Distributed Systems (H0N08A)

Report: Project Part 2

Thibaut Mertens *(r0762702)*
Ignasi Juez Guirao *(r0974979)*

January 31, 2024

Important note We noticed from the feedback session that we made a mistake with fixing the inconsistencies when there is a crash. For the first part of the project we made a function fixInconsistencies which gets called everytime the server starts up. This was done by having a bookingTemp collection in the database, which was also confirmed as a good solution by the assisstants. This also works completly fine, but the pub/sub will also keep retrying until it gets a 2xx response. As a result one of them will fail when retrying. Because we didn't see the problem at first we now have some unintended behaviour. For example when sending a mail after a crash it will probably both send a mail that the booking succeeded and failed. Because we noticed the problem so late we decided not too fix it because it would take too much time and potentially bring in a lot of other problems when trying to do it quickly. It was also more a part of level 1 and hope this won't be taken into account for the second part. A better way to implement this would be just changing the subscription to also check if some seats are already reserved by the same customer.

Question 11 NoSQL databases are designed for horizontal scalability and they can easily scale out by adding more servers or nodes to handle increased load. High availability is achieved with data replicated across multiple regions and zones, as we decided to use multi-regional Firestore. In addition to data scalability, NoSQL databases often provide scalable authentication and authorization mechanisms. As Google Cloud states as his first benefit of using Firestore: "*Effortlessly scale to meet unpredictable demand: Firestore automatically scales up and down based on demand. It requires no maintenance, and provides high availability of 99.99–99.999% achieved through strongly consistent data replication.*"

Question 12 The data for the internal TrainCompany is structured in a collection "InternalTrainCompany". In this collection there are documents of trains with their train id as the document name. Each train document contains 5 fields (image, location, name, trainCompany and trainId) and a collection "Seats". This collection has each seat as a document with their id as the document name. The reason for this is that it is easy to query a seat and we can only fetch the required seats. Each seat document contains the following fields: name, price, seatId, time, trainCompany, trainId, type and available. Available determines if the seat is already booked or not.

Question 13 One way we are limited in with the Cloud Firestore is the way we store data because queries can't span accross multiple collections. Another way is that it is way more complex to get number of tickets for each customer which would be possible with one query in a relational database.

Question 14 Once we could implement transactional behaviour for our internal TrainCompany there was no need for our rollback implementation made in part1 for the internal TrainCompany since the transaction functionality achieves the same goal, atomicity. It should be emphasized that the rollback mechanism is still active for non internal TrainCompanies because we have no database access.

Question 15 Our feedback channel was implemented with SendGrid which help us to send custom mails to our customers. With SendGrid product and a specific class in our project we were able to notify customers of a successful completion or failure of a new booking request. Customers that had a successful booking will receive the information of their tickets, TrainCompany and ticketId, and customers that the completion of the booking was unsuccessful will receive an informative text about it.

Question 16 We did not change anything to the configuration to improve scalability. It should be changed when there is a better idea of how much traffic it experiences. If there are a lot of users the amount of concurrent requests can be enhanced. Further changes might be necessary when specific performance or scaling issues are observed.

Question 17 (1) There are several benefits for running an application or service as a web service. It allows for accessibility where users can access using only a web browser and remotely from different locations, cross-platform compatibility, convenient updates where updates can be applied on the server side without requiring any user interaction, scalability and enhanced security measures as encryption and authentications. (2) And the benefits for running an application or service in the cloud offers advantages in terms of scalability and cost efficiency as cloud platforms provide the ability to scale resources up or down based on demand with the pay-as-you-go model that is so attractive for all sized business, flexibility, a broader reach since cloud has a lot of data centers around the globe providing a closer connection to customers, it also provides reliability and security.

Question 18 The pitfalls that we have experienced in our transition from local to cloud were: We had to think about or data model to make it appropiate for the scalability of the cloud. The data transfer from locally to the cloud may take a while also at first so that can be time-consuming and involve costs. Local applications may not be designed to be deployed on cloud so we had to adapt the application functionalities to the cloud specification and API. That includes everything from features like pub/sub to security considerations. Last but not least, the cloud is billed based on usage, where costs can rise very quickly, so optimizing configuration and setting budgets & alerts.

Question 19 The extent of our tie-in with Google Cloud Platform depends on various factors, including the architecture and design choices made during the development and deployment to GPC. Where the pub/sub implementation, Firestore database, how we have adapted our application to be able to import data from local to the cloud infrastructure, transactional operations and authentication methods were made based on GCP documentation. Which may differ from other cloud providers.
Migrating from GCP to another cloud provider would make us consider the following changes: An infrastructure transformation would be the first thing since the foundation of our cloud environment is our infrastructure, storage, and networking components.
An application adaptation of our cloud-based applications which may need modifications to work correctly with the new cloud platform, the new API and libraries.
The data might take an important place as well where we would transfer all of our data in a safe way, but the database may be adapted to the new provider.
Connections, authentication, and security configurations may also need to be adapted to the new cloud provider.

Question 20 With the help of cost estimation tool on Google Cloud console in multi-region location type, assuming that each user generates an average of 1MB of data per day, a 1000-user application would require approximately 1GB of storage per day. This would translate to around 30GB of storage per month. And assuming that each user performs an average of 10 operations per day, a 1000-user application would generate approximately 10,000 operations per day. This would translate to around 300,000 operations per month. Which gives us a Workload estimate of $3.78/month. The scalation for a million users cost would be Workload estimate$783,000.00/month. We would think about optimizing our infrastructure, negotiating with our cloud provider for a deal, monitoring the usage to identify spikes or anomalies, optimize the number of Compute Engine instances and the amount of resources used by each instance, use Cloud Storage's lifecycle management features to automatically archive or delete data that is no longer needed and use BigQuery's partition and cluster features to improve query performance and reduce costs.

Question 21 https://distributed-systems-406415.ew.r.appspot.com/