Distributed Systems (H0N08A)

Report: Project

Oana-Eliza Alexa *(r0973974)*
Julia Szymanek *(r0975074)*

December 8, 2023

Question 1: How does using a NoSQL database, such as Cloud Firestore, affect scalability and availability?

In terms of scalability, Cloud Firestore uses horizontal scalability instead of vertical scalability. Vertical scalability means that you can add more data to already existing node, while horizontal scalability means adding new nodes at any time. Because of this, when a lot of workload is required, Cloud Firestore will add more nodes and distribute the work to speed up the tasks. Thus Cloud Firestore is highly scalable.

In terms of availability, Cloud Firestore has multiple regions where the data can be stored, so the data is replicated. In case of failure of the chosen service, data will be retrieved from one of the others. Because of this, Cloud Firestore is solving both failure issues and transaction speed. Data will be retrieved from the closest region to the client. Thus Cloud Firestore is highly available.

Question 2: How have you structured your data model (i.e. the entities and their relationships), and why in this way?

We have not changed the structure of Booking, Quote, Ticket, User and Seat but we have changed the structure of Train entity. Initially, a Train object did not have a list of train times, so loading of internal trains times and seats was slow. To solve this problem we created a TrainTime class which has an id, a train company, a train id, a time, and a list of Seat objects that share the same TrainTime. Inside Train object we have added new field: a list of LocalDateTime objects that contains all available times for that train.

Question 3: Compared to a relational database, what sort of query limitations have you faced when using the Cloud Firestore?

In Cloud Firestore you are not able to make JOINs, there are no many-to-many relationships between classes and inequality filters are limited to one field. Since data is distributed, transactions are needed for ensuring consistency, which was usually not needed in a relational database.

Question 4: How does your implementation of transactional behaviour for Level 2 compare with the all-or-nothing semantics required in Level 1 ?

For Level 2 we implemented different booking processes depending on the origin of the train company (external vs internal). For internal ones, we do not call an API to book or check availability of the seat. Therefore, we only check if the seat is not booked yet in our database. If any of the seats is already booked, we do not have to cancel rest of the internal train tickets because they had not been booked beforehand (contrary to external tickets). Only if all of the seats are available, real booking is added to our database. What is more, our addBooking method runs a transaction to ensure that all-or-nothing will be booked.

Question 5: How have you implemented your feedback channel? What triggers the feedback channel to be used, how does the client use the feedback channel and what information is sent?

To implement a feedback channel we created a new Service called EmailService. It uses SendGrid to send emails to customers after successful or unsuccessful booking. EmailService is triggered by PubSub Subscriber when it completes a booking process. Confirmation email is sent to the customer on success or on failure. It includes information about user, booking time and all the details of the tickets: train company, location, time and seat. If the Subscriber does not manage to fetch details of a given ticket (e.g. due to external server error), it sends only train company, train id and seat id for this ticket.

In practice, our SendGrid account could not be activated and it does not send any emails. However, requests sent by our application are visible on SendGrid dashboard. To see the email contents, we print them on the terminal after the moment of "sending".

Question 6: Did you make any changes to the Google App Engine configuration to improve scalability? If so, which changes did you make and why? If not, why was it not necessary and what does the default configuration achieve?

In the Google App Engine configuration we changed max_concurrent_requests to 30 and target_cpu _utilization to 0.7. Both of the parameters have been changed to reduce load on one instance and make it more efficient. After some testing, it seems like the application works faster. However, we were not able to check the performance of our application with a lot of concurrent users. Results of tests with more load probably would be more reliable in terms of scalability.

What is also worth mentioning here is the fact that loading some pages, especially bookings can take some time (sometimes it takes around 15 seconds), even after changing scaling parameters.

Question 7: What are the benefits of running an application or a service (1) as a web service instead of natively and (2) in the cloud instead of on your own server?

(1) Running an application or service as a web service is advantageous in terms of accessibility, scalability, flexibility. It can be accessed over the Internet and it often uses standardized communication protocols like HTTP and REST which promotes interoperability between different systems. Web services can be also easily scaled by adding new instances. What is more, it is characterized by flexibility because different functionalities can be developed independently as services.

(2) When you are running your application in cloud, the application becomes more reliable, if the server fails another server will take its place. Moreover, data is distributed in multiple regions so the application can access the closest service in terms of geographical distance to the user in order to speed up transactions. Cloud uses horizontal scalability, so when the application sends a lot of requests, it adds more nodes to speed up the waiting time. For all of these situations, working with your own server is harder.

Question 8: What are the pitfalls when migrating a locally developed application to a real-world cloud platform? What are the restrictions of Google Cloud Platform in this regard?

Migrating a locally developed application to a real-world cloud platform can be time consuming because a lot of settings need to be made. Moreover, a lot of data has to be transfer into the cloud database from your local one, depending on how much data exists locally. Unexpected errors might occur because of the scalability of the application. Thirdly, migrating an application to cloud implies billing, which might be high because of multiple reasons: bad management of resources, redundant data, wrong configurations. Lastly, security issues might occur.

Google Cloud Platform has a restricted quotas and limits for data transfer, but when exceeding these limits additional charges will be made. For security issues, Google Cloud Platform has Identity and Access Management (IAM) where you can specify and attribute roles. For the billing problem, Google Cloud Platform let you see how much money you have spent on each service, and you can set alerts for safety.

Question 9: How extensive is the tie-in of your application with Google Cloud Platform? Which changes do you deem necessary for migrating to another cloud provider?

Our application is not very tie-in with Google Cloud Platform, but a few changes would be required if this decision is made. Firstly, the repositories will need to work with different kind of databases, now they use Firestore interface and use its methods. Secondly, for the Pub-Sub the cloud endpoint will be different, but the implementation is the same. Lastly, the verification of the token will be different, since now we use the fact that it is a Google signed token.

Question 10: What is the estimated monthly cost if you expect 1000 customers every day? How would this cost scale when your customer base would grow to one million users a day? Can you think of some ways to reduce this cost?

The total costs is split in the table below based on number of customers per day and services. The estimations were made considering that each user will make 2 bookings per made and each booking consists of 5 seats.

	App Engine	**Cloud Pub-Sub**	**Firestore**	**Total Cost**
1000 users/day	€50	€9	€22	€81
1 million users/day	€44000	€9300	€23000	€76300

Costs might be reduced if the App Engine is modified: instance class could be changed along with the scaling type and the number of max instances. Also, the location of these services are important, prices might change.

Question 11: Include the App Engine URL where your application is deployed. Make sure that you keep your application deployed until after you have received your grades.

Our application is deployed at `https://train-companies-ds.ew.r.appspot.com`