PotionTalk

Luca Huber

November 2024

1 Introduction

PotionTalk is a real-time, scalable chat application designed to enable seamless written communication for individuals and groups. Built with the Elixir programming language and Phoenix framework, it leverages Elixir's unique capabilities in handling high concurrency, fault tolerance, and real-time updates, providing a robust and responsive chat experience. Inspired by popular chat platforms, PotionTalk combines familiar functionalities with the resilience of Elixir's backend architecture.

2 Motivation

This is well suited to be done in Elixir since a popular application called Discord is partly written in elixir. The language provides great concurrency features and is very fault tolerant. Due to processes in elixir being light weight it is easy to scale if needed. Elixir also has a framework for webapps, which will be used to create a simple interface.

3 Key language features

Channels of the Pheonix framework can be used for real time communication. Pheonix Presence can be used to check for what users are online. PubSubs will be used to inform clients of new messages for real-time update once new messages are sent.

4 Basic system Architecture/Design

The frontend will be written in Elixir using Pheonix LiveView. It will be connected using Websockets for chats and a simple REST API for login and creating chat rooms. The backend will be written using Elixir. It will make use of Pheonix channels for chats. As for storing chats and user data, it will be done in memory using Erlang Term Storage. A basic overviw can be seen in Figure 1.

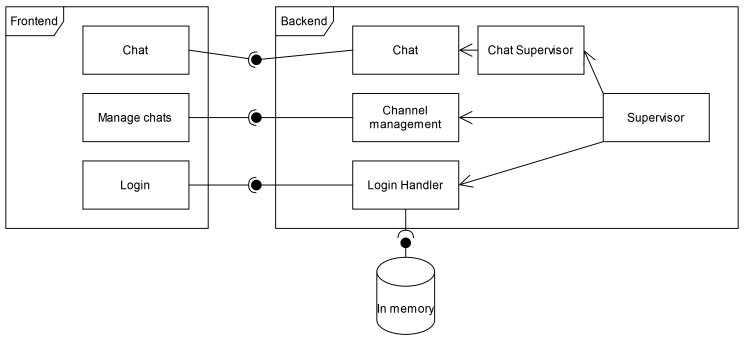

Figure 1: Basic system architecture

5 Scope

must-haves

- The user can connect to my service using a browser.
- The user can login. In this stage it can be by providing a username only.
- The user can create and join chat rooms.
- The user can send messages in a chat room.
- The user view is updated when new messages have been sent.
- The state of chat rooms is held in memory

nice-to-haves

- The user can see other users online.
- The storage is permanent instead of in memory.
- Tokenized authorization

out-of-scope

- Extending to other form of media beyond text.
- Admin tool
- Different user roles
- Scaling the backend to different nodes (clustering)
- Message reactions

6 Potential challenges

One challenge is to work with Pheonix, which I have never worked with. It should be able to handle my needs with templating, but in case it does not I might have to use some JavaScript. Another potential challenge could be to work with WebSockets for communication, even though Pheonix Channels are supposed to solve that in an abstracted way.