Chattie

Mobile Dev App

Michal Stokluska

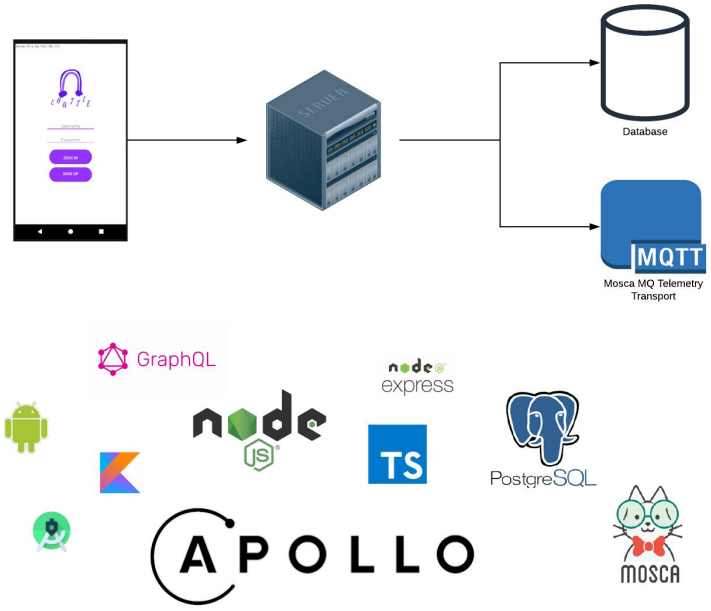
IT 3 - 20079174

21st April, 2020

Introduction

Chattie is an application that I had created for my Mobile App Development and Automated Cloud Services projects in year 3 of Information Technology. It is a very simple and plain communication application that has some of the functionalities of apps like Viber, WhatsApp and so on. It allows for a real-time one to one communication between potentially thousands of devices.

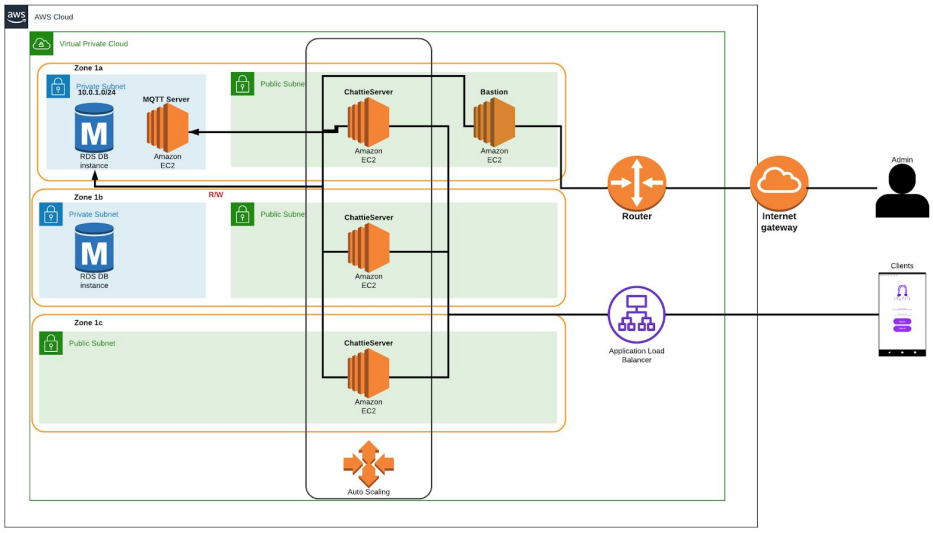
The Idea



The idea of chat application started developing in my head years ago, where I didn't even know what programming in java looked like but I felt that coding might be an interesting thing to do for living. I've watched many youtube videos but there was one that I liked the most, where an experienced programmer explained why every programmer should do a communication app at least once in their career. I promised myself back then that one day, I will do a communication app too. Didn't know how, but all that cleared out during my internship in Red Hat. I was exposed to TypeScript, PostgreSQL, NodeJS and many more but most importantly, Apollo and GraphQL. After my internship I was able to build a GraphQL server and I knew what it is capable of. In Red Hat I've built a server for a web application, and a simple Android app using Java. Android app, although worked, wasn't build correctly so when I found out that I'm going to have a mobile app module where I will learn about android app development - I thought, why not combine all of what I know and make one fully working app from scratch, back end and front end, cloud implementation etc.

So that's how I came up with Chattie.

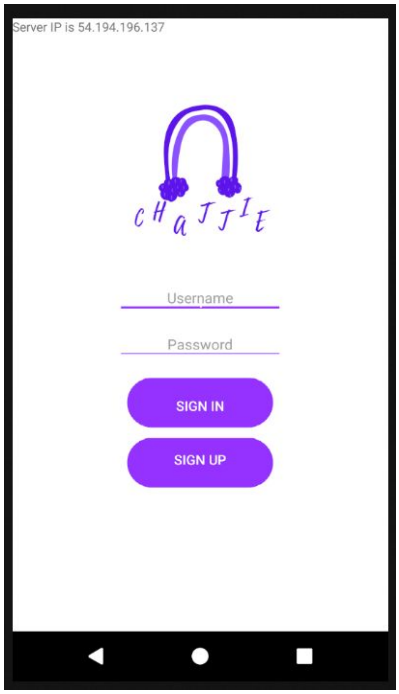
Architecture



While application can be run on any android device or emulator, server runs on Amazon EC2 instance, that EC2 instance is linked with Amazon RDS for PostgreSQL support and another EC2 instance for MQTT support.

A very brief look at the architecture, however, if you would like to know more about the architecture I can make my Automated Cloud Services report fully available to you, the report describes the above picture in detail. Autoscaling rules, security groups, load balancing and many more.

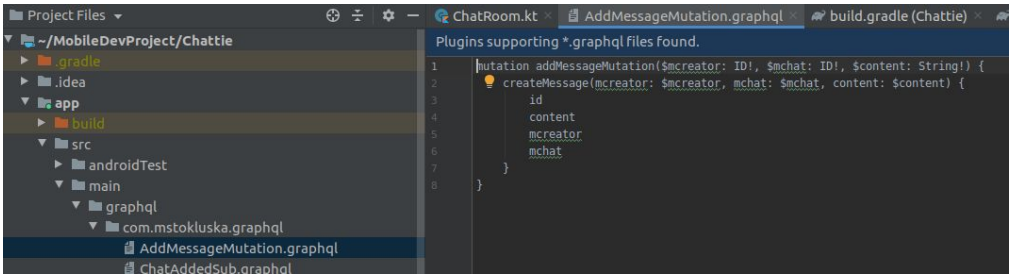
Front End



Chattie consists of few activities and 3 different data models, one for chats, one for users and one for chat rooms.

It uses the Apollo schema downloaded from exposed by the server endpoint. Schema was downloaded using apollo codegen and it contains the data model, list of queries (GET) mutations (POST, UPDATE, DELETE) and subscriptions that can be available to user.

Then based on schema I am constructing how does my graphql call looks like and what it returns:



And finally, Apollo auto generated classes from all the .graphql files, so that I can use them in my project to make calls to the GraphQL server that I got the schema from!

Back End

As I know this project isn't about back end, a very quick overview of the server:

It is a Express NodeJS server wrapped by Apollo Server written in TypeScript that uses GraphQL technology. All queries, mutations and subscriptions are described in schema file.

Logic for the queries, mutations and subscriptions is covered by the resolvers, which make calls to PostgreSQL for persistence. I am using docker-compose locally for PostgreSQL and Mosqua MQTT broker.

MQTT messaging broker is used to handle subscriptions from more than one server (which happens during auto scaling on AWS).

Server exposes port 4000 or another port can be provided by environment variables.

The Good

There's a lot of good stuff about this project that I love, the architecture, the complexity, scalability and load balancing. I think my android project structure is clear as well, I am trying to use res values where I can, I finally understand (considering its done right :)) the adapters and listeners. I'm very happy with the way subscriptions are working with adapters, figuring out how to change colors if the message is mine or not was a big deal to me. I think I got a good understanding now of how Android projects are structured. I am also happy with the activities and graphical parts of it ie roundy buttons, menu icons, the drop down menu for logout / edit and the flow in general. Parcels are very cool also, however, they are so cool that I've realized at some point that I don't need to use them that much! For current users for example, instead I could just use the global currentUser variable. There's been a lot of small wins that I had during the development of the app which I am very proud of. Of course, I've used a lot of google, stackoverflow, apollo docs and a lot of other documentations.

The Bad

Some stuff I am not a fan of, and I know that in a few years time, these things are going to make me feel embarrassed...

Server schema, as you can see, for example in type Chat, the creator, should be simply a type of User, the recipient, a type of User, so that when chat is accessed my application gets automatic access to the users related to this particular chat. Same for message, I should link to user object and chat object. The reason why I am not doing it the right way is because my subscriptions were freezing the server for some reason! And I couldn't figure it out! If the client subscribed to newChat, which would populate chat array if chat was added, worked, but when same user went into that chat room and subscribed to message subscription, going back to chats screen would hang the server. So I made this hacky and ugly workaround.

```
type User{
    id: ID!
    username: String!
    name: String!
    password: String!
}

type Chat{
    id: ID!
    creator: String!
    recipent: String!
}

type Message{
    id: ID!
    mcreator: String!
    mchat: ID!
    content: String!
}
```

Another bad thing is the login and security, there's absolutely no security in place, passwords are not encrypted, passed in as raw strings back and forth from the server. There is no sessions and that also leads to another issue, when users are subscribing to addChat subscription for example, and user1 creates chat with user2, every single application user gets the chat created. Each app then checks if current user of application is either creator or recipient of the chat room, if he is not, he doesn't display the chat. Problem with it is that it brings so much unnecessary traffic! However, I am not familiar with sessions and security aspects of Android or GraphQL.

Another issue, I feel that the memStores I am using are wrong, correct me if I am wrong but do I even need memStores if I am using GraphQL calls to the server ?

I did userMemStore to practice it and I really like the idea, although I couldn't really find any more use for it rather than a simple check if the user is logged in.

The Ugly

I feel that my UI could use a bit more colorful scheme, although I wanted to match the Viber color scheme, because I had no background it looks very plain and simple. Too simple!

For example, in chat room activity, I would love to have my adapter to display all my messages on the right side, all recipient messages on the left side, been googling it for some time and couldn't figure it out.

Code wise, I feel that I could have used some kind of functions more often, to keep the activities clearer. Right now some of the activities have multiple calls and I can only imagine how messy it would look if I added a few more. There must be a different way of organizing it, I must look through MVP patterns, which I heard might be a good solution ;/

The Future

I'd love to implement an MVP pattern, sessions and security. Fix the freezing server issue when subscribing to more than one channel. From a UI point of view, I'd like to implement lastMessage, that would display the last message on chat lists activity. Implement unified push support that would trigger a push notification pop up even when the app is closed. Some improvements in the way the app looks and also, allow for sending pictures or uploading their own pictures as their profile picture, this would be so cool! Pictures or/and emojis support!

Hopefully I'll get a chance to get back to it at some point!

How to Run

Application front end client has been set to use server local endpoint. You can change server URL by going to helpers/client. To run the application locally please follow the instructions below to launch the server locally. However, I am also able to provide a URL to the cloud server but I need to know in advance when you are planning to use it as it literally drains money from my AWS account. For the demo purpose I am going to have cloud set up and will use the cloud url.

Prerequisites:

- docker & docker-compose
- Node for yarn
- Android Studio

To run locally:

Go to server folder and run:

$ yarn install

This will install all the required dependencies. If you don't have yarn installed you can install yarn by:

$ npm install -g yarn

This will install yarn globally on your machine. However, to use npm you need to have nodejs installed.

Next run:

$ docker-compose up -d

To run docker-compose postgreSQL and MQTT broker

Finally run:

$ CREATE_DB=True yarn start

On initial launch we need to create database tables so use CREATE_DB=True variable, any other time after you can just use yarn start to start the server.

Conclusion

Chattie is by far the best project I've done in college. I am very happy I was able to do it the way it is! I'm very happy that I could use one project for 2 different modules and although it took a solid amount of hours to do, I enjoyed every single minute of it.

My only regret is that we have only covered so much and that I won't have mobile app development 2 !

Amazing assignment !

Thank you

Links:

Github link: https://github.com/MStokluska/MobileDevProject