Qub

Conor Barry

19385371

Daniel McCarthy

19350093

Dawid Tkaczyk

19386321

Marvin Schmid

22208886

Synopsis:

Our system is a hotel application like AirBnB or Booking. The user should be able to find hotels for a given timeframe, location and persons. Moreover the user should be able to get detailed information about a hotel such as price, description, pictures, user reviews. Furthermore the user should be able to rate different hotels.

The application domain is within the hotel market sector. The good thing about it is that it should be able to be extendable, so that new functionalities can be added easily.

Technology Stack

List of the main distribution technologies you will use

- *Ionic v7 with Angular Standalone Components for the Frontend*
- *MongoDB*
- *MySQL*
- *Netflix Eureka?*
- *Spring Boot*
- *Swagger: documentation of the API endpoints from all services*
- *Compodoc: documentation of the Angular Frontend*

Highlight why you used the chosen set of technologies and what was it about each technology that made you want to use it.

Swagger: We used swagger to document our API endpoints. We choose that technology, because it provides a useful user interface to explore and test the api endpoints. We want to use it within every service. A nice future addon would be to combine all documentation.

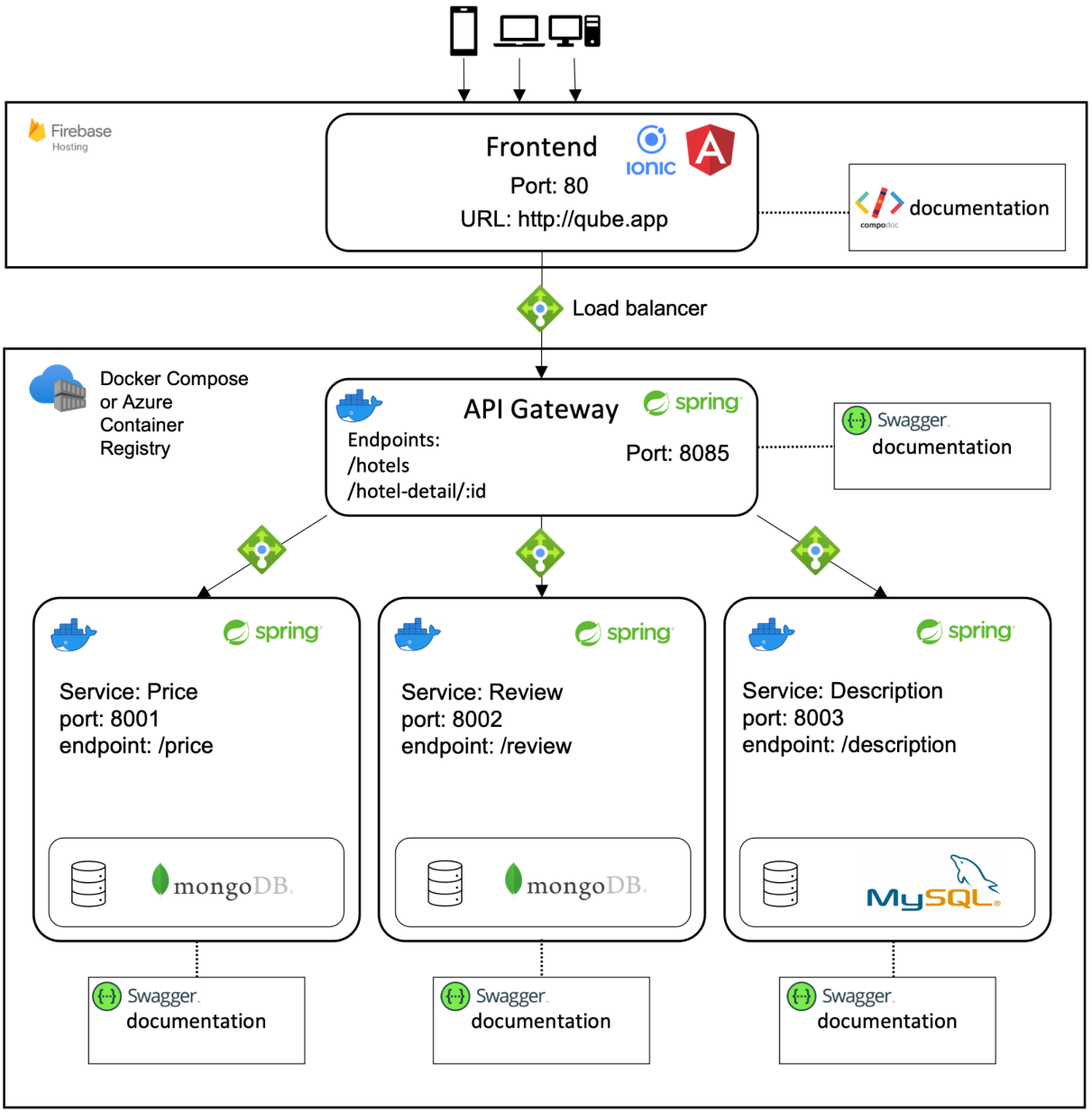
Compodoc: We used Compodoc because it helps with creating documentation for Angular Applications. This is especially useful when you don't use a standalone component, because it can generate relationship and dependency diagrams between your components. Why we used it: “Good code doesn’t document itself”.

Ionic 7: We selected Ionic 7 as our framework. One of its new notable features is the inclusion of Inline Overlays. These overlays, such as Modals, Popovers, Action Sheets, Alerts, Loadings, Pickers, and Toasts, can now be easily incorporated into application templates using a declarative approach. This means that data can be passed as properties without requiring a separate controller, resulting in simplified code. Moreover, Ionic 7 introduces new properties like isOpen and trigger, which facilitate the presentation and dismissal of overlays. Another enhancement in Ionic 7 is the Simplified Form Control Syntax. Working with form controls, such as Toggles or Inputs, has become more streamlined. The requirement for Item and Label components has been eliminated, and each form control now directly handles the label content. Ionic Angular can also anticipate improved component initialization times. The general improvements are described in the system overview.

MongoDB and MySQL: The use of MongoDB and MySQL as the databases in our framework provides data persistence for each service in case of container failure or restart and yields scalability in terms of data storage. The separation of storage between these databases also makes the system more secure as the compromise of the data for one service does not mean the compromise of all. Connection to the databases is independent of service deployment which allows testing and hosting on a number of different systems.

System Overview

Describe the main components of your system.



Frontend: The combination of Ionic and Angular provides features such as native mobile functionality, cross-platform support and good integration with our backend Gateway. In terms of distribution and scalability we used angular standalone components. Angular standalone components offer advantages such as reusability, encapsulation, testability, scalability, flexibility, and improved performance. They are self-contained modules of code that can be used multiple times, making it easier to manage and maintain the codebase, and they offer better performance by reducing code load times. Scalability and fault tolerance was therefore an integral part of the frontend from the beginning.

Deployment: We focused on a cloud native deployment. Therefore we dockerized our different services and the gateway. Docker simplifies the distribution of applications by encapsulating within a separate container.We decided that we could use Microsoft Azure Load Balancer, as an public load balancer to handle the public ip port as well as an internal

load balancer for our hidden services. The advantage of this approach is that we increase the availability, can balance on multiple ip addresses and also monitor our traffic and application health with it. It could also serve as a security barrier that could protect DDOS attacks. When an azure container registry would detect a failure of a docker container, it can automatically restart the container which could lower the downtime and increase the fault tolerance at the same time.

The review module is a dockerized RESTful service which stores and permits adding of hotel ratings/reviews into an external online MongoDB store instance and keeps up to date the average user star rating value for each hotel. It features retrieval of data through the use of POST requests by the gateway which it combines with description and price information retrieved from the other services, thereby providing security and modularity to the system. The use of an online MongoDB store provides data persistence incase of container restart or failure and allows connection from a dockerized environment.

The descriptions module is also a dockerized RESTful service which permits the storage and retrieval of descriptions for each hotel using a MySQL database. The authentication details for the database are stored in an .env file to separate sensitive information from the codebase.Docker allows the entire description service and a MySQL environment to be packaged , including its dependencies and configurations, into a single container. Through the use of a volume, data persistence is provided in the event of a container restart.

The price module is, like the other modules, a dockerized RESTful service, which controls everything to do with booking a hotel room. It is stored on port 8081, using endpoints /price and /price/{id}. The booking of a hotel room is achieved through a POST request to /price, creating a “Booking”, which gets stored in a MongoDB database. It also computes the price of booking the room for the specified number of guests for the specified number of nights. It is also possible to view previously made bookings through a GET request to /price.

Contributions

Daniel: Price Service, documentation

Marvin: Frontend, recording, documentation.

Conor: Descriptions service, documentation.

Dawid: Review service, documentation.

Reflections

One of the major challenges we faced was integrating the individual services into a cohesive application. The differences in programming languages, frameworks, and architectures posed a significant hurdle. To overcome this challenge, we focused on effective communication, coordinating our efforts, and meticulous planning. Regular meetings and clear communication channels ensured that everyone understood the integration requirements, and we worked together to devise a seamless integration strategy. We decided on compatible interfaces and assigned each service its own port and hostname.

Time management was another significant challenge. Coordinating the work of multiple team members, juggling individual deadlines, and ensuring timely completion of each service wasn’t easy. To tackle this, we adopted agile project management techniques, such as

kanban, which helped us track progress, set realistic milestones, and adapt to changing circumstances. We used Trello as a free tool to organise this. By implementing effective time management strategies, we were able to keep the project on track, although it was sometimes really hard because of the exam week.

Reflecting on our project, there are a few things we would have done differently if we could start again. First and foremost, we would have invested more time in the planning phase. By conducting detailed architecture design and clearly defining interfaces and communication protocols from the outset, we could have minimised integration issues and improved project efficiency. But also take more time in general. Because the limited time during exam week was the most important factor holding us back from our and the applications' full potential.

In terms of the technologies we used, this project provided us with invaluable learning experiences. We gained a deeper understanding of various technologies and frameworks by working with these. We also learned a lot about distributed systems in general and the different approaches used to ensure scalability and fault tolerance as an integral part of the system design. The project also highlighted the complexities involved in integrating different software components. We learned valuable techniques such as API design, message passing, and data serialisation to overcome challenges related to interoperability, data sharing, and communication. These insights will be invaluable in future projects that involve integrating diverse software components.

We also learned about the limitations and trade-offs associated with different technologies. Each language or framework has its strengths and weaknesses. For example, the new ionic 7 doesn’t allow importing global dependencies and therefore no external libraries like “HTTP Client” for API Communication in a service. These limitations sometimes took a lot of time to fix.

We also had some issues at the start because our databases, especially the id’s weren’t consistent for e.g. hotel1, hotel2,...

docker/microservices allows us to use scalability and fault tolerance. We could use kubernetes clusters to scale our service over the world with different locations to minimise response latency and we could also restart containers if they fail automatically and with logging tools like NewRelic we could monitor our application status and logs, so that we know when an error occurs in multiple instances.