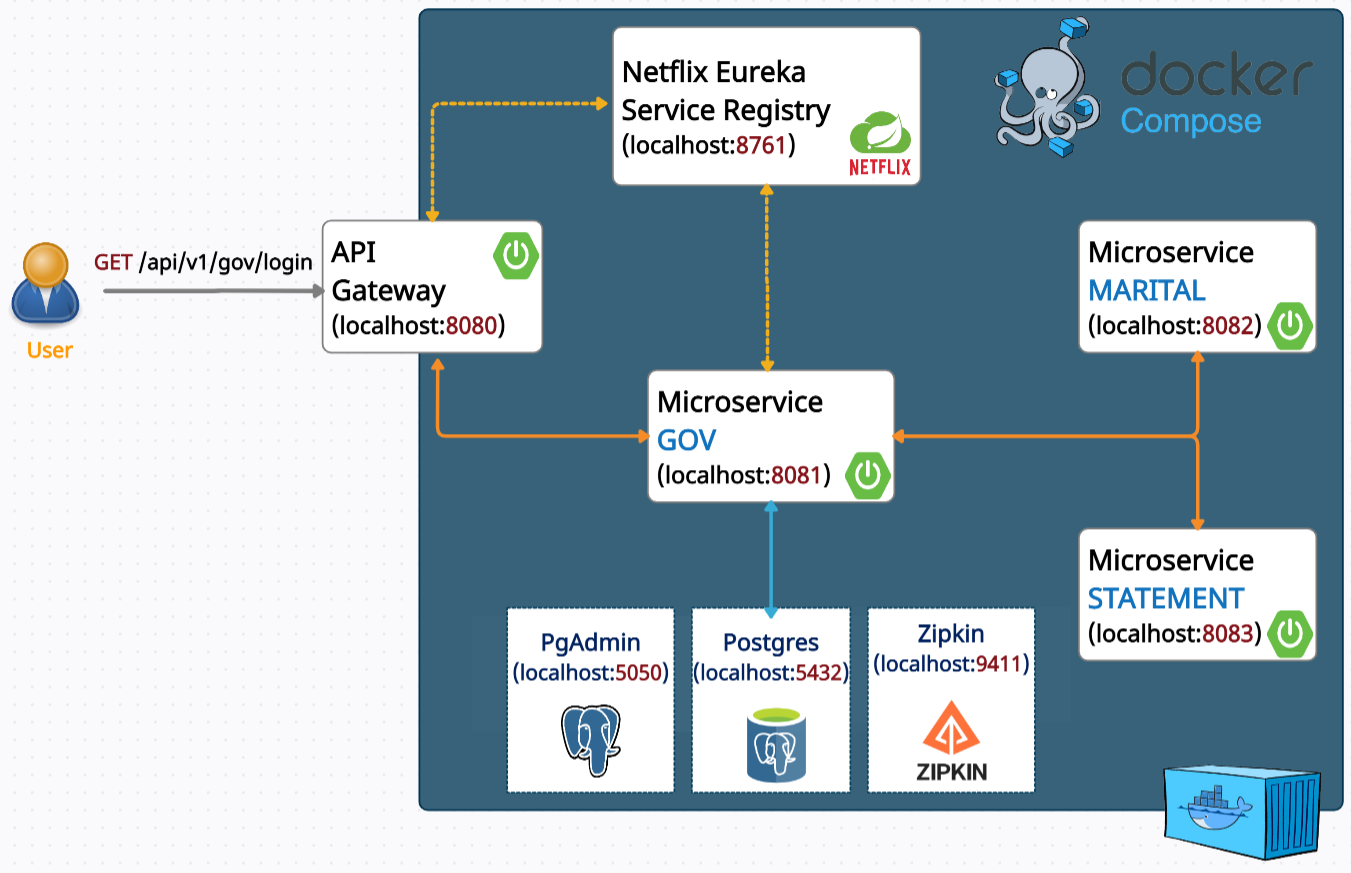
**Chapter 2. Set Up…........................................................................................ 6**

**Chapter 3. Application Demo......................................................................... 6**

**Chapter 1. System Architecture**

The purpose of this coursework is to create microservices which will simulate the a government page named gov, to which every registered user will be able to download in pdf format a responsible statement or certificate of marital status.

Following is a high-level architecture of our system:

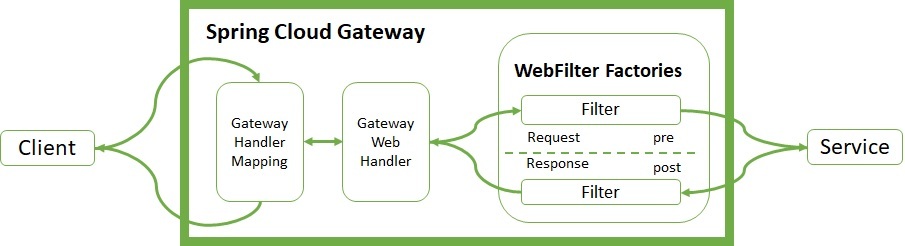


*Figure 1-1. System Architecture*

The system consists of the following major components - microservices:

* **API Gateway**: An API Gateway acts as a single entry point for a collection of microservices. Any external client cannot access the microservices directly but can access them only through the application gateway. So, this improves the security of the microservices as we limit the access of external calls to all our services. Also concerns like authentication, monitoring/metrics will be needed to be implemented only in the API Gateway as all our calls will be routed through it. So, API Gateway acts as a reverse proxy built before the API.

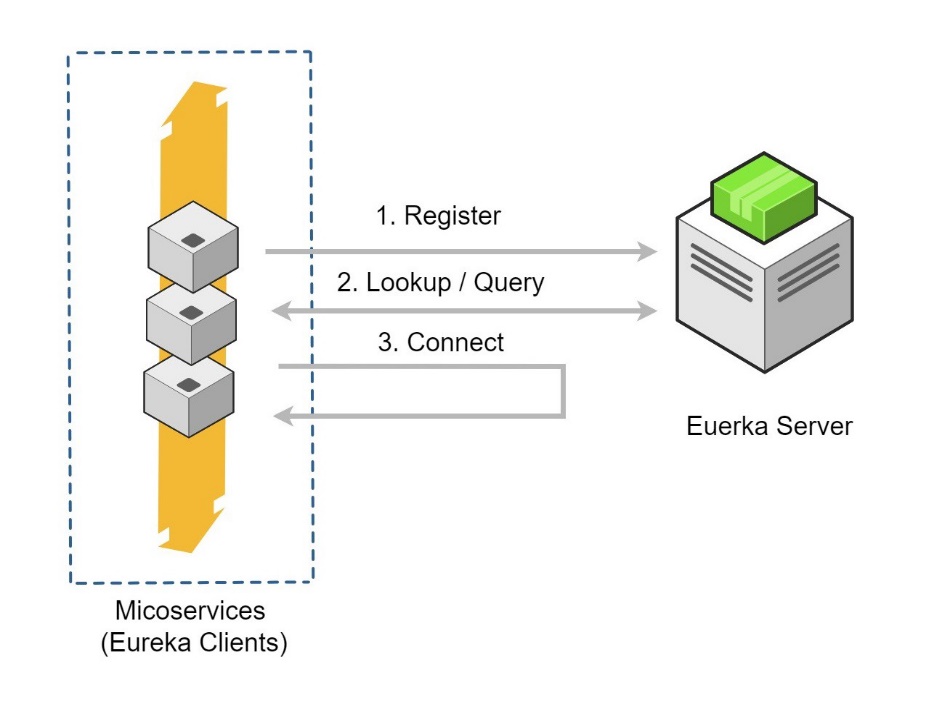
Spring Cloud Gateway is a non blocking API which means a thread is always available to process the incoming request. These request are then processed asynchronously in the background and once completed the response is returned.



*Figure 1-2. Spring Cloud Gateway Architecture*

Clients make requests to Spring Cloud Gateway. If the Gateway Handler Mapping determines that a request matches a route, it is sent to the Gateway Web Handler. This handler runs the request through a filter chain that is specific to the request. In our app, the api gateway communicates directly with GOV microservice which handles all the incoming requests.

* **Registry Service**: Netflix Eureka is a lookup server (also called a registry). All micro-services (Eureka clients) in the cluster register themselves to this server.

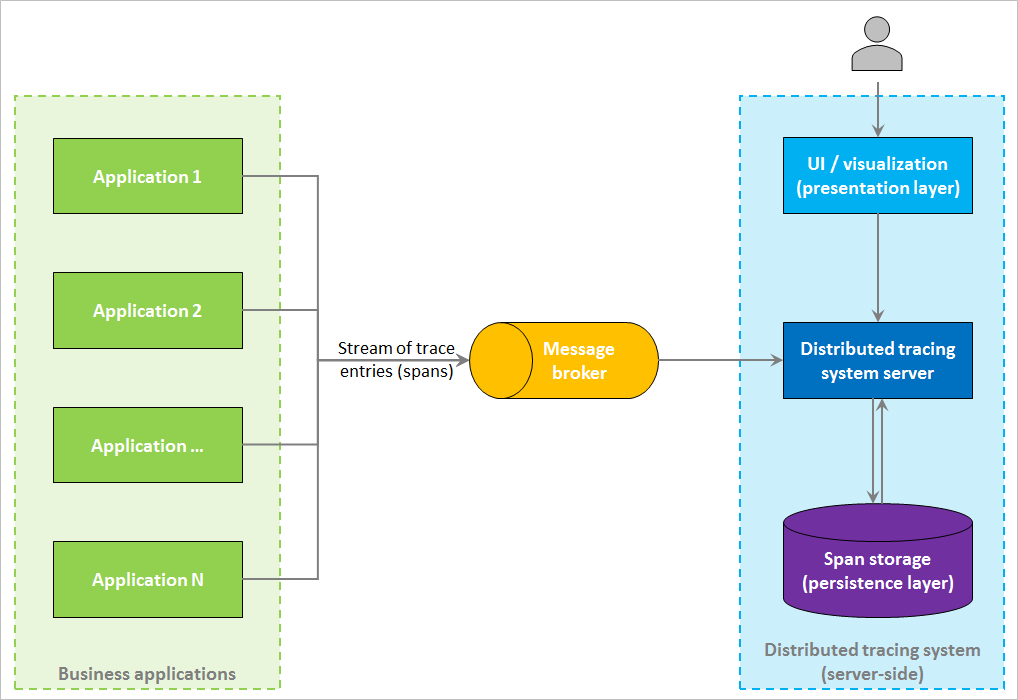


*Figure 1-3. Service registration and discovery pattern*

* **GOV Microservice**: is the major microservice is our application which performs the following operations:
* registration of a user
* login of a user
* download a declaration statement in pdf format
* download a marital status certificate in pdf format

Note that each user needs to register and then login in the application to download the above documents. The user data is recorded on a postgres database. Also, the GOV microservice calls the STATEMENT and MARITAL microservice to get extra information that is required to download a declaration statement and marital status certificate respectively.

* **STATEMENT Microservice**: is a microservice that provides additional information for the declaration statement.
* **MARITAL Microservice**: is a microservice that provides additional information for the marital status certificate.
* **Zipkin**: is a Java-based distributed tracing system to collect and look up data from distributed systems.



*Figure 1-4. Tracing System Architecture*

**Chapter 2. Set up**

For ease of the application setup process and avoid possible errors that may occur due to different specifications of the running environment, I created the required images of the microservices of our system and stored the in the docker hub. So in order to run the application, the user has to go to the root directory of the application and run the command:

**docker compose up -d**

which it will download the images of the docker compose file and then launch the containers.

**Chapter 3. Application Demo**

The steps that a user must take to download a given file are as follows:

**STEP 1:** Every new user must register first by going to:

[**http://localhost:8080/api/v1/gov/registration**](http://localhost:8080/api/v1/gov/registration)

**Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα**

*Figure 3-1. Registration Form*

**STEP 2:** Each registered user should be authenticated:

**http://localhost:8080/api/v1/gov/login**

**Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα**

*Figure 3-2. Login Form*

**STEP 3:** Then the user can choose to download a certificate of marital status or a responsible statement:

[**http://localhost:8080/api/v1/gov/options/1**](http://localhost:8080/api/v1/gov/options/1)

**Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα**

*Figure 3-3. Services Page*