KU LEUVEN

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN

CAMPUS GROEPT

DGST Final Project

Date: 16/06/2024

Wout Van Loo – r0810000
Arnoud Stas – r0837095
Joran Vleugels – r0850643
Niels Coopmans – r0837072

Academiejaar 2023-2024

This is our final report for our distributed systems project. We have successfully implemented almost everything from level 1 and managed to start with some level 2 requirements regarding the google cloud deployment of our system.

1. Distributed System Architecture and Deployment Diagram

We chose to implement a distributed car shop application that allows users to order packaged products which are cars and exhaust systems from multiple suppliers. The Microservices Architecture includes several components which are:

- **Web application**: Used by the end user to interact with the system, send HTTP requests to the server

- **The Firebase Authentication service and the Security Filter**: Handle the protection of the user and the applications infrastructure, makes sure that HTTP requests are only responded to if users are authenticated. Firebase Authentication also handles the making and storing of accounts and login users into the site.

- **Google Cloud Service**: This is where server-side code is deployed, handles incoming requests and manages requests between the server and the supplier services.

- **Firestore Database:** Stores the orders that clients have created successfully

- **Car and Exhaust Rest service:** The service created by the product suppliers. Each supplier has their way of implementing the rest service, so they are not identical with various products. To work with this the server has extensive controllers to communicate with these Supplier services.
  Both supplier services are rest services. Their operations include ordering/reserving products, checking the stock, getting all the products to display on the web application, cancelling orders and getting specific products using an ID. The REST services are also protected by an API key to prevent unauthorized access, ensuring that only known brokers can access these endpoints. The services also made use of REST Hypermedia (HATEOAS) to make it easier for other developers to find the different endpoints.

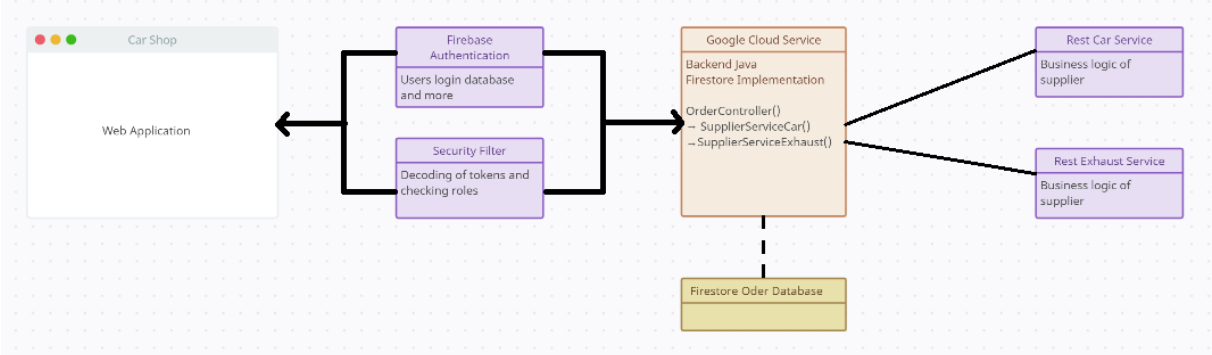

Figure 1: Distributed System Architecture and Deployment Diagram

2. Implementing Access Control for Manager Methods

To enforce and implement access control, ensuring that only authorized users can access manager methods, a role-based access control system was utilized. Users authenticated via Firebase Authentication are assigned roles, with certain users designated as managers. The system distinguishes between regular users and managers by their roles. Managers have the privilege to view all orders, leveraging their elevated permissions. In contrast, regular users are restricted to viewing only their own orders. This access control mechanism is implemented through a security filter that decodes tokens and checks user roles before granting access to specific endpoints. By assigning roles and implementing these security measures, the application ensures that only authorized users can access and perform manager-specific functions. The following code snippets show how this is implemented in our code.

```
if (user.isManager()) {
                return getAllOrders();
}

if (!user.isManager()) {
throw new AuthorizationServiceException("You are not a manager");
}
```

3. Scenarios Affecting ACID Properties in Order Implementation

When a new order is made by a client, our code keeps trying to do the order every 10 minutes. When making the order our code checks if the products are in stock, if one of them isn't, the whole order is cancelled. Normally you can't order an item that is out of stock so this should not really happen. If the stock check is positive for a car order and an exhaust order, but the car order still somehow fails, the exhaust order is not rolled back. This is not an atomic operation in this case. Use the cancel method?

4. Handling External Supplier Service Failures

In the case that an external supplier forgets to fill in fields, or the rest service messes up and doesn't send a field like “brand” or “price”, our code will filter out these products and not send them to the browser of the customer.

If the service of the supplier suddenly becomes unavailable or doesn't respond anymore will the website update and tell customers that there is a problem, and it will also keep retrying to deliver orders that were in the process of being made. Eventually our service will stop trying to order the

products that a client wants so if the supplier service does not come online fast enough, the order will not be made.

Our code does a lot of checking if items are available and if actions that the customer want to do are allowed. This means that for operations like ordering products, the full rest service of the supplier needs to be available, making sure that there are actual items in stock for the client to order.

5. Pitfalls and Restrictions of Migrating to Google Cloud Platform

You are almost forced to use their native authentication. Using any other authentication service beside firebase results into much work to achieve. It becomes unpractical to not use firebase.

6. Extent of Tie-In with Google Cloud Platform and Migration Considerations

The authentication and data storage in the broker are achieved using google provided services. These would have to be replaced by an alternative when migrating to another cloud provider both back and front end. Aside from those two is our application not tied-in with the Google Cloud Platform.

7. Deployment Details and Access Credentials

The application is deployed on Google App Engine and can be accessed using the following URL:

URL: https://car-shop-426409.ew.r.appspot.com

The following login credentials can be used:

Normal user:

Email: user@example.com

Password: example

Manager user:

Email: anothermanager@example.com

Password: admin123

8. TEAM MEMBERS' ROLES, TASKS AND TIME CONTRIBUTIONS

Name	Tasks	Total hours spend
Wout Van Loo	• Front end and JS • Authentication and signing up • Implementing ACID properties in existing code • Attempting to work with custom claims for manager only operations • Token decoding • The report • Working on handling client orders • debugging	48
Arnoud Stas	• Car rest service with functionality to list, reserve and order cars. Java classes and controllers corresponding with the Car rest service in broker application and external supplier. • Deploying the car rest service on Azure VM (Japan-east) • Implemented API key interceptor • The report • Implemented a reservation step in the order process to ensure atomicity. • Testing and reporting issues • First draft code to display cars	50
Joran Vleugels	• Exhaust rest service that can handle get requests and orders, Java classes and controllers corresponding with this • Made a logger for the Exhaust rest service • Made sure that order request got valuable http answers from the server • Implemented API key interceptor in exhaust rest service • Deploying the exhaust rest server on Azure VM (US-west) • Made sure that you are not able to order exhausts when the stock is 0	48

	• Testing & reporting problems • The report	
Niels Coopmans	• Deploying the project on google cloud • Back end: • Verifying tokens • Java classes and controllers for (order, Item, exhaust & customer) • Firebase setup for LEVEL 2 • Setting up manager users • Front end: • Displaying the items (exhaust, cars) • The shopping cart • Authentication of the user • Debugging and error fixing	58

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
CAMPUS GROEP T LEUVEN
Andreas Vesaliusstraat 13
3000 LEUVEN, België
tel. + 32 16 30 10 30
iiw.groept@kuleuven.be
www.iiw.kuleuven.be