Stegord

Secure Chat Application through Steganography

Roles

Avi Lance -Project Manager / Full Stack Developer

Aleks Stevens - Backend Developer / DB Manager

Isaac Perkins - Frontend Developer

Jaxon Simmons - Frontend Developer

Why use Steganography for sending and receiving messages?

- **Added layer of security for messages**
  - Text encryption through steganography as well as general encryption similar to other secure messaging apps.
- **A piece of Misdirection:**
  - If packets are getting intercepted, it will not look like text data, it will instead feign as image data.

User Stories

- Journalist trying to send sensitive information who lives under a dangerous government
- Government officials sending messages
- People who need a more secure messaging app

Quick Demo

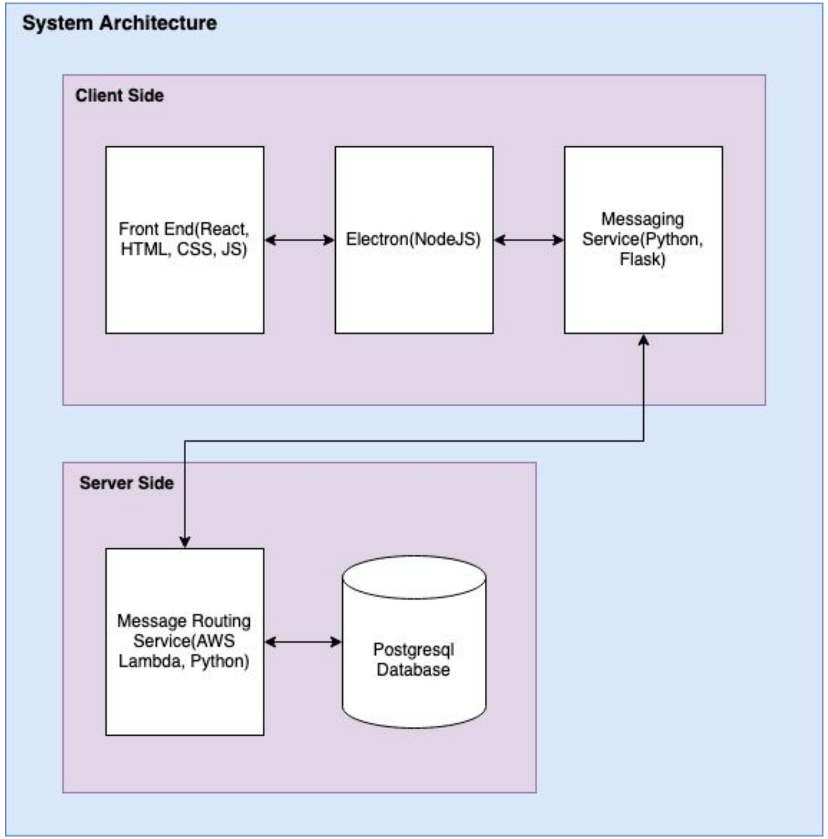
Function - User Requirements

User Requirements:

- Ability to login and sign up to a personalized profile
- Ability to find other users to friend and eventually chat with them.
- Ability to chat with friends using steganography
- Customizable profile bio and picture

System Architecture

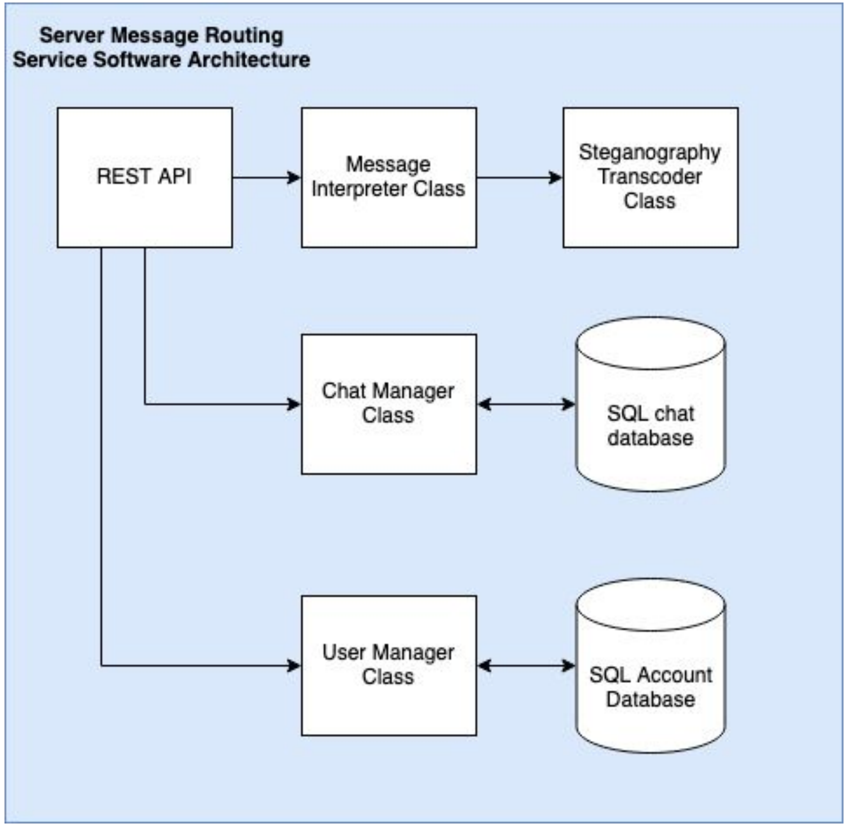
- A desktop application powered by Electron and Node JS
- The front end is coded in React using Typescript and Redux and communicates with Electron as a middleman with Flask
- AWS services (lambda functions, cognito, and amplify) act as the backend
- We use a PostgreSQL database for user data.


System Architecture
Client Side
Front End(React, HTML, CSS, JS)
Electron(NodeJS)
Messaging Service(Python, Flask)
Server Side
Message Routing Service(AWS Lambda, Python)
Postgresql Database

Backend and Integration Continued

Server Message Routing Module:

- Consolidate all server-side functionalities into a service accessible to a REST API
- Makes it easy for the client message routing module to send/receive messages
- Easy to interact with our PostgreSQL DB


Server Message Routing Service Software Architecture
REST API
Message Interpreter Class
Steganography Transcoder Class
Chat Manager Class
SQL chat database
User Manager Class
SQL Account Database

Testing and Development

- We all developed locally using github to pull new changes. We would also each pull down our own AWS Amplify authentication key to utilize the backend when developing.
- Both the frontend and backend would utilize test users to test every component
- Since many interactions are user-to-user, we would have two instances of our app running on different ports and login as different users

Team Organization

- Divided team into 3 groups, frontend, backend, and system architecture which allowed for asynchronous development
- Heavily utilized Jira to enforce Scrum Agile methodologies like sprints and bi-weekly meetings and checkups on work.
- Best fit for learning new technologies and understanding the changing state of our app through development
- Utilized github for version control
- Utilized Discord for team communication