Distributed Systems (H0N08A)

Report: Developing a locally deployed distributed application

Triet Ngo *(r0869104)*
Stijn Martens *(r0855156)*

November 25, 2021

Question 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.

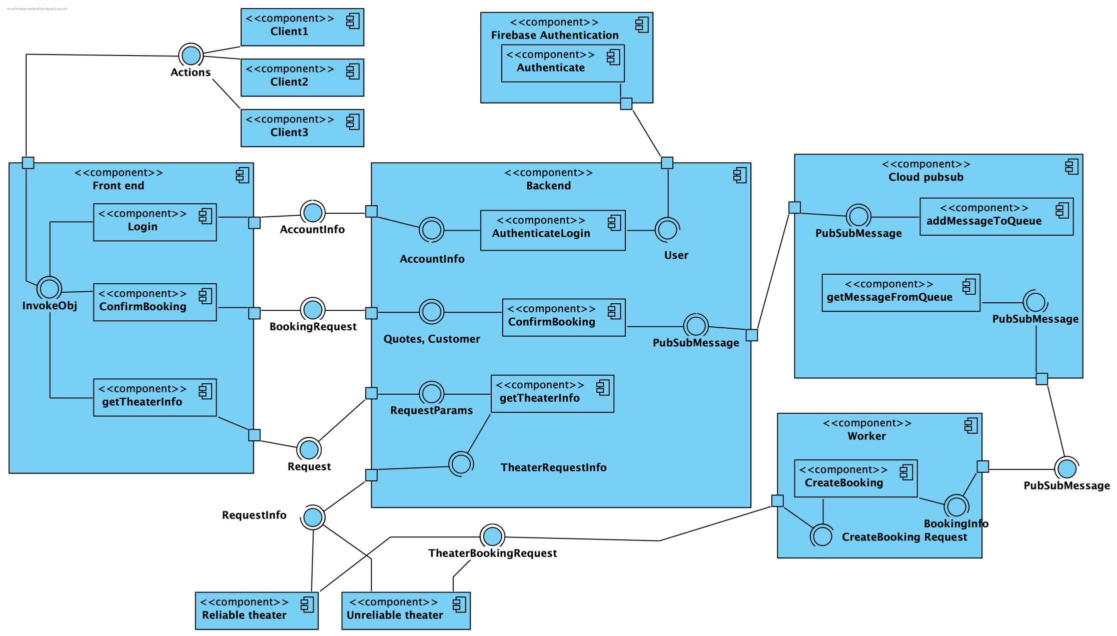
<<component>> Client1
<<component>> Client2
<<component>> Client3
Actions
<<component>> Firebase Authentication
<<component>> Authenticate
<<component>> Front end
<<component>> Login
<<component>> ConfirmBooking
<<component>> getTheaterInfo
InvokeObj
AccountInfo
BookingRequest
Request
<<component>> Backend
<<component>> AuthenticateLogin
User
AccountInfo
Quotes, Customer
<<component>> ConfirmBooking
PubSubMessage
RequestParams
<<component>> getTheaterInfo
TheaterRequestInfo
<<component>> Cloud pubsub
PubSubMessage
<<component>> addMessageToQueue
<<component>> getMessageFromQueue
PubSubMessage
<<component>> Worker
<<component>> CreateBooking
BookingInfo
CreateBooking Request
PubSubMessage
RequestInfo
TheaterBookingRequest
<<component>> Reliable theater
<<component>> Unreliable theater

Figure 1: Component diagram for distributed cloud theater booking app

Diagram explanation: Clients interact with the application by loading the website (front end) into their browser. Authentication is handled by the Firebase Auth middleware which provides an UI to login/register and stores the user credentials. When a user sends a normal request, it will be received in the back end and if it is necessary, requests will be send to the theatre companies for extra information. We only use indirect communication for confirm booking. Therefore, only on a BookingRequest a message will be published by the back end, a subscriber will receive it and request the theatre companies to confirm the booking.

Question 2: Where in your application were you able to leverage middleware to hide complexity and speed up development?
Firebase Auth is used for the authentication of users. Firebase provides us with an UI where the users can login or create an account. It stores the user authentication credentials in its own remote database, so we also don't need to implement our own database and find out how we can store the information safely. We can also link the Firebase authentication to our Spring framework, so we can use the information provided by Firebase to easily limit parts of our website to users who are authenticated and/or have a certain role.

Question 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.
At the step where we create a booking. When a POST request is received to confirm the cart and create a booking, we publish a message to the publisher endpoint (http://localhost:8080/confirm-quote). In the setup of the application a subscriber was created that listens to the publisher endpoint. When a message is received, the background worker will call a method that tries to confirm the quotes by interacting with the two theaters with their APIs.

Question 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?
We passed the list of quotes that need to be confirmed and the customer id as data to the background worker. In the confirm booking process, the quotes are used as a cart representation. When the cart is confirmed, a booking will be created in the database contains all the information of the quotes. Therefore, the booking object makes the quotes redundant. In other words, quotes are used as representation object in communication (If we save it in database, we would need to delete it after create booking). Therefore, It is unnecessary to persist such object.

Question 5: How does your solution to indirect communication improve scalability and responsiveness (especially in a distributed/cloud context)?
We can scale up the amount of publishers and subscribers. This allows more requests to be processed simultaneously, thus multiple users can create bookings at the same time. This increases the scalability, but also the responsiveness towards the user, who needs to wait less before their booking is processed.

Question 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?
The add or remove from cart operation doesn't require any interaction with a theatre company. It is only necessary to update the cookie of the user containing the quote information. Using indirect communication here would more overhead than the perceived benefits. Furthermore, the requests that are related to getting information from the theater would not be beneficial from indirect communication either. For example, when the user loads the movies page from all theaters, it is unnecessary to use pub/sub since the data from the theater are required to make up the page. It would not be ideal and user-friendly for the user to actively refresh the page to see whether a worker processed their request or not.

Question 7: Is there a scenario in which your implementation of all-or-nothing semantics may lead to double bookings of one seat?
It is not possible because when a request to create a booking is called, for each quote a request will be send to the corresponding theatre company. If two quotes for the same seat, the same show on the same date are trying to get confirmed, the one that is processed first will receive the ticket. The other one will fail and all quotes that are already confirmed for the booking will be reverted.

Question 8: How does role-based access control simplify the requirement that only authorized users can access manager methods?
When a user tries to access a part of the site, we just need to check if he has the right role/the role has the right access privileges, which are granted after login and stay in cookies/session information. Otherwise we would need to look up the user in a database and check if he has the right access rights for each interaction with the website.

Question 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?
In the case of Spring boot, we need to provide an implementation of the interface Authentication which contains the following methods:

- getAuthorities
- getCredentials
- getDetails
- getPrincipal
- isAuthenticated
- setAuthenticated

Therefore, if we use another authentication provider, the provider must have a way to extract the above information so Spring boot Security can work. The developer will have to implement an instance of the Spring boot Authentication interface. In this way, regarding of what authentication provider the user uses, spring boot will ensure that it still works as intended (correctly enforce access control rules). The change of authentication provider will be more difficult if we cannot gain that information easily.

Question 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?
If a request fails, we try again after two seconds. This will repeat for ten times. Afterwards, it really fails. We think that after this amount of retries, there will be a significant impact on the responsiveness of the application and the contacted theatre company must be severely overloaded or down.