Distributed Systems (H0N08A)

Report: Project

Andi Škrgat *(r0876363)*
Daniel Rey Šparemblek *(r0883565)*

November 26, 2021

1. Imagine you were to deploy your application to a real cloud environment, so not a lab deployment where everything runs on the same machine. Which hosts/systems would then execute which processes, i.e., how are the remote objects distributed over hosts? Clearly outline which parts belong to the front and back end, and annotate with relevant services. Create a component/deployment diagram to illustrate this: highlight where the client(s) and server(s) are.

The remote objects are distributed in a well organized fashion. The App on the front end is connected to the PubSub on the back-end which is called by the Worker class. Firebase authentication is hosted on a seperate Google server. The objects of TheatreService and handled through direct communication

2. Where in your application were you able to leverage middleware to hide complexity and speed up development?

Spring Security is a powerful framework that we use for authentication and authorization. It allows customizing role-based access control; practically all complex behavior can be defined within two classes. Building rest controllers is also trivial with Spring Boot so add-ons like adding cookies support, specifying which HTTP method will be used, and redirecting users are easily done. We would also like to point out Web Client and its easy configuration for hypermedia API.

3. At which step of the booking workflow (create quote, collect cart, create booking) would the indirect communication between objects or components kick in? Describe the steps that cause the indirect communication, and what happens afterwards. We think that the most beneficial would be to introduce indirect communication when creating bookings because this is the moment when quotes have to be saved to persistent storage which is a slow operation. Creating quotes and collecting carts are both computational tasks that can be executed quite fast. Indirect communication is done in the following steps:
1. When the application is started, a new topic is created for all messages related to confirming a cart.
2. Push subscription is created - Pub/Sub broker will send all messages to link /pubsub
3. Sometime during the application's lifetime, the client confirms the cart by executing a POST request on route /confirmCart.
4. The message is written to persistent storage.
5. ThePub/Sub broker sends a message to its subscribers
6. Once the message is acknowledged, it is deleted from persistent storage.

4. Which kind of data is passed between the application and the background worker when creating a booking? Does it make sense to persist data and only pass references to that data? A list of quotes is passed, together with information about the customer. In a real-world application, it would make sense to persist the data and pass the references to it in

order to keep the data backed up in case of an outage. Writing important data on volatile storage can be considered harmful.

5. How does your solution to indirect communication improve scalability and responsiveness (especially in a distributed/cloud context)? In a real-world scenario, there would be thousands of time-consuming requests from different parts of the world. Our system has to be capable of handling a high volume and spread of users so the load can be distributed as equally as possible. If we would use direct communication, this can be quite a big overhead, and the Pub/Sub broker solves that by using a broker which is capable of choosing channels without unnecessarily overloading nodes. This way, users are always provided with the last updated information.

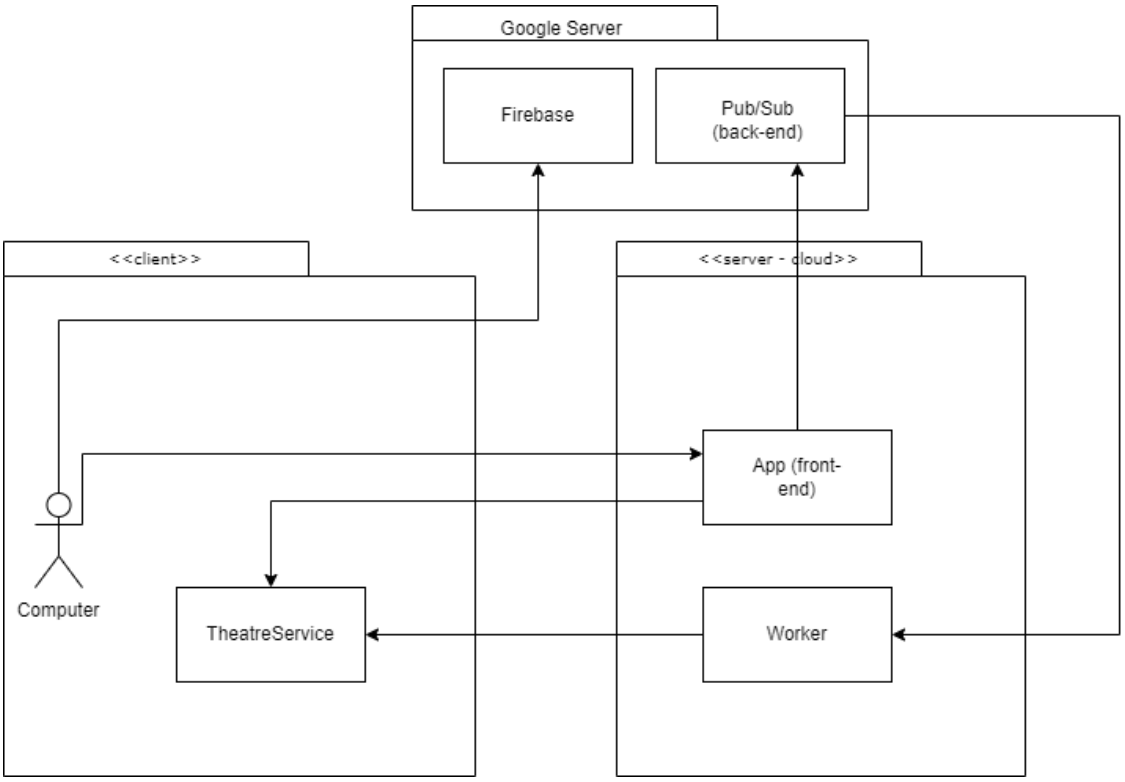
6. Can you give an example of a user action where indirect communication would not be beneficial in terms of the trade-off between scalability and usability? One of the greatest advantages of a pub/sub system is the ability to handle a high volume of data by directly on-demand scaling. However, sometimes it is simply unnecessary to use for some smaller operations so brokers could focus on scheduling more important tasks and subscribers on solving. For example, a user accessing his account is a trivial, simple-scale operation that will not influence the overloading of some server node. Using indirect communication for such operations could lead to system instability and network saturation.

7. Is there a scenario in which your implementation of all-or-nothing semantics may lead to double bookings of one seat? One seat cannot be booked twice in our implementation due to the fact that our Model class continues to other functions after the first booked seat. The better check happens in PubSub since it's checking that for every message, another message didn't arrive with the same ID.

8. How does role-based access control simplify the requirement that only authorized users can access manager methods? It simplifies it in the way that it is enough to specify a claimed role, which can be done in the Firebase emulator, and read it in doFilterInternal. By using role access it is enough to specify role permissions; and not for each user separately, which greatly reduces the time needed for adding a user or eg. changing the logic of giving access.

9. Which components would need to change if you switch to another authentication provider? Where may such a change make it more/less difficult to correctly enforce access control rules, and what would an authentication provider therefore ideally provide? Currently, we use the Firebase authentication provider. Some of the alternatives are Backendless, Amazon Cognito, Okta, Azure AD... If we would switch to another authentication provider we would have to change the login component of our system, which is currently called in the Firebase backend, and the code where we set the authentication context that Firebase Authentication uses. An authentication provider would ideally be able way to easily configure claims, which is a core idea when using access control rules. It should also be compatible and easy to set up with Spring-Boot middleware, specifically Spring Security module.

10. How does your application cope with failures of the Unreliable Theatre company? How severely faulty may that company become before there is a significant impact on the functionality of your application? It copes with well with any kinds of failures. It tries a REST request five times and if it doesn't succeed, it'll just return an empty list for that specific request, ie. time, show, seat, etc. No severe faults can significantly impact the functionality.


Google Server
Firebase
Pub/Sub
(back-end)
<<client>>
<<server - cloud>>
App (front-
end)
Computer
TheatreService
Worker