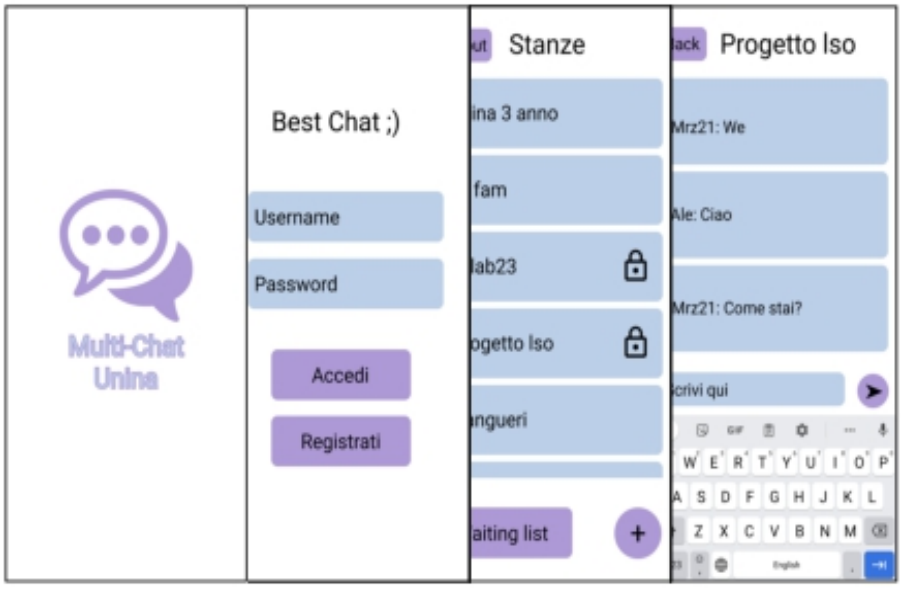


Marzia Pirozzi

OS Lab project
Multi Chat Unina

Requirements

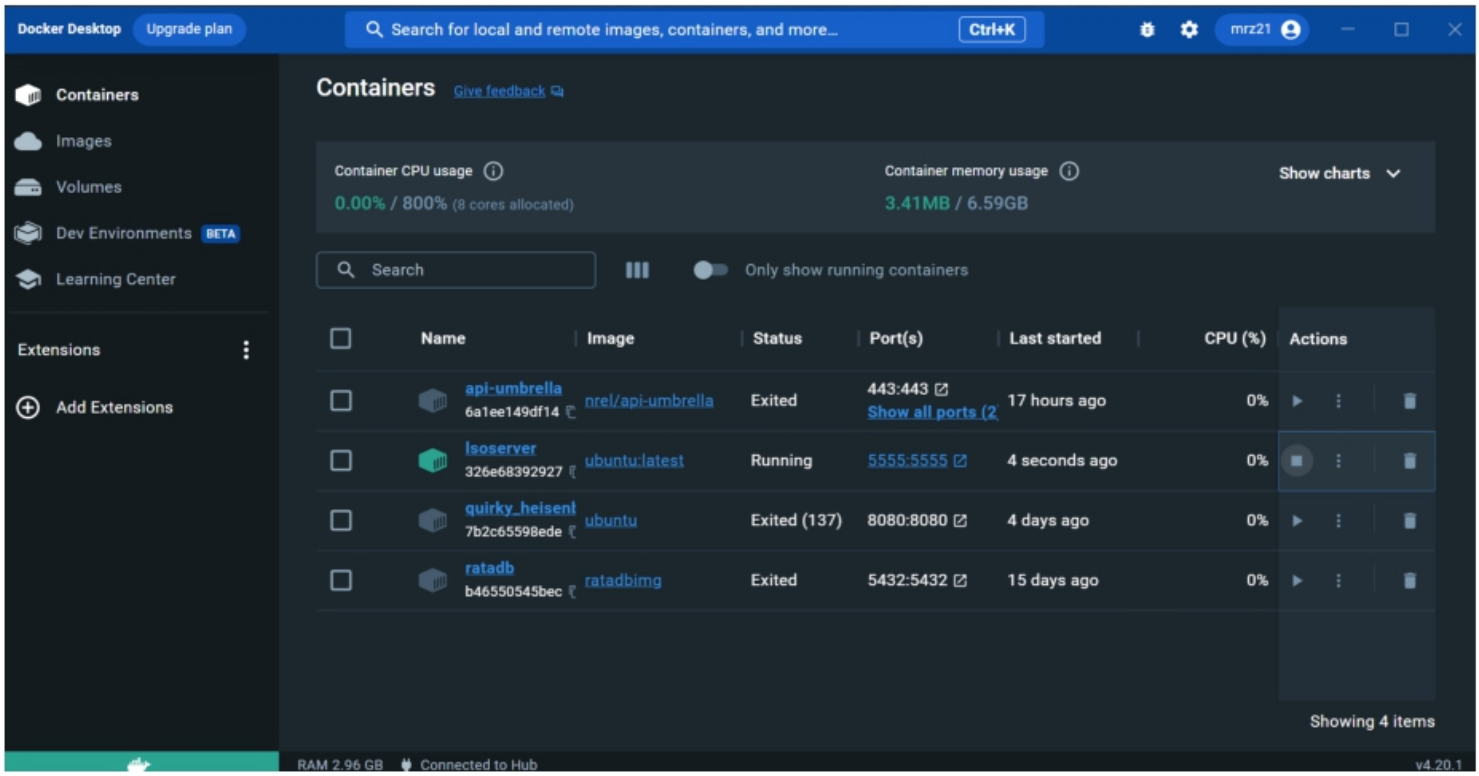
- Android Client (Java)
- C Server
- N users can access the system
- Each user has to register/log in
- Database can be SQL or a file
- Users can create rooms and can only chat with people in the same room
- Users can access public rooms or request to enter private ones
- The owner of the room can accept or deny users' access to the room

Extra features

- Compatibility with more devices (phones and tablets)
- PostgreSQL Database
- Backend hosted on cloud (AWS)

Backend choices

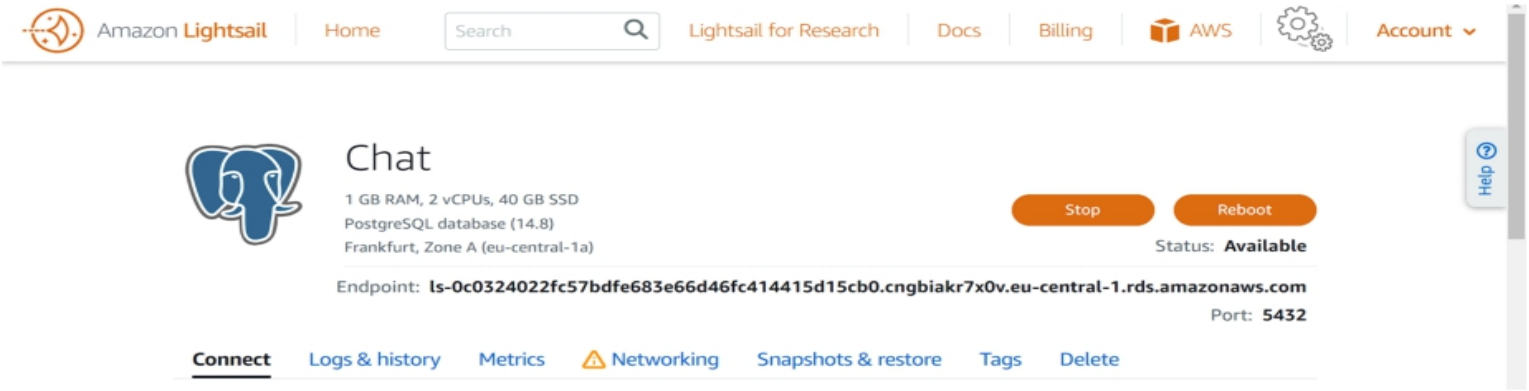
A Docker container, with an ubuntu image, was used to simulate a linux environment.

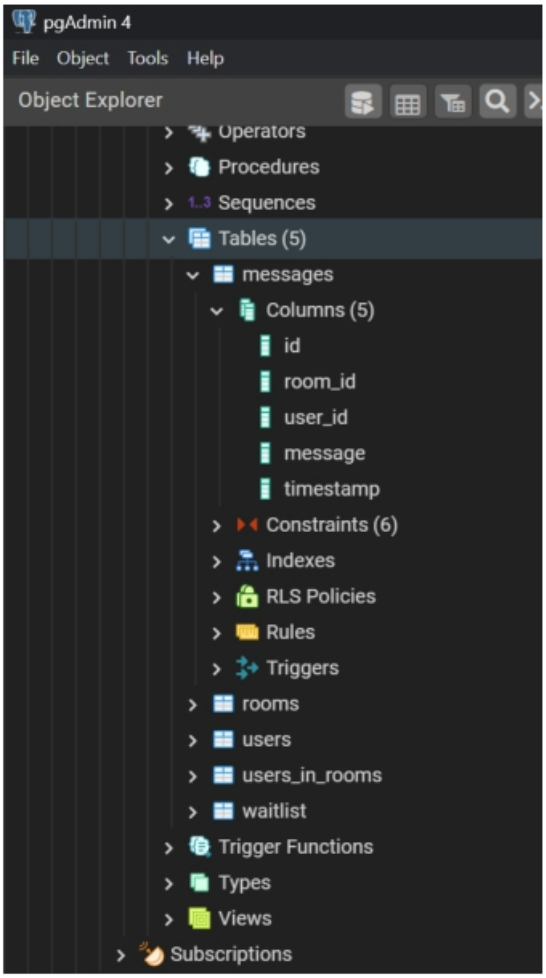


First thing I did was run the terminal and install all the necessary updates and services: nano, C compiler, Postgres library (libpq-dev).
Once tested the functionality on the local machine the server was moved in the cloud using the AWS service EC2 opening the ports to communicate with the database and with android clients from whatever ip address.
EC2 allows developers to use an instance of a machine which is secure and scalable. It also offers a free tier which is perfect for a university project because it’s very unlikely that it will exceed the maximum storage and data transfer limits.

Database

The relational database PostgreSQL was used and hosted on Amazon Lightsail.





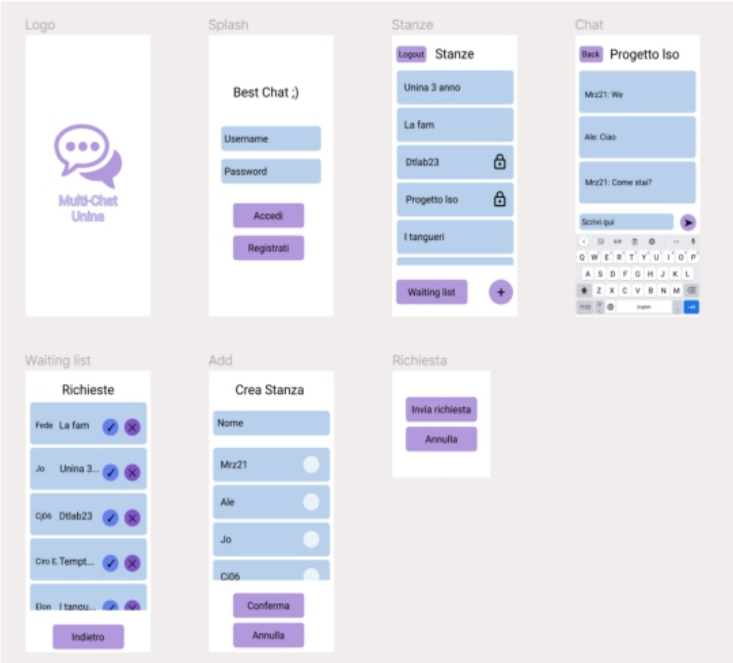
The following entities were recognised:

- Users
- Rooms
- Messages
- Wait lists
- Relation User-in-rooms

And the following constraints were defined:

- Users' usernames are Unique
- Rooms' names are Unique
- Each entity has an ID attribute which is Primary Key
- When necessary said IDs are used as Foreign Keys to determine relations between entities

Frontend choices

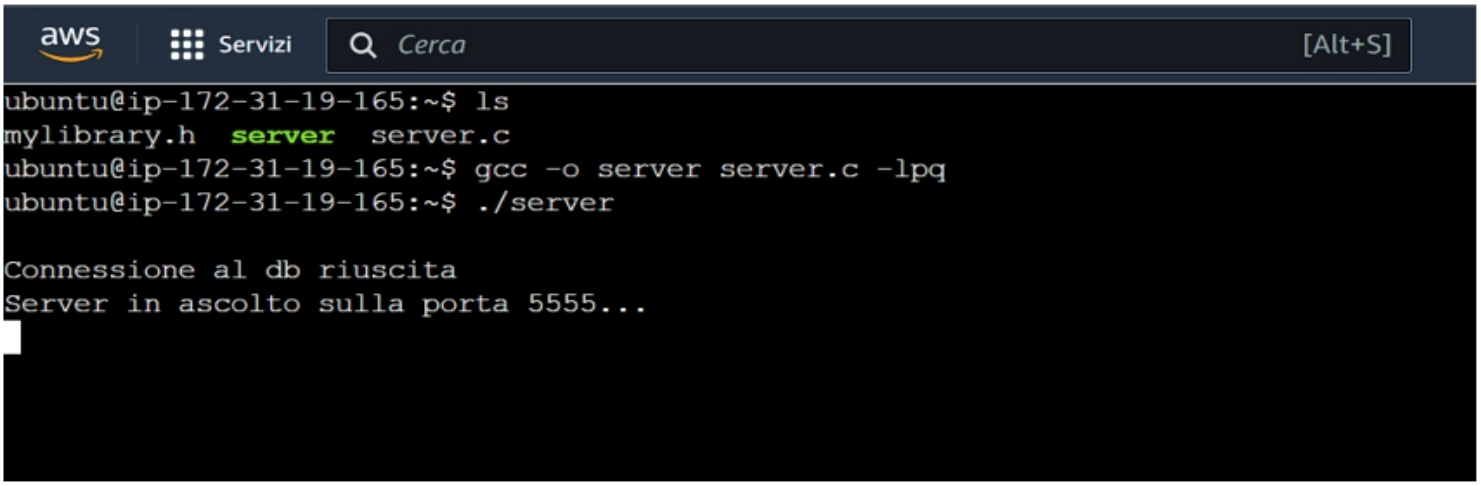


After reflecting on what are the essential features of the app the software Figma was used to prototype the interface. I tried to stay consistent the colors and the shapes.

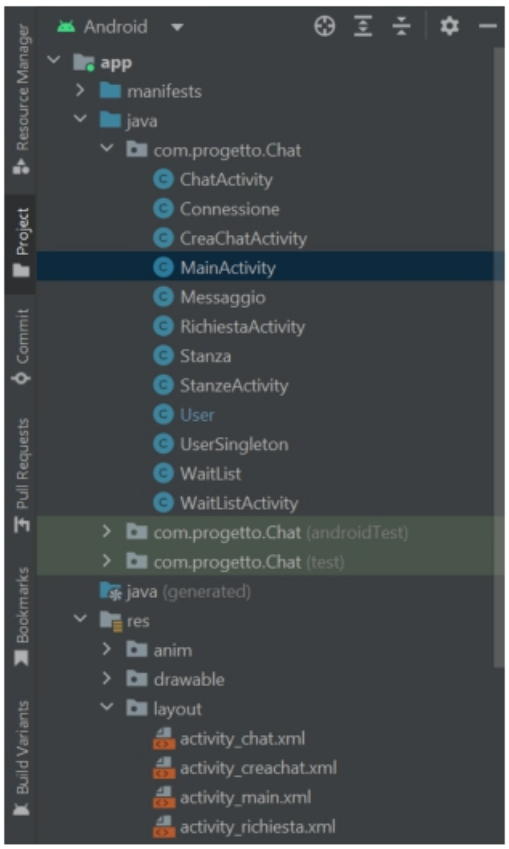
Then accessibility was checked using Adobe Colors, a system that checks contrast and daltony. The interface passed both tests.

Backend structure

The server is written in C and consists of only two files, one is the main and the other is a .h file that contains all the functions, making the code clean and modular. The port 5555 was opened for the server to listen on.



Frontend structure



The business logic was separated from the presentation logic. We have the following classes:

- **Model**: they represent the entities
- **View**: they represent the activities that the users interact with
- **Controller**: it represents the link between the server and the client so it takes care of the communication between the two. In particular the class UserSingleton is the controller and the reason why this design pattern was used is because from the moment the application is launcher the identity of the user is unique and shared between all the views in the application

How it works

To enable the communication with the client the server creates a socket that is a tunnel through which the data travels in, in both directions.
To handle many clients at the same time the server executes a fork and calls the function handleClient.
This function receives a string formatted in the following way:

“n%argument1%argument2...”

“N” represents the number of the operation to execute and the following are all the arguments that the operation needs.
The server then “decrypts” the message unpacking the string in an array of strings which is passed to a switch-case function that calls a specific operation based on the first string “n”.
After the function is executes the server sends a message to the client containing either the database response or a message of success/error.

Testing and known problems

- Usernames, rooms’ names and messages that contain “%” or “@” may not display correctly because of the convention chosen to format the communication string
- The app was tested with multiple users connected at the same time with no problems
- The app could contain unhandled exceptions, for example when a user tres to log in or register while the server is turned off the app just crashes

Final reflection

This application was a project I chose between 10 others at university for my operative system laboratory so the requirements where already set for me but I expanded from them. One of the boundaries was to use a server written in C which limited me and challenged me to find a way to make it work. Of course this would've been easier using an SDK like Firebase. Instead I hope it showcases my knowledge of relational databases, networking, cloud, docker and concurrency as well as operating with more processes and threads.