Design Documents for "LinguaConnect"

PROJECT TITLE: LinguaConnect

Contact information:

Project Owner: Antonios Chatzinikolas Georgas
Email: antonis_132000@hotmail.com
Phone Number: (347) 744-3680

Elevator Pitch

LinguaConnect is a web application designed to foster language exchange and cultural connections among individuals worldwide. My project objective is to create an intuitive and interactive app that allows users to connect, practice languages, and organize language exchange events. The scope includes developing user profiles, real-time chat functionality, an interactive map for location-based connections, and features for creating and managing language exchange events.

Technical Implementation Overview

My technical implementation plan for LinguaConnect includes the following elements:

- **User Registration and Authentication**: Implement user registration and authentication for secure access.
- **User Profiles**: Create user profiles with customizable details, including profile pictures, bios, and language interests.
- **Real-Time Chat**: Develop real-time chat functionality with individual and group chat options.
- **Interactive Map**: Integrate an interactive map with location sharing and user markers for real-world connections.
- **Event Management**: Allow users to create, manage, and attend language exchange events.
- **Notification System**: Implement a notification system to keep users informed about messages, etc.

USER STORIES

- As a new user, I want to be able to create an account and log in so that I can start using the app.
- As a registered user, I want to create and personalize my user profile, including adding a profile picture, a brief bio, and my language interests.
- As a user, I want to see a list of online users, including their profile pictures and usernames.
- As a user, I want to be able to click on a user's profile to view more details about them.
- As a user, I want to initiate a real-time chat with another user.
- As a user, I want to see a list of my ongoing chat conversations on the Chat Page.

- As a user, I want to receive real-time notifications for new messages.
- As a user, I want to share my location on the map so that I can discover nearby language exchange partners.
- As a user, I want to view markers on the map representing other users, click on them to view profiles, and initiate chats.
- As a user, I want to have control over my privacy settings, including whether to share my location.
- As a user, I want to be able to create language exchange events and specify event details.
- As a user, I want to see a list of language exchange events on the app and be able to RSVP to events I'm interested in.
- As a user and event organizer, I want to manage my events, including editing event details and viewing RSVPs.

These user stories cover the essential functionality of LinguaConnect and serve as a foundation for defining features and workflows.

FEATURES

User Registration:

- Develop a user registration form that collects essential information such as email, username, and password.
- Implement server-side validation to ensure data integrity and security.
- Hash and securely store user passwords in the database.

User Authentication:

- Implement user authentication mechanisms.
- Create login functionality that verifies user credentials against stored data.
- Develop a user session management system to keep users logged in.

Profile Creation:

- Create a user profile form with fields for profile picture upload, bio input, and language interests selection.
- Implement server-side validation for profile data.
- Store profile information in a database associated with the user's account.

Profile Picture Upload:

- Develop a file upload feature for profile pictures.
- Handle image storage, resizing, and optimization.

User List Retrieval:

- Implement a database query to retrieve a list of online users.
- Fetch user profile pictures and usernames.
- Sort and display the list in the user interface.

User Profile Viewing:

- Create a user profile page or modal that displays detailed user information.

- Fetch and populate the user's profile based on the selected user.
- Provide options for users to navigate back to the previous view.

Real-Time Chat Initialization:

- Implement a chat initiation mechanism when the user clicks on another user's profile.
- Open a chat window or conversation with the selected user.
- Enable the exchange of real-time messages between users.

Chat Conversation Retrieval:

- Develop a system that retrieves and lists all active chat conversations for the logged-in user.
- Display user profile pictures, usernames, and the last message in each conversation.

Real-Time Notifications:

- Implement a real-time notification system (WebSockets).
- Notify users about new chat messages and chat requests.

Location Sharing:

- Develop a feature that allows users to share their location with the app.
- Use geolocation services to determine and store user coordinates.
- Ensure user consent and privacy controls for location sharing.

Map Integration:

- Implement an interactive map using a mapping library (Mapbox).
- Display markers representing other online users on the map.
- Enable marker click events to view user profiles or initiate chats.

Events:

- Develop a user-friendly event creation form with fields for event title, description, date, time, and location.
- Allow users to specify the languages that will be practiced during the event.
- Store event details in the event data table, associating the event with the user who created it.
- Develop a section on the map page that lists language exchange events, including event titles, dates, and locations.

Event Management:

- Allow event organizers to manage their events (edit or delete).
- Allow all users to manage their events list by adding events they are interested in or attending and removing events from the list when desired.

"SCHEMA" OUTLINE FOR MY DATABASE (MongoDB)

User Collection (users):

Users can have multiple relationships and actions within the app.

Fields:

- _id: User's unique identifier.
- username: User's username for authentication and identification.

- email: User's email address for communication and login.
- password_hash: Securely hashed password for authentication.
- profile: An embedded subdocument for user profiles with fields like name, profile_picture_url, bio, and languages to store personalized user data.
- notifications: An embedded subdocument for notification settings, including fields like chat_message to manage user notification preferences.
- location: An embedded subdocument to store the user's current location, which can be updated as the user moves. This enables real-time location sharing.
- other user-related fields…

Relationships:

- Users have a one-to-many relationship with events they organize. Each user can organize multiple events, but each event has one organizer (user).
- Users have a one-to-many relationship with chat messages they send or receive. A user can send and receive multiple chat messages, but each chat message is associated with one sender and one receiver (user).
- Users have a one-to-one relationship with their location. Each user has one current location.
- Users have a many-to-many relationship with events through the "user_event_categories" join collection. A user can attend multiple events, and events can have multiple attendees. This join collection represents the user's relationship with events and event categories of interest.

Event Collection (events):

Stores information about language exchange events created by users.

Fields:

- _id: Event's unique identifier.
- organizer_id: Reference to the user who created the event.
- title: Title of the event.
- description: Description of the event.
- date: Date and time of the event.
- languages: An array of languages to be practiced during the event.
- attendees: An array of subdocuments, each containing the user's _id and status (attending, interested).

Relationships:

- Events have a one-to-many relationship with the user who organizes them. Each event is organized by one user, but a user can organize multiple events.
- Events have a many-to-many relationship with users who attend or express interest in them. Multiple users can attend or show interest in multiple events. This relationship is represented through the "attendees" array within each event document.

Chats Collection (chats):

Handles chat conversations between users.

Fields:

- _id: Chat's unique identifier.
- participants: An array of user references who are part of the chat.
- messages: An array of chat message subdocuments, each containing the sender's _id, content, and timestamp.

Relationships:

- Chats have a many-to-many relationship with users. Multiple users can participate in a single chat conversation, and each user can be part of multiple chat conversations. This relationship is represented through the "participants" array within each chat document.
- Chats have a one-to-many relationship with chat messages. Each chat can contain multiple chat messages, but each chat message belongs to one specific chat conversation.

Location Collection (locations):

Stores users' locations for the map integration.

Fields:

- _id: Location's unique identifier.
- user_id: Reference to the user associated with this location.
- coordinates: Geospatial coordinates (latitude and longitude).

Relationships:

- Locations have a one-to-one relationship with users. Each location is associated with one user, representing the user's current location.

Event Categories Collection (event_categories):

Define event categories.

Fields:

- _id: Category's unique identifier.
- name: Name of the event category (Created by User, Attending, Interested).

Relationships:

- Event categories have a one-to-many relationship with events. Multiple events can be categorized under the same category, but each event category can be associated with multiple events.

User-Event Categories Join Collection (user_event_categories):

Associates users with events and event categories.

Fields:

- _id: Unique identifier for the join document.
- user_id: Reference to the user.
- event_id: Reference to the event.

- category_id: Reference to the event category.

Relationships:

- This join collection establishes many-to-many relationships between users, events, and event categories. Users can attend multiple events, and events can fall into multiple event categories. This join collection associates users with specific events and their corresponding event categories.

USE CASES

User Registration and Authentication:

- User registers with email and password.
- User logs in and authenticates.

User Profile Management:

- User updates their profile information, including name, profile picture, and bio.
- User manages their language interests.
- User changes password.
- User shares their location.
- User views their own profile.

Event Management:

- User creates a new event.
- User updates event details.
- User removes an event they organized.
- User attends an event.
- User expresses interest in an event.
- User removes an event they were interested in/attending from their event list.

Chat Functionality:

- User sends a chat message.
- User initiates a chat with another user.
- User views their chat conversations.
- User receives real-time chat notifications.

Location Sharing:

- User shares their location.
- User views nearby users on the map.

Event Discovery:

- User views a list of events.
- User views event details.
- User sees events on the map.

Notifications:

- User receives chat message notifications.

TECHNICAL IMPLEMENTATION

User Registration:

Client-Side Implementation:

- Users provide their registration details (email, username, password) through a registration form.
- The client app performs basic client-side validation (e.g., password strength).
- It sends a POST request to the server's registration endpoint.

Server-Side Implementation:

- The server validates the registration data, checking for unique email and username.
- It securely hashes the user's password and stores the user's information in the database.
- Upon successful registration, the server generates an authentication token and sends it to the client.

User Login:

Client-Side Implementation:

- Users enter their credentials (email and password) in the login form.
- The client app sends a POST request to the server's login endpoint.

Server-Side Implementation:

- The server verifies the user's credentials, comparing the provided password hash with the stored hash.
- If the credentials are valid, the server generates an authentication token and sends it to the client.

Authentication:

- Upon receiving the authentication token, the client app stores it securely in local storage.

User Profile Creation:

Client-Side Implementation:

- Users access their profile settings and fill out the profile information, including profile picture, bio, and language interests.
- The client app validates the data and sends a PATCH request to the server's profile update endpoint.

Server-Side Implementation:

- The server verifies the user's authentication token.
- It updates the user's profile information in the database.

User Profile Viewing:

Client-Side Implementation:

- Users can view other users' profiles by clicking on their usernames or profile pictures.
- The client app sends a GET request to the server's user profile endpoint.
- Server-Side Implementation:

- The server retrieves the requested user's profile information from the database and sends it to the client.

User Creates a New Event:

User Interaction:

- The user clicks on the map to initiate the event creation process.

Client-Side Implementation:

- The client app renders an event creation form on the "My Events" list next to the map.
- The user enters event details, including title, description, date, time, location, and languages.

Client Sends Event Creation Request:

- Upon submitting the event creation form, the client app sends a POST request to the server's event creation endpoint.
- The request includes all the event details provided by the user.

Server-Side Implementation:

- The server receives the event creation request.
- It verifies the user's authentication token to ensure that the user is authorized to create events.

Create Event:

- The server creates a new event document in the database, associating it with the user who organized it.
- The event document includes all the details provided by the user, such as title, description, date, and more.

Response to Client:

- The server responds to the client with a confirmation message and the event's unique identifier (ID).

Client-Side Event List Update:

- After successfully creating the event, the client app updates the event list to include the newly created event in the "Created by Me" category.

Display Event on the Map:

- Simultaneously, the client app updates the map view to display the marker representing the new event at the specified location.

User Edits an Existing Event:

User Interaction:

- When a user wants to edit an event they organized, they navigate to the "My Events" list next to the map and click on the options menu (three vertical dots symbol) next to the event.

Client-Side Implementation:

- The client app provides an "Edit Event" option.
- The user clicks on "Edit Event" to open an event editing form pre-filled with the existing event details.

Client Sends Event Edit Request:

- Upon making edits, the user submits the edited event details using a PUT or PATCH request to the server's event edit endpoint.
- The request includes the updated event details and the event's unique identifier (ID).

Server-Side Implementation:

- The server receives the event edit request.
- It verifies the user's authentication token to ensure that the user is the organizer and authorized to edit the event.

Update Event:

- The server updates the event document in the database with the new event details provided by the user.

Response to Client:

- The server responds to the client with a confirmation message indicating that the event details have been updated.

User Deletes an Existing Event They Organized:

User Interaction:

- The user clicks on the three vertical dots symbol next to an event in their "My Events" list.
- The user selects the "Delete Event" option.

Client-Side Implementation:

- The client app sends a DELETE request to the server's event deletion endpoint.
- The request includes the unique identifier (ID) of the event to be deleted.

Server-Side Implementation:

- The server receives the event deletion request.
- It verifies the user's authentication token to ensure that the user is authorized to delete events.

Delete Event:

- The server identifies the event to be deleted based on the provided event ID.
- The server removes the event document from the database.

Remove Event from User's List:

- Simultaneously, the server removes the event from the user's "My Events" list by updating the user's document in the database.
- The event is removed from the "Created by Me" category.

Remove Event from the Map:

- The server also removes the event's marker from the interactive map, ensuring that it is no longer visible to other users.

Response to Client:

- The server responds to the client with a confirmation message indicating that the event has been successfully deleted.

User Attends an Event:

User Interaction:

- The user interacts with the map interface and clicks on an event marker representing an event they want to attend.A popup displays with event details and options: "attending," "Interested," and "Close."
- The user selects the "attending" option from the popup, indicating their intention to attend the event.

Client-Side Implementation:

- The client app sends a POST request to the server's event attendance endpoint.
- The request includes the unique identifier (ID) of the event the user is attending and the user's authentication token.

Server-Side Implementation:

- The server receives the event attendance request.
- It verifies the user's authentication token to ensure that the user is authorized to attend events.

Update User's "My Events" List:

- The server adds the event to the user's "My Events" list under the "attending" category by updating the user's document in the database.
- This update associates the event with the user and places it in the "attending" category.

Response to Client:

- The server responds to the client with a confirmation message indicating that the user has successfully attended the event.

User Expresses Interest in an Event:

User Interaction:

- The user interacts with the map interface and clicks on an event marker representing an event they are interested in.
- A popup displays with event details and options: "attending," "Interested," and "Close."
- The user selects the "Interested" option from the popup, indicating their interest in the event.

Client-Side Implementation:

- The client app sends a POST request to the server's event interest endpoint.
- The request includes the unique identifier (ID) of the event the user is interested in and the user's authentication token.

Server-Side Implementation:

- The server receives the event interest request.
- It verifies the user's authentication token to ensure that the user is authorized to express interest in events.

Update User's "My Events" List:

- The server adds the event to the user's "My Events" list under the "Interested" category by updating the user's document in the database.

- This update associates the event with the user and places it in the "Interested" category.

Response to Client:

- The server responds to the client with a confirmation message indicating that the user has successfully expressed interest in the event.

User removes an event they were interested in/attending from their event list:

User Interaction:

- The user clicks on the three-dot symbol next to an event that is under either the "attending" or "Interested" category on their "My Events" list.
- A context menu is displayed with options, including "Remove from my list."
- The user selects the "Remove from my list" option from the context menu, indicating their intent to remove the event.

Client-Side Implementation:

- The client app sends a DELETE request to the server's event removal endpoint.
- The request includes the unique identifier (ID) of the event the user wants to remove and the user's authentication token.

Server-Side Implementation:

- The server receives the event removal request.
- It verifies the user's authentication token to ensure that the user is authorized to remove events.

Remove Event from User's "My Events" List:

- The server removes the event from the user's "My Events" list, whether it was under the "attending" or "Interested" category.
- This removal is performed by updating the user's document in the database, specifically removing the reference to the event.

Response to Client:

- The server responds to the client with a confirmation message indicating that the event has been successfully removed from the user's list.

User Sends a Chat Message:

Client-Side Implementation:

- When a user types and sends a message in the chat interface, the client app formats the message content.
- The app then sends the message to the server via an API request.

Server-Side Implementation:

- Upon receiving the message from the client, the server performs several actions:
- Authenticates the user sending the message.
- Identifies the recipient of the message.
- Stores the message in the database, associating it with the relevant chat conversation.

Real-Time Notification:

After storing the message, the server broadcasts it to the recipient in real-time.

- The server pushes the new message to the recipient's chat interface via WebSockets or a similar technology.
- The client app receives the real-time message and displays it in the chat conversation.

User Receives a Chat Message:

Real-Time Messaging:

- The client app establishes a WebSocket connection with the server upon login.
- This connection allows the client to listen for incoming chat messages in real-time.

Server-Side Notification:

- When a user receives a chat message, the server identifies the recipient of the message.
- The server sends the message to the relevant recipient via the WebSocket connection.

Client-Side Handling:

The client app receives the real-time message and handles it accordingly:

- If the chat conversation is currently open, it displays the message in real-time.
- If the chat conversation is closed, it notifies the user of the new message, with a notification badge or sound.

User Views Message History:

Client-Side Implementation:

- The chat interface includes a scrollable message history section.
- As the user scrolls back in time, the client app detects the request for older messages.

Server-Side Request Handling:

- When the client requests older messages, it sends a request to the server for a specific range of messages (e.g., messages 50-100).

Database Query:

- The server queries the database for the requested message range based on the chat conversation ID.
- It retrieves and sends the messages back to the client.

Client-Side Display:

- The client app receives the older messages and displays them in the chat interface.
- It may use pagination or infinite scrolling to load more messages as the user continues scrolling back.

User Sends a Read Receipt:

Client-Side Implementation:

- When the user opens a message in a chat conversation, the client app marks the message as "read."
- It sends a request to the server to update the message status.

Server-Side Handling:

- The server receives the "read" status update request.
- It updates the message status in the database to indicate that the message has been read by the recipient.

User Initiates a Chat with Another User:

User Interaction:

- When a user wants to initiate a chat with another user, they typically navigate to the user's profile page or use a search function to find the user they want to chat with.

Client-Side Implementation:

- The user interacts with the client-side application to trigger the chat initiation process.
- This interaction occurs by clicking on the message icon in the profile of the user they wish to chat with.

Client Sends Chat Initiation Request:

- Upon user interaction, the client app sends a request to the server's chat initiation endpoint.
- The request includes information about the target user, such as their user ID or username.

Server-Side Implementation:

- The server receives the chat initiation request.
- It first verifies the user's authentication token to ensure that the user initiating the chat is authenticated and authorized to use the chat feature.

Check Existing Chats:

- The server checks if there is an existing chat conversation between the two users.
- If an active chat already exists, it might return the existing chat ID to the client.

Create New Chat if Necessary:

- If no existing chat is found, the server creates a new chat room or conversation.
- It generates a unique chat ID for this conversation and associates both users with it.

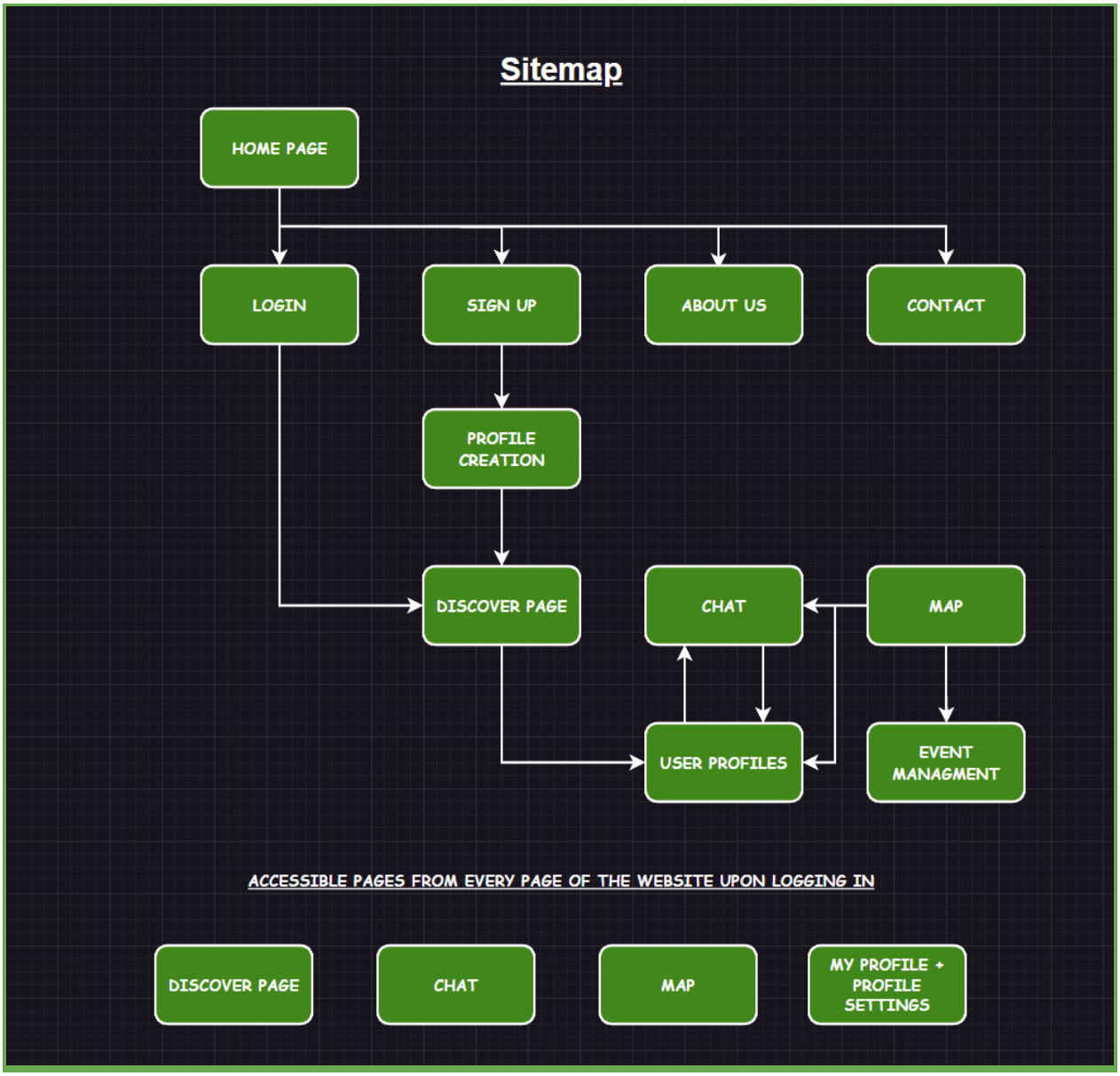
Response to Client:

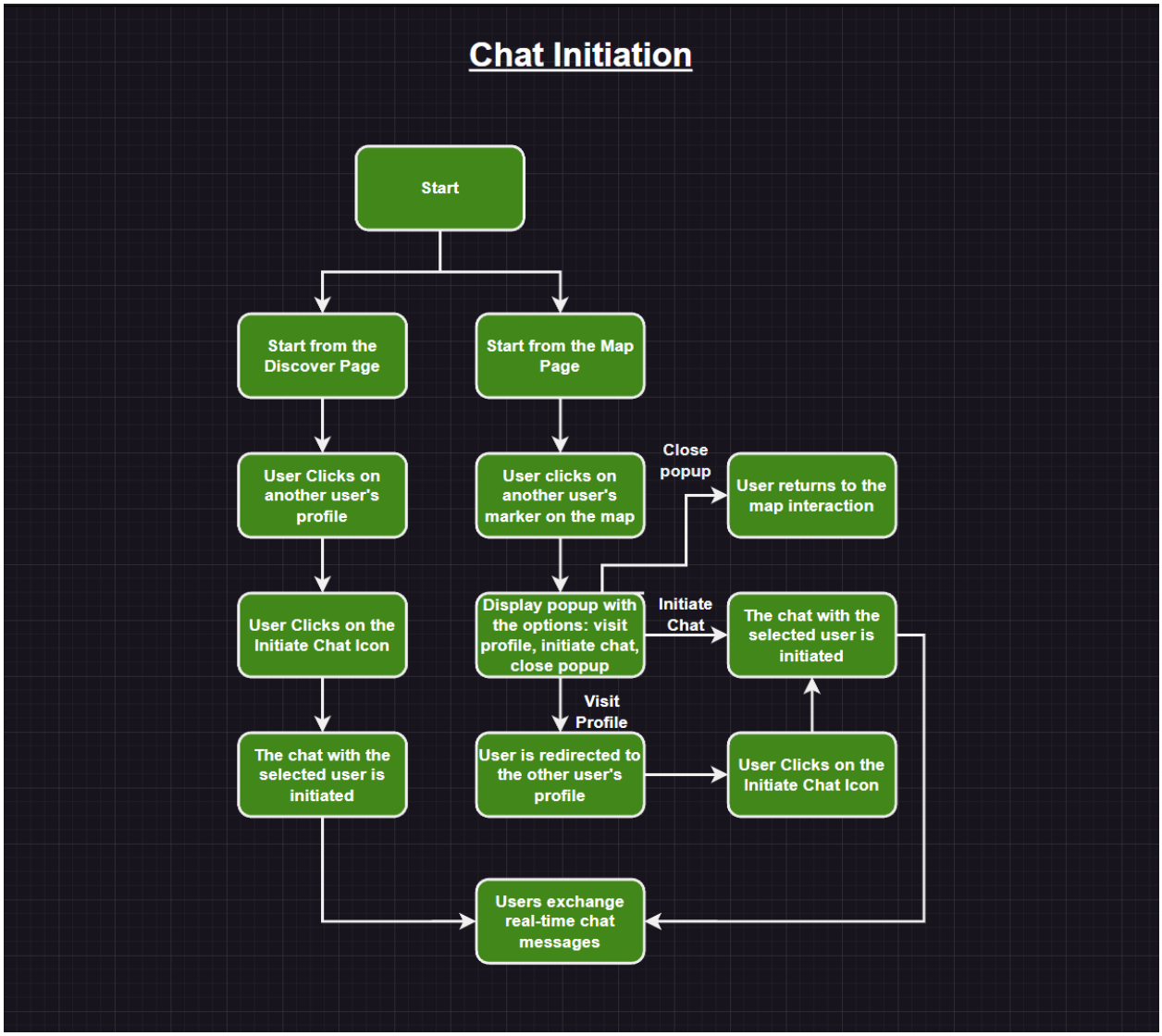
- The server responds to the client with the chat ID or information about the newly created chat room.

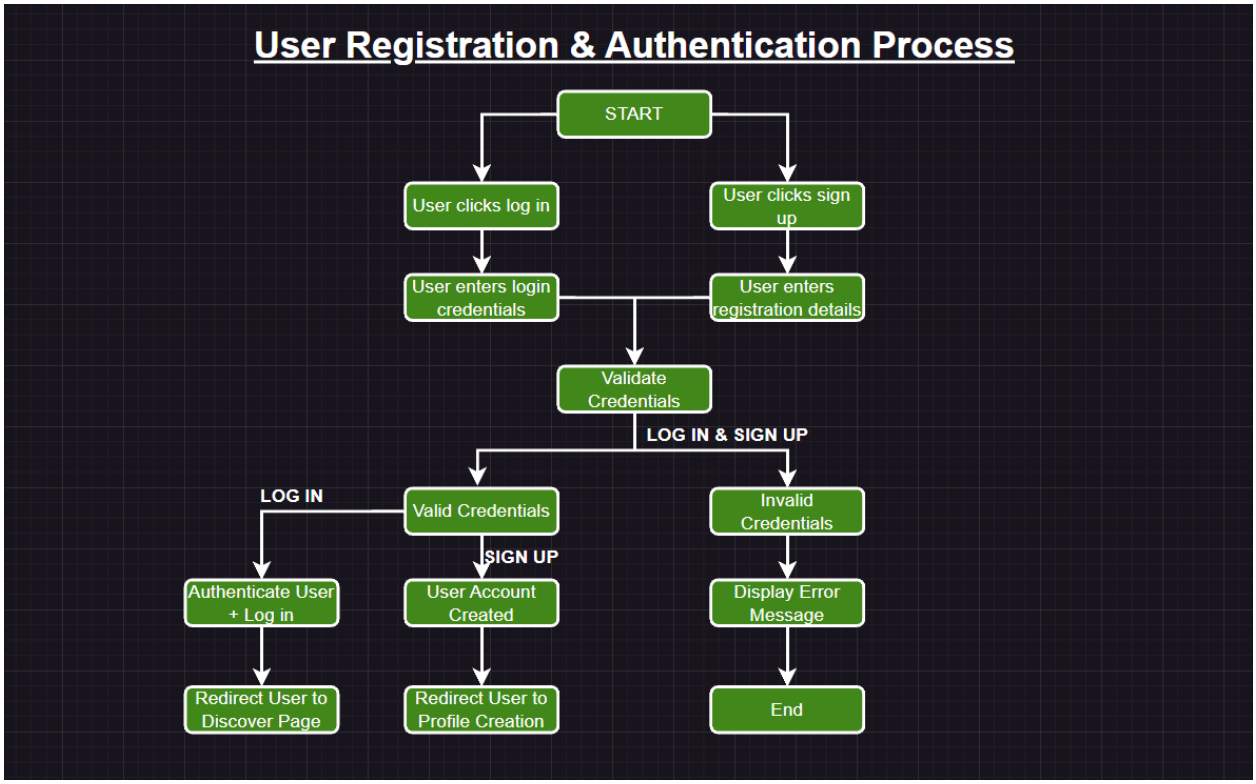
Client-Side Chat Interface:

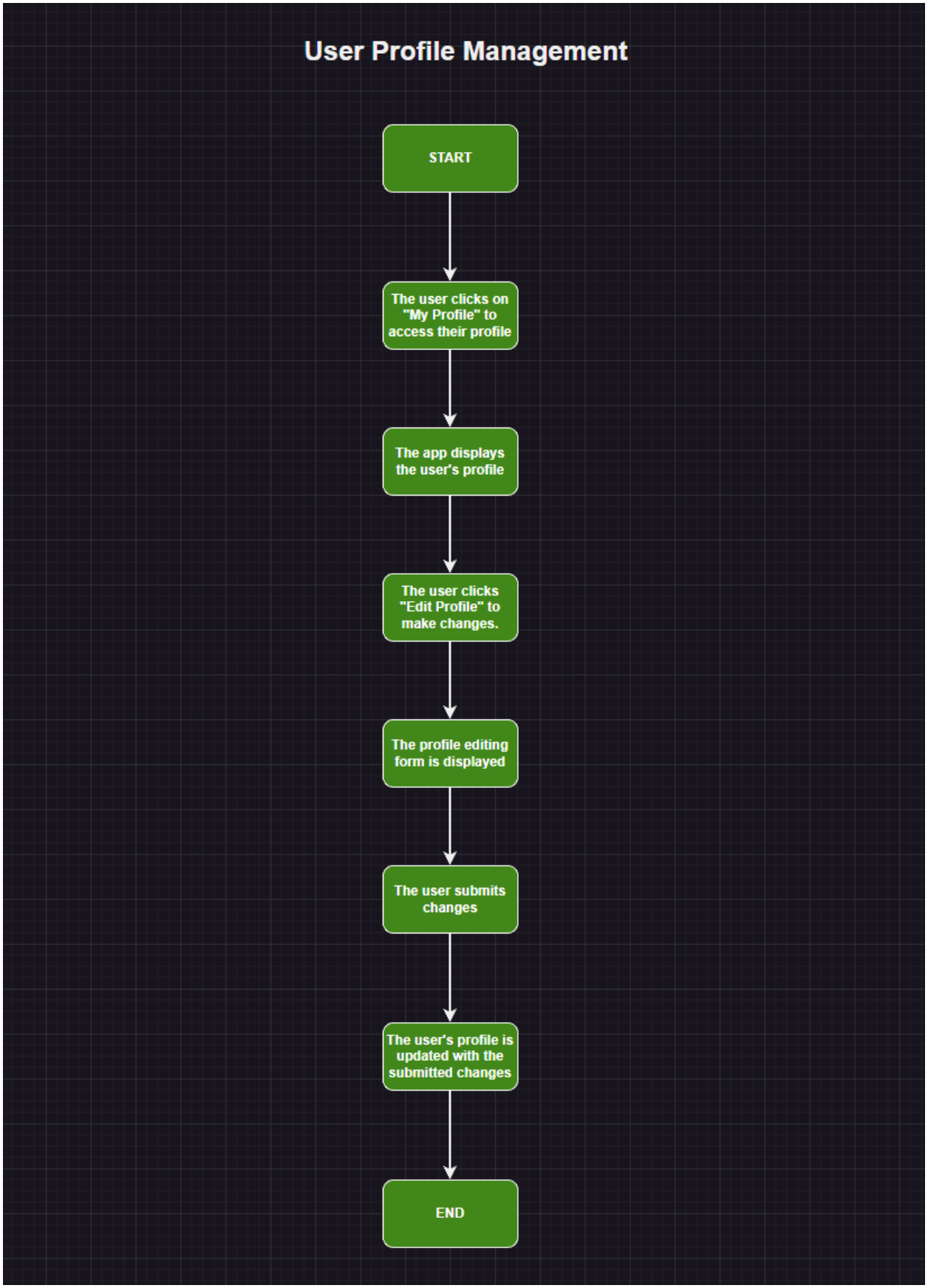
- The client app updates the user interface to display the chat room or conversation with the target user.
- It may open a chat window, provide an input area for sending messages to the targeted user.

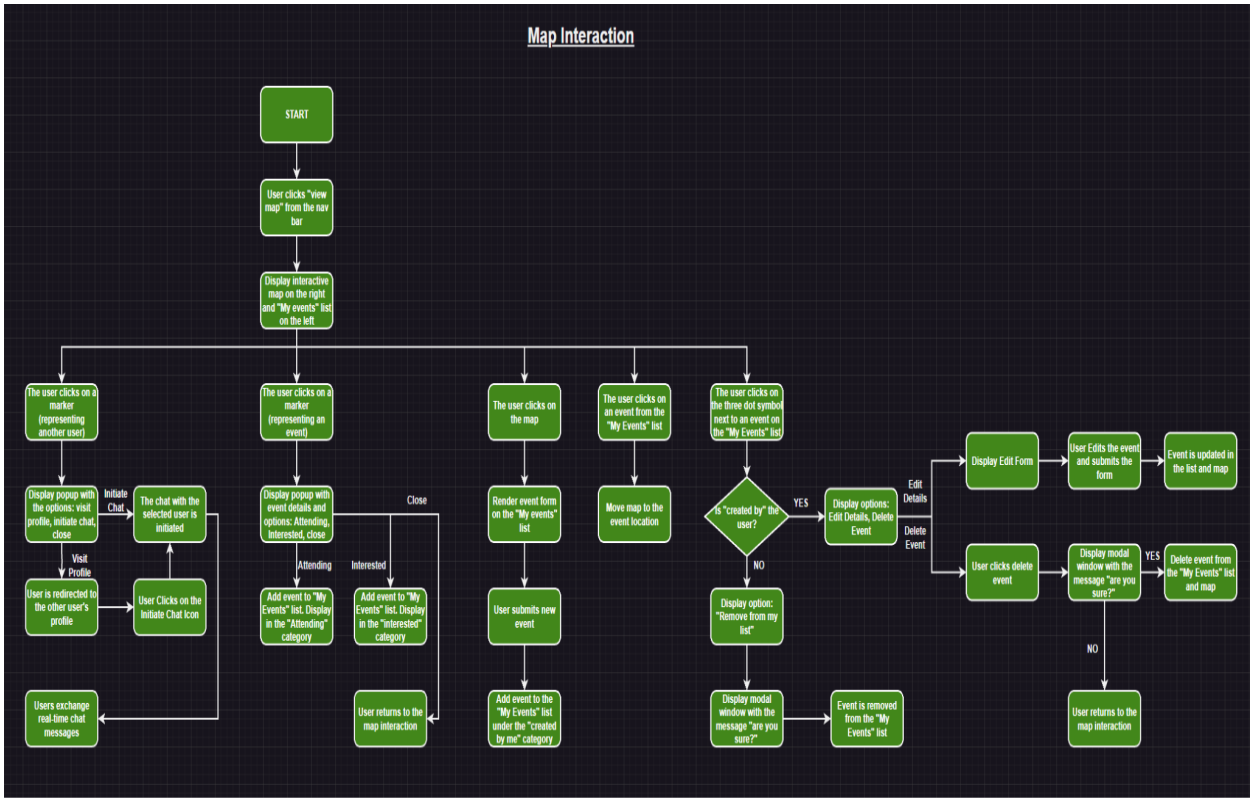
SITEMAP & FLOWCHARTS



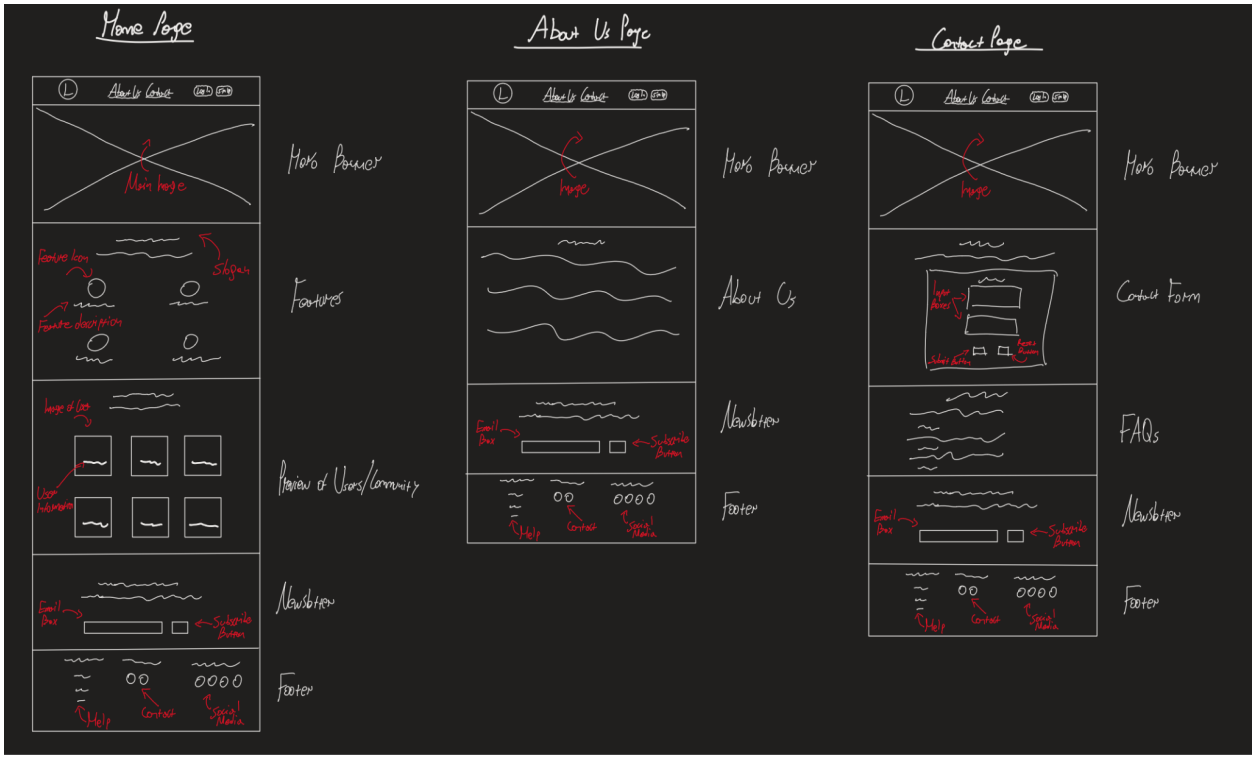

Chat Initiation
Start
Start from the Discover Page
Start from the Map Page
User Clicks on another user's profile
User clicks on another user's marker on the map
Close popup
User returns to the map interaction
User Clicks on the Initiate Chat Icon
Display popup with the options: visit profile, initiate chat, close popup
Initiate Chat
The chat with the selected user is initiated
Visit Profile
The chat with the selected user is initiated
User is redirected to the other user's profile
User Clicks on the Initiate Chat Icon
Users exchange real-time chat messages

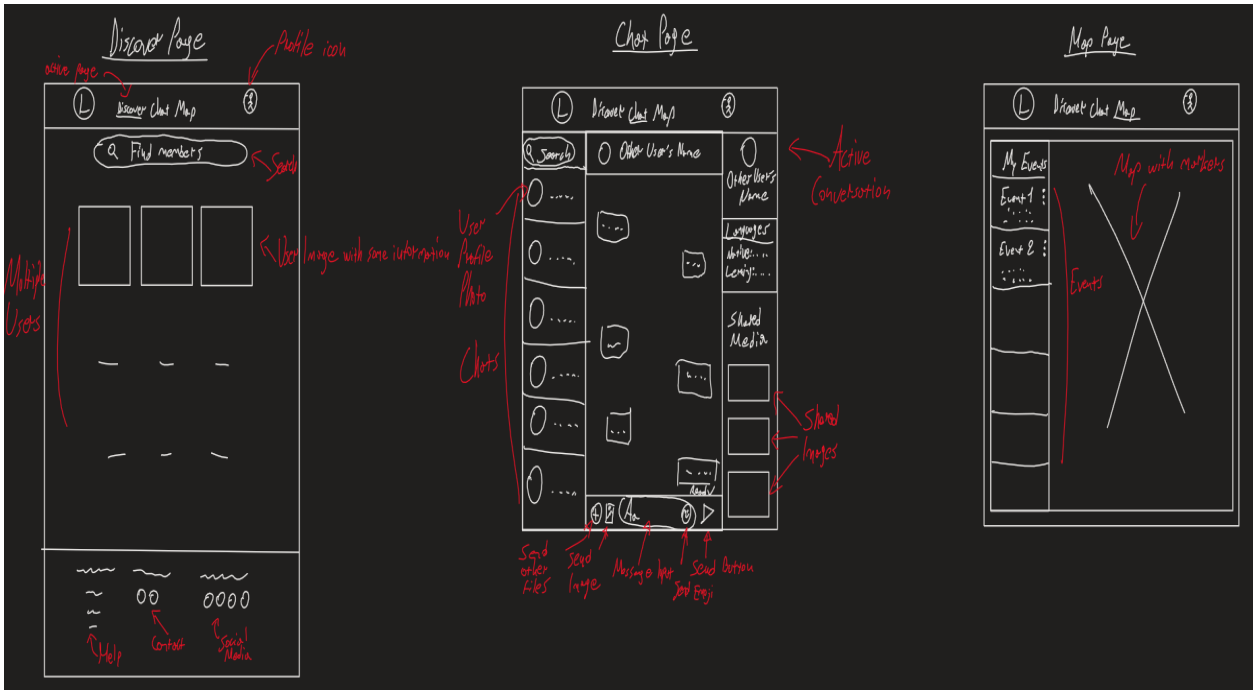

User Registration & Authentication Process
START
User clicks log in
User clicks sign up
User enters login credentials
User enters registration details
Validate Credentials
LOG IN & SIGN UP
LOG IN
Valid Credentials
Invalid Credentials
SIGN UP
Authenticate User + Log in
User Account Created
Display Error Message
Redirect User to Discover Page
Redirect User to Profile Creation
End


User Profile Management
START
The user clicks on "My Profile" to access their profile
The app displays the user's profile
The user clicks "Edit Profile" to make changes.
The profile editing form is displayed
The user submits changes
The user's profile is updated with the submitted changes
END


Map Interaction
START
User clicks "view map" from the nav bar
Display interactive map on the right and "My events" list on the left
The user clicks on a marker (representing another user)
Display popup with the options: visit profile, initiate chat, close
Initiate Chat
The chat with the selected user is initiated
Visit Profile
User is redirected to the other user's profile
User Clicks on the Initiate Chat Icon
Users exchange real-time chat messages
The user clicks on a marker (representing an event)
Display popup with event details and options: Attending, Interested, close
Close
Attending
Interested
Add event to "My Events" list. Display in the "Attending" category
Add event to "My Events" list. Display in the "interested" category
User returns to the map interaction
The user clicks on the map
Render event form on the "My events" list
User submits new event
Add event to the "My Events" list under the "created by me" category
The user clicks on an event from the "My Events" list
Move map to the event location
The user clicks on the three dot symbol next to an event on the "My Events" list
Is "created by" the user?
YES
NO
Display options: Edit Details, Delete Event
Edit Details
Delete Event
Display Edit Form
User Edits the event and submits the form
Event is updated in the list and map
User clicks delete event
Display modal window with the message "are you sure?"
YES
Delete event from the "My Events" list and map
NO
User returns to the map interaction
Display option: "Remove from my list"
Display modal window with the message "are you sure?"
Event is removed from the "My Events" list

WIREFRAMES


Home Page
About Us Page
Contact Page
Hero Banner
Features
Preview of Users/Community
Newsletter
Footer
About Us
Contact Form
FAQs



Tentative Schedule

* I can dedicate 25-40 hours per week to the project. *

Week 1 (September 11 - September 17): User Registration and Authentication

- Develop user registration form
- Implement server-side validation for user registration
- Implement user authentication mechanisms

Week 2 (September 18 - September 24): User Registration and Authentication (Continued)

- Create login functionality
- Develop user session management system

Week 3 (September 25 - October 1): User Profile Management

- Develop user profile form
- Implement server-side validation for profile data
- Develop file upload feature for profile pictures

Week 4 (October 2 - October 8): User Profile Management (Continued)

- Implement profile information storage

Week 5 (October 9 - October 15): Real-Time Chat Functionality

- Implement chat initiation mechanism
- Create chat window and real-time messaging

Week 6 (October 16 - October 22): Real-Time Chat Functionality (Continued)

- Develop chat conversation retrieval system
- Implement real-time notifications

Week 7 (October 23 - October 29): Location Sharing and Map Integration

- Develop location sharing feature
- Implement map integration with markers
- Enable marker click events

Week 8 (October 30 - November 5): Location Sharing and Map Integration (Continued)

- Develop privacy controls for location sharing

Week 9 (November 6 - November 12): Event Creation and Management

- Develop event creation form
- Implement event details storage

Week 10 (November 13 - November 19): Event Creation and Management (Continued)

- Develop event listing and RSVP functionality
- Implement event management

Week 11 (November 20 - November 26): Testing and Bug Fixing

- Comprehensive testing
- Identify and address bugs and issues

Week 12 (November 27 - December 3): Documentation and Deployment

- Write user documentation and guides
- Prepare for deployment, configure servers, and set up databases

Week 13 (December 4 - December 10): Final Testing and Deployment

- Final testing and quality assurance
- Deployment to production servers

Week 14 (December 11 - December 13): Final Preparations and Submission

- Final checks and preparations
- Submission of the project

Data Sources and Nature of Data

LinguaConnect relies on user-generated data. Users create profiles with information such as usernames, email addresses, profile pictures, bios, and language interests. They also generate data through chat conversations, event creation, event attendance, and location sharing.

Data Sources

- **Chat Messages**: Chat messages are user-generated content that includes message content, timestamps, and sender/receiver information. These messages are part of chat conversations between users.
- **Event Data**: Events are created by users and include information such as event titles, descriptions, dates, times, languages to be practiced, and organizer details. Users can attend or express interest in events, creating associations between events and users.
- **Event Categories**: Event categories define event types, such as "Created by User," "Attending," and "Interested." These categories help organize events.
- **Location Data**: Users can share their location. This data is used for the map integration to display nearby users and events.

Nature of Data

- **Structured User Data**: User data consists of structured information, such as usernames, emails, and profile details (name, bio, languages).
- **Unstructured Chat Messages**: Chat messages are unstructured text data, including user messages, timestamps, and sender/receiver details.
- **Structured Event Data**: Event data is structured and includes fields such as event titles, descriptions, dates, times, and language preferences. Event categories also provide structure for organizing events.
- **Geospatial Data**: Location data is in the form of geospatial coordinates (latitude and longitude) used for mapping and location sharing.
- **Relational Data**: The application manages relational data to establish relationships between users, events, and event categories. This includes many-to-many relationships between users and events, as well as events and event categories.
- **User Interaction Data**: Data related to user interactions with events, such as attendance status (attending, interested), is tracked to facilitate event management.