ChatMaps Deliverable 5

Software Architecture

Stephen Goodridge, Clark LaChance, Nicholas Pease, Joseph Gallant, Aidan Bradley

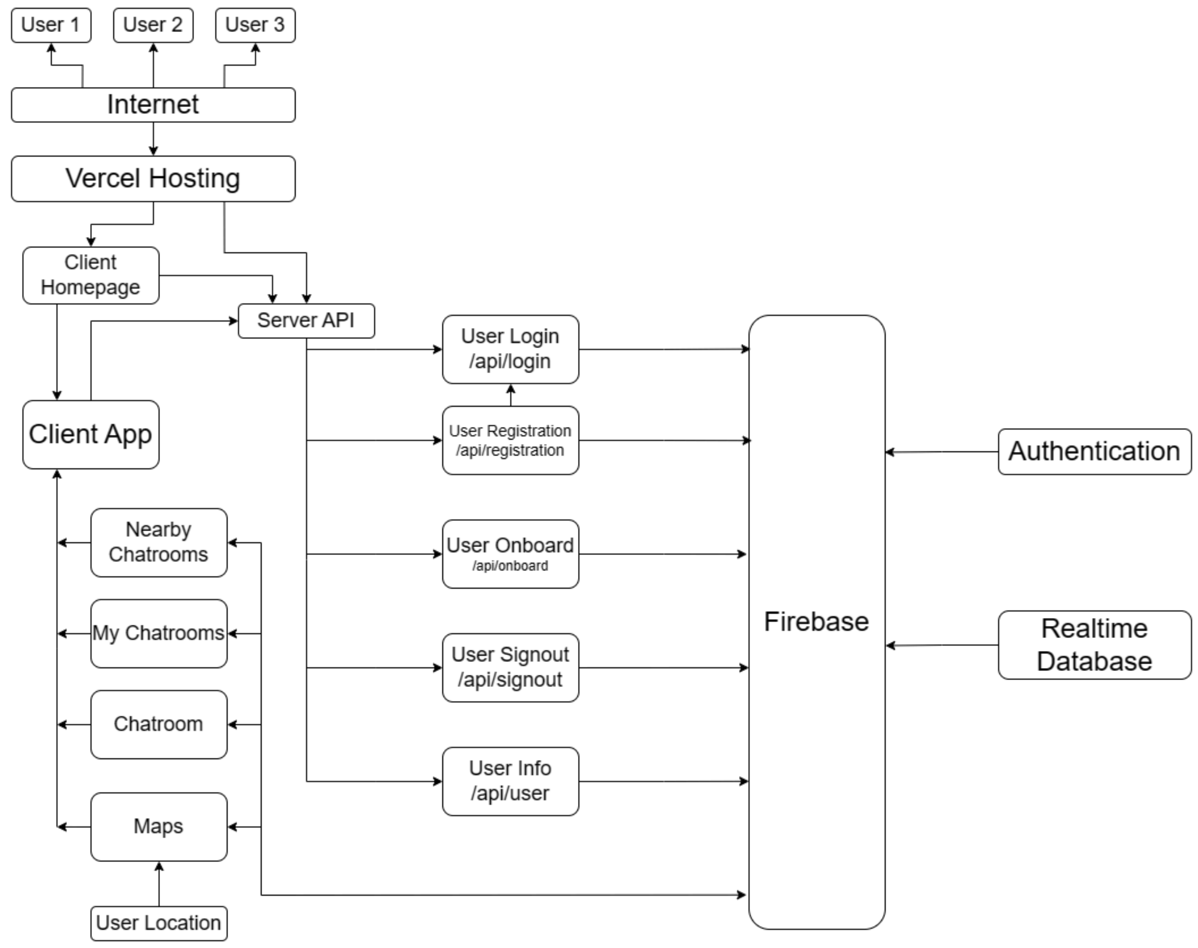
COS420

21 April 2024

Github: https://github.com/ChatMaps/ChatMaps

Kanban: https://trello.com/b/TaygvBv7/chatmaps

Architectural Design Document



For ChatMaps, we chose a client-server architecture as depicted in the diagram above. In a client-server architecture, all systems in an application are accessible at differing endpoints or via different servers. At the top you can see very little is rendered on the users end, and most of the app is hidden behind our hosting provider Vercel.

We chose to design our app this way to maximize the performance on the user's end, thus improving the user experience. Due to the structure of React, on the server side we have a client and a backend area. The client area consists of most everything that the user directly interacts with, whereas the backend area (labeled as the Server API) exists as a layer between our application and our external authentication and database provider Firebase. When using our application, we perform a handoff process where after authentication is verified, control is shifted to the Client App box, which utilizes all of the subsystems listed underneath to provide interactivity to the user.

On the bottom left, Nearby Chatrooms, My Chatrooms, Chatroom, and Maps are all linked to the client app. These each represent different pages and data being rendered within the client application. They each pull data from different parts of the backend where most functionality is implemented.

The API endpoints each represent a different “server” in the client-server architecture depicted, as they are each handled independently by the backend.

- /api/login represents existing user login
- /api/registration represents new user registration
- /api/onboarding represents the page where new users enter personal information
- /api/signout handles the logout process
- /api/user handles pulling user data from the database

Users connect to the service via the internet, being routed through the chosen hosting service (Vercel) to the main client homepage, which is the initial landing page. From here, users navigate to the client app where all user actions take place. As requests are made to each server endpoint, data is directed from Firebase back to the client app depending on the action performed. Although each action is performed on the same server, this architecture was appropriate as every user action is handled independently and could be hosted on several different servers with scalability in mind.

Firebase handles the rules for incoming and outgoing requests of data, and pulls its information from the Realtime Database. Authentication is also handled separately, where Firebase connects with the authentication service and then can send data back to the client application.