Faculty of Informatics and Information Technologies
Slovak University of Technology
Ilkovičova 2
842 16, Bratislava 4

Convo

Development of Progressive Web Applications

Author: Erik Roganský, Peter Sartoris

ID: 120985, 120877

Year of study: Third

Degree course: Computer Science

Study field: 9.2.1. Computer Science

Academic year: 2024/2025

Contents

1. Introduction 2
2. Assignment 2
3. Database diagram 3
4. Architecture diagrams 5
5. Design decisions 5
6. Screenshots 5

1. Introduction

In an era where instant communication drives both personal and professional interactions, the need for efficient and versatile messaging platforms has never been greater. This project aims to explore and develop a text-based communication application inspired by the principles of IRC (Internet Relay Chat), with functionality tailored for modern use, akin to a simplified version of Slack. The application seeks to balance simplicity with robust features, providing users with an intuitive interface and seamless communication capabilities.

The primary objective of this project is to create a functional messaging platform that accommodates user registration, channel-based communication, and administrative control, while emphasizing usability and technical efficiency. By leveraging key concepts of web application development, this project will serve as an opportunity to deepen practical understanding and implement dynamic features that align with the evolving needs of communication technology.

Through the development of this application, we aim to demonstrate proficiency in designing user-centric interfaces, handling real-time data processing, and managing the complexities of multi-user systems. Moreover, this endeavor underscores the role of technology in fostering connectivity and collaboration, contributing to a growing body of knowledge surrounding text-based communication systems.

2. Assignment

The goal of this assignment is to design and develop a functional text-based communication application modeled on the principles of IRC (Internet Relay Chat), with added features inspired by modern messaging platforms like Slack. The application will provide a robust framework for channel-based communication, user management, and real-time interaction, offering an opportunity to explore practical aspects of web application development.

Users should be able to register, log in, and log out seamlessly, with each user profile containing essential information such as a unique nickname, first name, last name, and email address. The application must ensure secure handling of user data while maintaining simplicity in the user management process. An intuitive interface should allow users to navigate through features effortlessly, ensuring a positive experience.

Channel interaction is a central aspect of the application. Users must have the ability to view a list of channels they belong to, create new channels, leave channels, or delete channels if they have administrative privileges. Channels are categorized as public or private, with distinct rules governing membership and access. Private channels restrict membership changes to administrators, while public channels allow broader interaction, enabling members to invite or remove users under specific conditions.

Messaging functionality is implemented via a fixed command-line interface, where users can send messages and commands. Messages can be directed to specific users with the use of @nickname, triggering highlighted notifications for the recipients. The application should support a complete message history accessible through infinite scroll, ensuring efficient and

user-friendly navigation. Notifications must inform users of new messages when the application is inactive, providing sender details and a preview of the message content.

User status management is another key feature, allowing users to set their status as Online, Do Not Disturb (DND), or Offline. Notifications are tailored to user preferences, with options to restrict alerts to direct messages. Offline users do not receive messages during their inactive period, but all updates are synchronized upon returning online.

To enhance usability, the application should include features such as a command to view the real-time list of channel members (/list) and indicators showing who is typing in active channels. Additionally, users can view drafts of messages being typed by others in real-time, fostering transparency and engagement within channels.

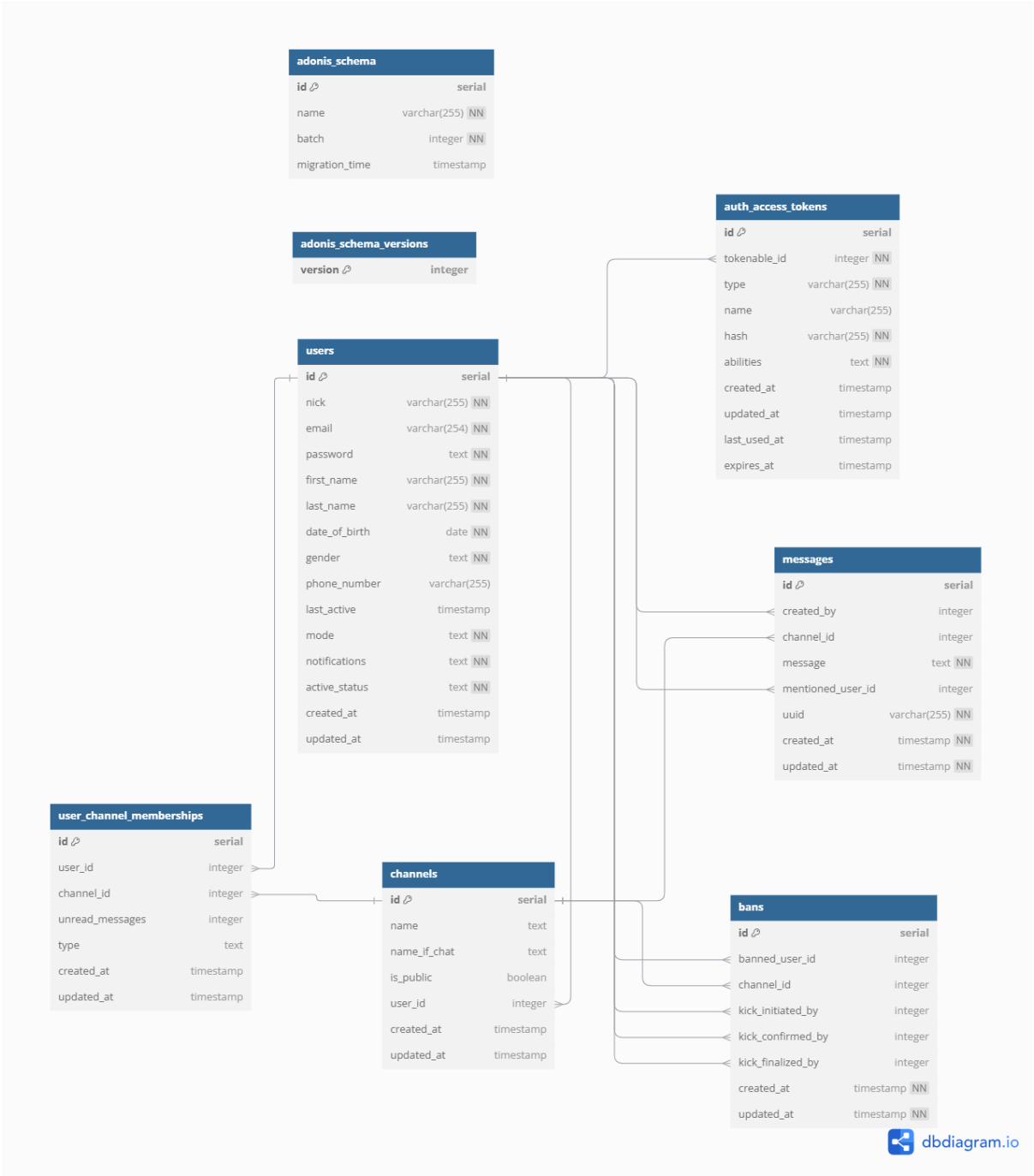
This assignment emphasizes the integration of core functionalities with user-centric design principles, aiming to deliver a platform that is both technically robust and intuitive to use. Through its development, the project offers an opportunity to address the challenges of building real-time, multi-user communication systems while refining skills in modern web development techniques.

For this project, the chosen domain is a communication platform named Convo, merging functionalities from Messenger and Slack.

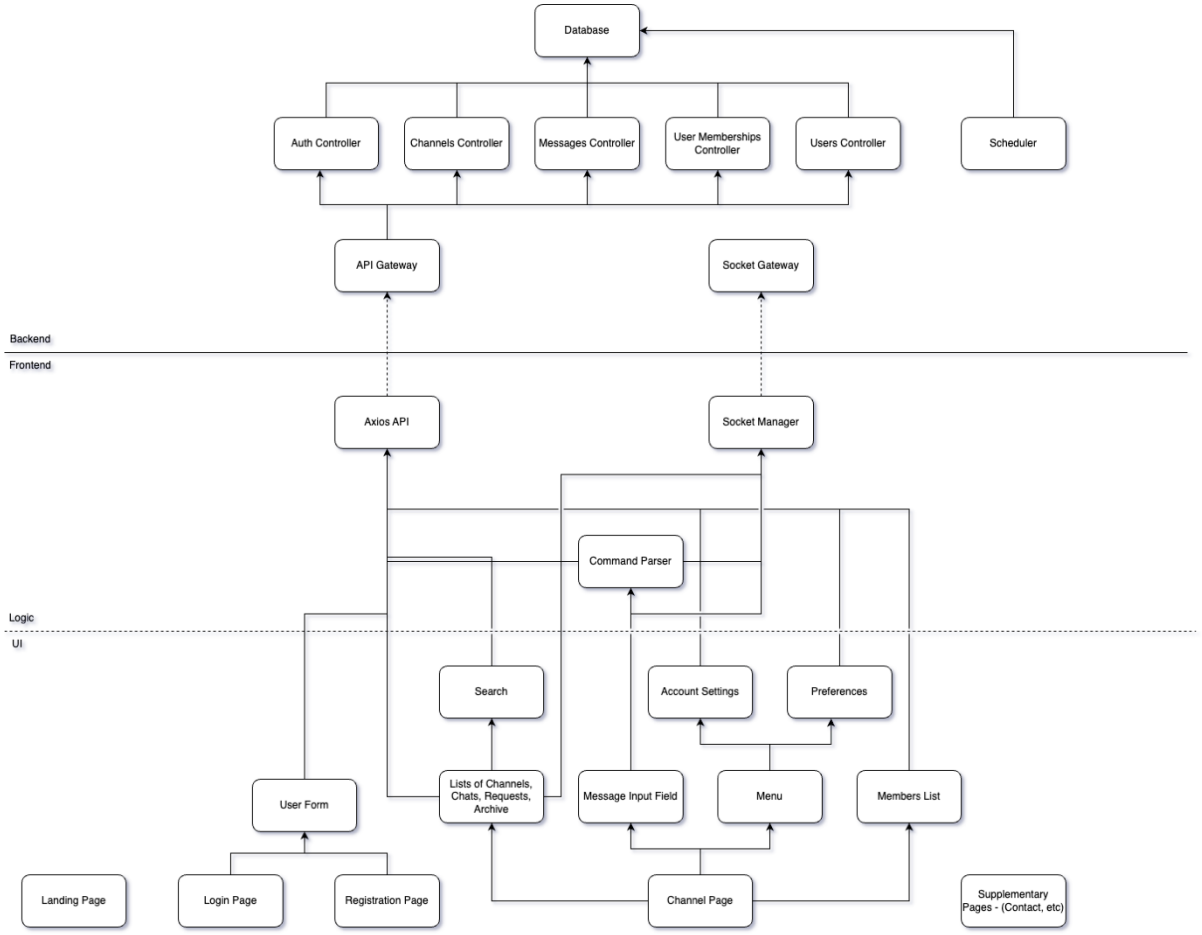
3. Database diagram

In comparison to the logical diagram created during Phase 1, only a few changes were implemented in the updated design. The primary modifications are as follows:

1. A uuid column was added to the messages table to ensure unique identification of messages. This change was introduced primarily to address the issue of duplicate messages being added to the chat. The uuid is generated using the uuidv4() function in AdonisJS, and the system checks for its presence to confirm whether the message has already been displayed.
2. The column names related to bans were revised to improve clarity and better reflect their purpose.
3. The type column was relocated from the channels table to the user_channel_memberships table. This adjustment was made because a single channel can have varying types depending on the user (e.g., a channel could be a request for one user, archived for another, and a basic chat or channel for a third). Maintaining a generic type for all users was not practical.
4. Additionally, AdonisJS built-in tables were automatically added, including a table for authentication purposes to store tokens.


adonis_schema
id serial
name varchar(255) NN
batch integer NN
migration_time timestamp
adonis_schema_versions
version integer
auth_access_tokens
id serial
tokenable_id integer NN
type varchar(255) NN
name varchar(255)
hash varchar(255) NN
abilities text NN
created_at timestamp
updated_at timestamp
last_used_at timestamp
expires_at timestamp
users
id serial
nick varchar(255) NN
email varchar(254) NN
password text NN
first_name varchar(255) NN
last_name varchar(255) NN
date_of_birth date NN
gender text NN
phone_number varchar(255)
last_active timestamp
mode text NN
notifications text NN
active_status text NN
created_at timestamp
updated_at timestamp
messages
id serial
created_by integer
channel_id integer
message text NN
mentioned_user_id integer
uuid varchar(255) NN
created_at timestamp NN
updated_at timestamp NN
user_channel_memberships
id serial
user_id integer
channel_id integer
unread_messages integer
type text
created_at timestamp
updated_at timestamp
channels
id serial
name text
name_if_chat text
is_public boolean
user_id integer
created_at timestamp
updated_at timestamp
bans
id serial
banned_user_id integer
channel_id integer
kick_initiated_by integer
kick_confirmed_by integer
kick_finalized_by integer
created_at timestamp NN
updated_at timestamp NN
dbdiagram.io

4. Architecture diagram



5. Design decisions

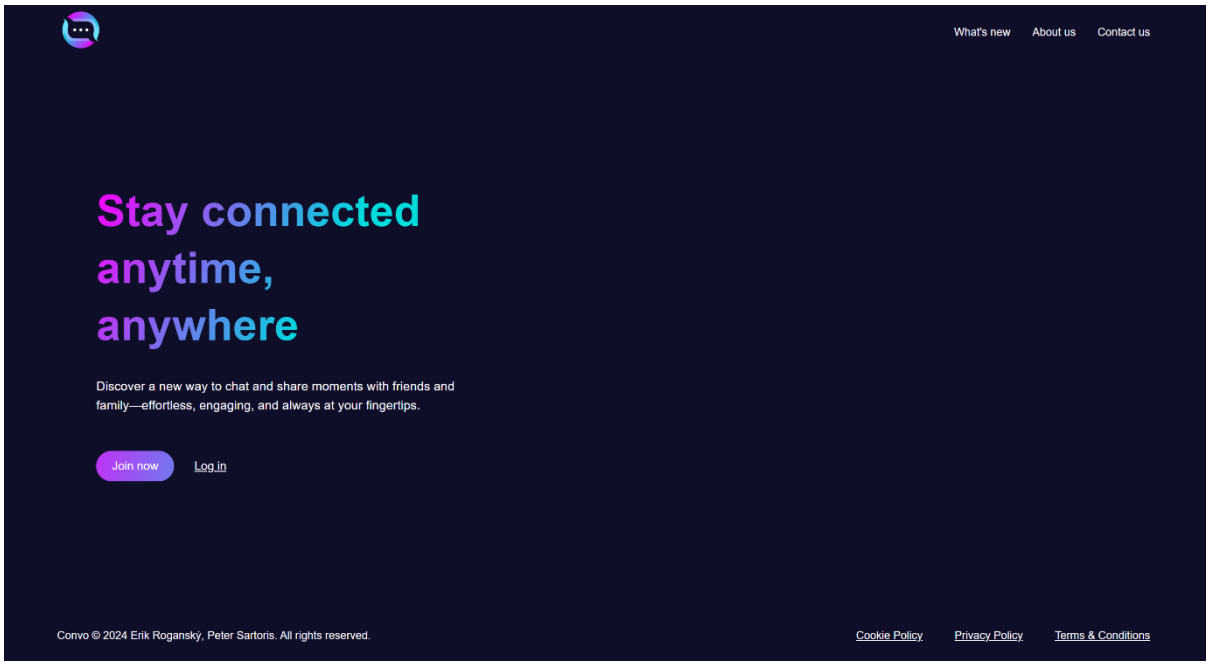
For our project, we chose to utilize the latest technologies, incorporating the most recent versions of Node.js, AdonisJS, and Quasar. While this decision added complexity—particularly due to the limited availability of tutorials, especially for AdonisJS—it allowed us to stay current and expand our knowledge base by working with modern tools.

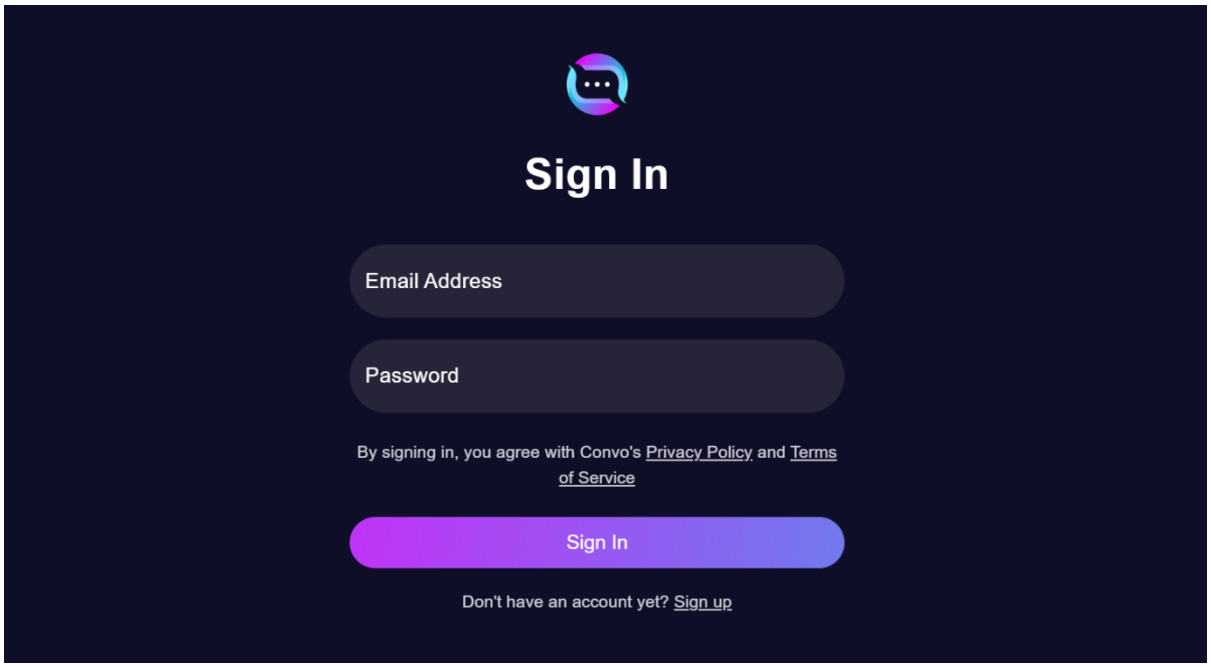
Regarding Quasar, we opted for the Composition API with the `script setup` configuration. Although it initially posed a steeper learning curve, this approach minimized redundant code and ultimately proved more efficient and easier to manage. For state management, we used Pinia, as it is the most up-to-date solution available. In AdonisJS, we applied the strictest ESLint settings to ensure our code adhered to a consistent and clean format. For database management, we chose PostgreSQL due to our prior familiarity with it.

In terms of specific implementations, we used the `cron` library to handle the scheduling of channel deletions after 30 days, as it provided a straightforward solution. For real-time socket communication, we employed Socket.IO. Given the robust solutions already offered by AdonisJS and Quasar, we found minimal need to rely on external or custom libraries.

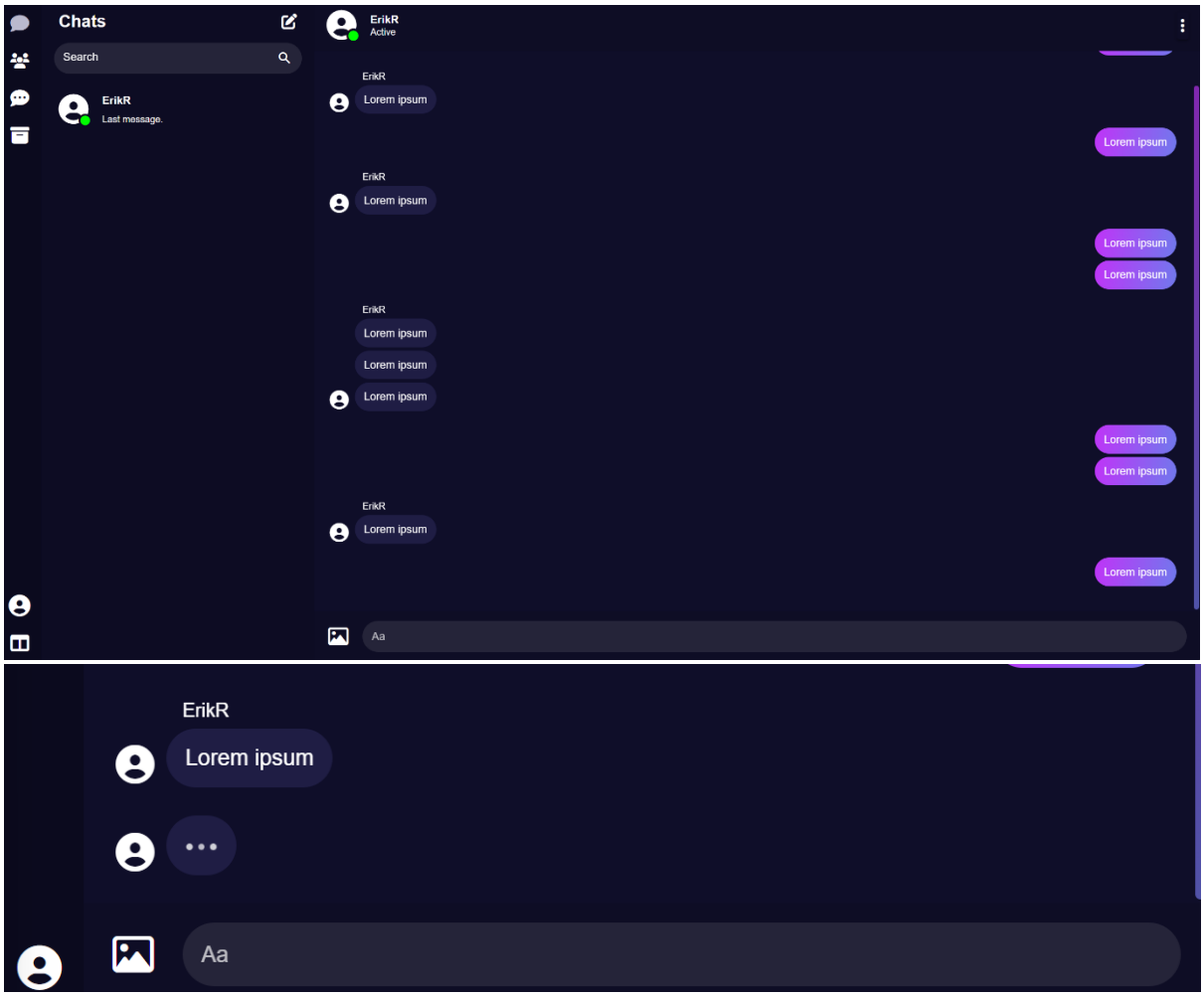
6. Screenshots

6.1. Landing and sign-in page

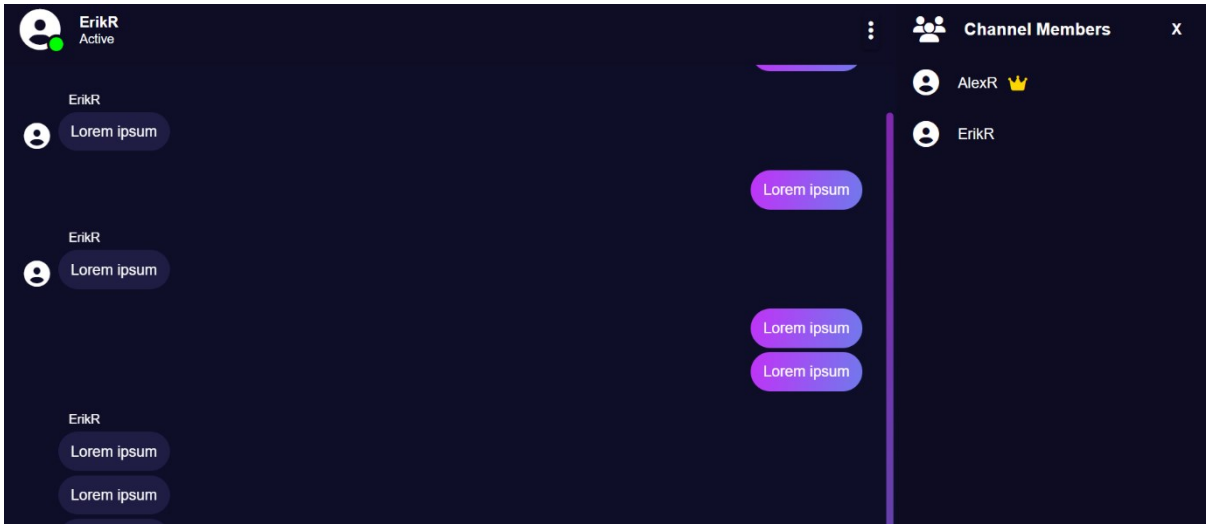




6.2. Chat and is typing demonstration



6.3. Channel members display



6.4. Offline mode