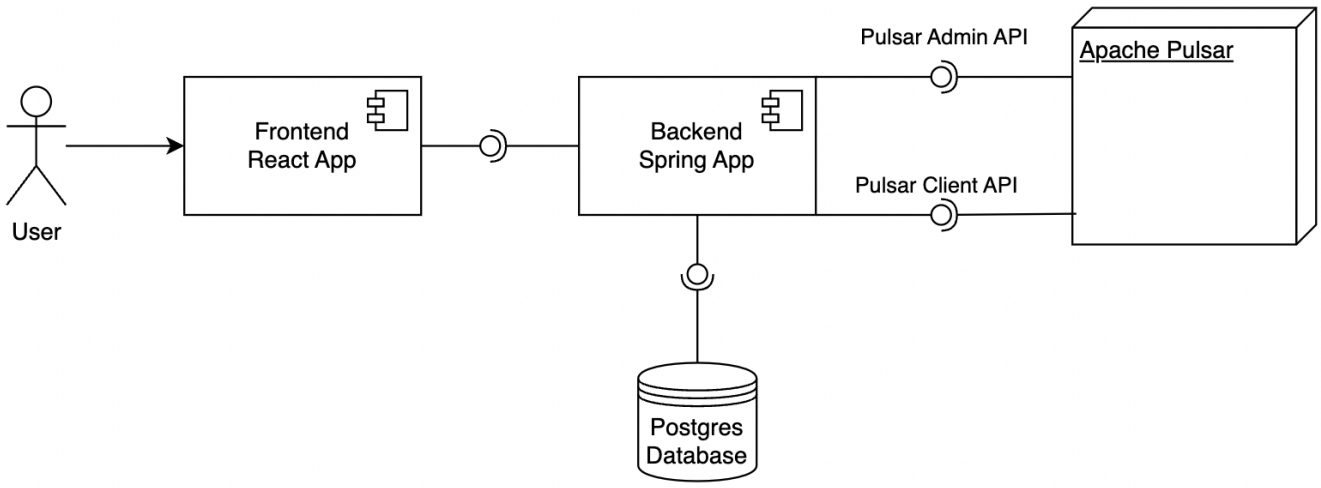
# Runtime Components



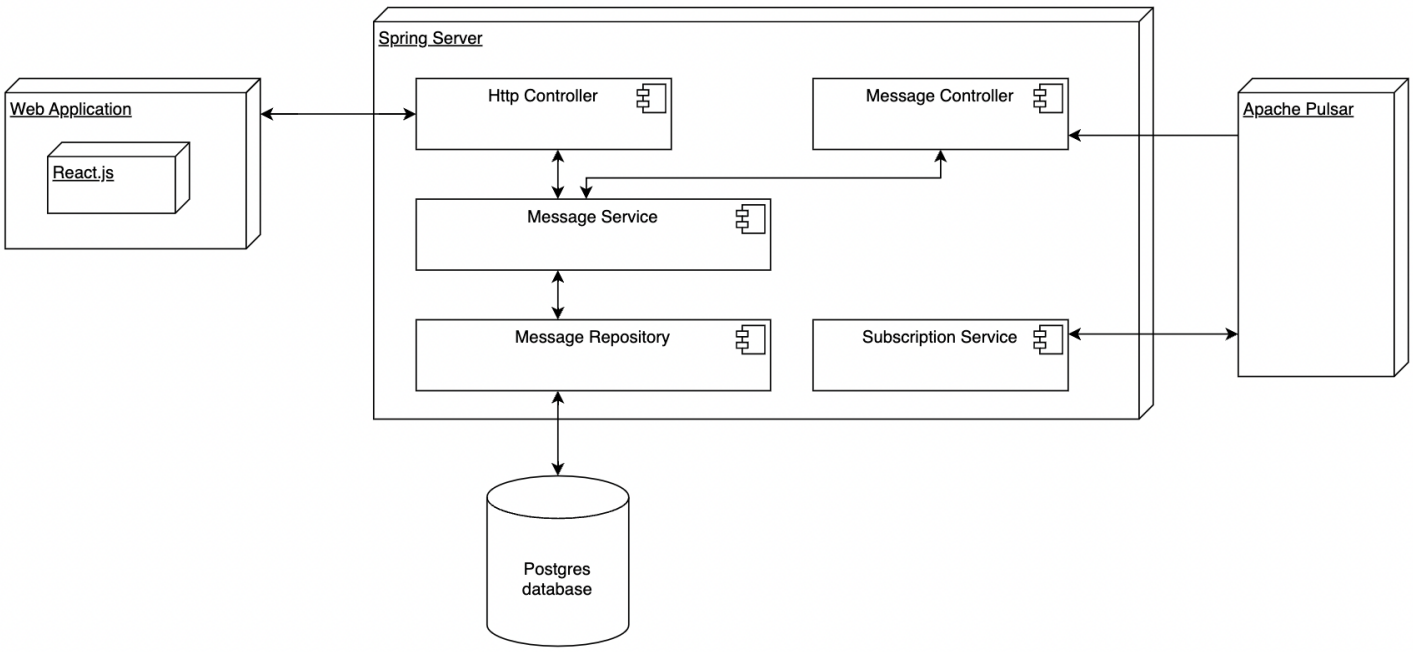

## Description

The runtime components diagram is fairly simple at the moment because the developers still need some time to iron out the details. Nonetheless, it is clear that our frontend React App will to communicate with the backend Spring App, which in turn will act as a “proxy” for querying instance information from Apache Pulsar. Lastly, this information will be displayed in the client to our users and remain updated thanks to the PostgresSQL database.

A particularly challenging part of the process will definitely be using the Pulsar Admin API to receive the necessary instance-related information in the correct format. Additionally, based on the requirements defined by the company partner, the data traveling between frontend, backend, and Apache Pulsar needs to be dealt with extreme care, so that the information displayed to the user, that stored in the database, and that coming from Pulsar remain truthful.

While the core of the architecture will most likely remain intact, it may be possible that, after experimenting with the Pulsar Admin API, and gaining more experience with the involved technologies, some minor changes might occur during the upcoming weeks.

## Code components




## Description

Our current idea for a rough composition of the frontend and the backend of our application can be seen in the diagram. The web application poses as a Client and can request information (messages) from the backend.

As agreed with our partner company, to develop the User Interface (in other words, what the users will see) and make the requests to the Spring Server we will adopt the React.js framework/library. The requests will be handled by the HTTP-Controller that, after receiving a request, will read the stored messages from the Message Service and send the information back to the Client.

The Message Service uses the Message Repository to read and store information in the database. The reason behind this type of architecture was to apply the “separation of concerns” principle.

On the right side of the Service Server container we highlighted the services that interact with the Apache Pulsar instances. More precisely, the Subscription Service subscribes to all topics on a particular Apache Pulsar instance, so that we will be notified on new messages in the stream. On the other hand, the Message Controller reacts to published messages and stores them via the Message Service.

## Technology Stack

Programming languages: Typescript (Frontend), Java (Backend)

Frontend:


React.js


Tailwind + SCSS


Axios


Jest


Material UI


Redux Toolkit

Backend:


Spring


PostgresSQL


Apache Pulsar


JUnit

Code:


Prettier


Github


Lombok

Design:


Figma

## Description

The rationale behind this technology stack comes from each developer's individual strengths. We tried as much as possible to let everyone work with software they were already comfortable with. Nonetheless, we also encouraged each other to experiment with new libraries, frameworks, and technologies (f.e. Tailwind and Jest in the frontend).

As stated previously, in the frontend there is a React.js application. By using SCSS, Tailwind, and Material UI, we will create and style our main components. In addition, to manage the core application logic, we will make use of the features provided by the Redux Toolkit. Next, Axios will be the HTTP client for the application. As for the backend technologies, we already explained the roles of Spring, PostgresSQL, and Apache Pulsar in the "Code Components" section. For testing purposes, we will use Jest (for React Components) and JUnit (for Java code).

Our code will be maintained through Github and formatted through Prettier. In addition, to optimize Java code, we will adopt Lombok. Lastly, we will use Figma to design the necessary user interfaces and wireframes.