Quillyz

Social Network & Travel Web Application

Final Grade Project
Grade 2023-2024

Authors
Marcos Connolly Lopez
Steven Mallqui Aguilar

DIRECTOR
Ramón González del Campo Rodríguez Barbero

Grade in Computer Science
Grade in Software Engineering
Facultad de Informática
Universidad Complutense de Madrid

September 2024

Quillyz

Red social & Aplicación web de viajes

Trabajo Fin de Grado
Curso 2023-2024

Autores
Marcos Connolly Lopez
Steven Mallqui Aguilar

DIRECTOR
Ramón González del Campo Rodríguez Barbero

Grado en Ingeniería Informática
Grado en Ingeniería del Software
Facultad de Informática
Universidad Complutense de Madrid

Septiembre 2024

Quillyz

Social Network & Travel Web Application

Final Grade Project in Computer Science
Final Grade Project in Software Engineering

Autors
Marcos Connolly Lopez
Steven Mallqui Aguilar

DIRECTOR
Ramón González del Campo Rodriguez Barbero

September 2024

Grade in Computer Science
Grade in Software Engineering
Facultad de Informática
Universidad Complutense de Madrid

12 September 2024

DEDICATED

To our families, friends and those who are

no longer with us

THANK-YOU'S

A special thank you to our families, who stood by our side, supporting us from the very beginning of our academic journey and to their influence motivating us and giving us the opportunity, year after year, to pursue what we are most passionate about. Your trust has been invaluable in helping us achieve our goals, and for that, this work is dedicated to you.

We would also like to extend our gratitude to our advisor Rámon González, for guiding us and believing in the project from the moment we first shared our idea. This project has been made possible through many hours of work, late nights, and most importantly, a great deal of passion.

ABSTRACT

Quillyz is an innovative web application that merges travel planning with social networking. We initially noticed that the current group travel applications made no effort towards their trip recommendation. Where each individual user had to skim through every trip or manually, checking their details one by one.

After seeing this, we decided to offer them a different and simpler way to find trips, so that they're able to focus on enjoying the trip while making it easier to find friends along the way. Giving them the ability to join a trip that they'll fully enjoy or the chance to make their own, that is focused on grouping like minded people.

Additionally, we expanded it into a social network, to help develop communities and give them the ability to share and comment their experiences and follow each other's progress, helping them keep in contact and develop a meaningful long lasting relationship, while pushing them to keep travelling together and make new memories.

With this, we hope to give them a new and original application to travel.

Keywords: Social Network, Travel Planning, Web Application, Clustering Algorithms, Genetic Algorithms, Recommendation Algorithm, MERN, Posts Sharing, Data Analysis.

RESUMEN

Quillyz es una innovadora aplicación web que combina la planificación de viajes con una red social. Inicialmente notamos que las aplicaciones actuales de viajes en grupo no hacían ningún esfuerzo en cuanto a la recomendación de viajes. Cada usuario tenía que revisar manualmente cada viaje y verificar sus detalles uno por uno.

Después de observar esto, decidimos ofrecerles una manera diferente de encontrar viajes de una forma más sencilla, para que puedan enfocarse en disfrutar el viaje mientras les facilitamos encontrar amigos por el camino. Les damos la opción tanto de unirse a un viaje que disfrutarán como de crear uno propio, con un enfoque en agrupar personas con intereses afines.

Además, al expandir la plataforma hacia una red social, podemos ayudar a desarrollar comunidades y darles la capacidad de compartir y comentar sus experiencias, seguir el progreso de otros y mantener el contacto. Esto les permitirá desarrollar relaciones significativas y duraderas, motivándolos a seguir viajando juntos y creando nuevos recuerdos.

Con esto, esperamos ofrecerles una aplicación nueva y original para viajar.

Palabras clave: Red Social, Aplicación de Viajes, Aplicación Web, Algoritmos de Clustering, Algoritmos Geneticos, Algoritmo de Recomendación, MERN, Posts, Análisis de Información.

Content Index

Chapter 1 - Introduction **1**
1.1 Motivation 1
1.2 Objectives 2
1.3 Work Plan 3
Chapter 2 - State of the art **5**
Chapter 3 - Software Development Technologies **7**
3.1 Core Technologies 7
3.2 Development Tools 8
3.3 Relevant Dependencies and Libraries 9
Chapter 4 - System Design, Architecture, and Implementation **11**
4.1 System Architecture 11
4.2 Database Design 16
4.3 Frontend Implementation 19
4.4 Backend Implementation 24
4.5 Features Implemented 29
4.5.1 User Module 29
Login 29
Register 29
Profile 29
Change Profile Picture 29
Upload post 30
View Followers 30
View Following 30
4.5.2 Trip Module 31
Recommended Trips 31
Create Trip 31
Join Trip 31
Update - Cancel Trip 31

Disjoin Trip 31
4.5.3 Social Network Module 32
Search People 32
Chat 32
Trip Album 32
Chapter 5 - Recommendation Algorithm 33
5.1 Matchmaking Algorithm 34
5.2 Algorithm Design and Graphs 35
5.3 Omega : Algorithm Controller 36
5.4 Evaluation variants 36
5.5 Algorithm Design and Graphs 37
Chapter 6 - Conclusions and Future Work 38
6.1 Conclusions 38
6.2 Future Work 39
6.3 Continuous Improvement 41
Chapter 7 - User Experience and Feedback 42
7.1 Interviews & Questionnaires 42
7.2 Analysis of User Feedback 42
Chapter 8 - Personal Contributions 44
8.1 Contributions by Marcos: 44
8.2 Contributions by Steven: 44
Bibliography 45
Books 45
Online Resources 45
Technical Documentation 45
Web Screenshots Examples 47
Login 47
Register 48
Profile 48
Change Profile Picture 49
Upload post 50
View Followers 51
View Following 52
Recommended Trips 52

Create Trip 53
Update - Cancel Trip 54
Join Trip 54
Disjoin Trip 55
Search People 56
Chat 56
Trip Album 58
Algorithm Examples **60**
Alpha example 60
Gamma example 61
Omega Evolution Stages 62
Omega example 62
Questionnaire **63**
Tables **66**
Questionnaire 66

Graphs & Figures Index

Web Screenshots Examples **48**

Login 48

Register 49

Profile 50

Change Profile Picture 51

Upload post 51

View Followers 53

View Following 53

Recommended Trips 54

Create Trip 54

Update - Cancel Trip 56

Join Trip 56

Disjoin Trip 57

Search People 58

Chat 58

Trip Album 60

Algorithm Examples **62**

Alpha example 62

Gamma example 63

Omega Evolution Stages 64

Omega example 64

Questionnaire Diagrams 65

TABLE INDEX

Questionnaire 52

Chapter 1 - Introduction

The *Quillyz* project focuses on developing a web application that integrates travel planning with a social networking platform.

1.1 Motivation

In today's digital world, social networks and travel apps have become essential tools for planning and sharing travel experiences. While numerous online platforms have made travel more accessible, many still offer a limited and generic experience, primarily focusing on vacation packages that do not satisfy the unique preferences and interests of individual travelers. This lack of personalization presents a significant gap in the market, motivating us to develop a new type of travel application, one that blends personalized trip recommendations with a vibrant social networking component.

The influence of social networks has transformed how people connect and share experiences, including travel. However, most travel platforms do not effectively integrate social features, focusing instead on transactional functions like booking. Our application aims to bridge this gap by promoting a social environment where users can connect with like-minded travelers, share experiences, and collaboratively plan trips. This approach not only enhances user engagement but also creates a community-driven travel experience.

To offer a more personalized experience, our platform uses a recommendation algorithm that evolves based on users' past trips and preferences. As digital platforms increasingly rely on artificial intelligence for personalized content, such algorithms are essential for engaging users with relevant suggestions. By learning from user's travel history, preferences, and social interactions, our application provides customized recommendations, making each trip more engaging and tailored to individual interests.

We chose the MERN stack, a powerful combination of MongoDB, Express.js, React.js, and Node.js, to build this innovative platform. This technology stack is ideal for creating dynamic, responsive web applications. MongoDB offers a flexible database solution, Express.js and Node.js provide a robust backend framework, and React.js ensures a seamless, interactive user interface. The MERN stack enabled us to develop a scalable and efficient application that delivers personalized travel experiences and facilitates social interaction.

1.2 Objectives

Our travel application was developed with several key objectives in mind, all aimed at delivering a unique, user-centered experience. These objectives are divided into primary and secondary goals, each contributing to the overall vision of the project.

The primary objective of this project was to build a travel application that perfectly integrates personalized trip recommendations with social networking features, as outlined in the previous section. We wanted to build an application that not only suggests travel options based on the user's individual preferences and past experiences but also promotes a community where users can connect, share experiences, and plan trips together. By leveraging advanced algorithms and social features,our application aims to provide a more engaging and personalized travel planning experience than what's currently available.

A secondary goal was to experiment with and refine a robust recommendation algorithm suited for a travel-based social network. In this context, we chose the k-means clustering algorithm because it's effective at finding patterns in large datasets and grouping users with similar travel preferences. This approach allows the application to dynamically segment users based on their behavior and interests, enhancing the relevance of our travel suggestions. By focusing on a well-established algorithm like

k-means, we aim to achieve a balance between efficiency and recommendation quality.

In addition to these user-focused objectives, we also set several technical objectives to ensure the application meets high standards for usability, performance, and scalability:

- **Usability**: The application was designed with user-friendly interfaces and intuitive navigation to make it easy for everyone to explore, connect, and engage. Special attention was given to designing a responsive interface using React.js, ensuring the application works well across all devices and screen sizes.
- **Performance**: High performance was essential to ensure fast, responsive interactions, especially given the dynamic nature of our recommendation engine and the social networking features. We implemented various optimizations, such as database indexing in MongoDB and efficient server-side processing with Node.js and Express.js, to reduce latency and speed up the app.
- **Scalability**: Scalability was also a key consideration, as we wanted the application to handle a growing number of users and data. The application was built using the MERN stack, which provides a flexible, scalable architecture that can easily handle increased loads. The use of MongoDB's flexible schema design allows for efficient scaling of the database, while Node.js's event-driven architecture ensures the server can handle many simultaneous requests.

1.3 Work Plan

The development of our travel application was an iterative process that took advantage of the strengths and interests of our small team. Instead of following a formal development methodology like Agile or Scrum, we relied on a flexible, collaborative approach that allowed us to focus on the areas where we were most

skilled in and passionate about. This flexibility helped us to work efficiently and maintain a high level of motivation throughout the project.

Our work plan was organized around three key areas: modeling the application's data entities, designing the architecture and full-stack development, and developing the recommendation algorithm. This structure enabled us to cover both the technical and user experience aspects of the application, ensuring a well-rounded development process.

We started by defining the core data model that would support the application's functionality. This phase involved identifying the essential data structures and relationships needed for the application to function effectively. Establishing a clear and robust data model from the beginning provided a solid foundation for the rest of the development process. Before starting with the core development tasks, we first completed a project setup phase. This involved configuring the tools, environments, and frameworks needed for coding and testing. We set up version control, established coding standards, and selected the appropriate libraries. This preparation was essential for minimizing technical issues and ensuring a smooth, unified workflow. Once the setup was complete, we began working on the application's design and architecture alongside the development of the recommendation algorithm. The design phase involved planning the application's overall structure, including the database schema, API endpoints, and a cohesive front-end layout, aiming for a modular and scalable architecture that could support both current features and future enhancements. At the same time, we researched different algorithms and chose the k-means clustering algorithm, training and testing it locally to ensure it provided accurate trip recommendations based on user preferences and past behaviors. Throughout this process, we divided tasks according to our strengths but remained actively involved in all aspects of development. This collaborative approach allowed us to integrate the

full-stack architecture with the recommendation algorithm smoothly, resulting in a well-rounded and user-friendly application.

Chapter 2 - State of the art

The landscape of social network travel applications has evolved significantly, with several platforms offering a range of features tailored to different aspects of travel and social interaction:

- **Tripmakery**: This platform focuses on custom trip planning, allowing users to create and personalize their travel itineraries. However, its social features are limited, emphasizing itinerary management over encouraging user interaction.
- **Travello**: Travello combines travel planning with social networking by allowing users to connect with other travelers, share experiences, and receive local recommendations. Its social features are stronger compared to Tripmakery, but the platform is mainly centered around user-generated content and local tips rather than deep personalization.
- **Steller**: Known for its emphasis on visual storytelling, allowing users to create and share travel stories through rich media. Although it offers social features like following and commenting, it lacks advanced personalization in its recommendation system, focusing instead on the visual and narrative aspects of travel.
- **WeRoad**: WeRoad takes a different approach by focusing on group travel and community building. It organizes trips for groups and supports social interactions among travelers. Although it is effective in arranging group travel experiences, it lacks advanced personalized recommendations.

These platforms demonstrate the various ways social networking can be integrated with travel functionalities. However, there is a noticeable gap in combining robust

personalized recommendations with comprehensive social features, which is a key area of interest in current research and development.

Recommendation algorithms are essential for enhancing user experience on digital platforms by providing personalized content. Algorithms like collaborative filtering, content-based filtering, hybrid models, and clustering methods such as k-means are commonly used to group users with similar preferences and provide more accurate suggestions. Leading platforms like Netflix, Amazon, Spotify, YouTube, and Airbnb effectively utilize these techniques to boost user satisfaction through more targeted recommendations.

In the world of web development, the MERN stack, which includes MongoDB, Express.js, React.js, and Node.js, has become a go-to choice for developing dynamic and robust web applications. This stack is behind major platforms like Facebook, Instagram, and Airbnb. Social media platforms like Facebook use it to manage massive amounts of user data and deliver real-time interactions. E-commerce giants like Amazon and eBay utilize the stack to offer seamless shopping experiences and provide personalized recommendations based on user activity.

The appeal of the MERN stack lies in its use of JavaScript across both client and server sides, which streamlines development and boosts productivity by minimizing the need to switch between different programming languages. MongoDB's flexible architecture and Node.js's non-blocking I/O make the stack particularly scalable, capable of managing large datasets and high traffic volumes efficiently. The stack's versatility is evident in its wide adoption across various applications, from project management tools like Trello to streaming services like Netflix, highlighting its ability to support a diverse range of robust web solutions.

Chapter 3 - Software Development Technologies

3.1 Core Technologies

MongoDB: MongoDB is a NoSQL database that we used to manage and store diverse and complex data structures, such as user profiles, travel itineraries, and interactions. Its schema-less design provided us with flexibility in data modeling, allowing for efficient storage and retrieval of information. MongoDB's ability to handle large volumes of data was perfect for the scalable and dynamic nature of our application.

Express.js: We used Express.js, a web application framework for Node.js, to build and manage RESTful APIs within our application. It made handling HTTP requests, routing, and integrating middleware straightforward, ensuring smooth communication between the client-side and server-side components.

React.js: React.js, a popular JavaScript library, was used to develop the user interface of our application. It allowed us to create reusable UI components and efficiently render dynamic content. Its component-based architecture made the UI more dynamic and responsive, supporting single-page application functionality and enhancing overall usability.

Node.js: Node.js served as the JavaScript runtime for executing server-side code in our application. We employed its non-blocking, event-driven architecture to handle multiple user requests simultaneously, which was vital for maintaining performance.

WebSockets: We implemented WebSockets to support real-time features like live chat and notifications. By providing a persistent connection between the client and server, WebSockets made the app more interactive and engaging by instantly updating users with new information.

Python: We also used Python to develop the recommendation algorithm within our application. Python's extensive libraries and frameworks for data analysis and machine learning made it ideal for implementing sophisticated recommendation systems.

Flask: Flask is a lightweight Python web framework that we used to connect our backend with the Python service running the recommendation algorithm. It facilitated communication between our Node.js backend and the Python-based recommendation engine, ensuring that data and recommendations were efficiently passed between systems.

3.2 Development Tools

Vite: Vite is a modern build tool that provides a fast development environment for web applications. We used it in our project to improve the development workflow by providing instant hot module replacement (HMR). This feature allowed us to see changes in the code in real-time, directly in the browser, saving a lot of time during development and debugging.

Docker: Docker is a platform that enables developers to create, deploy, and run applications in containers, which are lightweight, standalone, and executable packages of software. By containerizing our application, we avoided the usual "it works on my machine" problems. Docker simplified our development process and made it more predictable, giving us peace of mind that the application would behave the same way across different environments.

Git: Git is a distributed version control system that allows multiple developers to work on the same project efficiently. It helped us keep track of changes, collaborate more effectively, and maintain a comprehensive history of our work. The ability to manage branches and merge changes was particularly helpful in ensuring our code remained stable and up-to-date.

GitHub: GitHub is a cloud-based hosting service for Git repositories. We used GitHub to store our codebase and facilitate collaboration. It provided version control, issue tracking, code review, and project management features, which helped us coordinate tasks, review code changes, and maintain the overall quality of the project.

NPM (Node Package Manager): NPM is the default package manager for Node.js, allowing developers to install and manage libraries and dependencies for their projects. We used NPM to handle all the JavaScript libraries and frameworks required for our application. It simplified the process of updating packages and maintaining a clean project structure, which is crucial for long-term maintainability.

Postman: Postman is a popular API testing tool that allows developers to design, test, and document APIs. We used Postman extensively during the development phase to test the RESTful APIs created with Express.js. This tool helped us simulate HTTP requests, validate responses, and debug issues with server-side logic, ensuring robust and reliable API functionality.

Visual Studio Code: Visual Studio Code (VS Code) is a lightweight yet powerful source code editor. Its speed, versatility, and wide range of extensions made it an ideal choice for us. We appreciated its built-in support for debugging, linting, and version control, which made development more efficient.

Tailwind CSS: Tailwind CSS is a utility-first CSS framework that provides low-level utility classes for building custom designs. It was particularly helpful in maintaining a consistent design across the application and keeping our stylesheets clean and manageable.

3.3 Relevant Dependencies and Libraries

Zod: Zod is a TypeScript-first schema validation library. We used it to validate and ensure the integrity of client-side data structures in our application.

Axios: Axios is a promise-based HTTP client for making server requests. It simplified our handling of API requests and responses, making data fetching and integration straightforward.

JWT (JSON Web Tokens): JWT is a compact, URL-safe token format for securely transmitting information between parties. We used it for user authentication and session management, allowing secure access control across our application.

JS-Cookies: JS-Cookies is a library for managing cookies in the browser. It helped us store user-specific data, like authentication tokens and preferences, ensuring a consistent user experience.

Multer: Multer handles file uploads in multipart/form-data. We used it to manage image uploads from users.

Zustand: Zustand is a small, fast, and scalable state management solution for React. We used it to manage global state efficiently, thanks to its simple API and performance benefits.

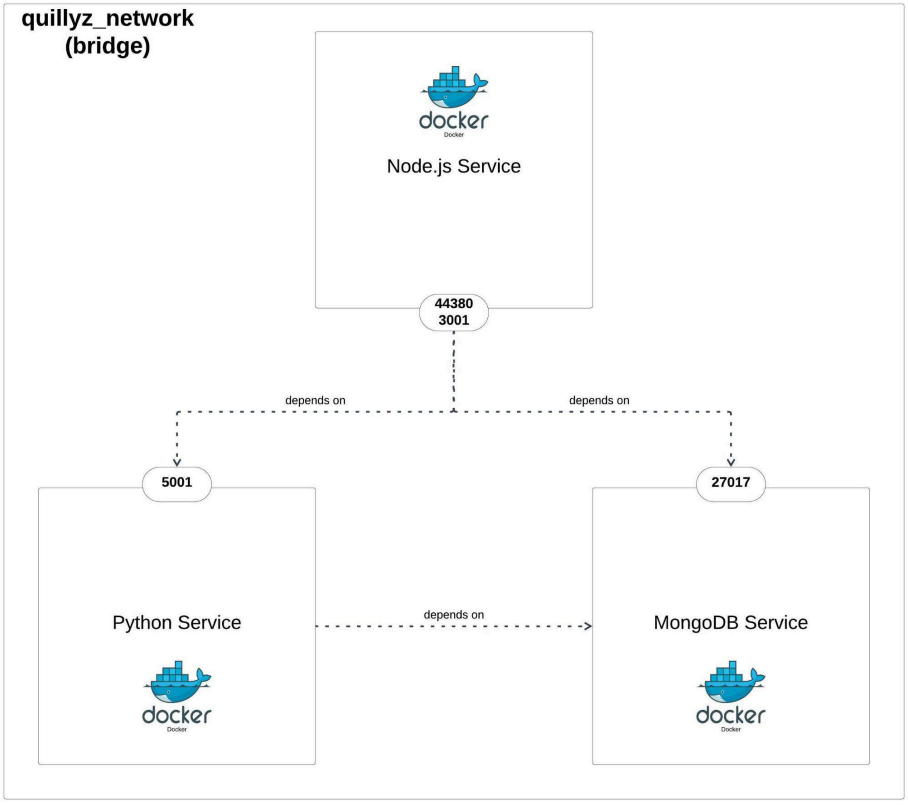
pymongo: PyMongo is a Python driver for MongoDB. It facilitated interaction with MongoDB in our Python-based recommendation algorithm, allowing for efficient data retrieval and manipulation.

numpy: NumPy is essential for scientific computing in Python. We used it for numerical operations and data processing in our recommendation algorithm, leveraging its array-handling and mathematical functions.

Sklearn: An essential library used in one of our algorithms, that contains clustering functions and data examples used for testing purposes.

Chapter 4 - System Design, Architecture, and Implementation

4.1 System Architecture



When designing our application, we wanted to make sure that our development and testing environments were consistent and reliable. To achieve this, we decided to dockerize our application. Docker made it easier for us to bundle all the necessary software components, dependencies, and configurations into portable containers. This approach not only allowed us to work smoothly across different operating systems, like macOS and Linux in our case, but also simplified our deployment process, making it more manageable and predictable.

Why We Chose to Dockerize Our Application:

- **Consistency Across Environments**: One of the biggest benefits of using Docker was that it gave us a consistent development environment. With Docker containers, we were able to create an environment that works the same way everywhere, whether it's on our local machines or on a production server.
- **Isolation and Security**: Each component of our application runs in its container, isolated from the others. This isolation provides a layer of security, as processes are contained within their environment and cannot interfere with each other. It also allows us to update or modify a specific component without affecting the rest of the system.
- **Development Flexibility**: Another big advantage of Docker was the flexibility it gave us in using different technologies. For example, we used Node.js for most of our application logic and Python (with Flask) for our recommendation algorithm. Docker made it easy to run these different components together and ensured they could communicate effectively.

To manage all these containers, we used Docker Compose, which helped us define and run our multi-container application. In our docker-compose.yml file, we set up the following services:

- **MongoDB**: We used MongoDB as our NoSQL database to manage and store diverse data structures. The MongoDB service is set up with a specific image (mongodb/mongodb-community-server:6.0-ubi8), and we mapped the default MongoDB port 27017 to ensure proper communication. The data is stored persistently using Docker volumes, so the data is not lost when the container is stopped or removed.
- **Python Service**: This service runs the Python environment necessary for our recommendation algorithm. The Python container uses the python:3.10-alpine image. We configured the Python service to install required dependencies (Flask,

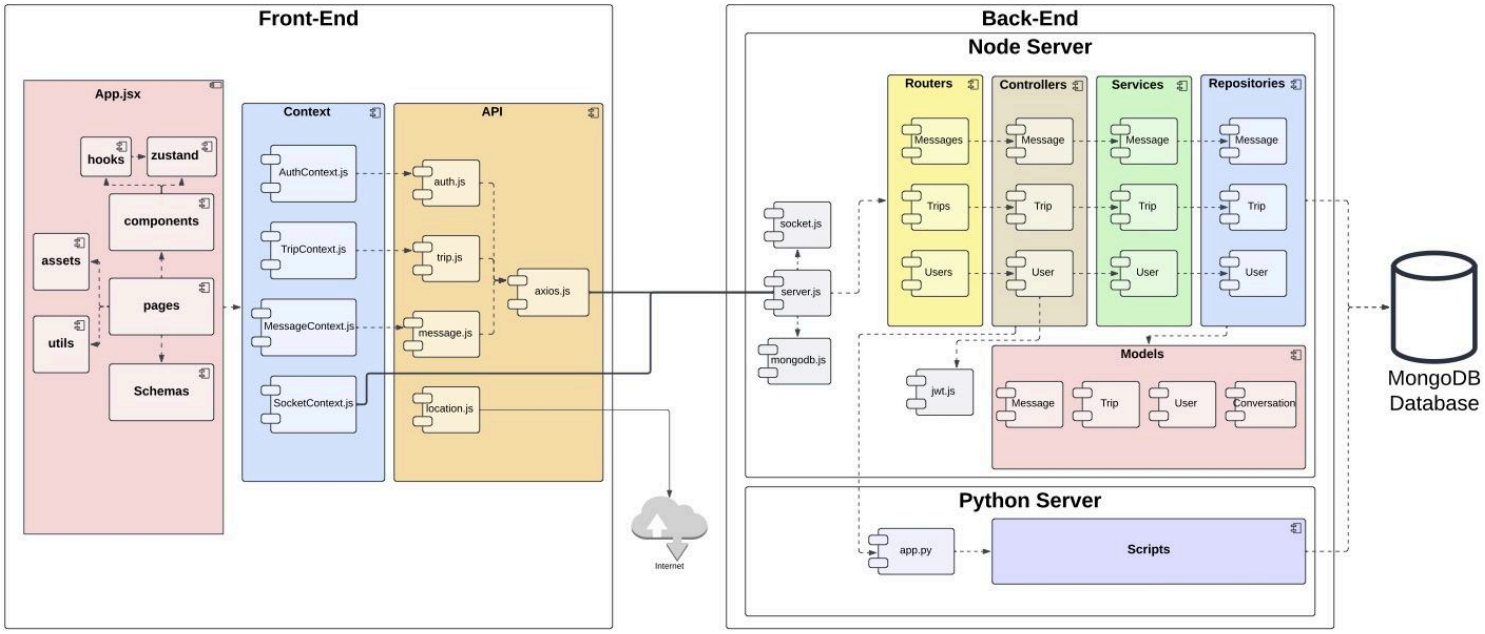
geopy, pymongo, numpy, sklearn, typing) and execute the recommendation algorithm with Flask providing a simple web server to handle requests from our Node.js backend. The Python service communicates with MongoDB to fetch and process data for generating recommendations.

- **Main Application (Node.js)**: The main service for our application, developed in Node.js, handles the core logic and serves the frontend built with React. We used the node:alpine image for a minimal Node.js environment. The application is designed to interact with both the MongoDB database and the Python service, facilitating data exchange and displaying recommendations to the user. We exposed ports 44380 and 3001 to handle HTTP and WebSocket connections, respectively.

All these services are connected through a Docker network (quillyz_network), which allows them to communicate with each other smoothly. We managed environment variables using an .env file to store sensitive information, such as database credentials and configuration settings. This approach kept sensitive data secure and allowed us to change configurations without directly modifying the source code.

We also used Docker volumes for data persistence and bind mounts for the Python and Node.js services, linking our local file system to the containers. This setup allowed us to see real-time code updates during development.

By using Docker and Docker Compose, we created a flexible, scalable, and reliable architecture that meets our current needs and can easily adapt to future changes. Docker helped us focus more on building and refining the features of our application without worrying about environment-specific issues or dependency conflicts.



Our application's architecture is thoughtfully designed to balance flexibility, scalability, and maintainability. At the core of our setup, we have a React-based frontend. We chose React because its component-based structure allows us to build reusable UI elements that come together seamlessly to form dynamic, user-friendly pages. In React, we utilize contexts to manage global state efficiently and establish connections with the backend. These connections are made through either Axios for handling standard HTTP requests or WebSockets when we need real-time, two-way communication between the client and the server. On the backend, we opted for Node.js, structured in a layered architecture that keeps our codebase clean and organized. Everything starts with the server, which brings in different routers. These routers are the entry points for incoming requests and are mapped to specific controllers. The controllers play a critical role, they handle the logic of processing requests and determining responses. But they don't do it alone. They rely on services, which encapsulate the core business logic. These services then call upon repositories, which are solely responsible for interacting

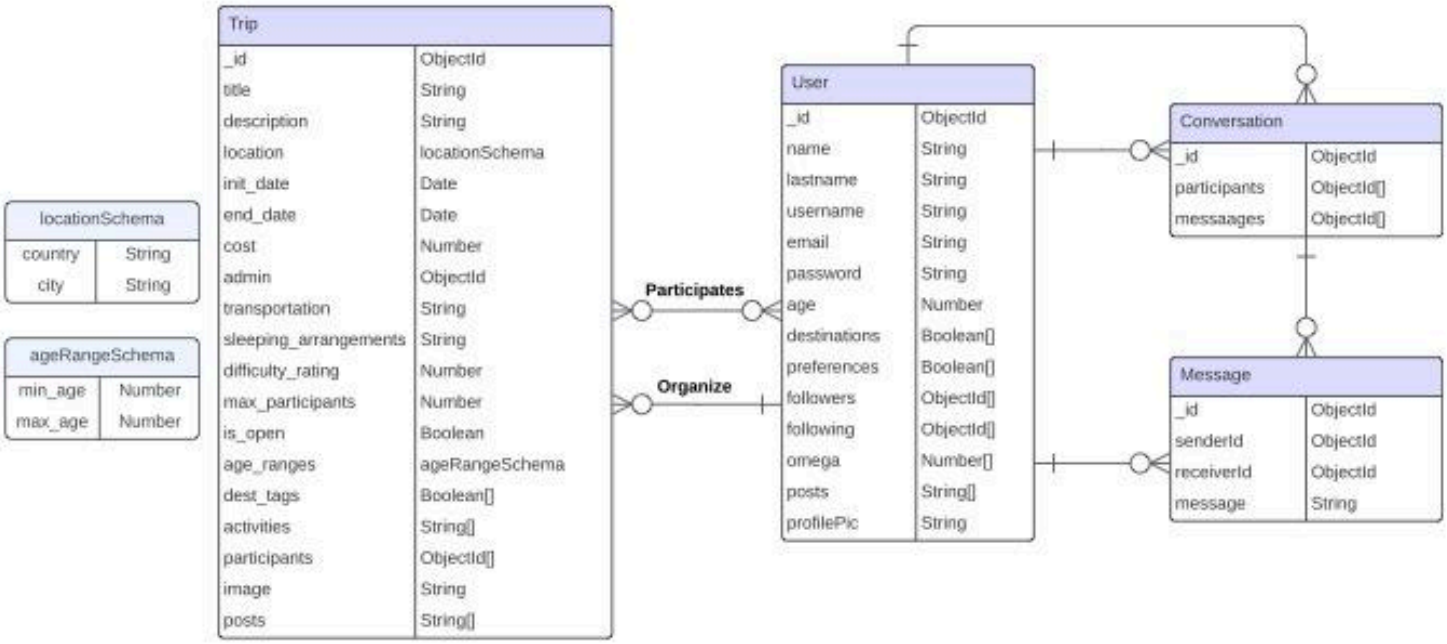
with our MongoDB database. This structure not only ensures a clear separation of concerns but also makes our backend more modular and easier to maintain.

To enhance the application's functionality, particularly in delivering personalized content, we integrated a Python-based recommendation system. This service, developed separately, is powered by Flask, which acts as a bridge between the Python environment and the Node.js backend. Through Flask, our Python service connects to the same MongoDB database, ensuring a unified data flow across the entire application. This setup allows the recommendation engine to access all the necessary data, run its algorithms, and provide personalized suggestions directly to the user interface.

Speaking of data, we kept things streamlined with just four core models: Message, Trip, User, and Conversation. This decision was intentional, focusing on simplicity and efficiency.

Overall, our architecture reflects a careful balance between using the right technologies and structuring the system in a way that is both robust and adaptable.

4.2 Database Design



Our application relies on four key data models: User, Message, Conversation, and Trip. Each of these models represents a different aspect of the app, but they're all connected to capture the relationships between users, their conversations, and the trips they organize or join.

User Model

At the heart of our application is the User model. It captures all the necessary information about a user, such as their personal details (name, lastname, username, email, etc.), credentials (password), and preferences (destinations, preferences, omega). Additionally, it includes fields that represent social features, such as followers and following, which are self-referential relationships. Each follower or following is represented as an ObjectId that references another user within the same User collection. Users also interact with other parts of the app. They can send and receive messages, participate in conversations, and create or join trips. These interactions are

captured by referencing the User model in other models, making it easy to track who is doing what within the application. For example, when a user joins a trip, their ID is added to the participants field in the Trip model, creating a direct link between the user and the trip.

Trip Model

The Trip model is designed to encapsulate all the information related to a travel experience. This includes basic details like the trip's title, description, location, and dates, as well as logistical information such as cost, transportation, sleeping arrangements, and difficulty rating. A trip is created and managed by a user, indicated by the admin field, which references the User model. This helps us keep track of who is responsible for managing each trip. Trips also involve multiple participants, which are again referenced by their user IDs. This allows us to see who is going on which trips, making it easy to pull up all the trips a particular user is participating in or managing. Additionally, trips can have associated posts, images, and activities that provide more context and engagement within the app.

Message and Conversation Models

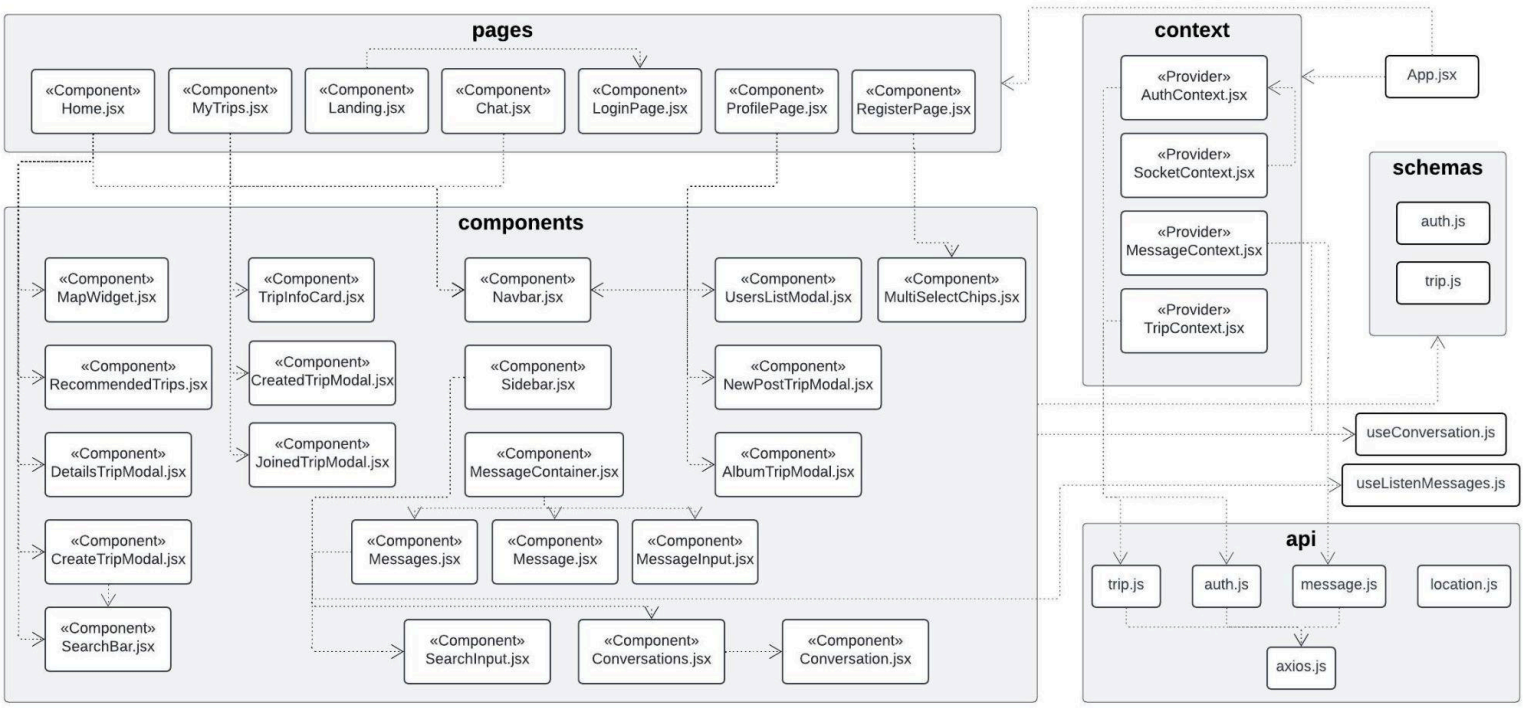
Communication between users is a crucial part of our app, and that's where the Message and Conversation models come into play. The Message model captures the content of each message, the sender, and the receiver, linking directly to the User model through the senderId and receiverId fields, ensuring that we can always see who was involved in any conversation. The Conversation model acts as a thread that groups related messages together. It keeps track of all the participants involved in a conversation and all the messages exchanged. This structure allows us to efficiently retrieve all the messages in a conversation and display them in the correct order.

The models are interconnected in a way that reflects the natural relationships in the application. For example:

- **Trips and Users (Many-to-Many Relationship)**: Users can join multiple trips, and each trip can have multiple participants. This many-to-many relationship between Trips and Users is managed using arrays of ObjectId references in the Trip model. Each Trip document includes an array of ObjectId references to the User documents representing the participants.
- **Trips and Organizers (Many-to-One Relationship)**: While multiple users can participate in a trip, each trip is organized by a single user. This is a many-to-one relationship where each Trip document contains a reference to a single User who is the organizer, represented by an ObjectId.
- **Users and Conversations (One-to-Many Relationship)**: Each user can participate in multiple conversations, but they only have one conversation with any specific other user. This relationship is managed in the Conversation model, where each conversation document contains references to User documents.
- **Conversations and Messages (One-to-Many Relationship)**: Within each conversation, multiple messages can be exchanged over time. This is a one-to-many relationship, where each Conversation document includes an array of ObjectId references to the Message documents that belong to that conversation.

Our database design relies heavily on a reference model approach, where documents are connected through references to other documents rather than embedding data within a single document. This approach offers several advantages for our application's architecture, especially considering the interconnected nature of our data.

4.3 Frontend Implementation



The frontend of our application was designed with a modular architecture to ensure a dynamic and responsive user experience. We chose React.js to build the user interface, which allowed us to create reusable components and manage the application's state effectively. Here's a closer look at how we implemented the frontend:

Component-Based Architecture

Our frontend is built around a component-based structure, where each page and feature is developed using React components. These components are responsible for rendering specific parts of the user interface and are designed to be reusable across different parts of the application.

Contexts and State Management

To manage the application state and handle data fetching, we used React's Context API. Contexts provide a way to pass data through the component tree without having

to manually pass props at every level. This approach simplifies state management and improves code efficiency.

Data Fetching and API Integration

For communication between the frontend and backend, we used Axios, a promise-based HTTP client. Axios simplifies the process of making HTTP requests and handling responses. It allowed us to perform many operations, such as retrieving user data, submitting forms, and interacting with other API endpoints.

WebSocket Integration

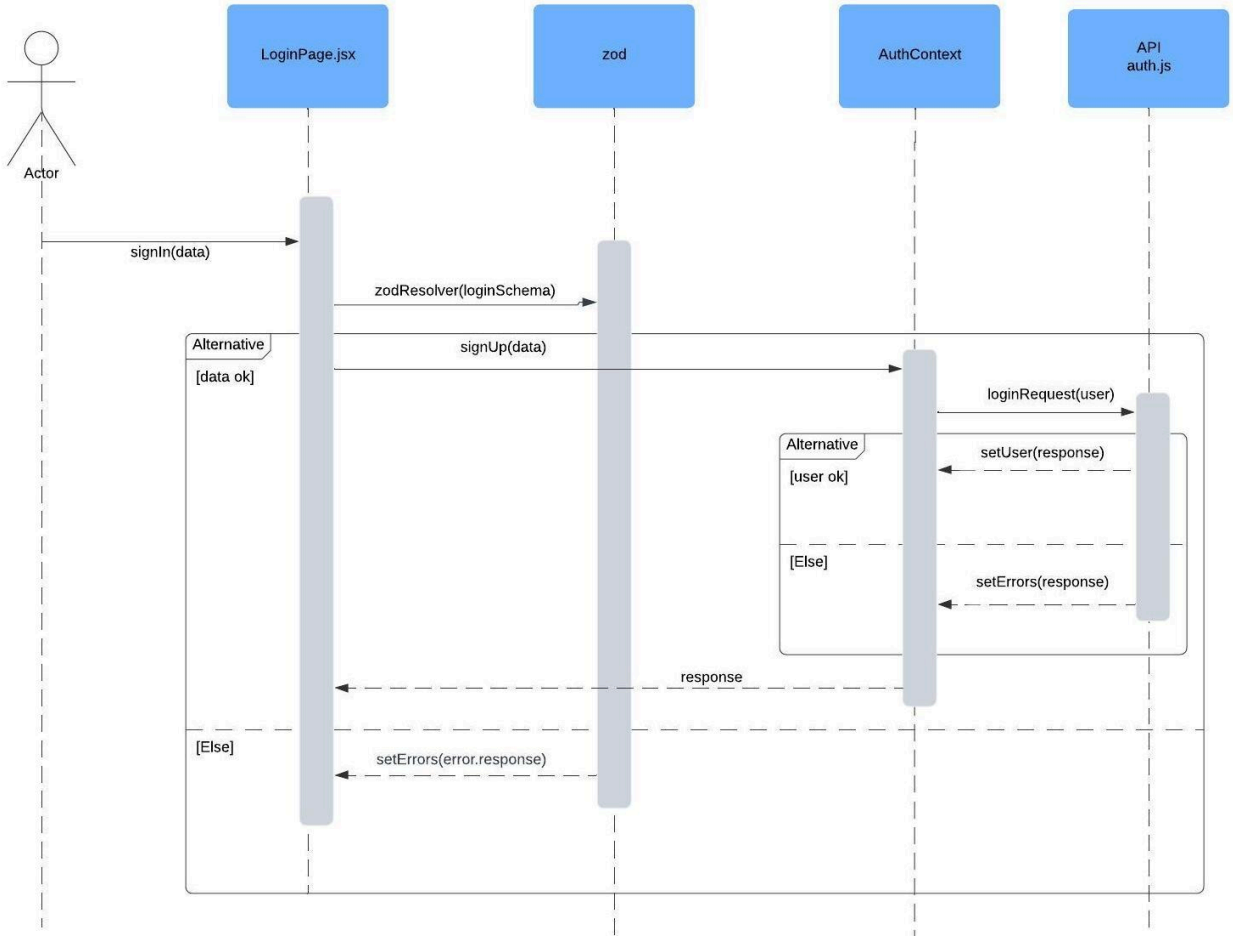
To enable real-time features such as chat functionality, we integrated WebSockets into our frontend. WebSockets provide a persistent connection between the client and server, allowing for two-way communication. Unlike traditional HTTP requests, which are stateless and involve a new connection for each request, WebSockets maintain an open connection, enabling the server to push updates to the client instantly.

Data Validation

Client-side validation was implemented using schemas to ensure data integrity and enhance the user experience. Schemas define the structure and constraints of data inputs, allowing us to validate user input before it is submitted to the server. This pre-validation helps catch errors and enforce correct data formats, providing immediate feedback to users and reducing the likelihood of invalid data reaching our backend.

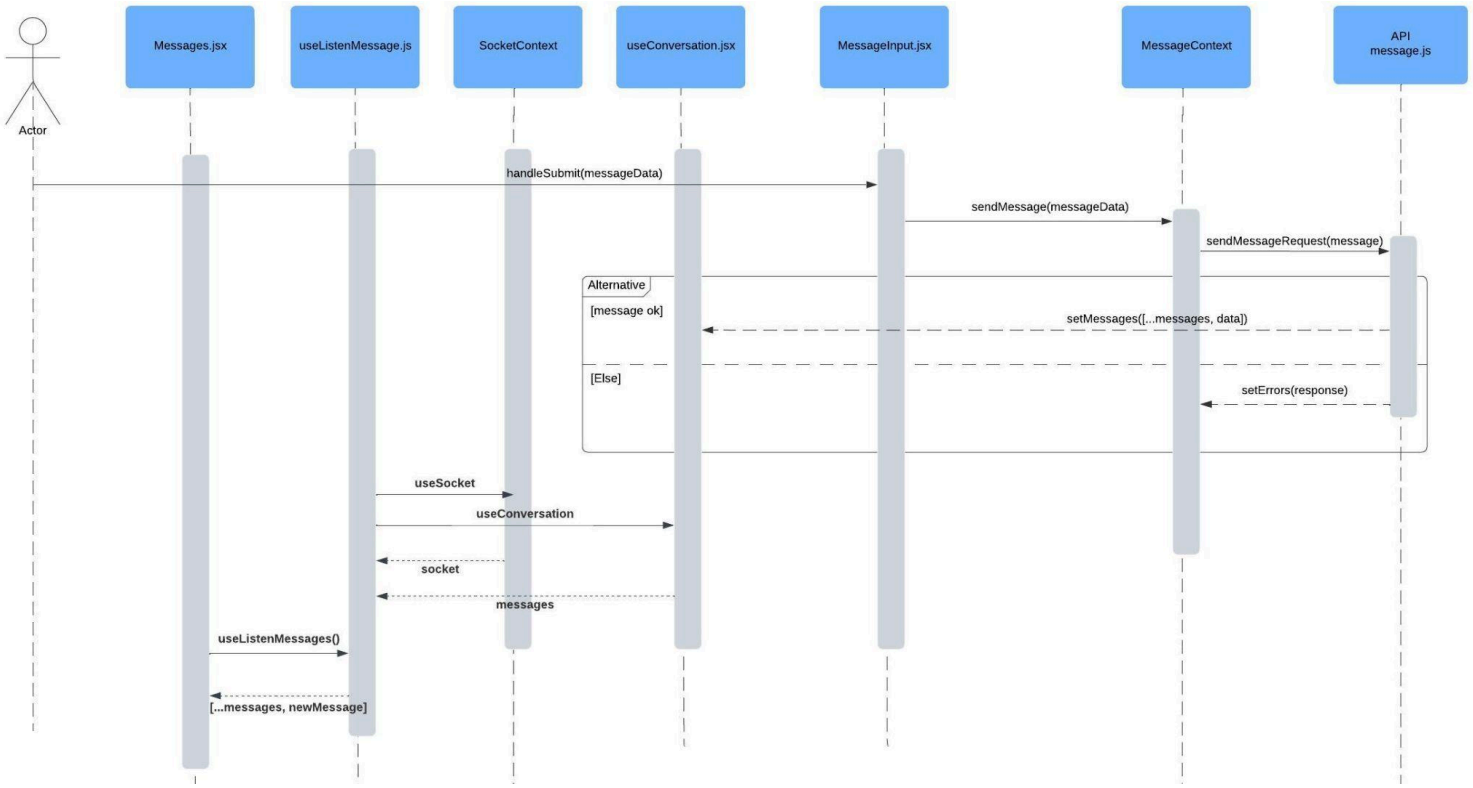
Here's how the frontend operation flows work for the three examples:

Login Request



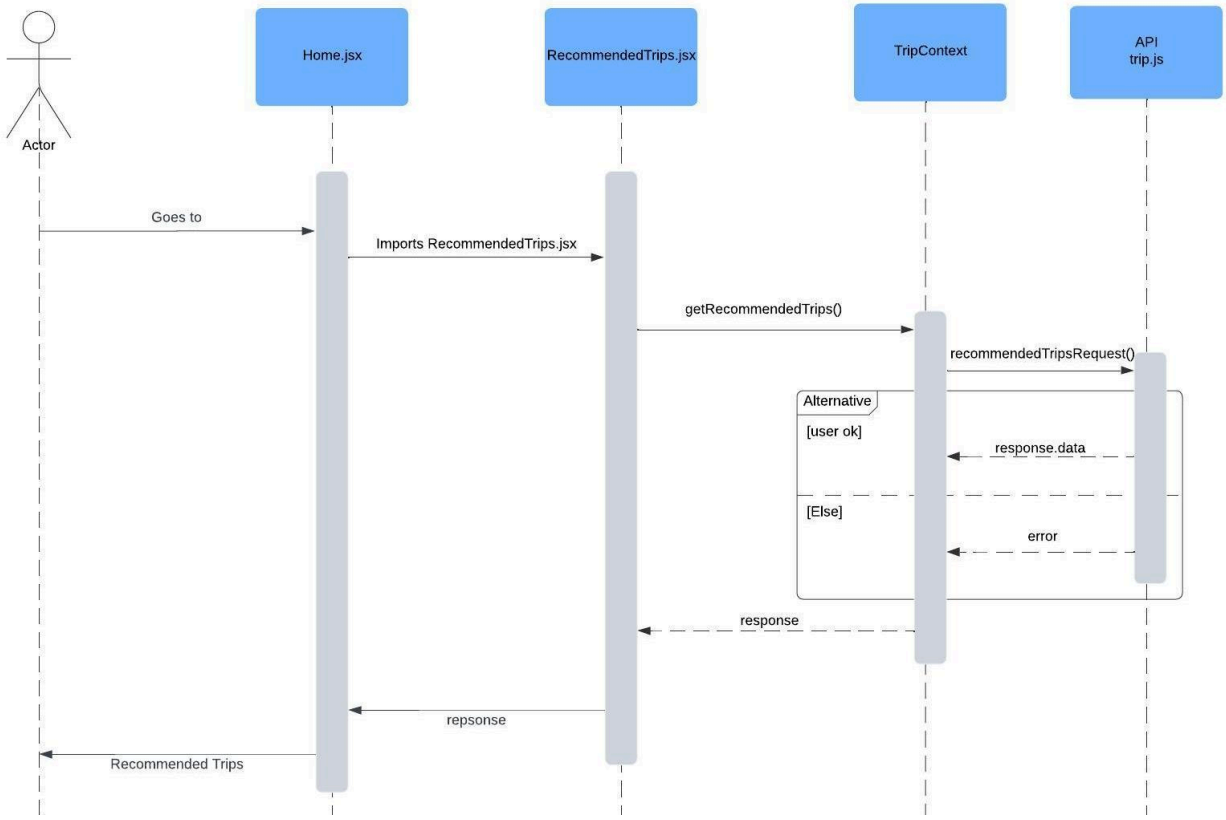
When a user attempts to log in, they first enter their credentials into the login form. The process starts with Zod performing client-side validation on the input fields. If the inputs pass validation, the signUp method from the AuthContext is called. This method then triggers an HTTP request using Axios to the backend's login endpoint. Upon a successful response, the user's information is stored in the AuthContext, which manages the user's authentication state and can be accessed throughout the application.

Send Message



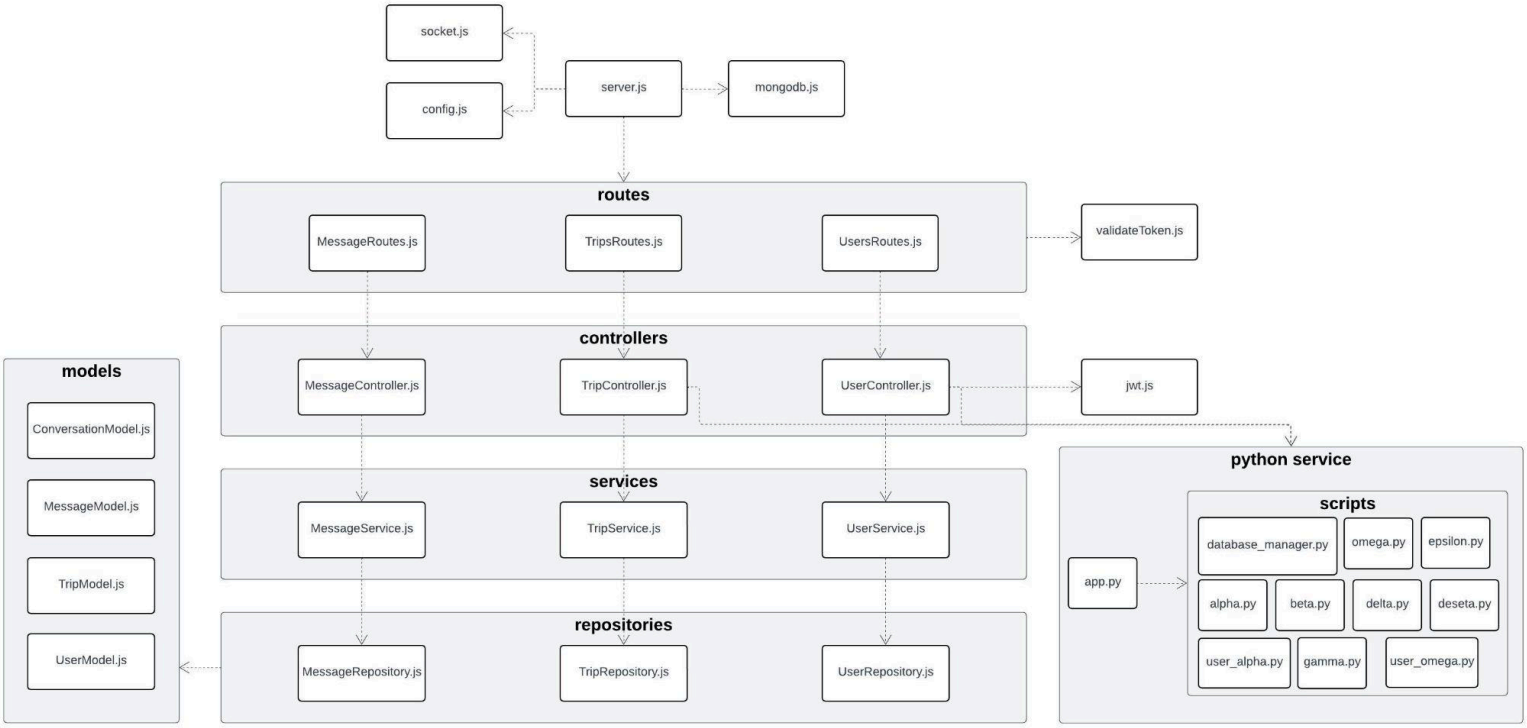
When sending a message, the user types in the MessageInput component and submits it. The sendMessage method from MessageContext takes the message data and calls the backend API using Axios with sendMessageRequest. Upon a successful response, the new message is returned and added to the conversation state managed by useConversation and Zustand. To display the updated messages in real-time, useListenMessage hook is used. This hook listens for new messages via WebSockets through SocketContext, updates the conversation state, and ensures the latest messages are visible in the UI.

Recommended Trips



For recommending trips, when the home page loads, the RecommendedTrips component triggers a useEffect hook that calls getRecommendedTrips from TripContext. This function makes an Axios call to recommendedTripsRequest to fetch the recommended trips. Once the trips are successfully retrieved, they are rendered in the user interface, providing the user with suggested travel options based on the backend data.

4.4 Backend Implementation



The backend of our application is designed with a clear, organized structure to keep things manageable and efficient. It all starts with server.js, which acts as the main entry point for our API. This file sets up the server and configures essential middleware to handle HTTP requests, manage sessions, and ensure security.

Server Initialization and Middleware Configuration

In server.js, the first step is to import socket.js. This file is crucial because it sets up our Express app with Socket.IO, which allows us to handle real-time communication, like chat messages, alongside standard HTTP requests. After setting up the Express app, we configure several middlewares:

JSON Parser: This handles any incoming data in JSON format, making it easy to work with.

CORS (Cross-Origin Resource Sharing): This is important for security and allows us to specify which domains can interact with our API.

Cookie Parser: This middleware helps us manage cookies, which are essential for maintaining user sessions.

Once everything is configured, the server starts listening for incoming requests and establishes a connection to our MongoDB instance, which runs in a Docker container. This setup ensures that our backend is ready to handle all sorts of requests efficiently and securely.

Routing and Controllers

Our backend uses a layered structure to organize the API logic. Routing is handled by separate route files that define endpoints for different parts of the application, such as authentication, user management, messaging, and trips. Each route is associated with a controller function that handles the specific business logic for that route.

The controllers serve as the middle layer, receiving requests from the routes, processing them, and preparing a response. If a request needs special handling, like uploading a file or checking if a user is logged in, the controller will use custom middleware functions like multer or authRequired.

Services and Repositories

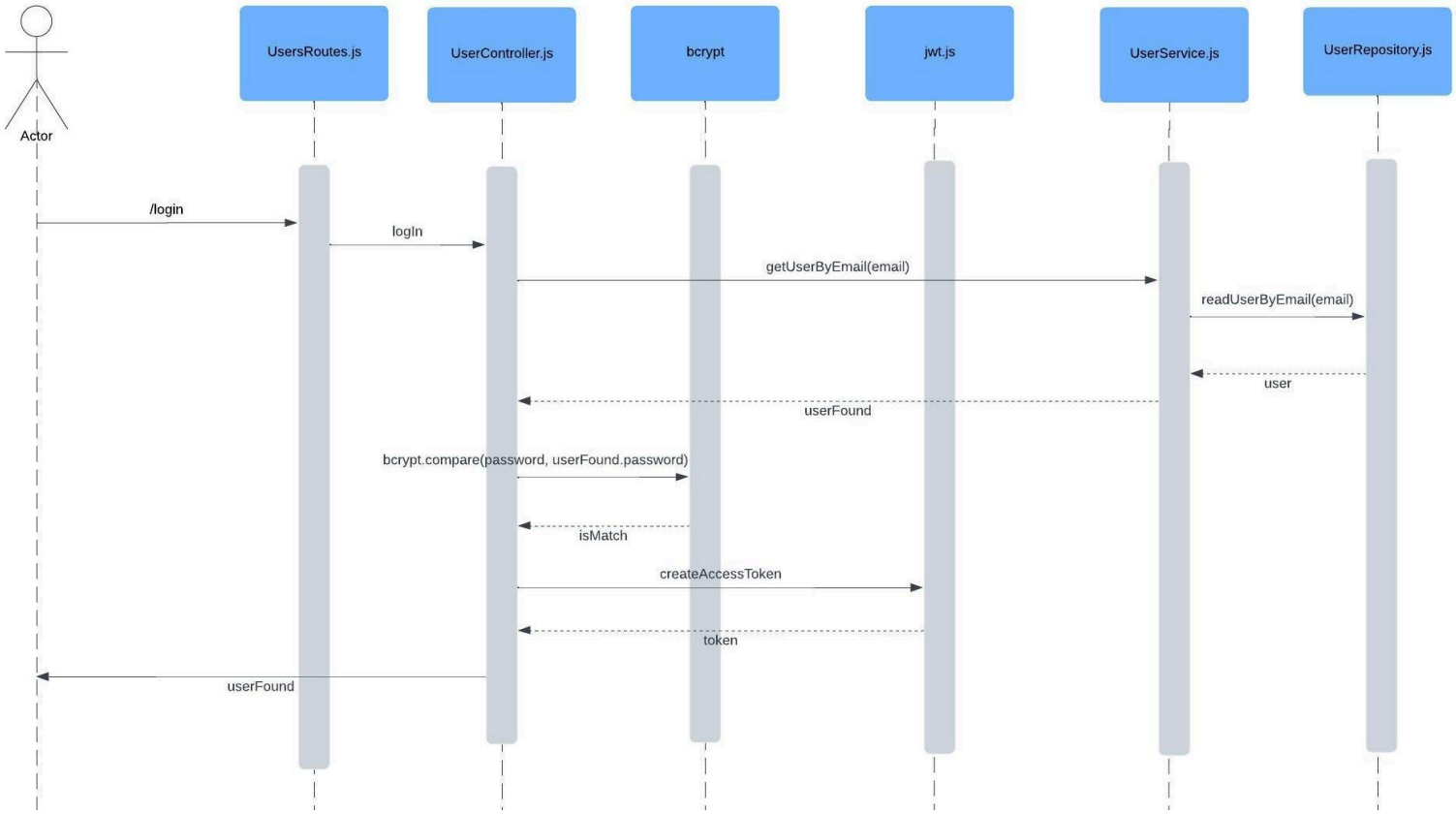
Controllers delegate the core business logic to services, which are responsible for processing data and implementing the rules of the application. When it comes to interacting with the database, services call repositories, which are responsible for the actual data operations.

The repository layer interacts with the MongoDB database using Mongoose, an Object Data Modeling (ODM) library for MongoDB and Node.js. Mongoose allows us to define schemas for our data models, ensuring data consistency and providing an API for CRUD operations.

Integration with Python Service

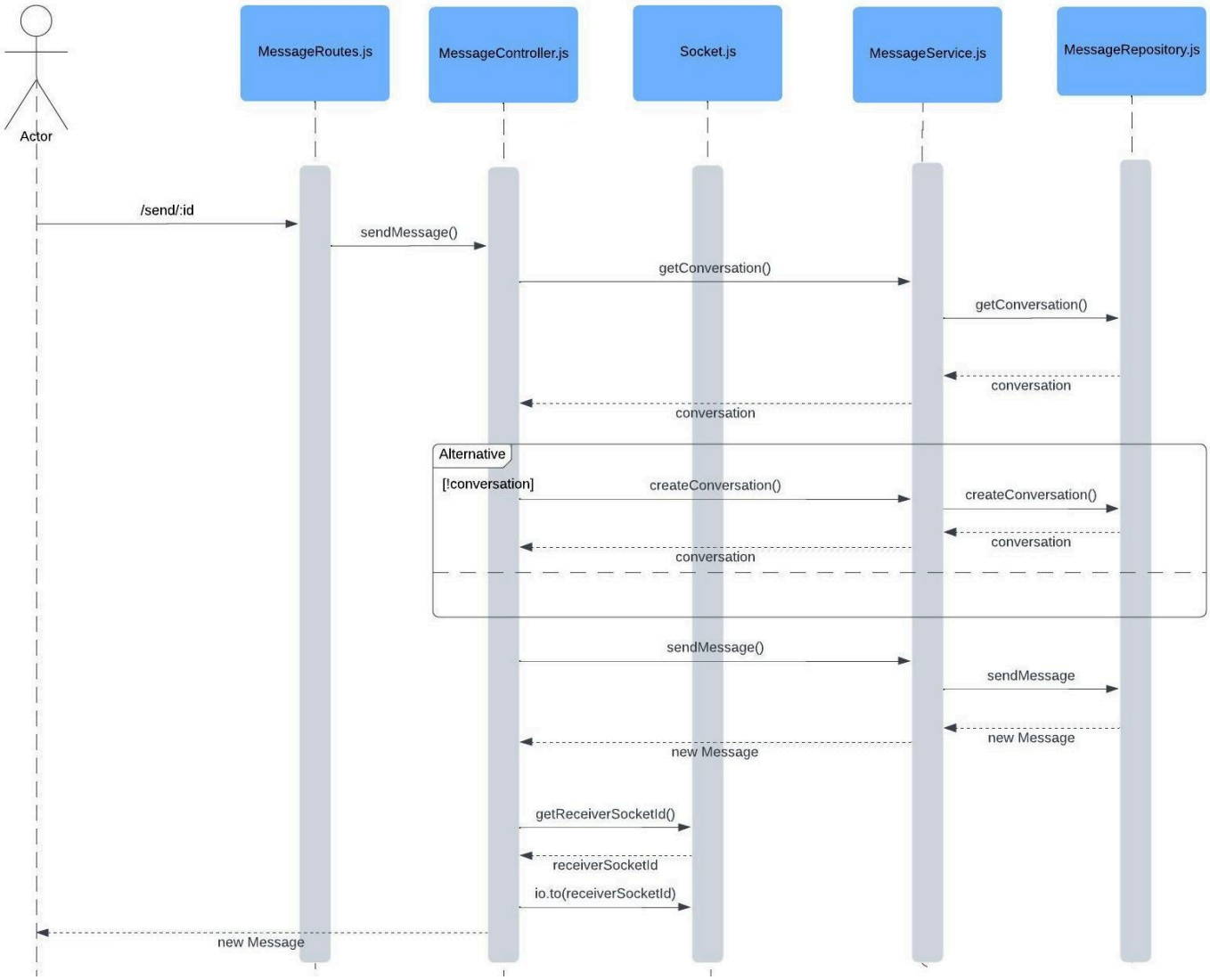
One interesting part of our backend is how it integrates with a separate Python service. This Python service runs our recommendation algorithms, which are used to suggest trips to users. We set up this integration using Flask, a lightweight web framework for Python, to create a simple API that our main Node.js backend can talk to.

Login



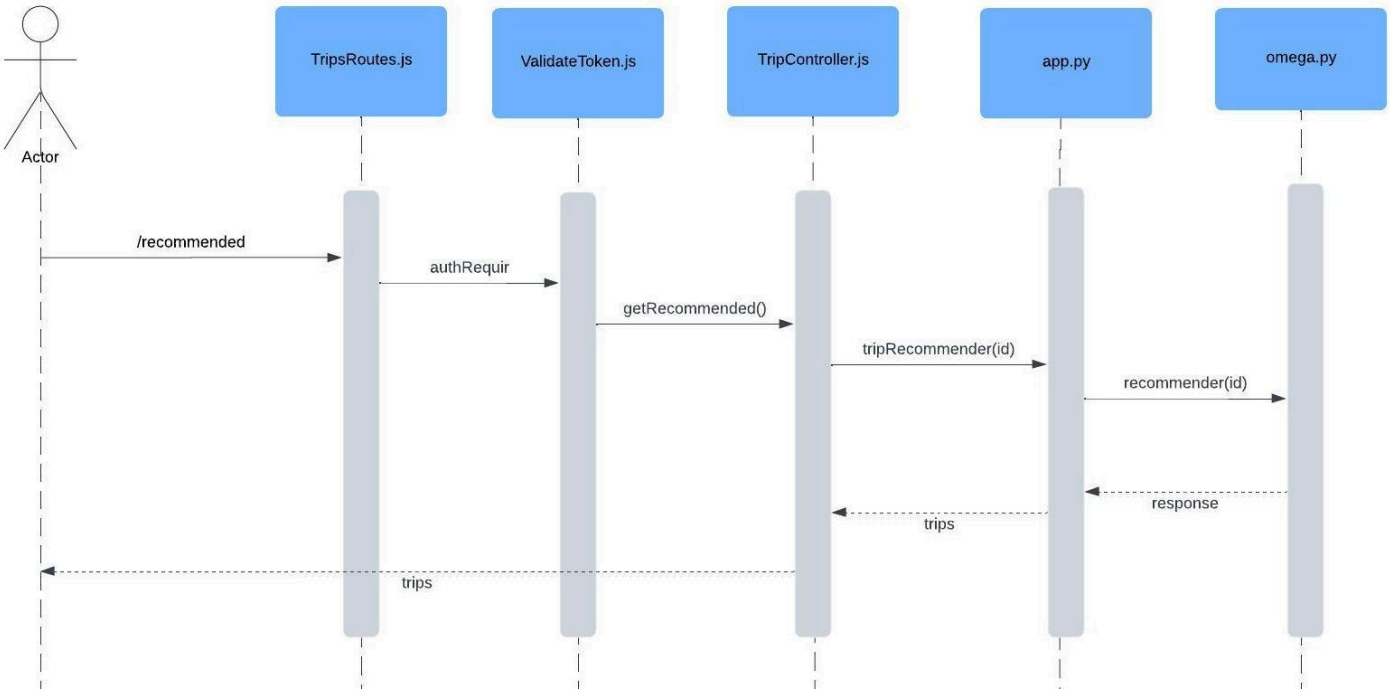
When a user tries to log in, the frontend sends their details to the /login endpoint. The server first checks the user's credentials by searching for their email in the database. This is done through a series of function calls that start in the controller, move to the service layer, and finally reach the repository, which directly interacts with our MongoDB database. If the user is found, the password is verified using bcrypt. If the password matches, we generate an access token for the user session using our JWT utility and send this token back to the client along with the user data. This ensures that subsequent requests are authenticated and authorized properly.

Send Message



When the frontend sends a message, it hits the /send/:id endpoint. The backend first checks if there's an existing conversation between the two users. If there isn't, it creates a new conversation record. If a conversation already exists, the message is simply added to it. Once the message is saved, we use Socket.IO to push the new message to the recipient in real-time, which is crucial for a smooth chat experience. This real-time functionality relies on WebSockets to maintain an open connection between the client and the server, allowing updates to be sent instantly. It's a great example of how our backend uses modern web technologies to enhance user interactivity and engagement.

Recommended Trips



Fetching recommended trips takes advantage of a different aspect of our system. When a user accesses the homepage, a request is sent to the /recommended endpoint. Before anything else happens, we make sure the request is coming from an authenticated user by checking their token. Once they're cleared, we fetch personalized trip recommendations by calling the Python service. This service runs our

recommendation algorithm and sends back a list of trips. The backend then relays this data to the frontend, providing users with relevant suggestions tailored just for them.

4.5 Features Implemented

4.5.1 User Module

Login

On the Quillyz landing page, there is a login form where you need to fill in your email and password to gain access. Any errors, such as incorrect credentials, empty fields, and more, are properly handled and users are notified accordingly. View Images

Register

To sign up, there is a "Register" button located at the bottom of the login form on the landing page. Clicking this will take you to the registration page, where you'll need to complete all the required fields in both the personal information and preferences sections before hitting the "Register" button. View Images

Profile

On the right-hand side of the navigation bar, you'll see your profile picture. Clicking on it will allow you to access your profile page, where you can manage your account. View Images

Change Profile Picture

To change the profile picture, go to your profile page and click on the icon above your current picture. This will allow you to upload a new image to update your profile picture. View Images

Upload post

To upload photos, head to your profile page and click the "+" icon in the upper right corner. This feature lets you upload posts to your feed or add photos to a specific album that you have. View Images

View Followers

To see your followers, click the "Followers" button. This opens a modal with cards representing your followers, each of which you can click to access their profiles. View Images

View Following

To view the people you're following, click the "Following" button. This will open a modal showing cards for everyone you're following, which you can click on to visit their profiles. View Images

4.5.2 Trip Module

Recommended Trips

On the main screen, there is a panel displaying trips recommended by our algorithm. These are represented by cards that show key information about each trip. View Images

Create Trip

To create a trip, go to the main page and look for the "+" icon in the bottom-right corner. Clicking this will open a modal where you can create a new trip by filling out all the required fields. View Images

Join Trip

To join a trip, simply go to the main page and select any of the recommended trips. This will open a modal with more detailed information about the trip, and from there, you'll have the option to join. Once you've joined a trip, you can view it in the "My Trips" section under "Joined Trips" in the navigation bar. View Images

Update - Cancel Trip

In the "My Trips" page, you'll have access to the trips you've created. By clicking on any of them, a modal will open that allows you to update the trip details or cancel it if necessary. View Images

Disjoin Trip

In the "My Trips" section, under "Joined Trips", you can click on any of the trips you've joined to open a modal. From there, you'll have the option to leave the trip. View Images

4.5.3 Social Network Module

Search People

Accessed via the "Search" option in the navigation bar, this feature allows you to discover people with similar interests or preferences, recommended by our algorithm. Users are displayed as cards, giving you three options: view their profile, follow them, or skip to the next suggestion. View Images

Chat

You can access this feature through the "Chat" option in the navigation bar. It allows you to engage in real-time conversations with people you're connected to, see who is currently active for chatting, and search for a specific person to start a conversation. View Images

Trip Album

From your profile page, using the image upload option, you can also upload a post to a trip album for trips you've either been part of or created. This applies to all trip members, as they can also contribute posts to the shared album. View Images

Chapter 5 - Recommendation Algorithm

At the core of Quillyz's functionality are its advanced recommendation algorithms, designed to enhance a user's experience by suggesting personalised trip options and potential companions. These algorithms analyse the user's data and evolve through time to tailor the recommendation options that match individual interests and tastes. The algorithms were designed and adapted to follow a k-means clustering implementation. We used a variety of algorithms, each with a specific focus and methodology, to fulfil the unique preferences of our users during the selection of a trip:

- **Alpha Algorithm**: This algorithm is based on an analysis of our potential user feedback that showed most users value trips by the price, distance from their own home and the planned activity. It employs a simple clustering technique to identify patterns in past user behaviour, grouping together a range of those three values, so that the users may repeat similar trips within the same distance, cost and activity preference.
- **Beta Algorithm**: Unlike Alpha, Beta is a fully automated clustering algorithm that uses any and all variables contained in the user trips history. Analysing automatically additional details like seasons, if transportation is included, overall ratings and more. But due to its large computational cost, its usage is limited, to avoid long waiting periods so that we don't diminish our user's experience.
- **Gamma Algorithm**: This algorithm provides a different view, instead of focusing on a user's past experiences, it groups the user's with those of similar taste, and or trip history. With this approach, those users who are adventurous and curious will be able to try out new trip ideas, granting our user's more options that should still be appealing.

- **Delta Algorithm**: Delta is based on the social aspect of travelling. It suggests trips where the user's friends are already listed, fostering shared experiences with those friends you've already made on the way. To avoid uncomfortable situations users will have full ability tu black-list or white-list certain users without having to block or ban them, in the end, our objective isn't to force interactions, only that each user may be comfortable while using our application.
- **Epsilon Algorithm**: To ensure diversity for our master algorithm, Epsilon reserves a small section as a mutation variable, introducing fresh trip ideas from various categories. It suggests new activities, with different price ranges, locations, activities, preventing users getting fixated and stuck into the same trips due to our algorithms.

Additionally all these recommendations have global filtering options to ensure a decent level of control to our users over the initial data pool used by each algorithm.

5.1 Matchmaking Algorithm

The matchmaking algorithm is designed to connect users within our social network applications. That filters each by their personal hobbies and interests, as well as their trip history. We wish to also join people through our blog application, having them share their experiences and giving them the chance to meet others through forums as an additional way to find friends.

The recommendation behind the matchmaking currently uses the same methodology from both the **Beta** and **Gamma** algorithms. Which are some of the most common techniques used in these social net applications.

5.2 Algorithm Design and Graphs

To facilitate the understanding of our recommendation system, we have produced a series of graphs that illustrate an example of the outcome from each algorithm. These diagrams provide a visual representation of how user information input is processed, how data is filtered, and how final recommendations are generated.

Please note that the data used in the following examples were loaded and adapted to suit our examples from public sources (sklearn). These images are only meant as guide in explaining their functionality.

- **Alpha Algorithm**: In the following image, we can view the clustering process and how similar trips are grouped. Here, each trip is positioned by cost and distance, while each colour represents a different activity. Two examples have been generated for different k distance values.
- **Beta Algorithm**: No analyzable data can be found from this recommender due to the data being too ambiguous to analyse without using multidimensional panels which are still unavailable to us.
- **Gamma Algorithm**: For the next image, Outlines the comparative analysis with other users. In this example the initial user only shows interest in the blue subgroup of trips, but the yellow is later added due to many similar users sharing a potentially enjoyable trip.
- **Delta Algorithm**: No analyzable data was included since this recommender is unique to each individual's connections.
- **Epsilon Algorithm**: No analyzable data was included due to the purpose of this recommender, that only adds new trips to promote mutations in the user's trip selection.

5.3 Omega : Algorithm Controller

The Omega Algorithm is dedicated to properly manage, learn and evolve to tailor each recommendation to a specified individual. This master algorithm is inspired by genetic programming, based on Darwin's theory of evolution. Adapting the weight and influence of each algorithm based on user behaviour and feedback. The necessary components used in a genetic algorithms must include:

- **Genotype**: Represents a code used to calculate the shares of each algorithm from Alpha to Epsilon. This item is composed from a number of genes.
- **Gene**: Is the individual value of an algorithm's influence on the overall recommendation.
- **Mutation**: Introduces random variation to the gene values, ensuring the system remains adaptable and innovative.
- **Fitness function:** Evaluates how well the current set of recommendations meets the desired outcome. In our case this job will be fulfilled by the user, analysing their activity and their interest during the session. This is quantified by the amount of trips the user has opened with the objective of looking up more details.

5.4 Evaluation variants

Ensuring the effectiveness of our recommendation algorithms is crucial. Which is why we have two evaluation selections used to optimise the algorithm:

1. **Static Evaluation**: This is based on historical data, analysing how well the algorithms perform in recommending trips that users have previously enjoyed.
2. **Session Evaluation**: During active user sessions, the algorithms are continuously evaluated and adjusted based on user interactions. This real-time feedback loop allows the system to learn and evolve, improving the relevance of recommendations over time.

5.5 Algorithm Design and Graphs

The overall process of a genetic algorithm is divided into several stages:

I. **Selection:** The initial selection will contain a single genotype based on the static evaluation of the user's preference history. The following recommendations will also include the previous selection activity, and will also depend on the fitness evaluation.

II. **Fitness calculation :** The fitness evaluation as previously mentioned, is dictated by the user's actions, which trips he finds interesting are defined by the opening of said trip. The fitness will receive the initial static evaluation with the two previous session evaluations, from these. The genotypes with the most activity will be chosen.

III. **Reproduction :** After choosing both optimum evaluations, their values will be randomly joined. The difference in fitness will also affect the amount of data that will be contained in the progenitor.

IV. **Mutation :** To ensure that the user is able to receive a variation of options from where to choose, a small mutation has a chance to take effect and change one of the genes of the genotype, this promotes new options that will hopefully improve user experience.

The following image shows a usage example of a user navigating during a single session. In which 40 different recommendations have been loaded. Sharing at first and equal amount of trips, to adapting and showing those calculated by the users interest.

Chapter 6 - Conclusions and Future Work

6.1 Conclusions

We developed Quillyz with the vision of providing an innovative platform that combines trip planning with social networking features, allowing users to discover new destinations, meet like-minded travellers, and share their experiences. Throughout the development process, we focused on:

1. **Comprehensive Trip Planning and Discovery**: Quillyz integrates a robust trip planning feature that leverages a recommendation algorithm to suggest travel destinations tailored to user preferences and interests. This personalization will hopefully enhance our user satisfaction by presenting travel options that align closely with each individual taste.
2. **Social Networking Integration**: By incorporating social networking elements similar to platforms like Instagram, Quillyz allows users to share their travel photos, post updates, and engage with other users. This feature not only enriches the user experience but also fosters a community of travellers who can inspire each other through shared experiences.
3. **User Matching Based on Interests**: Quillyz's matching algorithm enables users to connect with others who have similar hobbies and tastes. This functionality enhances social interaction and networking, providing users with opportunities to make meaningful connections with fellow travellers.
4. **User-Centric Design and Continuous Improvement**: A significant focus was spent on user experience, as evidenced by our feedback collection and iterative design. We hope that Quillyz remains intuitive, user-friendly, and aligned with the needs of its target audience.
5. **Scalable and Secure System Architecture**: The application's backend and frontend systems were designed to be scalable, supporting future growth in user

features and complexity. Security measures are yet to be implemented, but we will work in the future to protect user data and ensure a safe environment for sharing personal information and travel experiences.

Overall, we hope that Quillyz has successfully established itself as a versatile platform for both trip planning and social interaction, differentiating itself from other travel apps through its unique mix of personalised recommendations and social networking capabilities.

6.2 Future Work

While Quillyz has achieved its initial objective, there are numerous opportunities for growth and to further improve the platform's functionality, user experience and social reach. WE have plans to continue developing the next areas:

1. **Recommendation Algorithms**:
    - **Machine Learning Integration**: Future versions of Quillyz can incorporate more advanced machine learning models to enhance the accuracy and time spent. By analysing user behaviour patterns using feedback, the recommendation system can continuously improve and adapt to changing user preferences.
2. **Expansion of Social Features**:
    - **Live Streaming and Stories**: Adding live streaming and story features will enable users to share real-time travel experiences with their followers, creating a more dynamic and engaging platform.
    - **Group Trip Planning**: Implementing features that allow users to plan trips together, coordinate itineraries, and share expenses can enhance the collaborative aspect of Quillyz, making it easier for groups of friends or family members to organise their travels.

3. **Integration with Travel Services**:

   Partnering with airlines, hotels, and local tour operators to offer booking options directly within the Quillyz platform will provide a seamless travel planning experience. Users could book flights, accommodations, and activities without leaving the app, making Quillyz a one-stop travel solution.

4. **Enhanced User Matching and Community Features**:
   - **Interest-Based Communities**: Creating community spaces within Quillyz based on specific interests, such as adventure travel, cultural experiences, or food tourism, can help users connect with others who share their passions.
   - **Advanced Matching**: Based on matching up those who show similar activity in the same categories of forums, so that they may meet on group chats or on trips together.
5. **Localization and Global Expansion**:

   As this application grows, expanding its reach to include more languages, local travel content, and culturally relevant features could be essential. This will help Quillyz cater to a broader audience, making it a global platform for travellers from different regions.

6. **Data Analytics and Insights**:

   DEveloping tools that provide users with insights into their travel patterns, preferences, and social interactions can add value to their experience. These analytics can help users understand their travel behaviour and plan future trips more effectively.

7. **Improved Efficiency**:

   Improvements to both the computational power of the server and connection speed will be important to ensure our user's experience. With the additional algorithms and functions we wish to include, this upgrade would be essential.

8. **Improved Security and Privacy Measures:**

   As we continue to grow and collect more user data, ongoing improvements to security protocols will be essential. Implementing end-to-end encryption, two-factor authentication, and regular security audits will ensure that user data remains protected.

   Additional steps to ensure user authentication and ensure safety within the users who join the trip shall also be implemented. From ID verification to a validation and confirmation net used in verifying and rating each user.

6.3 Continuous Improvement

The journey of Quillyz will not end with its initial launch. Continuous improvement, based on user feedback and emerging technological trends, will be crucial to maintaining its relevance and competitiveness in the travel and social networking industry. Regular updates, user engagement initiatives, and a commitment to innovation will be the necessary steps to ensure Quillyz's long-term success.

By focusing here, Quillyz can continue to evolve, offering new features that meet the changing needs of our users worldwide. The platform's commitment to delivering a great user experience will remain at the core of its development strategy, ensuring that Quillyz remains a beloved tool for planning trips, meeting new people, and sharing travel experiences.

Chapter 7 - User Experience and Feedback

User experience is a critical component of Quillyz's design, as we wish to satisfy all sorts of different users, with different tastes, and yet we wish for them to stay engaged in our application. The following section details the strategies and methods we used to collect user feedback, and analyse that feedback to make informed improvements to the application.

7.1 Interviews & Questionnaires

For the interviews, a thorough analysis was first conducted on the application and the different users who might use it. These formulated questions helped us classify the users and their views about the application. It was decided that the ideal duration for the interview would be around 15-25 questions or 10-20 minutes. However, the actual duration of the interviews ended up varying on each of the 5 interviews.

An additional questionnaire was also shared through online forums to further analyse interested participants that may share some more information. These questionnaires hold fifteen questions, from which these five questions were the most informative (Additionally, the entire questionnaire can be found here).

7.2 Analysis of User Feedback

Once user feedback was collected, we analysed it systematically to obtain meaningful insights. From which we learnt:

1. **Target users**: Most users interested in travelling are between the ages of 16 and 25. Additionally most of those over the age of 20 are the ones that travel frequently.

2. **Social preferences**: Survey shows that most people enjoy meeting others and would be willing to travel and learn about new cultures.
3. **Trip selection details**:

   Most users select their trips depending on these following factors:

   - **Price**: The cost spent on the overall trip from start to finish.
   - **Location**: Where the trip takes place. Including users who want to avoid or wish to see specific locations.
   - **Activity:** The activity the user wishes to partake in during the trip.
   - **Transportation:** The cost and if the transportation is included in the announced trip cost.
   - **Accommodation:** The cost, and If accommodation is included in the announced trip cost.
   - Other variables include the trip's season, trip duration, the age group of the participants and more.

4. **Interest towards our application**: We've received positive reviews about our application idea, that show interest in this sort of application.
5. **Design preferences**: An additional optional comment section was included to our questionnaire that included many ideas and functionalities that users wanted present on our application. Some of these have been already added, or will be included in the future to further improve our user experience.

Chapter 8 - Personal Contributions

8.1 Contributions by Marcos:

Data Modeling and Design and Implementation of the Recommendation Algorithm

Chapters Written:

Chapter 5 - Recommendation Algorithm

Chapter 6 - Conclusions and Future Work

Chapter 7 - User Experience and FeedBack

8.2 Contributions by Steven:

Data Modeling and System Design, Architecture and Implementation of the Full-Stack Application

Chapters Written:

Chapter 1 - Introduction

Chapter 2 - State of the art

Chapter 3 - Software Development Technologies

Chapter 4 - System Design, Architecture, and Implementation

Bibliography

Books

- *"Introduction to Artificial Intelligence" by Philip C.Jackson, Jr. 3° Edition.*
- *"The Master Algorithm" by Pedro Domingos. Penguin Science/Tech.*
- *"Genetic Algorithms + Data Structures = Evolution Programs" by Zbigniew Michalewicz. Springer Verlag*
- *Clean Architecture by Robert C. Martin*
- *Refactoring by Martin Fowler*

Online Resources

- *Tutorials point clustering tutorial : https://www.tutorialspoint.com/artificial_intelligence_with_python/artificial_intelligence_with_python_unsupervised_learning_clustering.htm*
- *Authentication MERN APP: https://www.youtube.com/watch?v=NmkY4JgS21A*
- *Dockerizing MERN APP: https://redragno.hashnode.dev/dockerizing-a-mern-stack-web-application*

Technical Documentation

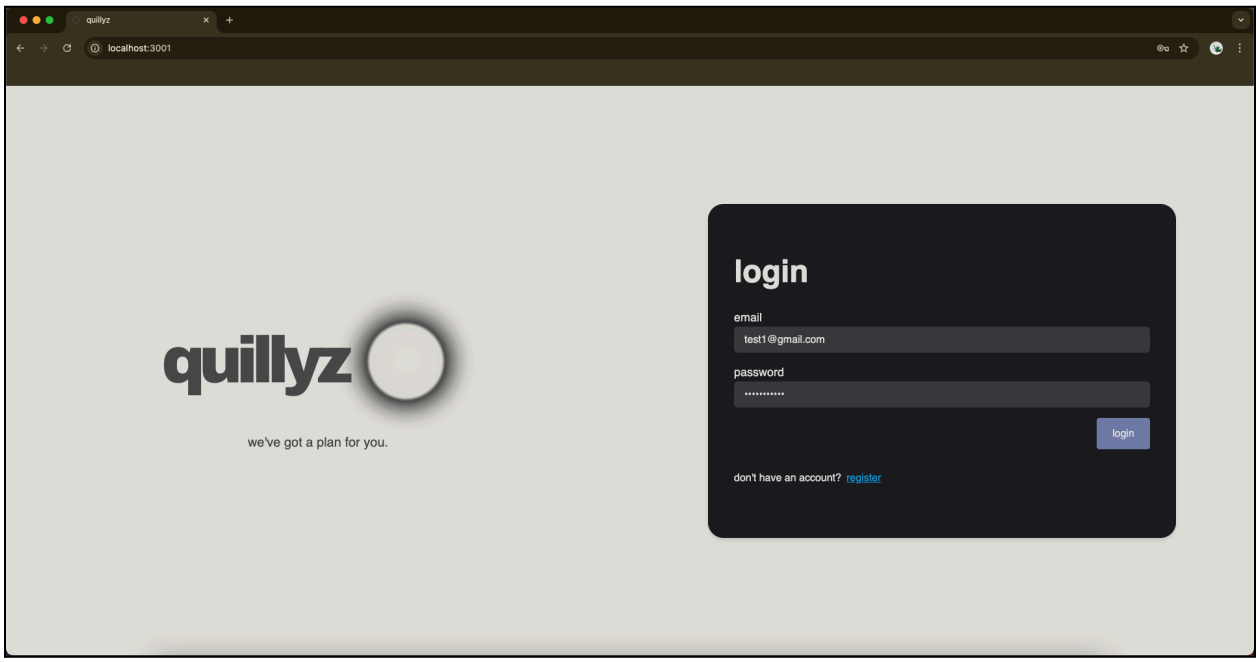
- *UML Diagrams: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/*
- *Tailwind CSS: https://tailwindcss.com/*
- *React.js : https://es.react.dev/versions*
- *MongoDB : https://www.mongodb.com/docs/*

- *Express.js : https://expressjs.com/*
- *Javascript : https://developer.mozilla.org/es/docs/Web/JavaScript*
- *Postman : https://learning.postman.com/docs/introduction/overview/*
- *WebSockets : https://github.com/websockets/ws*
- *Docker : https://docs.docker.com/*
- *Zod : https://zod.dev/*
- *Zustand : https://zustand.docs.pmnd.rs/getting-started/introduction*
- *Python : https://docs.python.org/3/tutorial/index.html*
- *Python clustering : https://machinelearningmastery.com/clustering-algorithms-with-python/*
- *NodeJS: https://nodejs.org/docs/latest/api/*

Graphs & Figures

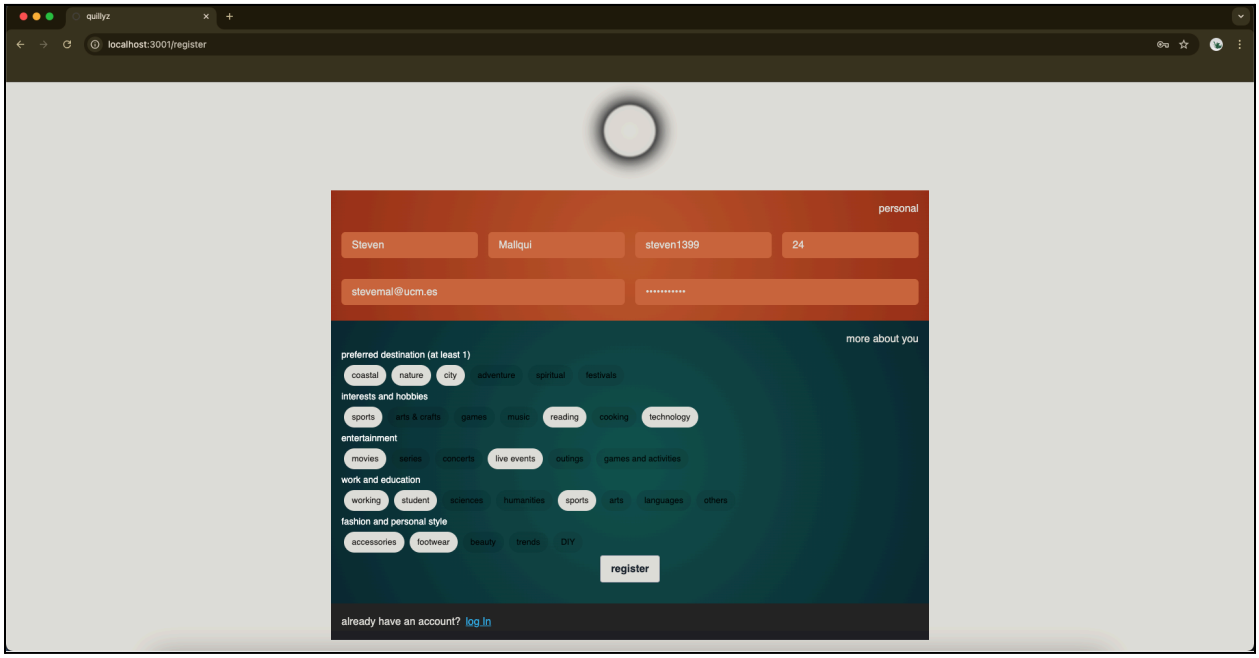
Web Screenshots Examples

Login



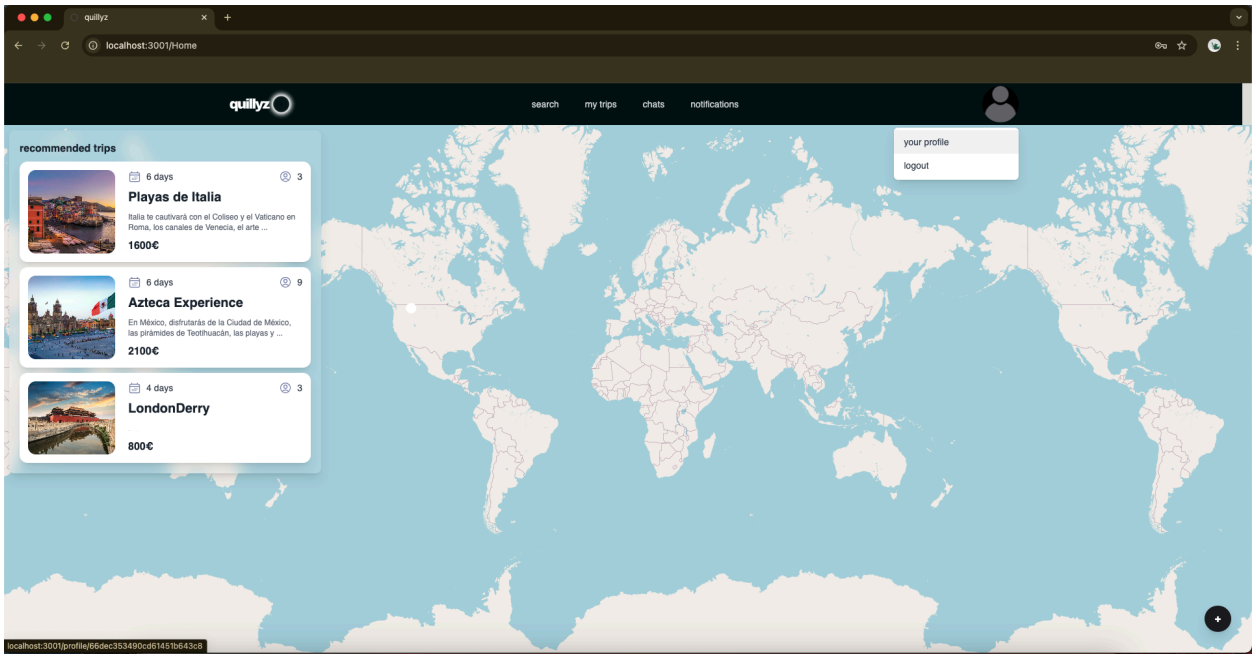
back

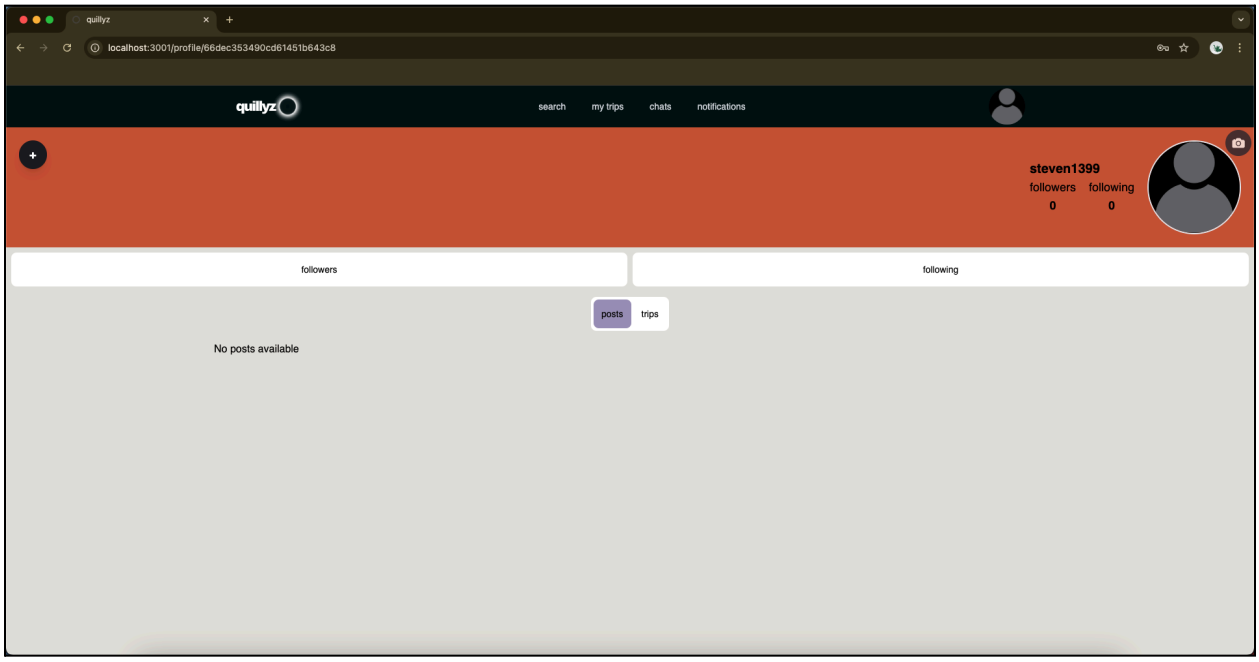
Register



back

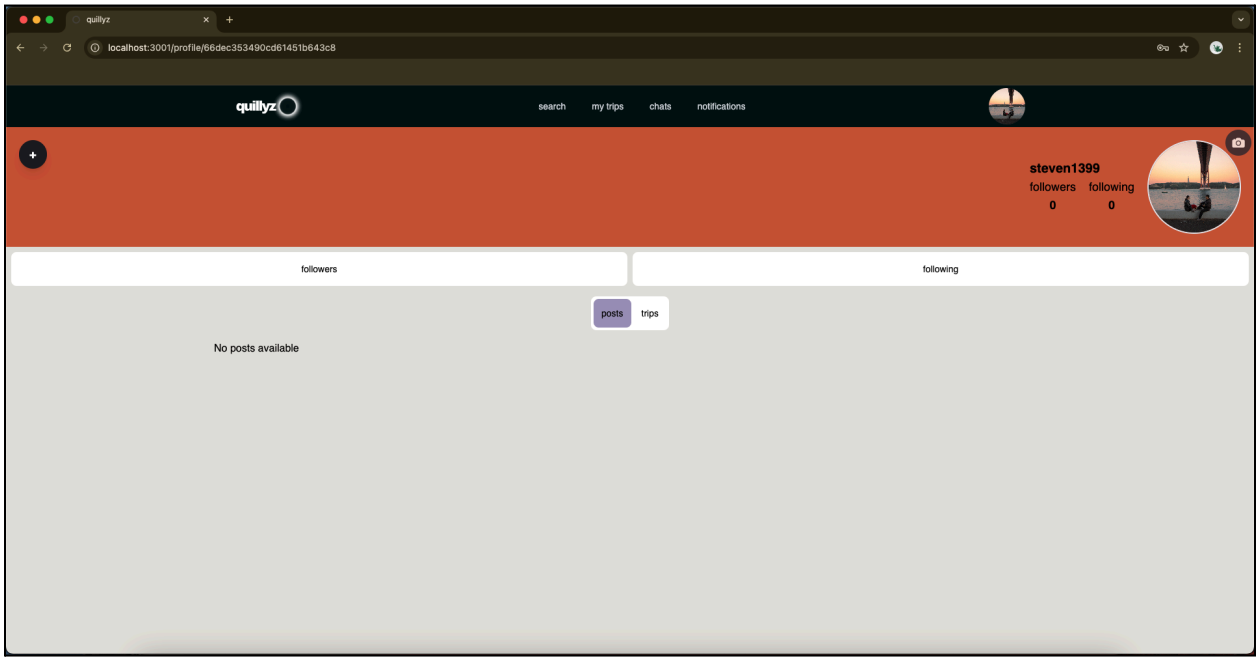
Profile





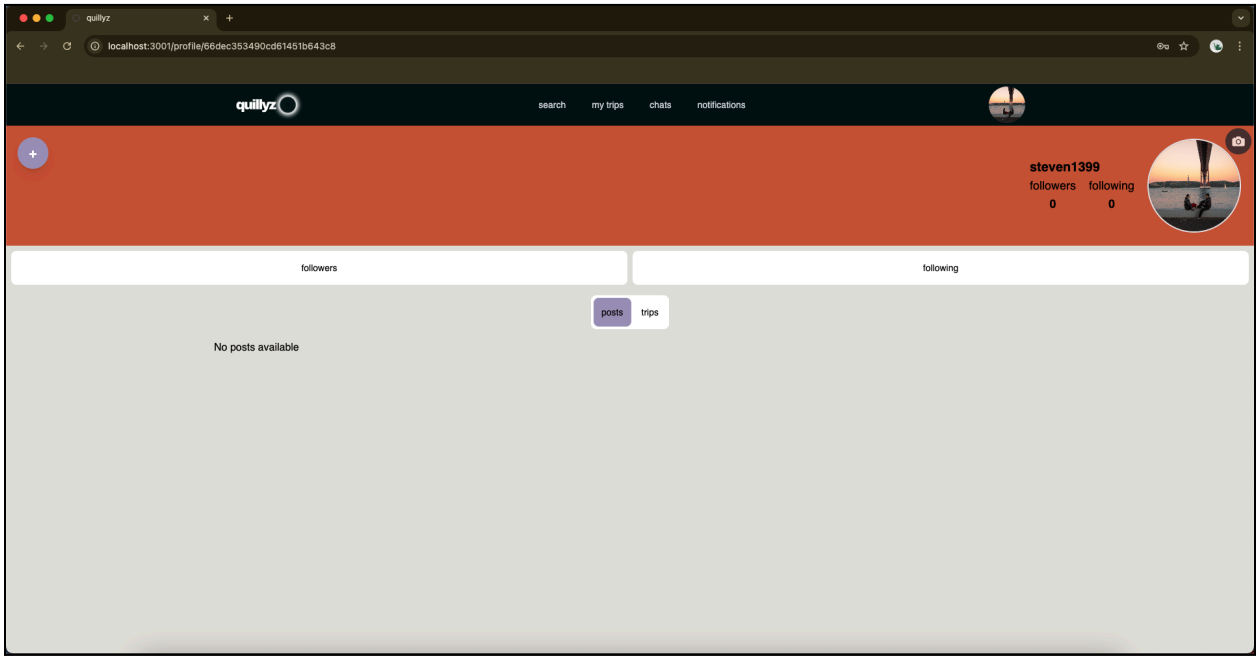
back

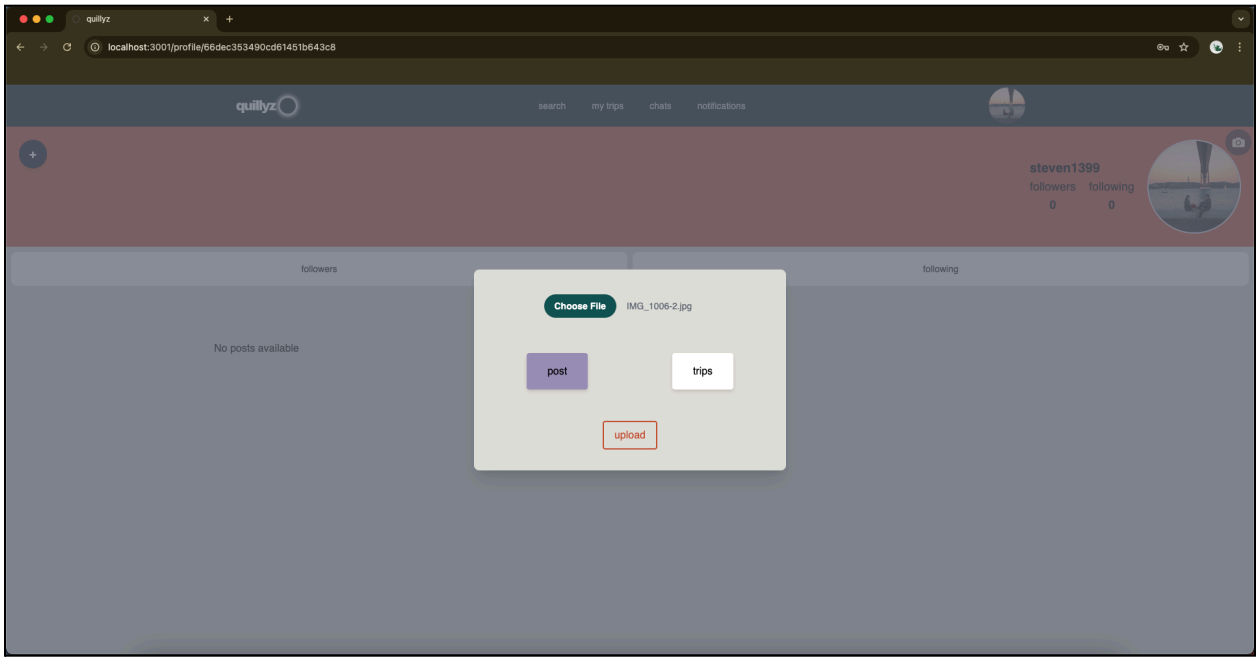
Change Profile Picture

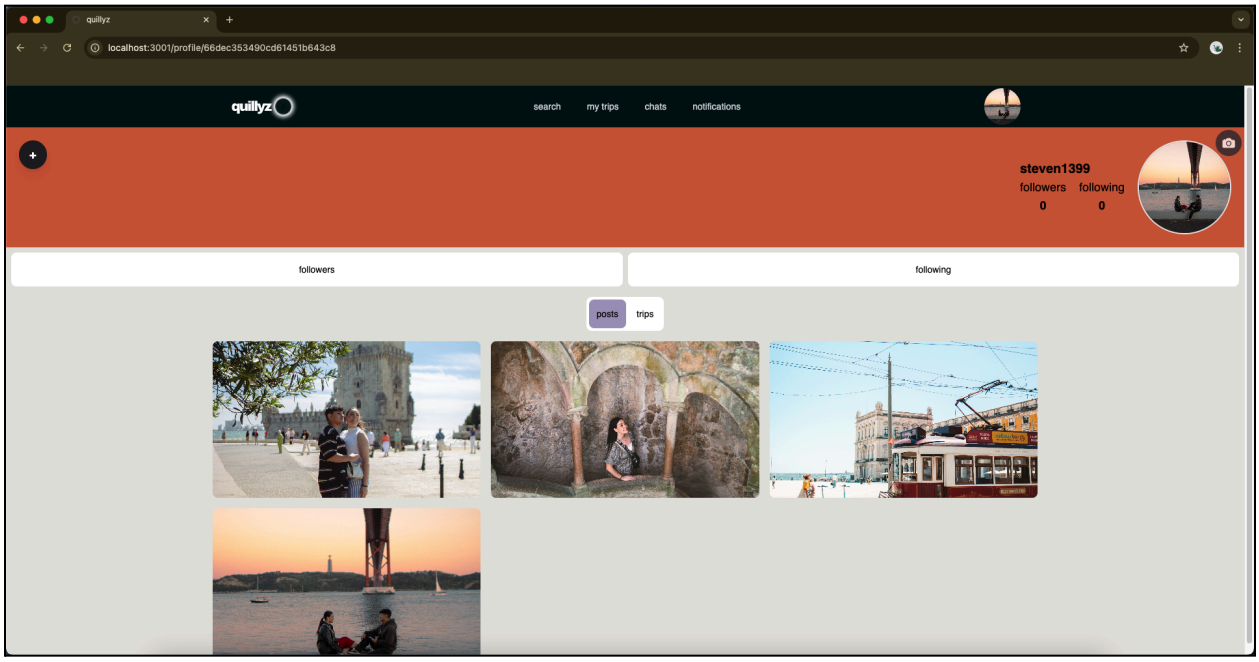


back

Upload post

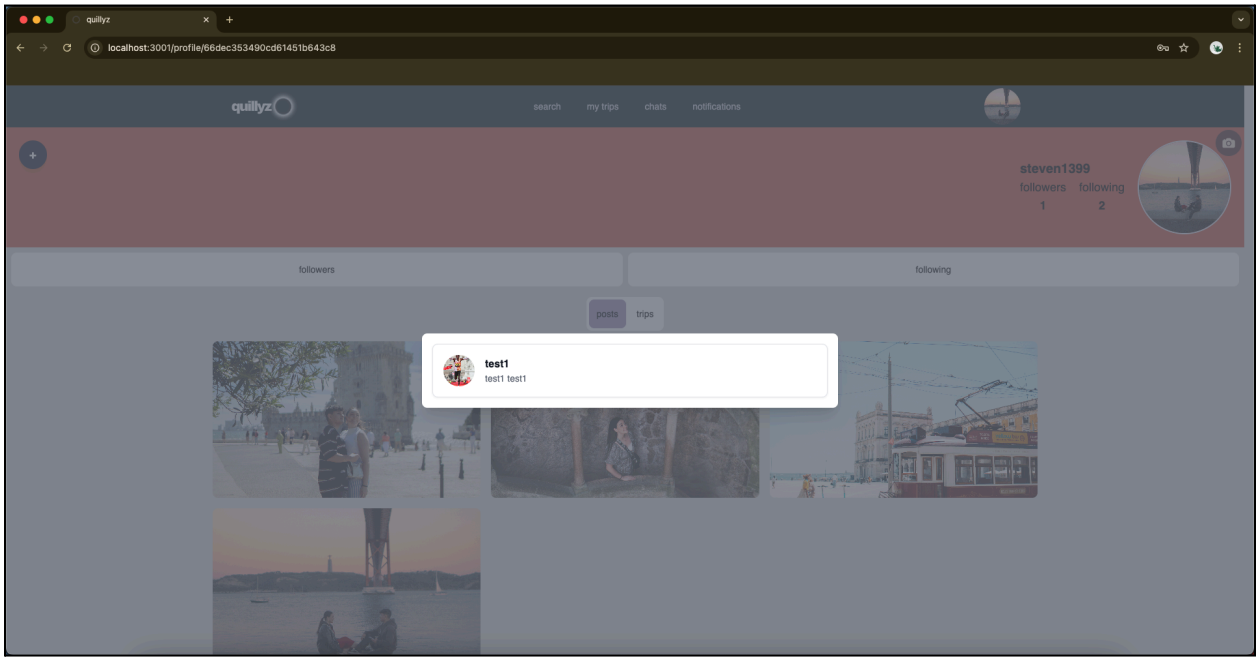

quillyz
localhost:3001/profile/66dec353490cd61451b643c8
quillyz
search
my trips
chats
notifications
steven1399
followers
following
0
0
followers
following
posts
trips
No posts available


quillyz
localhost:3001/profile/66dec353490cd61451b643c8
quillyz
search
my trips
chats
notifications
steven1399
followers
following
0
0
followers
following
No posts available
Choose File
IMG_1006-2.jpg
post
trips
upload



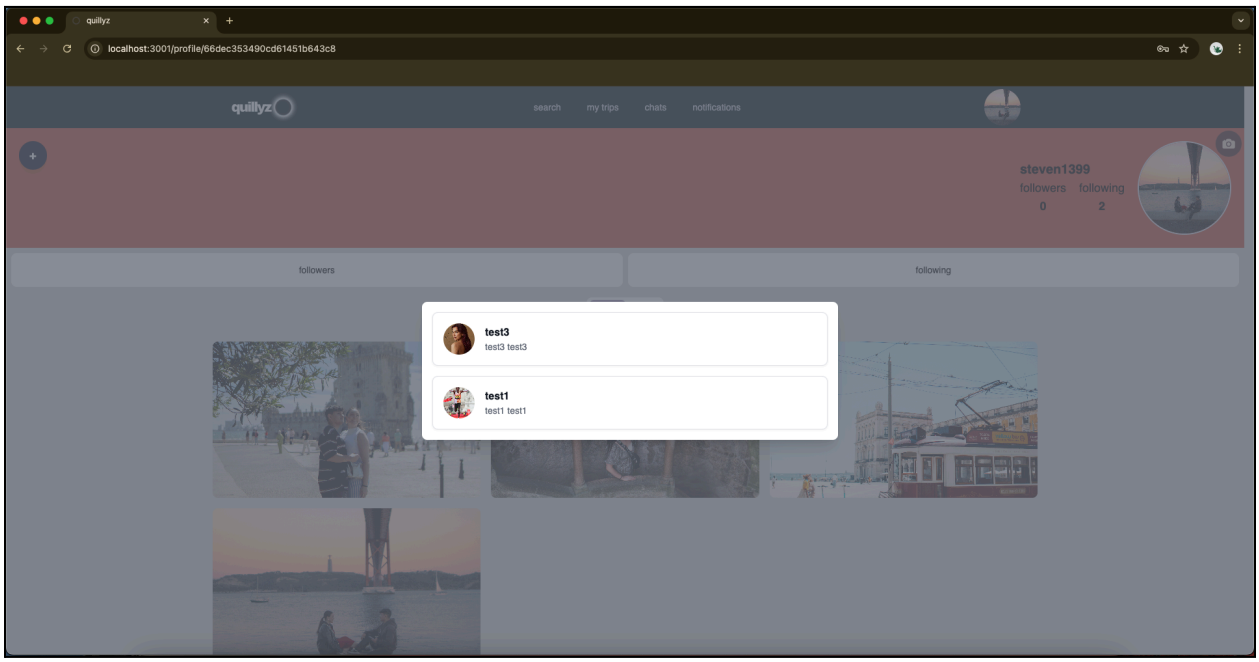
back

View Followers



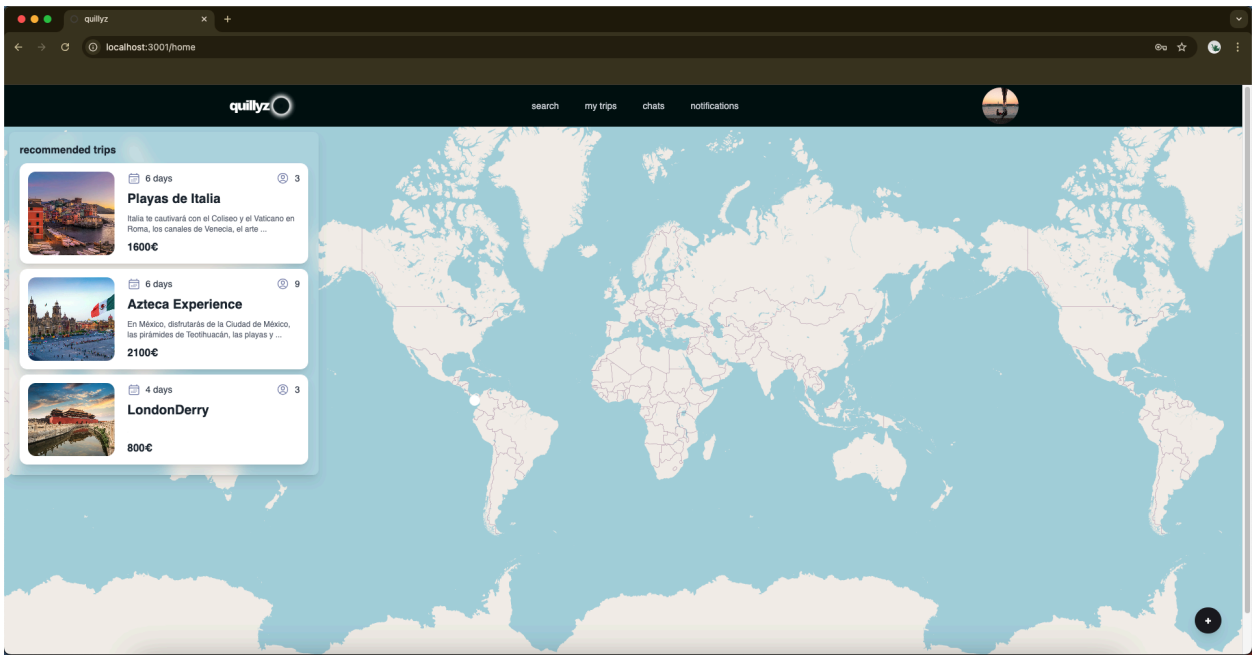
back

View Following



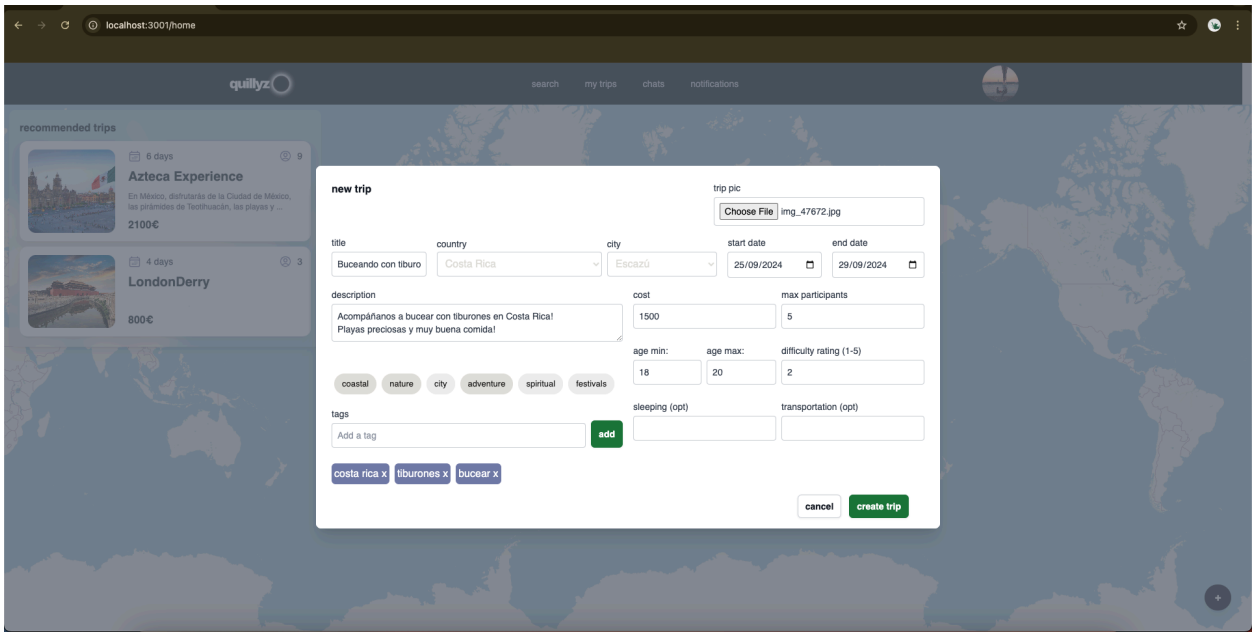
Back

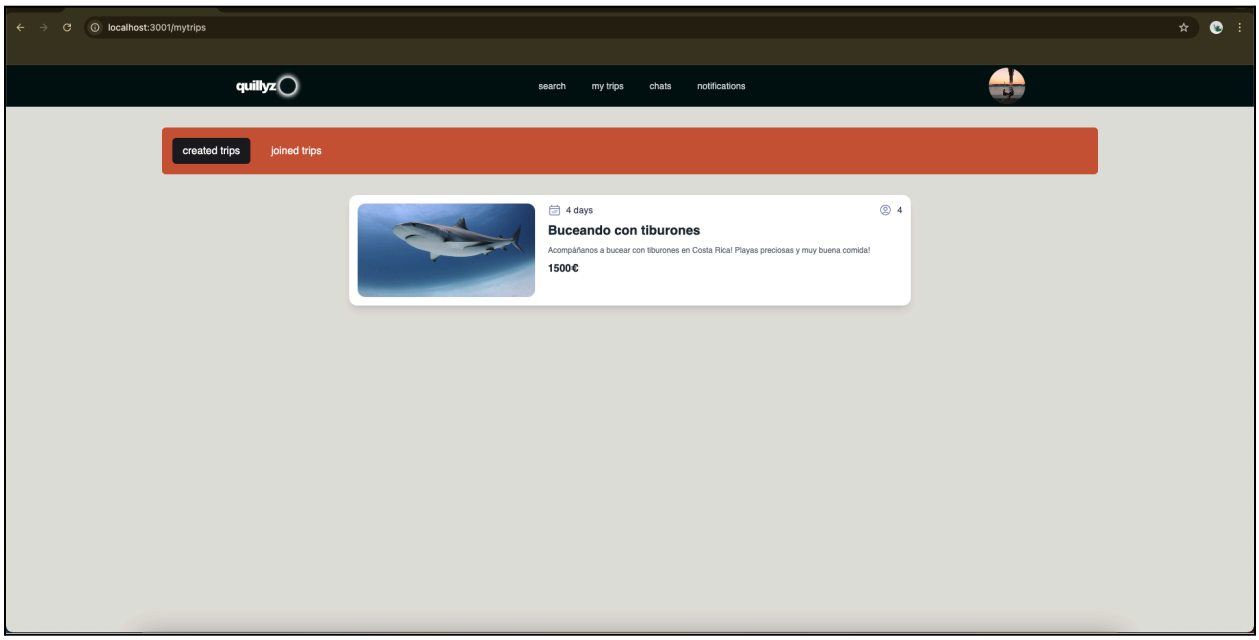
Recommended Trips



back

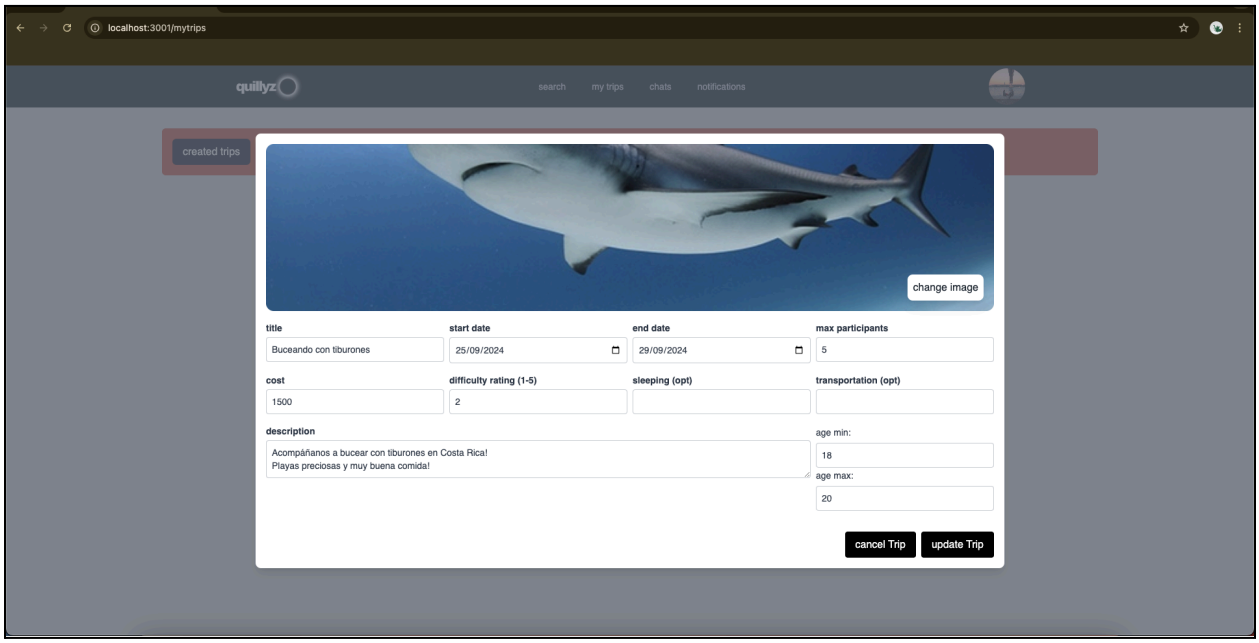
Create Trip





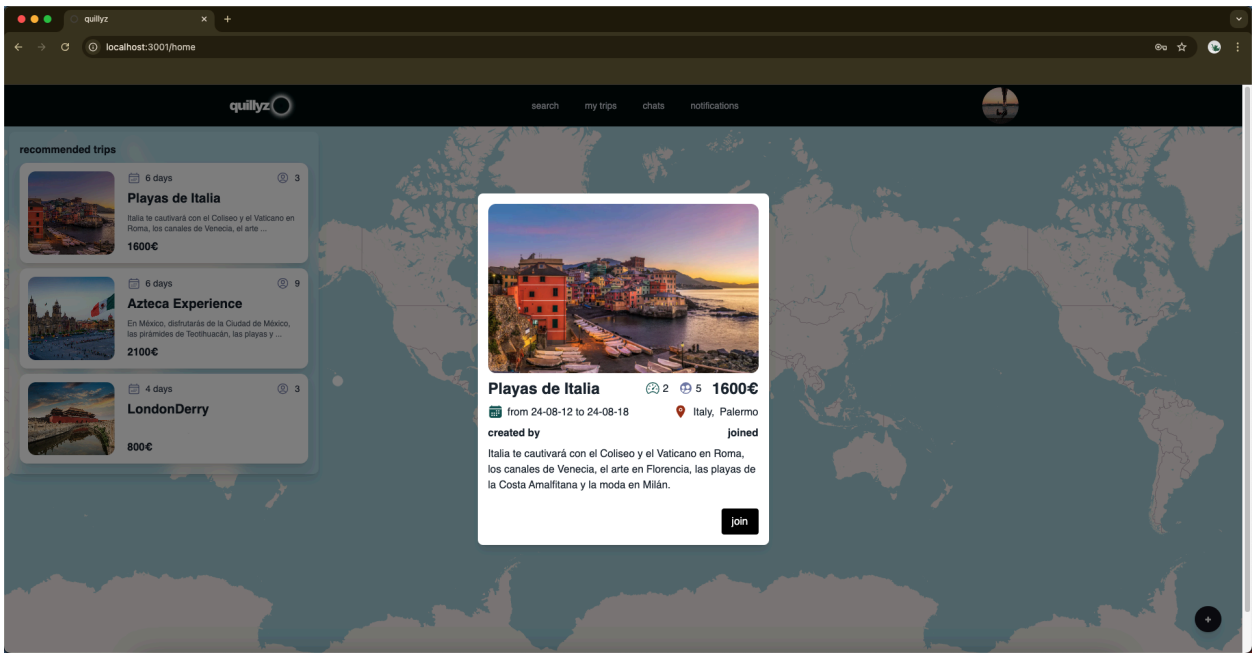
back

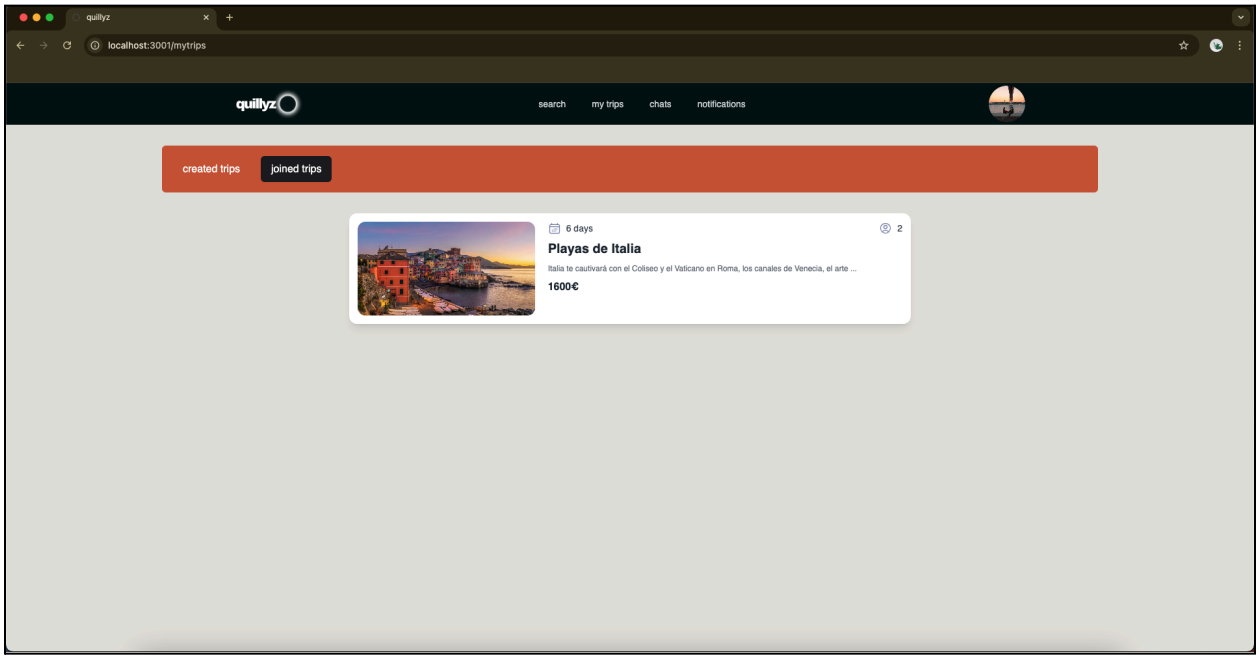
Update - Cancel Trip



back

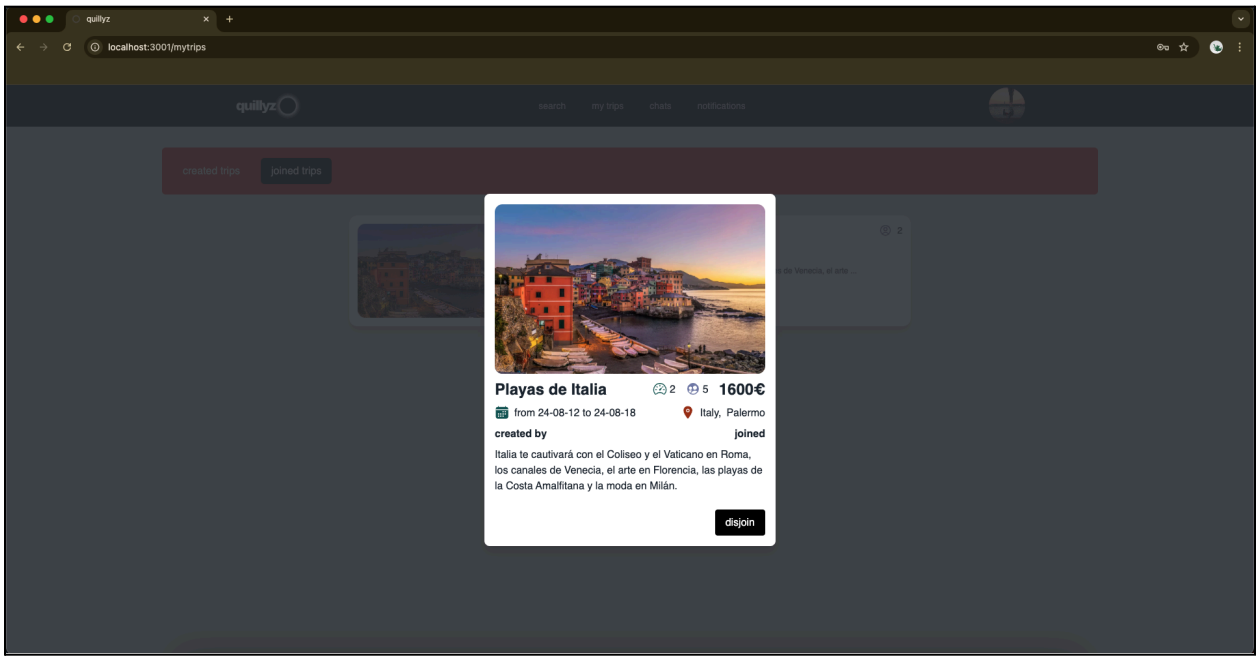
Join Trip





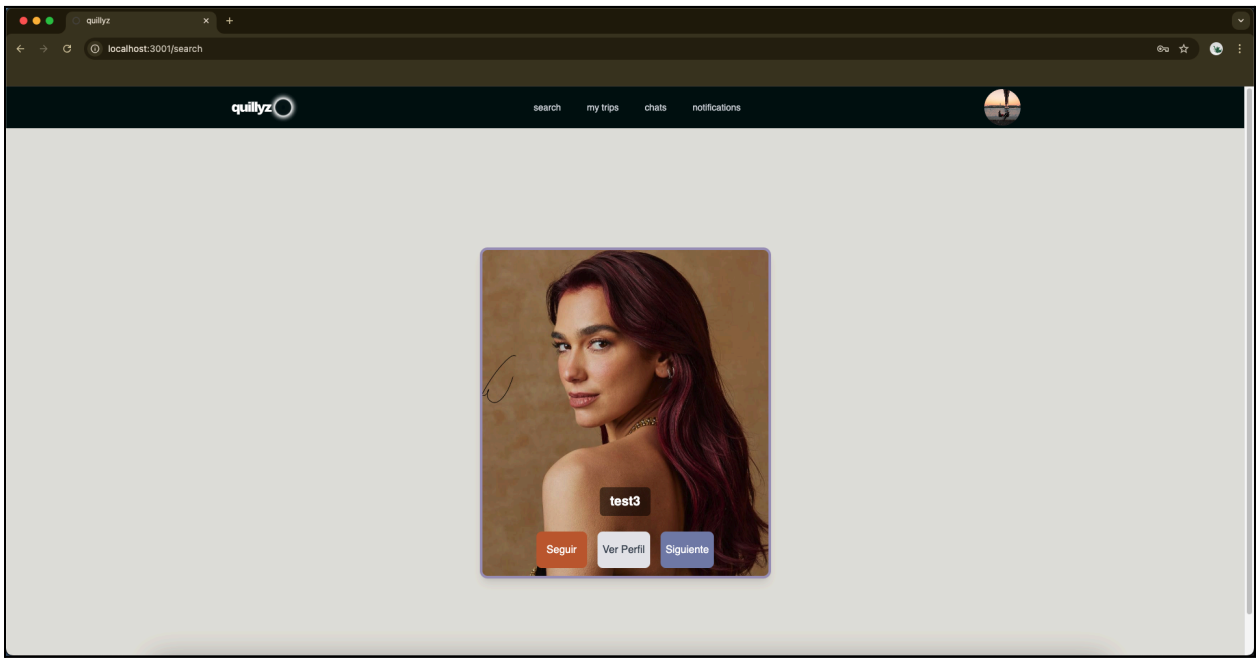
back

Disjoin Trip



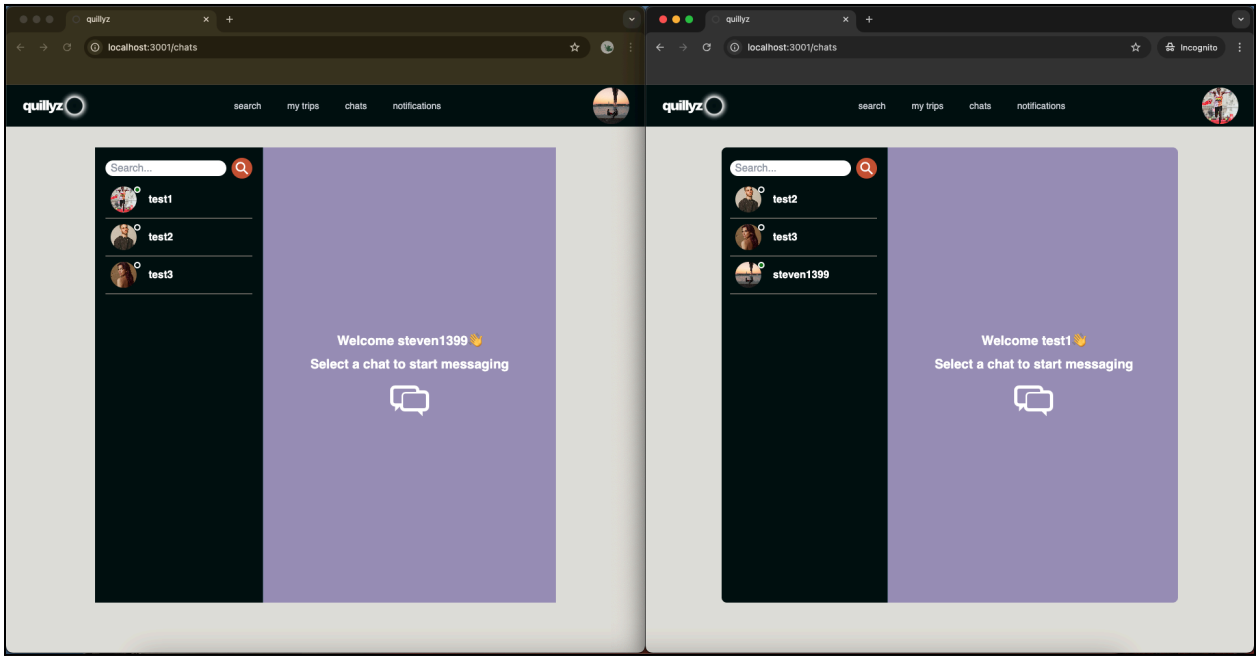
back

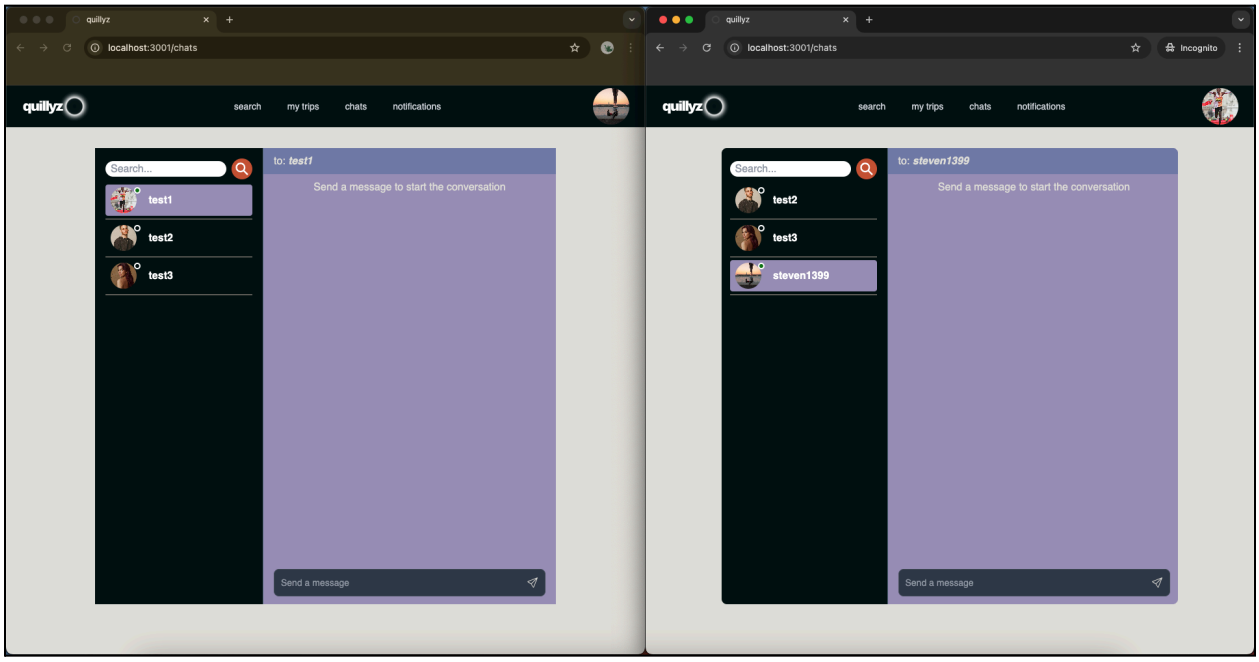
Search People

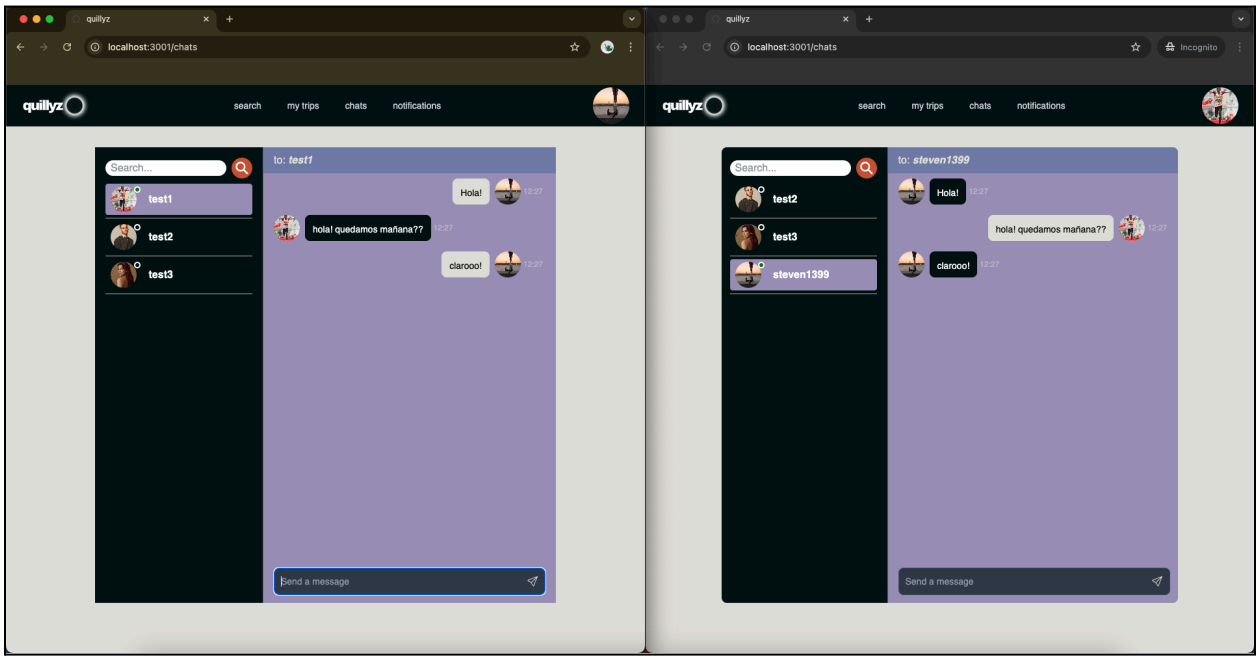


back

Chat

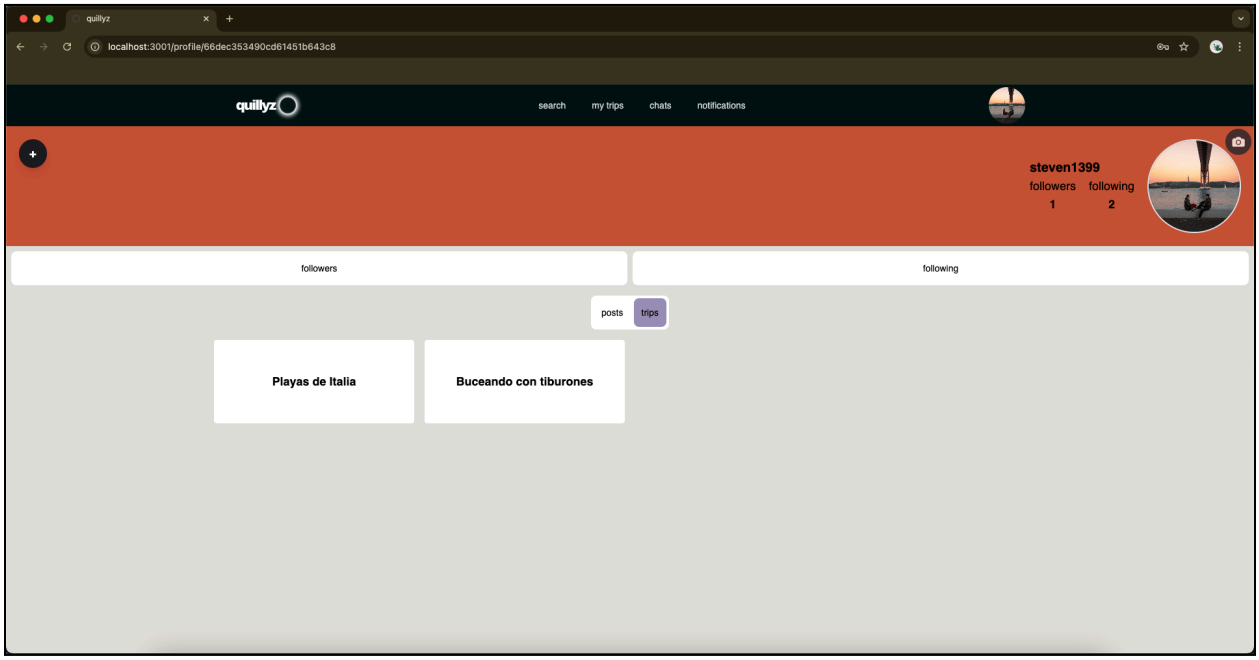


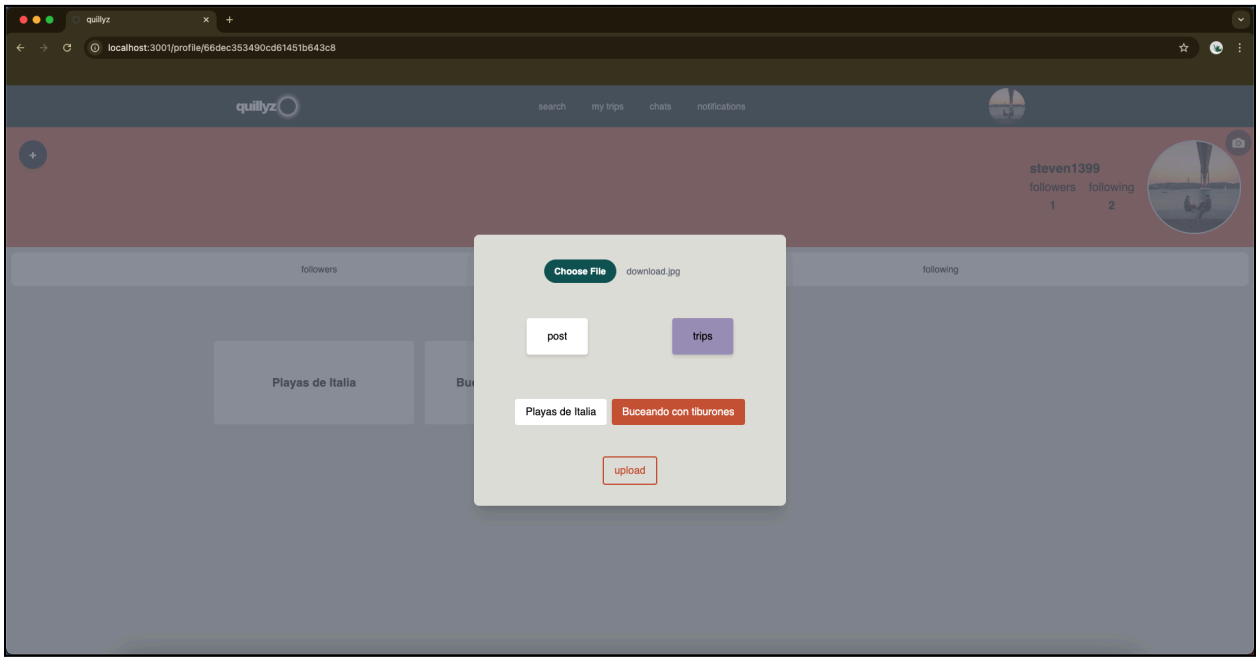

quillyz
localhost:3001/chats
search
my trips
chats
notifications
Search...
test1
test2
test3
to: test1
Send a message to start the conversation
Send a message
Incognito
steven1399
to: steven1399

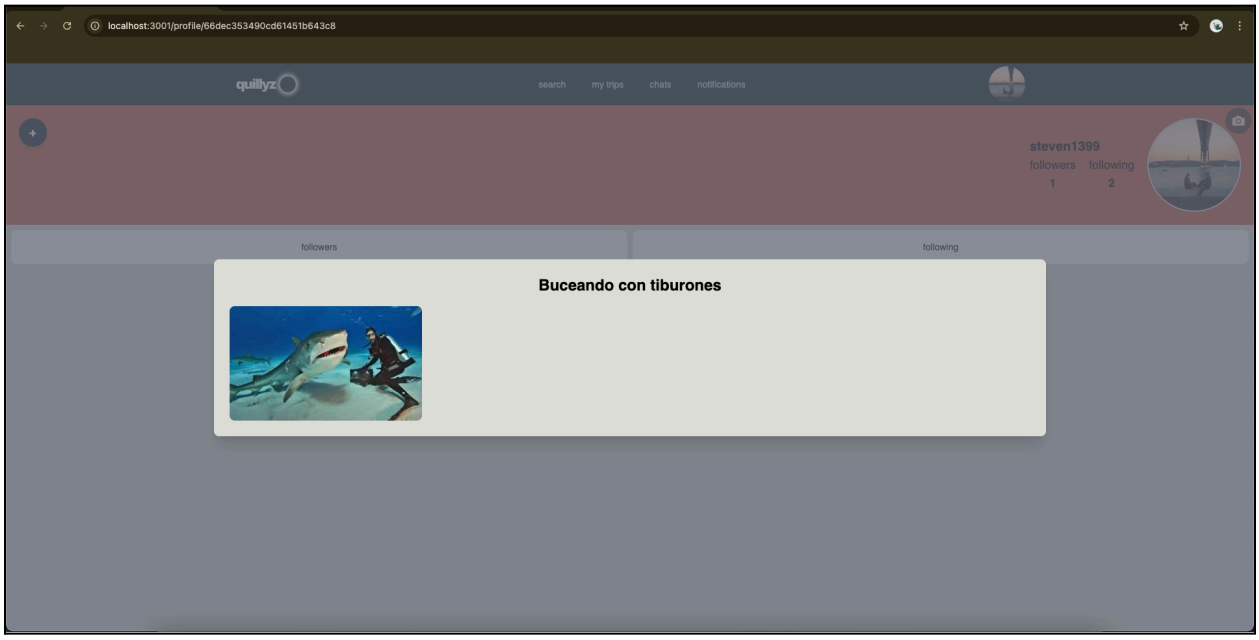

quillyz
localhost:3001/chats
search
my trips
chats
notifications
Search...
test1
test2
test3
to: test1
Hola!
hola! quedamos mañana??
clarooo!
Send a message
Incognito
steven1399
to: steven1399

back

Trip Album


quillyz
localhost:3001/profile/66dec353490cd61451b643c8
quillyz
search
my trips
chats
notifications
steven1399
followers
following
1
2
followers
following
posts
trips
Playas de Italia
Buceando con tiburones


quillyz
localhost:3001/profile/66dec353490cd61451b643c8
quillyz
search
my trips
chats
notifications
steven1399
followers
following
1
2
followers
following
Playas de Italia
Choose File
download.jpg
post
trips
Playas de Italia
Buceando con tiburones
upload


localhost:3001/profile/66dec353490cd61451b643c8
quillyz
search
my trips
chats
notifications
steven1399
followers
following
1
2
followers
following
Buceando con tiburones

back

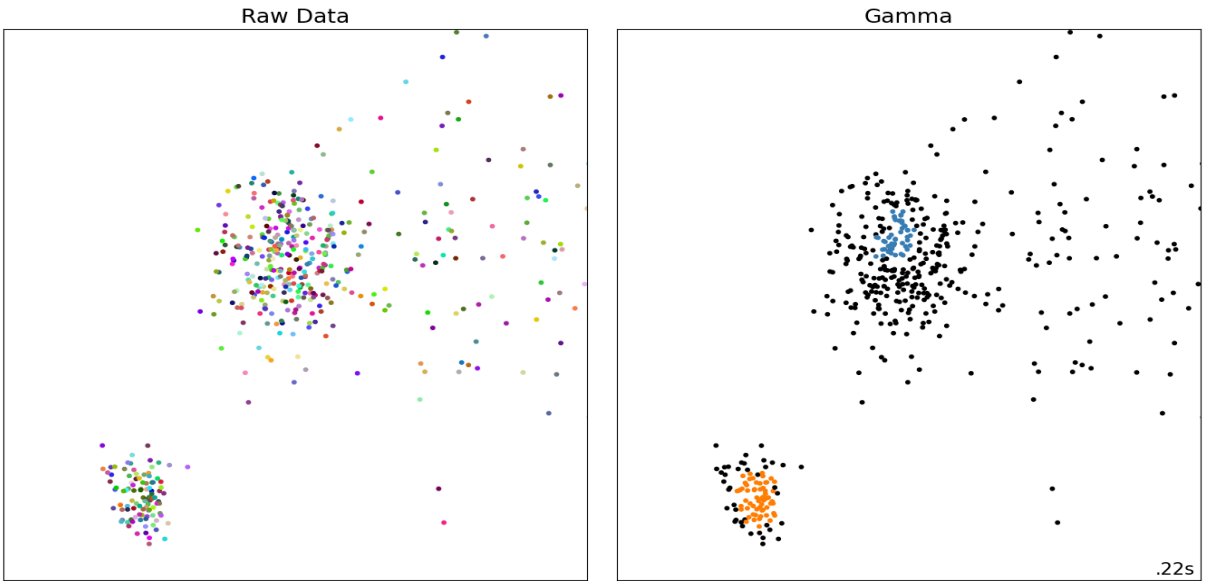
Algorithm Examples

Alpha example

Images generated using pyplot

back

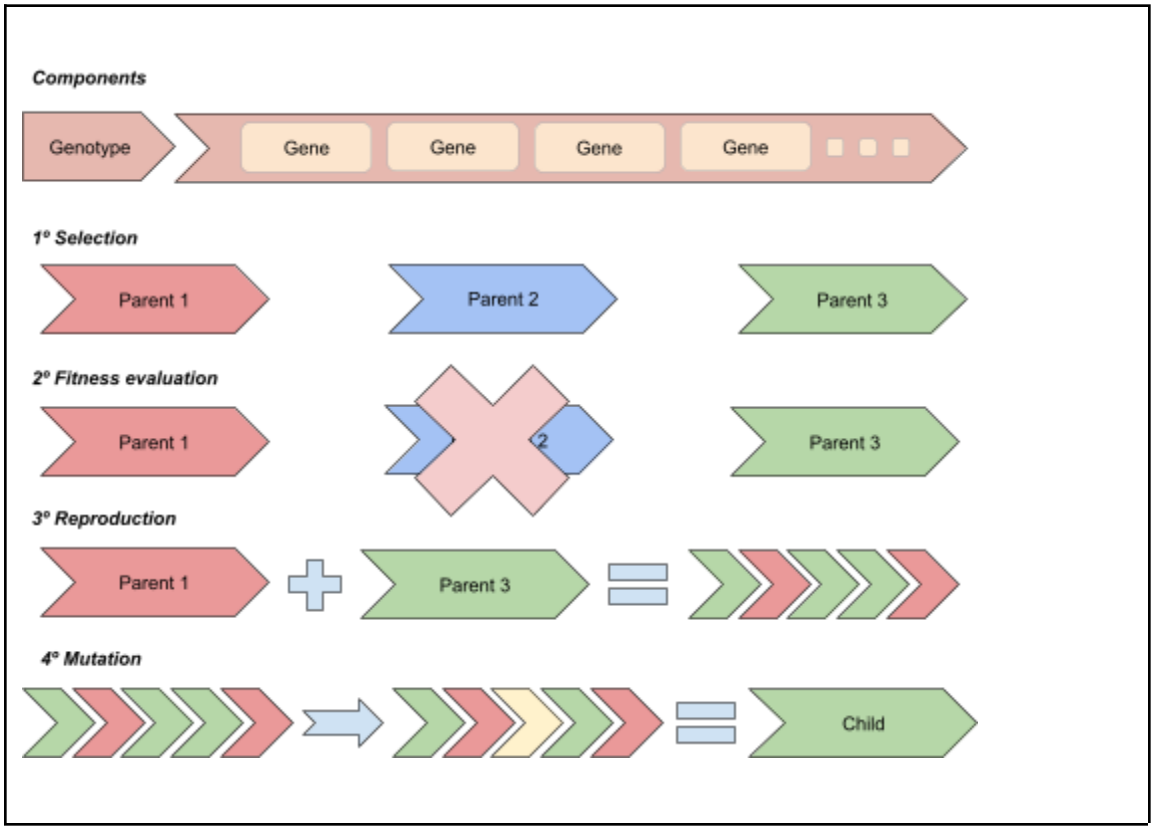
Gamma example



Images generated using pyplot

Back

Omega Evolution Stages



Back

Omega example

Details :

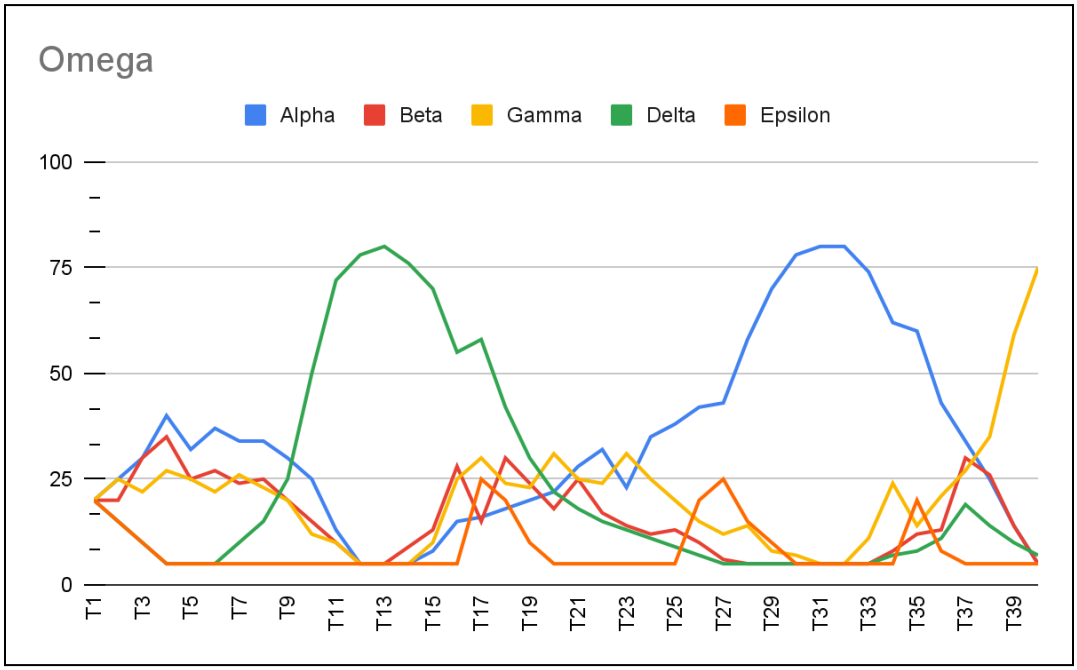
Stages T1 - T5: *Show arbitrary selection of recommended trips.*

Stages T6 - T13: *Show a continuous selection of friend oriented trips (Delta).*

Stages T14 - T23: *Show more arbitrary selections.*

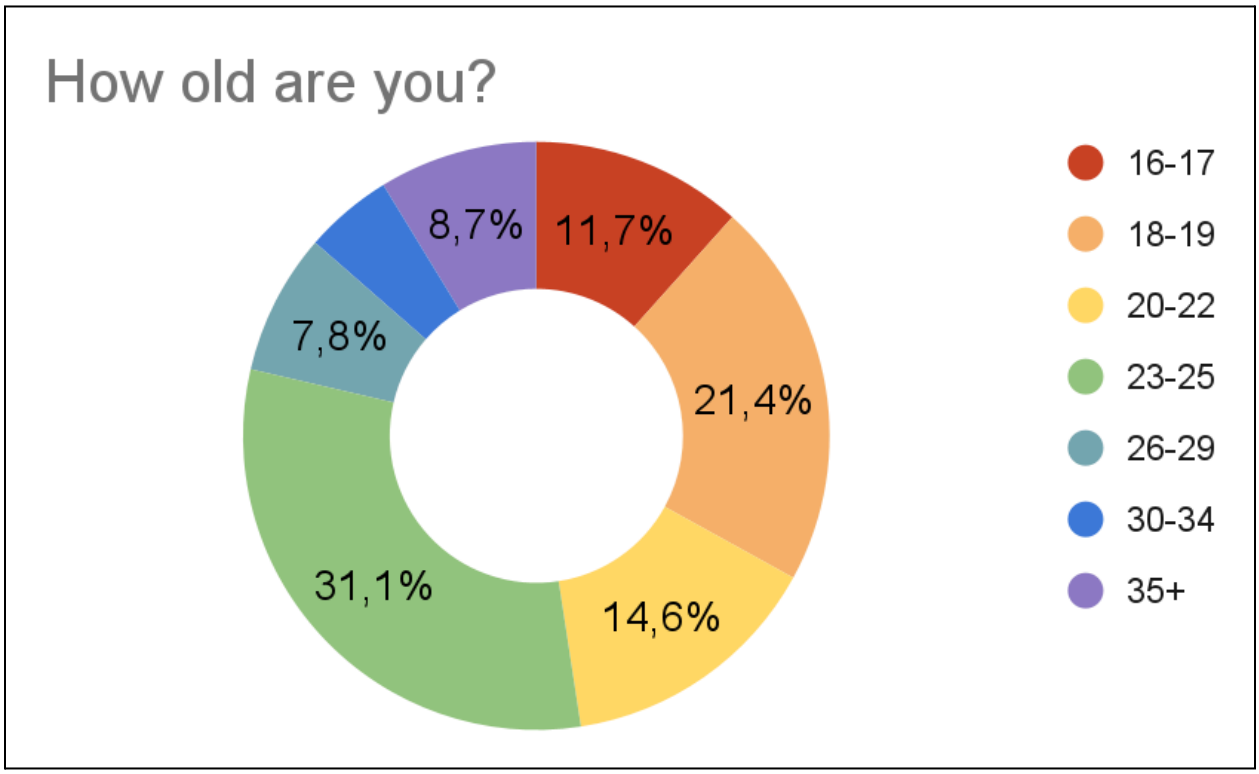
Stages T24 - T31: *Show focused selections of cost, distance oriented trips (Alpha).*

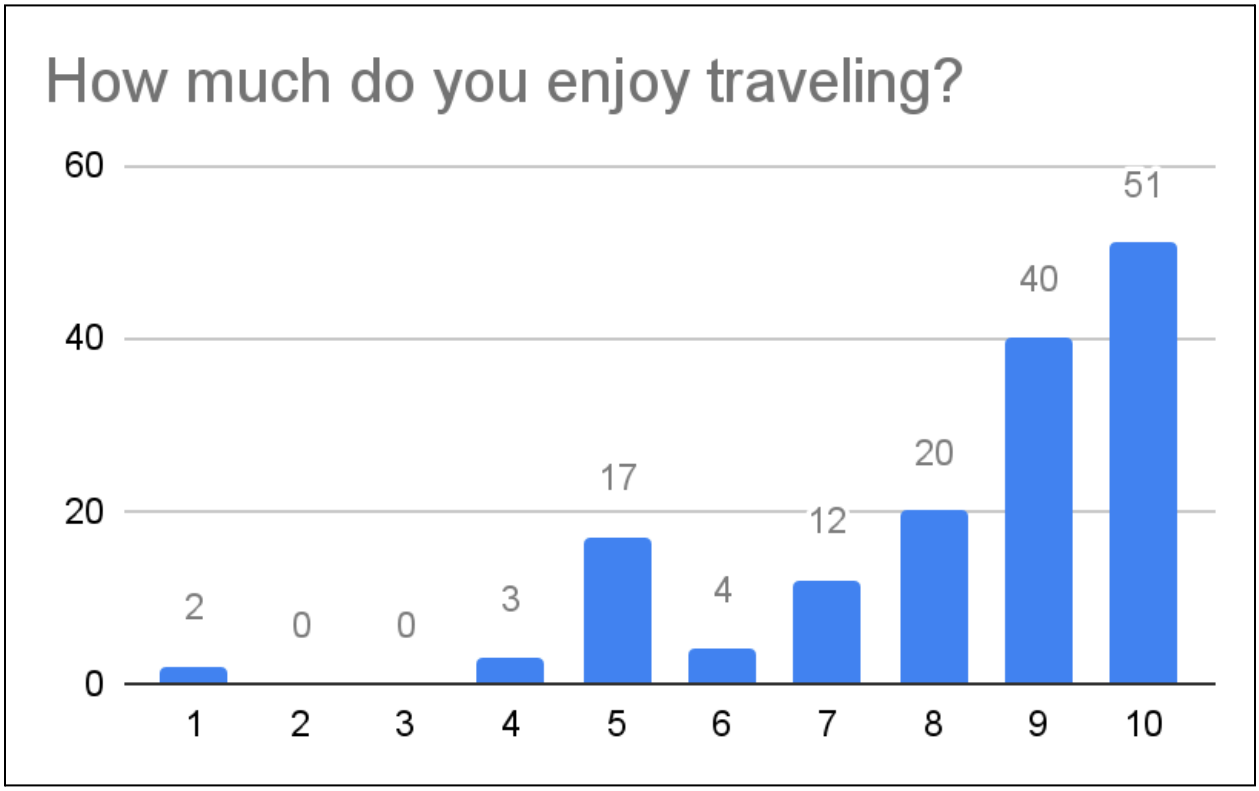
Stages T32 - T 40: *Show selections aimed towards a user matching recommended Selection (Gamma).*

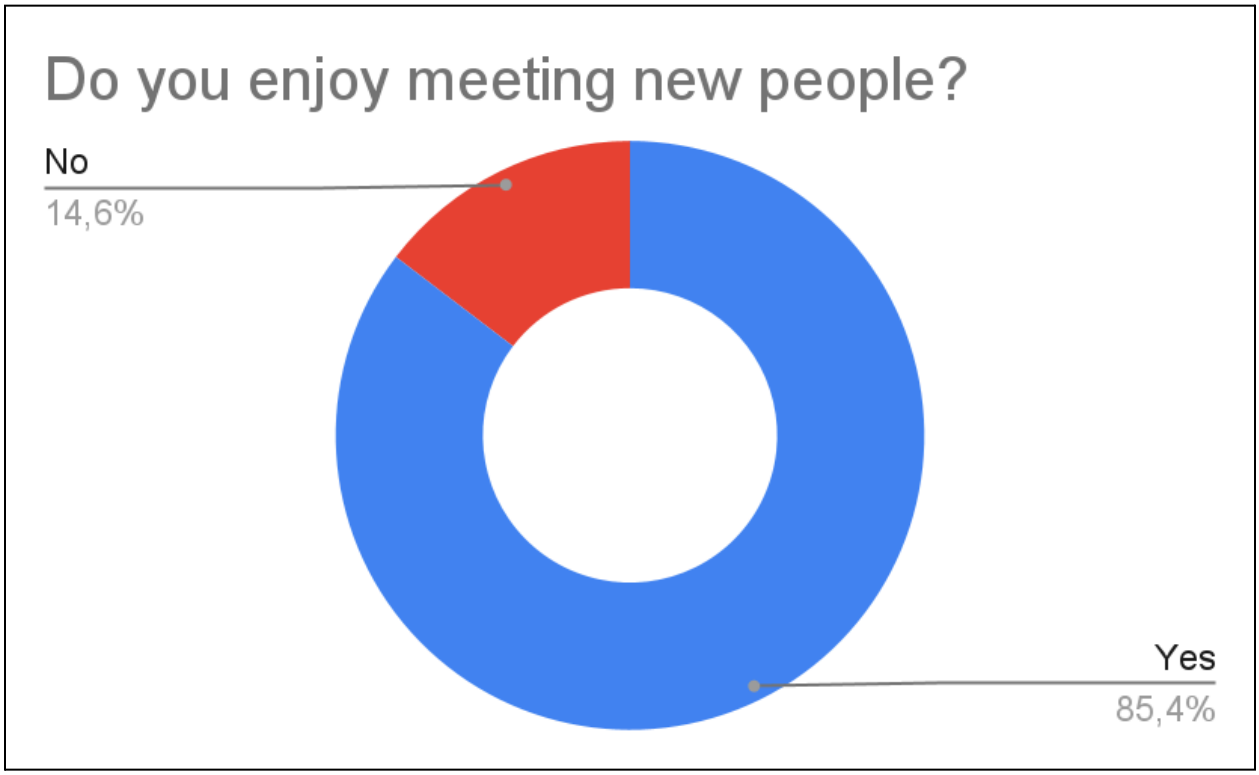


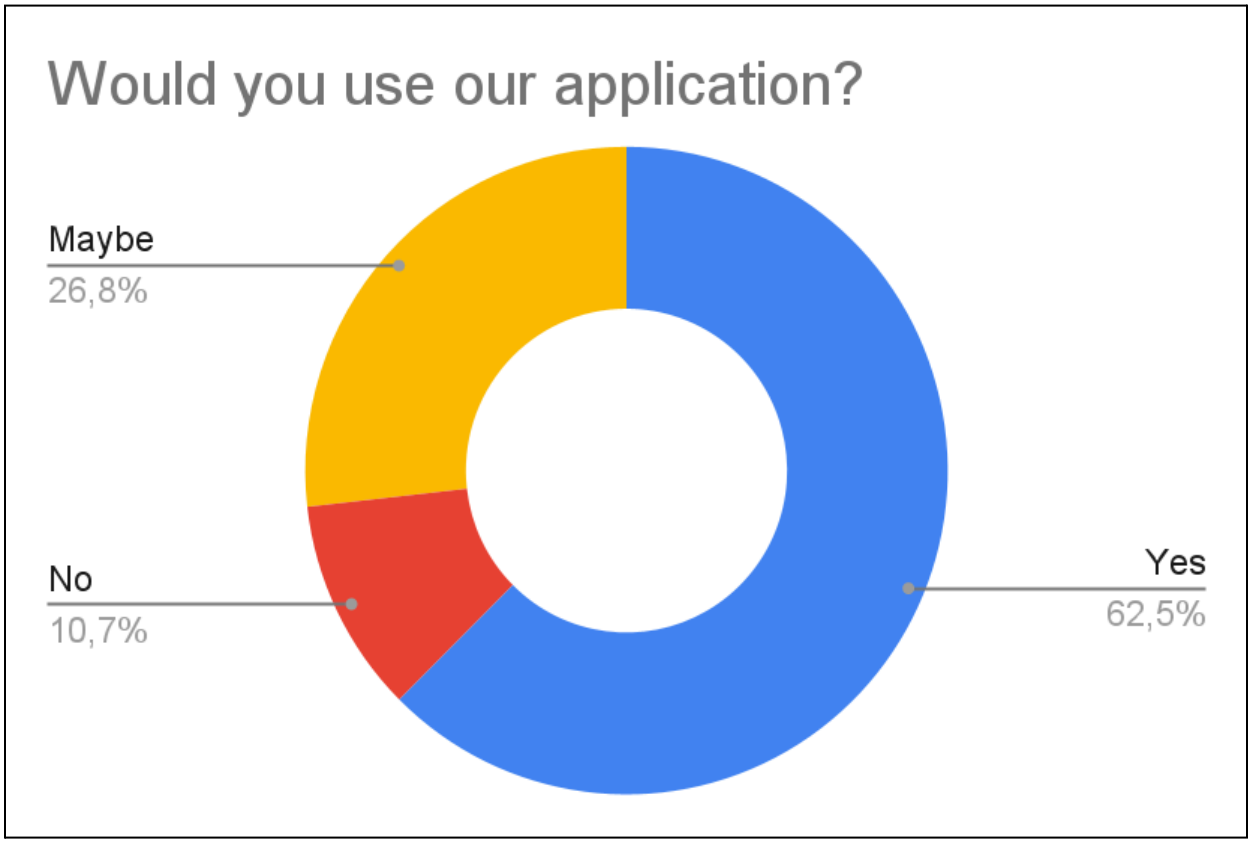
Back

Questionnaire

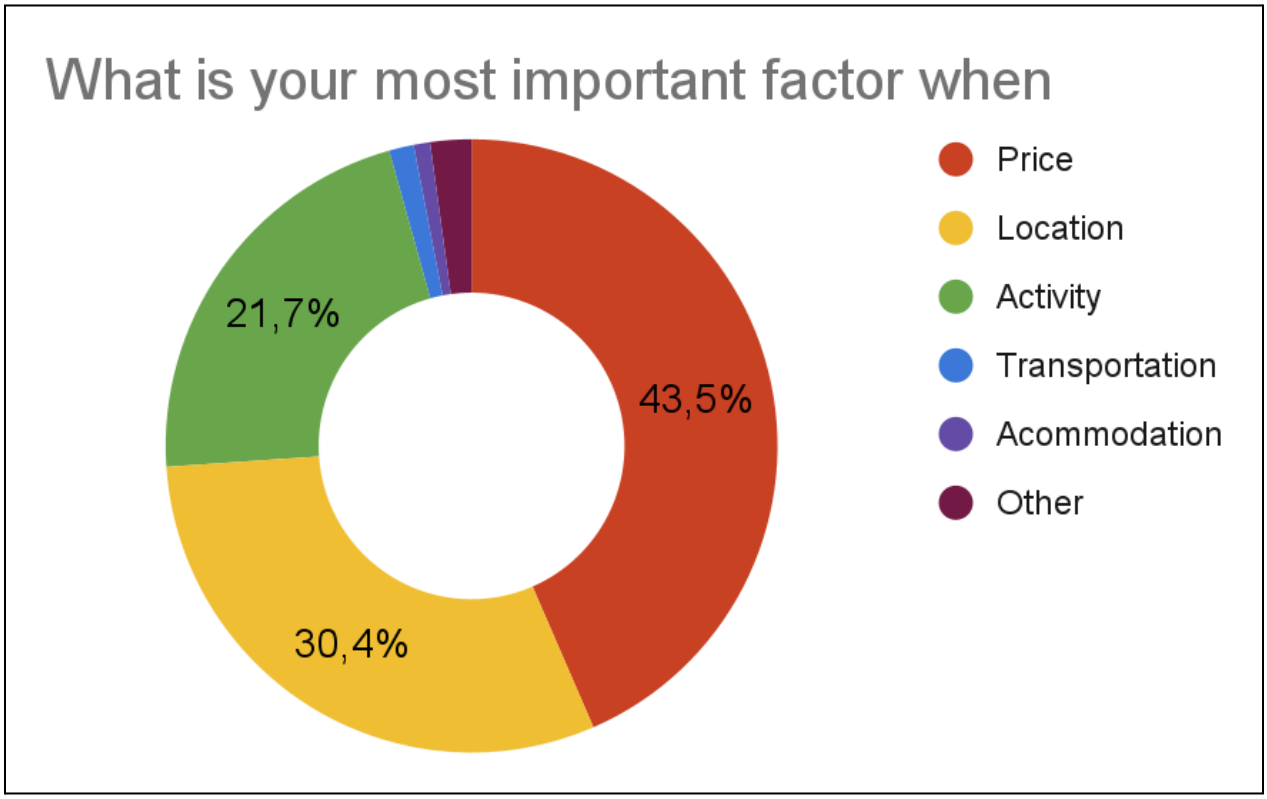





Do you enjoy meeting new people?
No
14,6%
Yes
85,4%


Would you use our application?
Maybe
26,8%
No
10,7%
Yes
62,5%

What is your most important factor when choosing a trip?



back

Tables

Questionnaire

Question	Options	Answer Analysis
How old are you?	"Open answers"	Most individuals fall under the young adult demographic around the ages of 23 to 25.
What are you currently doing?	Studying	The responses show that most people are currently workers. After that, the rest are students.
	Working	
	Neither	
How much do you enjoy travelling?	1 - 10	Most users reported that they love travelling, indicating that Quillyz has a highly engaged audience who are passionate about exploring new destinations.
Do you enjoy meeting new people while travelling?	Yes	Many users enjoy meeting new people, suggesting that the social networking aspect of the app is appealing.
	No	
Would you consider using our application to plan your travels and/or join group trips?	Yes	A majority of respondents showed a strong interest in using Quillyz.
	Maybe	
	No	
What is the most important factor to you while travelling?	"Open answers"	Pricing, location and activity are the most valued, but other values such as transport, accommodation, security and accessibility also were important.
How do you prefer to travel?	Solo	Most users prefer travelling with friends or family, After that, both solo travelling and travelling with our recommended participants are both viewed positively.
	With Friends	
	With Family	
	With Recommended People	

When planning a trip, do you prefer to:	Preplanned	Users generally prefer a mix of planning and flexibility, indicating that Quillyz should offer customizable itineraries that leave room for spontaneous activities.
	Spontaneously	
What type of trips do you prefer?	"Open answers"	Adventure trips are the most popular type of travel among users, as well as beach, city, cultural trips and many more.
How do you typically find travel companions?	I ask People I already know	Most users prefer travelling with people they already know, yet a high number of individuals show interest in using applications.
	I use apps and social platforms	
	I go solo	
How important is it for you to have recommendations tailored to your preferences?	1 - 10	Personalised recommendations are crucial for most users, indicating that Quillyz should focus on tailoring trip suggestions based on user profiles, interests, and previous activities.
Would you be interested in sharing your travel experiences on a social platform?	Yes	While many users enjoy sharing their experiences, they may prefer more private or controlled sharing settings, meaning Quillyz should offer both public and private sharing options.
	Maybe	
	No	
How often do you travel?	Every week	Most users travel multiple times a year, which indicates frequent opportunities to use Quillyz for both planning trips and staying connected with travel companions.
	Every month	
	Every Year	
What would make you more likely to use Quillyz?	Planning features	Users are most attracted to personalised trip suggestions. Quillyz should continue to refine its recommendation algorithm to provide highly relevant and appealing options.
	Social Network	
	Trip Recommender	
Which aspect of a travel app is most important to you?	Easy to use interface	The ease of use is a top priority for most users, meaning Quillyz should focus on providing a smooth, intuitive interface

	Customizable trip planning	that makes travel planning simple and enjoyable.
	Ability to connect with others	

back