WEST UNIVERSITY OF TIMIŞOARA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
BACHELOR STUDY PROGRAM: Computer Science in English

# Bachelor Thesis

**SUPERVISOR:**
Lect. Dr. Monica Sancira

**GRADUATE:**
Rareş-Cătălin Trif

**TIMIŞOARA**
**2025**

**WEST UNIVERSITY OF TIMIŞOARA**
**FACULTY OF MATHEMATICS AND COMPUTER SCIENCE**
**BACHELOR/MASTER STUDY PROGRAM: Computer Science in English**

# Excursia - A Social Web Application for Events

**SUPERVISOR:**
Lect. Dr. Monica Sancira

**GRADUATE:**
Rareş-Cătălin Trif

**TIMIŞOARA**
**2025**

# Abstract

This paper describes the design and development of a social web application for posting events. This project was implemented using React framework for the front-end, Node run time environment for the backend and OpenAI API for smart recommendations, with a goal in mind to make event planning more casual. Users can create events, scroll through the feed and swipe for recommendations, just like modern social applications.
The study examines the technologies and approaches taken during development process in order to create the web application. Additionally, the app offers the possibility to add friends and chat with them in real time, check their profile and apply for interesting events. Future work can be done to improve the recommendation system and bring more options to customize the creation of events and user profiles more authentic.

**Keywords:** Social web application, Recommendation system using LLMs, Real-time chatting, Swiping Interface, TypeScript

# Contents

**1 Introduction** **6**
- 1.1 Motivation . . . 6
- 1.2 Problem Domain . . . 6
- 1.3 Objective . . . 6
- 1.4 Thesis Description . . . 7

**2 Related Work** **8**
- 2.1 Analisys of existing competition . . . 8
- 2.2 Academic Foundations in Recommendation . . . 9

**3 Application Development** **10**
- 3.1 Application Design . . . 10
  - 3.1.1 Create Event . . . 11
  - 3.1.2 Recommendation Page . . . 12
  - 3.1.3 Chat . . . 13
- 3.2 Backend Architecture . . . 14
- 3.3 Frontend Architecture . . . 15

**4 Implementation** **17**
- 4.1 Technology Stack . . . 17
  - 4.1.1 Backend technologies . . . 17
  - 4.1.2 Frontend Technologies . . . 19
- 4.2 Feature Implementation Examples . . . 20
  - 4.2.1 Create Event . . . 20
  - 4.2.2 GPT-Based Event Recommendation . . . 20
  - 4.2.3 Swipeable Event Cards . . . 21
- 4.3 Authentication and Authorization . . . 22
  - 4.3.1 OTP Generation and Email Delivery . . . 22
  - 4.3.2 OTP Verification and Token Issuance . . . 23
  - 4.3.3 Route Protection with Middleware . . . 23
  - 4.3.4 Front-end Token Handling . . . 23
- 4.4 Database Integration . . . 23
  - 4.4.1 Database Initialization . . . 23
  - 4.4.2 Schema Overview . . . 24
  - 4.4.3 Data access and consistency . . . 24
  - 4.4.4 Design Rationale . . . 26
- 4.5 API Testing with Postman . . . 26

**5 User Manual** **27**
5.1 Login and Register . . . . . 27
5.2 OTP Verification . . . . . 27
5.3 Home Page . . . . . 28
5.3.1 Navigation bar . . . . . 28
5.3.2 Event Card . . . . . 29
5.3.3 Sidebar and Friend list . . . . . 29
5.3.4 Explore page . . . . . 29
5.4 Create Event . . . . . 29
5.5 User Profile . . . . . 30
5.6 Recommendation Page . . . . . 32
5.7 Notification page . . . . . 32
5.8 Chats page . . . . . 33
5.9 Event page . . . . . 34

**6 Conclusion and Future Directions** **35**
6.1 Future Work . . . . . 35
6.2 Closing Remarks . . . . . 35

**Bibliography** **37**

# Chapter 1

# Introduction

## 1.1 Motivation

For a long time, communities have gathered in town squares, parks or private homes to share experiences, exchange ideas and strengthen social bonds. As societies modernized, so did the means of planning, much of it shifting from face-to-face invitations to telephone calls, and with the internet becoming part of our daily life, planning moved online. Friends began creating events on different networks and sharing details in threaded comments. In response to the growing demand for dedicated tools, specialized event-planning applications appeared, offering calendars and simple recommendation engines based on user profiles or location. More recently, the rise of artificial intelligence has prompted platforms to experiment with it, trying to take advantage of semantic understanding and contextual cues to offer personalized suggestions that go beyond static filters.

## 1.2 Problem Domain

Event discovery and social coordination pose a complicated intersection of information retrieval, user-modeling, and real-time interaction. Users expect recommendations to respond to their changing preferences and tastes. Moreover, social features including profiles and chat, should blend as well, rather than feeling as if they were tackled on as part of a secondary effort. Creating something that accounts for all of these criteria involves careful design of back-end services and on-screen interaction, as much as well executed integration of AI components that the user can rely on and trust.

## 1.3 Objective

This work aims to develop an event-discovery platform that takes advantage of a large language model's contextual reasoning to deliver truly personalized recommendations. By integrating a swipe-based browsing interface, users can rapidly express interest or pass on suggested events, feeding back implicit preferences into the recommendation engine. Alongside this core innovation, the application provides the familiar social-app pillars: a continuously updated event feed, rich user profiles, and real-time chat functionality. The result is a unified environment where discovery, planning, and coordination occur seamlessly, guided by intelligent, context-aware suggestions.

## 1.4 Thesis Description

The rest of this thesis is organized as follows. Chapter 2 presents the related work and comparison with similar applications on the market, defining the competitive advantage of Excursia. Chapter 3 presents the system architecture, along with the reasoning for some of the more important design decisions. Chapter 4 presents narrative walkthroughs of the implementation of core features, which include the event creation, AI-driven recommendations, swipe interface, and chat. Chapter 5 offers a user manual that will explain how a user can interact with the application. Finally, Chapter 6 presents conclusions drawn from the project and suggests future areas for development and enhancement. The organization of the thesis should explain the motivations for Excursia's design, as well as a solid account of the final product achieved, a one-stop, secure, smart event-discovery platform.

# Chapter 2

# Related Work

## 2.1 Analisys of existing competition

Meetup and Eventbrite are probably the most common services for managing local events and activities. Meetup provides opportunity to join groups by topic and by keyword guess, but does not adequately support a recommendations engine (to find other relevant or similar groups) based on implicit feedback, or semantic matched items, largely relying on user declared interest and trending activity. Eventbrite, allows filtering by keywords and augmented by location and dates, likewise has the shortcoming of allowing members to sort through pages of irrelevant events on search result pages. Facebook Events, being part of the broader Facebook system, is a widely used tool which provides a very accessible way to discover and publicize events. It allows users to create public or private events, invite friends, track RSVPs [1], and engage with attendees through discussion threads. One of its key strengths is its tight integration with users social graphs, enabling event discovery through friend activity and mutual interests. This social aspect significantly enhances visibility and organic reach, particularly for informal gatherings or community-based activities, but at the same time, Facebook having such a broad content ecosystem, events can go unnoticed because the user is overwhelmed with so much information.

| Main functionalities | Meetup | Eventbrite | Facebook Events |
|---|---|---|---|
| Chat for users | YES | NO | YES |
| Casual event planning | NO | NO | YES |
| Event feed | YES | YES | YES |
| RSPV tracking | YES | YES | YES |
| Swiping system | NO | NO | NO |

Table 2.1: Comparison of the main features.

Despite the extensive capabilities of these platforms, each has its limitations (Figure 2.1). Meetup may be too formal or structured for casual users, while Eventbrite is focused more on professional organizers than on fostering interaction and Facebook Events, as socially powerful as it is, can be too shallow for users looking for a more tailored or event focused experience. These gaps present oppor-

[1]RSVP stands for *répondez s'il vous plaît*, a French phrase meaning "please respond"

tunities for new solutions that offer a hybrid approach, trying to bring solutions for more oriented towards casuals users, such as swipe-like system.

## 2.2 Academic Foundations in Recommendation

Academic research on recommendation systems has progressed from collaborative filtering (CF) and content-based methods to complex hybrids and neural architectures. CF techniques predict preferences by examining patterns of similar users' behavior [RIS+94], while content-based filters match item attributes to user profiles [LdGS11]. Both perform well with abundant data but suffer "cold-start" problems when new users or items lack history [RRSK11].

In parallel, social-based collaborative filtering methods exploit network relationships to mitigate data sparsity and cold-start issues. Algorithms like *Hete-CF* [LPW14] incorporate multiple types of social relations in heterogeneous networks to improve recommendation accuracy in event-based social platforms. More recent frameworks, such as Social Temporal Excitation Networks (STEN), directly model fine-grained temporal influences among friends' behaviors using temporal point processes, yielding explainable, event-level recommendations.

As we can see, there are many different approaches to the event recommendation problem. Large Language Models (LLMs) represent the latest frontier in recommendation research. Their ability to understand context and generate structured outputs has led to early explorations of conversational recommendation, where multi-turn dialogues refine suggestions interactively. Surveys highlight the potential (and open challenges) of LLMs for recommendation tasks [WZQ+24]. Crucially, function-calling APIs allow developers to enforce strict JSON schemas on LLM outputs, marrying the model's flexibility with the predictability required in production systems. Despite this promise, few consumer-facing event-discovery applications have adopted LLMs to deliver semantic, schema-compliant recommendations at scale.

# Chapter 3

# Application Development

## 3.1 Application Design

This section outlines the design of the application from the user's perspective. For a clear view, a use-case diagram (Figure 3.1) can be used and offers a high-level overview of the interactions a user can do. It serves as a foundation for understanding how the app is structured in terms of user experience and functional components.

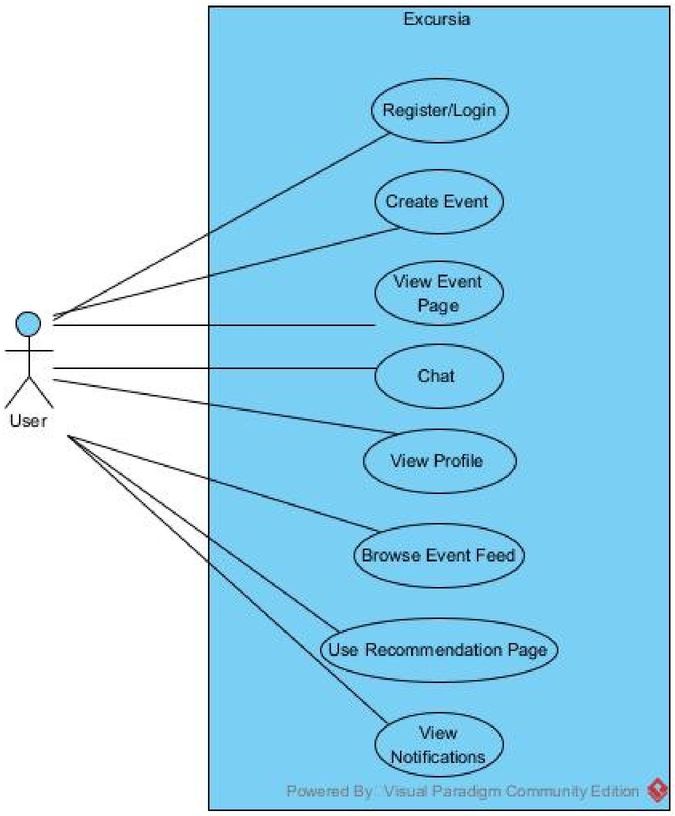


Figure 3.1: Use Case Diagram for Excursia

**Register/Login:** The first interaction of the user is with the login page, where he is asked to enter an email. Next step is to enter the OTP that was sent on the email address to be able to connect. The design makes it simple, automatically creating an account if the email is not found in the database.

**View Event Page:** This page offers offers all the details inserted in the create event form. The user can also see all the attendees and a comment section where he can ask questions regarding the event. It also has a three buttons, one for applying to the event, another for showing interest and a sharing the event.

**View Profile:** Users can view their own profile as well as the profiles of others. A profile contains basic information, such as name, profile and cover images, description, custom tags, a showcase of maximum six photos and finally a list of created events. If the current user is checking his profile, it has buttons that allow him to modify any information shown on the page. It serves as a social and identity layer within the app.

**Browse Event Feed:** The event feed acts as the home and explore pages, showing the events posted by the user's friends and any other public event posted. Users can scroll through the feed to discover events happening in their area or within their interest categories.

**View Notifications:** Notifications alert users to important updates such as friends requests and events applications. Also here he can accept or reject the requests. This keeps users engaged and informed about relevant activity on the platform.

### 3.1.1 Create Event

One of the key features of the application, is the ability for users to create events. The event creation form requires several input fields such as event title, description, date, time, maximum number of attendees, location, and an optional image upload. This feature enables users to contribute with content to the platform and initiate engagement from others.

As illustrated in the sequence diagram (Figure 3.2), once the user fills out the form, the frontend application attaches the authentication token and sends a 'POST' request to the server. The server validates the token through the authentication middleware, processes the uploaded image via the file upload middleware, and stores the complete event information in the database. Upon successful creation, the server responds with the event details, and the frontend confirms success to the user through a toast notification or redirect.

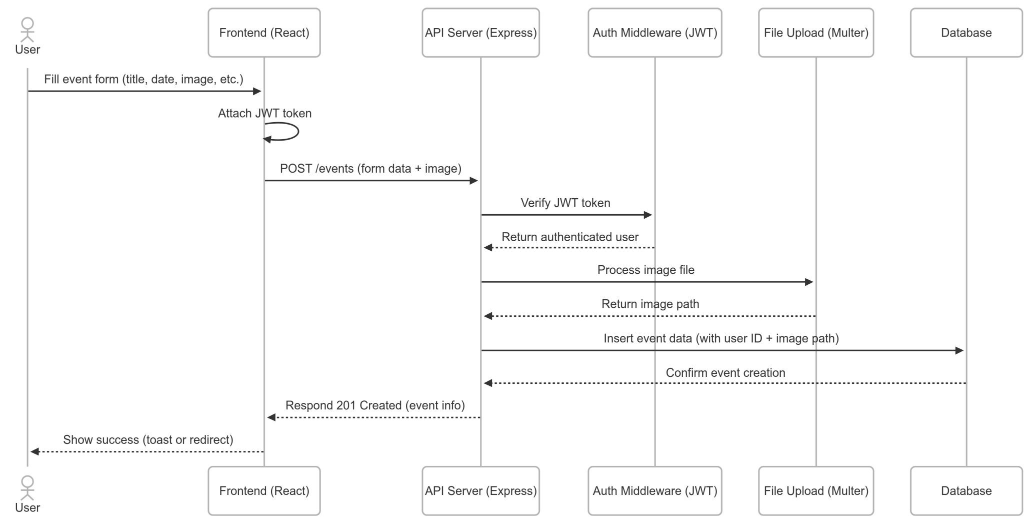

User
Frontend (React)
API Server (Express)
Auth Middleware (JWT)
File Upload (Multer)
Database
Fill event form (title, date, image, etc.)
Attach JWT token
POST /events (form data + image)
Verify JWT token
Return authenticated user
Process image file
Return image path
Insert event data (with user ID + image path)
Confirm event creation
Respond 201 Created (event info)
Show success (toast or redirect)

Figure 3.2: Sequence Diagram for Create Event page

### 3.1.2 Recommendation Page

An important feature of the application is the event recommendation system, which uses OpenAI's GPT model to filter the events based on user interests. This system provides a personalized experience by analyzing event data and matching it with user tags.

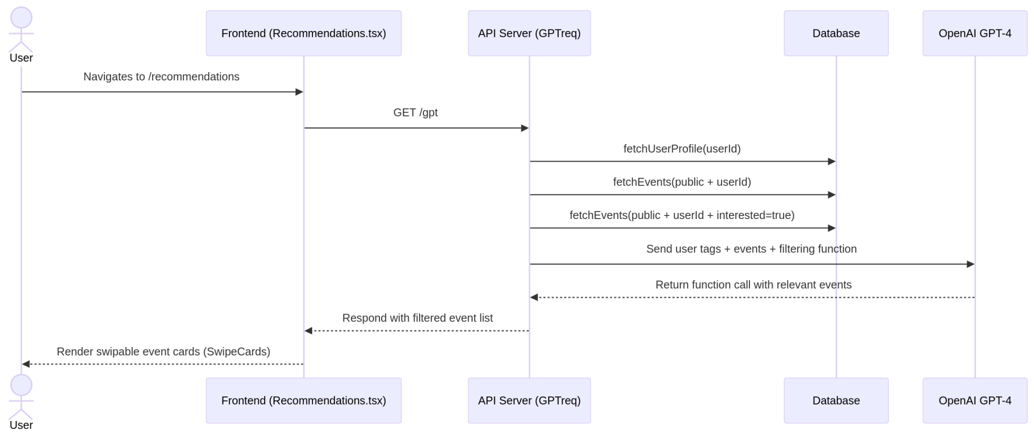

User
Frontend (Recommendations.tsx)
API Server (GPTreq)
Database
OpenAI GPT-4
Navigates to /recommendations
GET /gpt
fetchUserProfile(userId)
fetchEvents(public + userId)
fetchEvents(public + userId + interested=true)
Send user tags + events + filtering function
Return function call with relevant events
Respond with filtered event list
Render swipable event cards (SwipeCards)

Figure 3.3: Sequence Diagram for Recommendation page

As shown in the sequence diagram (Figure 3.3), when the user opens the Recommendations page, the front-end triggers a request to the server. The backend then retrieves the user's profile and a list of public events from the database. This data, including event details and user tags, is forwarded to GPT-4 using the function calling interface. GPT analyzes the event relevance based on semantic similarity and returns only those that match the user's interests. The server responds with the filtered list, which is then displayed to the user in an swiping interface.

### 3.1.3 Chat

The next sequence diagram (Figure 3.4) depicts the primary flow of a one-to-one chat session within Excursia's application design. The process begins when the user opens a chat window on the client side and the client requests the existing message history from the server. The server then requests the message history for that conversation, sends it back to the client and the client renders the previous messages.

Once the initial history is loaded, a real-time conversation takes place naturally in a loop: each time the user sends a new message, the client sends it to the server which then sends that broadcast back to all users in the chat (which is just those two users). As each broadcast is received, the client shows that latest incoming message in real-time, which creates an apparent easy exchange of messages, without page refreshes (or manual refreshes).

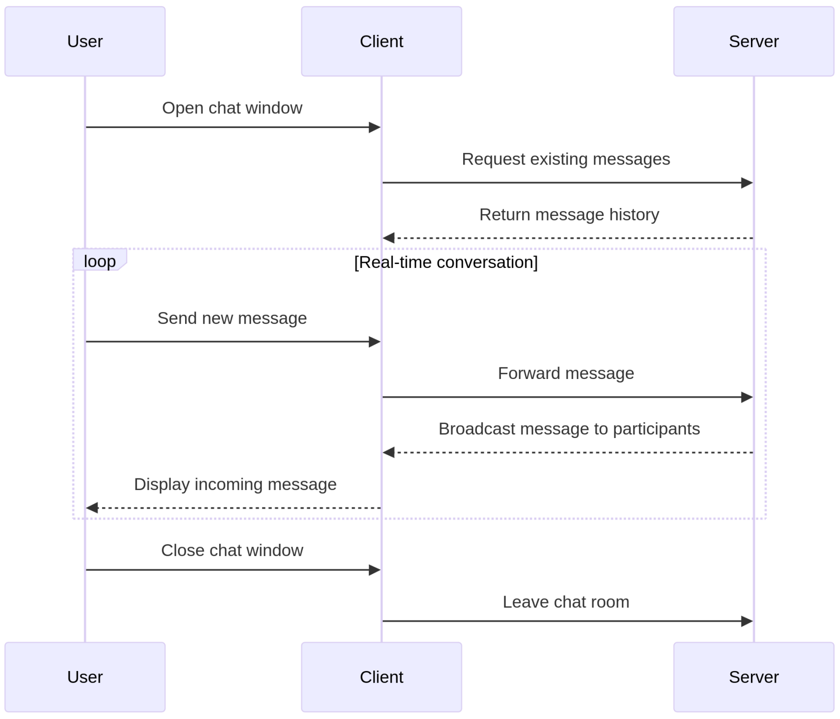


Figure 3.4: Sequence Diagram for Chat page

Lastly, when the user closes the chat window, the client notifies the server that it is leaving the chat room. This step makes sure the server can update presence indicators or resource allocations that are connected to active chat sessions. All of these interactions define a simple, reliable real-time messaging protocol that powers Excursia's chat feature.

## 3.2 Backend Architecture

The backend of Excursia is built using a modular architecture that promotes scalability and separation of problems. It is implemented in TypeScript using the Express.js framework and follows a layered structure (Figure 3.5) that organizes responsibilities into distinct categories such as routing, controllers, middleware, and database access.

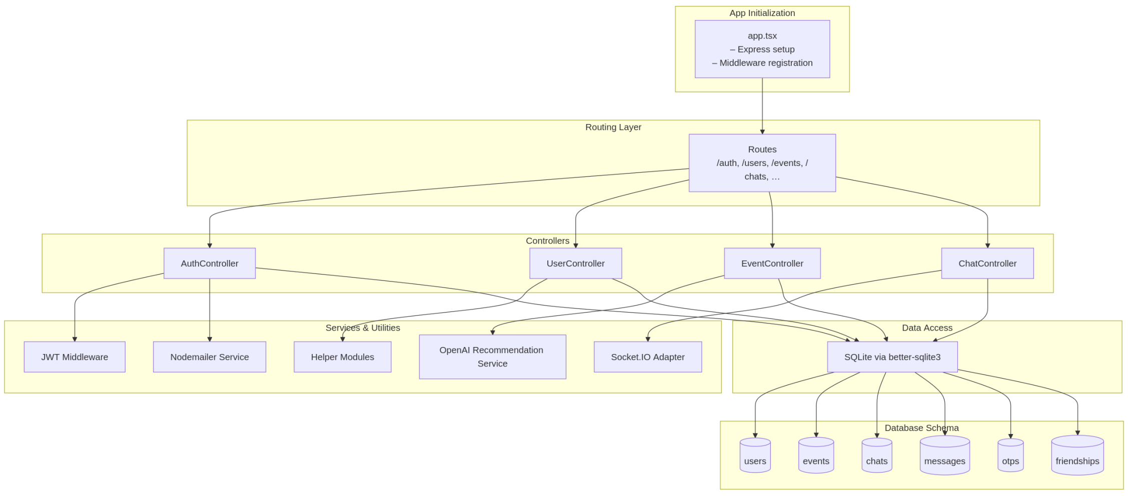


Figure 3.5: Flow diagram for Backend

At it's base, the app file sets up the Express application by registering middleware and attaching route handlers. The routing layer, defined in the routes directory, maps incoming HTTP requests to the appropriate controller functions, then each route is grouped by domain specific responsibilities, such as events, users, or authentication.

Controller logic resides in the controllers folder, where each one handles request processing, validation, and interaction with either utility modules or the database. These functions are intentionally kept thin and declarative, delegating business logic to reusable utilities or helpers when appropriate.

Authentication is handled using JSON Web Tokens (JWT), with a middleware module responsible for verifying tokens and attaching user data to requests. This layer ensures that protected routes can only be accessed by authenticated users.

All data persistence is managed through a SQLite database, interfaced via the `better-sqlite3` library. The database schema is initialized in a single module, where all tables and relationships are defined at application startup. The schema includes tables for users, events, chats, messages, OTPs, friendships, and more.

Additional functionality, such as OpenAI GPT integration for recommendations and Nodemailer for sending OTPs, is encapsulated within controller logic. These external services are used sparingly and are clearly separated from the main logic.

The overall architecture emphasizes synchronous control flow, reduced external dependencies, and quick iteration cycles, making it ideal for the scale and goals of this project.

## 3.3 Frontend Architecture

The frontend of Excursia is organized around a layered architecture (Figure 3.6)that gives priority to separation of concerns, modularity and extensibillity. In this project, Vite is used for development builds and hot-module replacement, and SWC is responsible for TypeScript and JSX compilation. This provides fast feedback loops without prescribing an architecture.

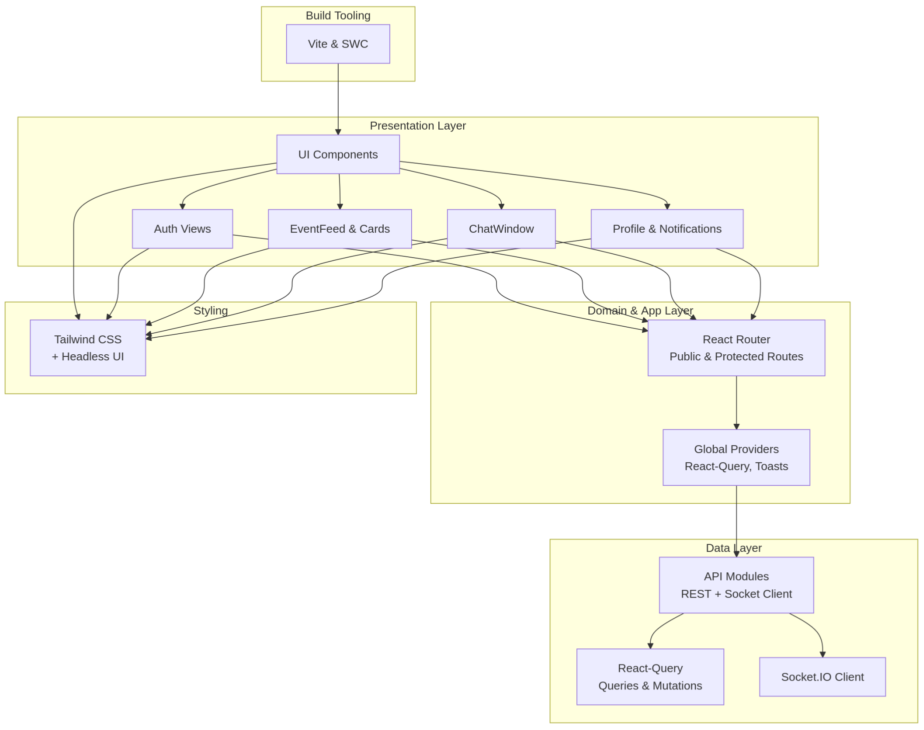


Figure 3.6: Flow diagram for Frontend

Above that, the application is structured into three primary logical levels. The first tier is the presentation layer, which contains reusable UI components and page views. Components are grouped by features (e.g. authentication, chat, event feeds, layout) and exposed to pages that correspond directly to routes. This organization assures that each domain can evolve independently.

The second tier is the domain and application layer where state management and routing lives. React Router allows for public and protected routes to be defined in a single shell component, enforcing authentication at the boundary instead of spreading guard logic throughout the codebase. Global concerns such as notification toasts and query caching contexts are injected via top level providers. This keeps the business logic separate from how it's consumed in the UI.

Lastly, the data layer abstracts away all of the interactions with backend services. A group of API modules wraps network calls and real-time socket setup, providing well-typed interfaces to the rest of the app. The data layer has the react-query library as its core, which configures fetches, mutations, caching, and background updates based on declarative rules, not event handlers. Real-time messaging is also not tightly coupled, using it's own socket client.

Styling is kept consistent by leaning on Tailwind and a UI component library. Developers will use utility classes to control layout, typography, and spacing. Any complex or cross-cutting UI element, such as modals or toasts, comes from a unified component set, ensuring the look and feel never drift between pages.

# Chapter 4

# Implementation

This chapter presents the implementation of Excursia, that comes with practical examples for the designs outlined in the previos chapter. While the Application Development chapter is more focused on the structure and user interactions, this section gives a deeper look into the actual technologies and coding practices used to build the app.

The implementation covers both client and server components, detailing how each part of the application was developed, integrated and tested. Functionalities such as event creation, real-time communication and recommendations system will be covered. Where applicable, code snippets, architecture diagrams, and component breakdowns are included to support the discussion and illustrate how specific challenges were handled during the development process.

## 4.1 Technology Stack

The implementation of the Excursia application is based on a modern web development stack, for both client-side and server-side technologies. The following presents an overview of the tools, languages, and libraries used in the development process.

### 4.1.1 Backend technologies

**TypeScript** - A high-level programming language that adds static typing and brings optional type annotations, interfaces, enums and advanced tooling to JavaScript. It offers significant advantages over plain JavaScript, that comes in handy for applications like Excursia. By enforing type safety at compile type, TypeScript reduces possible runtime bugs, makes code more easy to understand and offers editor support, such as auto-completion and inline documentation. This results in a maintainable and robust logic, inspiring for a more structured code base that is beneficial if a project grows larger. [FM14]

**Node.js** – A strong JavaScript runtime built on Chrome's V8 engine, Node.js helps with fast server-side development by handling operations asynchronously and without blocking the main execution thread. This model is particularly suited for scalable applications where high concurrency is required, such as real-time communication systems and API servers. In the context of this project, Node.js serves as the foundation of the backend, allowing smooth integration with newer tooling and libraries like Express, Socket.IO, and better-sqlite3. By using Node.js, the backend benefits from an environment that supports rapid prototyping by making use of its package ecosystem which offers a wide variety of options. [Nod25]

**Express.js** - One of the lightest web frameworks available on Node.js, Express reduces the development of the server logic by abstracting much of the boilerplate code required to set up and manage the routing, the HTTP requests, middleware and the response. Express has a very large feature set for creating RESTful APIs, and has middleware layering, allowing for separation of concerns in your backend logic modules. In this particular project, Express was the base of the backend API layer managing the endpoints for the user authentication flow, creating events, messaging functionality, and integration opportunities with other services like the OpenAI API. Overall, Express has a simpler API than other frameworks. It has relatively good support from the communities as well, so it is a safe option for developers when trying to rapidly develop a web application that users use in a fast, scalable and maintainable way.

**Better-sqlite3** - A library for SQlite in Node.js, that stands out as a superior alternative to the classic sqlite3 package. The traditional one is asynchronous and callback-based, while the better one uses a synchronous API that is significantly easier to use, and according to the official Github page, offers better concurrency. It also offers advanced features such as prepared statements, transactions and functions defined by the user. Its a good fit for small to medium sized apps and Excursia is placed in that category. [Wis25]

**Jsonwebtoken (JWT)** – A popular library that can be used to create and validate JSON Web Tokens, which is a lightweight data struct and self sufficient way to securely transfer information between parties. The JWT standard supports stateless authentication, as it adds essential user information, such as the user's ID or access scope, directly to the token itself and its signed digitally with the help of a secret key. In the Excursia application, JWTs are used to keep track of user sessions when clients communicate with the server and do not require the backend to remember session state.

**OpenAI API** – Used to provide intelligent, context-aware event recommendations by leveraging large language models. This API enables the application to offer personalized content that enhances user engagement and discovery. [Ope25]

**Multer** – Middleware for handling 'multipart/form-data', primarily used for uploading images and files. It simplifies the process of handling file uploads such as event images and user profile pictures.

**CORS (Cross-Origin Resource Sharing)** - Is a security feature for web browsers that restricts cross-origin HTTP requests initiated from scripts running in the browser. CORS configuration helps in avoiding unwanted access while enabling safe communication between client and server, by defining the allowed sources and headers. [MDN25]

**Socket.IO** – A powerful JavaScript library that enables real-time, bi-directional communication between clients and servers over WebSockets, with automatic fallback to HTTP-based transports when needed. Built on top of the Node.js runtime, Socket.IO abstracts the complexity of managing persistent connections, reconnections, and messaging. In *Excursia*, it is utilized to implement interactive features such as the real-time chat system, allowing users to exchange messages instantly without requiring page refreshes or polling. [Soc25]

### 4.1.2 Frontend Technologies

**React** – A declarative JavaScript library used for constructing composable user interfaces. The component-based nature of React allows for modular development, breaking UI elements into reusable pieces. This is an effective means of building dynamic and interactive features like event browsing, filtering, and user dashboards in *Excursia* [Met25]

**Vite** – A modern frontend build tool that offers near-instant server starts and lightning-fast hot module replacement (HMR). Vite improves developer productivity by drastically reducing build times and enabling real-time feedback during development. It also produces optimized production builds with code-splitting and tree-shaking out of the box. [YC25]

**Tailwind CSS** – A utility-first CSS framework that enables rapid UI development using composable classes. Rather than writing custom stylesheets, developers apply utility classes directly in markup, leading to a more consistent and maintainable design system. Tailwind's responsive and theme-aware classes help ensure Excursia offers a clean, adaptive user experience across devices. One more advantage that comes with using Tailwind is the broad list of different component libraries that is available.

**Shadcn/ui** - A headless component library that makes use of Radix UI and Tailwind CSS. It gives you access to unstyled, accessible primitives including dialogs, dropdowns, and modals, so you can style however you'd like, while still maintaining accessibility and usability standards. In *Excursia*, it simplifies the process of developing modern, consistent UI components without compromising flexibility. [sha25]

**Axios** - A promise based HTTP client that makes the process of making asynchronous requests to the backend AP as easy as possible. axios supports request/response interceptors, automatic JSON parsing, automatic error handling thus making it quite an effective way of full stack communication from the frontend application to the backend. In *Excursia*, axios is utilized for many requests, fetching event data, submitting user preferences, and authenticating a user session to name a few.

**Socket.IO Client** – The frontend interface for establishing WebSocket connections with the backend via Socket.IO.

## 4.2 Feature Implementation Examples

### 4.2.1 Create Event

The event creation feature of Excursia serves an essential function in enabling users to produce content and increase community engagement. This feature necessarily incorporates multiple components that exist on the back-end: authentication, file manipulation, input validation, and database interaction.

When a user requests to create a new event, the request goes through JWT authentication middleware. JWT authentication uses a middleware regarding the token to confirm the user is authenticated; the event-creation route should only accept authenticated requests. After authentication, any image file that is uploaded, (to be used for the event, if a promotional photo is available) is uploaded using Multer middleware, which is specifically designed for multipart/form-data. The component above allows the user to enter textual event details and optionally an image, in one go.

The event data generally includes title, description, date, and location. When the server receives the request, it checks the fields to make sure that the required fields are all present. If the validation succeeds, the back end builds a database insertion query using better-sqlite3. The reason to use that library to access the database is for a synchronous operation with as little overhead as possible, which makes it a better choice for applications that expect the middle traffic and low-latency responses.

Once the event is recorded in the database, the server can respond with the id of the newly created record. Optionally included fields — like image URL, and maximum attendees — are conditionally parsed into the database call when present; this way, the end user does not need to have that through the whole flow of adding an event and does not have to overcomplicate the flow of the application or hinder it with core functionality.

This implementation sets up the overall data integrity, enforces secure access control, and builds an efficient, friendly way to populate the app with meaningful content. e

### 4.2.2 GPT-Based Event Recommendation

To promote personalization and explore content, Excursia uses GPT-4 in the OpenAI API as a recommendation engine to suggest events to users using personalized interests. Unlike traditional methods that require filtering by keyword or predefined categories, this recommendation engine uses natural language processing capabilities to identify the meaning of event data, user interests and to determine semantically relevant events.

The recommendation engine is exposed through a secure API endpoint that is available only to authenticated users of the platform. When an authenticated user makes a GET request to the recommendation engine endpoint, the server fetches user profile information (userId), user tags (interest tags), and public events (for the current users tagged interests) that are saved in the database. If the user profile or user interest tags are not found then the server will halt and return a validation error, ensuring that recommendations are not generated unless there is appropriate and complete user data.

When the data is collected, the backend builds a structured request for GPT-4. Instead of simple text going into the model, we are now controlling the model through a functionality called function calling. The function calling means we can define an exact schema in JSON format for what we are going to output and that GPT is bound to the schema. In this case, the model is virtually "told" to call a function (returnFilteredEvents) that tells the model to return the events that are consistent with the interests of the user. The schema we define describes the properties we want to require in the event output, such as title, date, place, description, and other engagement-based metrics such as number of attendees and comments, while still giving GPT room for flexibility in not calling out every element exactly.

The prompt sent to GPT also includes a system-level instruction where we framed GPT (as an assistant) to be helpful, then we have the user-level message that contains the list of the individual's interest tags and all of the candidates' events. GPT looks at this data semantically and returns only the events it thinks are a good match.

By using function calling to invoke GPT, Excursia gets both the semantic capability of recommendations produced by AI alongside the technical safety to take those recommendations and derive user-facing features from them directly.

### 4.2.3 Swipeable Event Cards

In the mission of bringing a more dynamic and casual way of exploring events on the recommendation page, Excursia incorporates a swipe-based card interface inspired by familiar mobile app patterns. This feature is implemented in the swipe card component and, creating for users a way to navigate recommended events by swiping left to dismiss or right to show interest.

The main logic of the component handles drag gesture detection, utilizing both mouse and touch input. This is crucial so that the component is compatible on any desktop or mobile device, and maintains a consistent experience. React's useState and useRef are used to track the interactive state of the card, including where you start a drag, how much you drag the card, and whether the component is currently animated or not to avoid overwriting gestures.

When a user initiates a drag or swipe gesture, the component saves the initial coordinates and derives the movement deltas while the drag (or swipe) is made. The card then moves horizontally, following the user's finger (or mouse), until the user decides to release their gesture. Once the gesture is complete, the component checks whether the swipe distance has passed a predetermined threshold (e.g., greater than 100 pixels). If it did, the card is considered to have been "swiped" in that direction and the appropriate direction is then sent to the onSwipe callback. If not, then the card is transitioned back to its original position using CSS while making for a pleasant, seamless experience for the user.

The card itself is visually interned in such a way as to accommodate all event information, and the item's title, description, location, date, and participant metrics were all used. The card's information is presented in a vertically-stacked fashion, and icons from the lucide-react library were used for immediate visual recognition (e.g., calendar, pin for location, a count for users, and like/dislike). The event image, when present, is retrieved using the utility function and rendered (as a background and/or header visual feature).

The swipeable interface is not simply a UI improvement but crucially a point of functional integration with the application's backend GPT recommendation engine. Events that are shown as swipeable cards are sourced and edited by the backend GPT-4 semantic filtering, with each card representing a candidate match, and the swipe direction the user chooses provides information to the system about user preferences. When the user swipes right, there is an implicit user interaction that triggers the backend to record the user interaction or recommend events of a similar nature.

The implementation for the SwipeCard uses a balance of expressiveness of interaction design and precision of function. It uses React's declarative component functionality, and controlled managed state, which contributes to optimizing add swipe logic for performance at the same time as creating a compelling social front-end experience that complements the intelligent backend recommendation service.

In summary, the SwipeCard component serves to represent a critical user interaction paradigm that is intuitive and enjoyable, but also extensible—creating opportunities for increasingly richer user feedback capabilities, future gesture-based navigation states, or even variable user interface behaviours based on user history.

## 4.3 Authentication and Authorization

The authentication system in Excursia is built around the passwordless login concept, combining one-time passwords (OPT) with JSON Web Tokens (JWT), offering a safe login access. This approach modernizes the user login flow, moving users away from the traditional passwords, which many times can be weak, reused or poorly managed.

Passwordless authentication offers several advantages from a security standpoint, over classical solutions. Users no longer have to create passwords that they forget, or reuse for convenience, or simply fall victim to phishing or data breaches. They now log in with a temporary code that is sent to them via a user email and in most cases, offers a better user interface experience for the mobile user or the user in a shared environment.

### 4.3.1 OTP Generation and Email Delivery

When a user tries to login by entering the email, the backend generates an OTP formed from six random digits. This code is stored temporarily in the database with a strict expiration window of five minutes, to prevent possible abuse. The code is then sent to the user's email address with the help of a Simple Mail Transfer Protocol (SMTP) based mailing service.

This ephemeral credential acts as a secure and disposable key, assuring that only someone with access to the email account can proceed with the authentication. Because the OTP is not reusable and has a short expiry time, it dulls many risks associated with credential theft.

### 4.3.2 OTP Verification and Token Issuance

After a user receives their OTP and submits it, the back-end checks the record and expiration time in the database to verify the number. If the record exists and is within the expiration window, the corresponding OTP entry is removed from the database so that a valid code cannot be used again.

Once the OTP is verified, the server will create a JWT token for the user. If the user already had an account registered, the long-lived token will be used for external session management. If the user had not registered, a short-lived token is generated so the user can create a profile. These tokens are secured by encoding the user's information and signing it with a secret key so it cannot be manipulated on the client side.

### 4.3.3 Route Protection with Middleware

Secure access to the application is enforced by protecting backend API routes using middleware that can check for valid JWT tokens. Each protected API endpoint verifies whether a token is present in the Authorization header. If a token is found, it is verified. Then the API identifies the user's ID, and attaches that user's ID to the request object, so that the requested API endpoint can identify the user. Authentication is stateless: no sessions are maintained on the server, making it easier to manage scaling issues.

### 4.3.4 Front-end Token Handling

Upon successful authentication, the frontend side stores the JWT token in the user's browser local storage. The function of managing access tokens across the app is accomplished by a custom hook that retrieves the token. When accessing protected frontend routes, a wrapper component ensures that only users with valid tokens can access certain pages. If there is no valid token, the user is redirected to the login screen.

Although storing the authentication state in local storage may allow for manipulation on the client side, the app remains secure because the backend strictly enforces token validation. In other words, unauthorized users will be blocked at the API access level, even though they may bypass checks client-side.

## 4.4 Database Integration

The backend uses SQLite as a lightweight, file-based relational database system. All data is stored locally in a single file, `database.sqlite`, and accessed through the `better-sqlite3` Node.js library. This library enables efficient synchronous queries and supports SQL features such as prepared statements, foreign keys, and schema enforcement.

### 4.4.1 Database Initialization

A single module initializes the entire database, taking care of the connection, and creating the missing tables at startup. This way, the application can run right away and does not need a separate migration step. The schema is written with raw SQL in a file and executed at launch with synchronous methods to keep the initialization deterministic, fast and maintainable.

It checks to see if tables already exist, so the app can bootstrap itself but still have user data available instead of overwriting the previous version of the app. The synchronous .exec() and .prepare().run() methods provided by better-sqlite3 also have predictable outcomes at this stage.

### 4.4.2 Schema Overview

The database schema (Figure 4.1) supports all of the core features of Excursia, ranging from user authentication to event management to social features. The users table is at the center of it all, containing fundamental profile details such as name, email, role, and profile images. Additionally, ”user tags” and ”user images” will save profile information to facilitate profile personalization.

There are many tables related to event information to make it easier to manage for different features to enable flexibility. The events table stores the general information about each event (title, location, date, audience scope, etc.). The event attendees table tracks the people who will come to events, while the ”event interested” table records expressions of interest (only unique for each user) to events, and other tables track applications and attendance data. Users can comment on events for public discussion points, these are all saved in the ”event comments” table.

The ”otps” table supports brief, single-use codes for passwordless logins, while the ”chats” and ”messages” tables support real-time, one-to-one dialogs between users. Social relationships are maintained through the friendships table, which represents confirmed relationships and pending requests.

This relational schema is enforced via foreign key constraints that assure correctness of the data - for example, if a message comes from a chat, it must belong to that message’s chat; if a user is attending, that user must be a valid user; and if a user is attending, that event must refer to a valid event.

### 4.4.3 Data access and consistency

Database access is accomplished using prepared SQL statements, which provide the greatest performance and protection against injection attacks. The queries are compiled once, transforming them into executable code and multiple invocations only modify the parameters, so they can minimize editing and parsing overhead and clearly separate code and data. Foreign key constraints are enabled at the connection level in order to maintain referential integrity across a set of related tables—for example, ensuring that every record in an attendees table always has a matching user and event.

To illustrate these concepts in practice, let’s examine a couple of specific operations:

When retrieving a user’s friends, the application has to consider that friendship relationships are stored in a single friendship table (Figure 4.1), with columns for the two user IDs and a status flag indicating each relationship’s accepted status. The friendships table has a single row for a friendship, a single record with two user

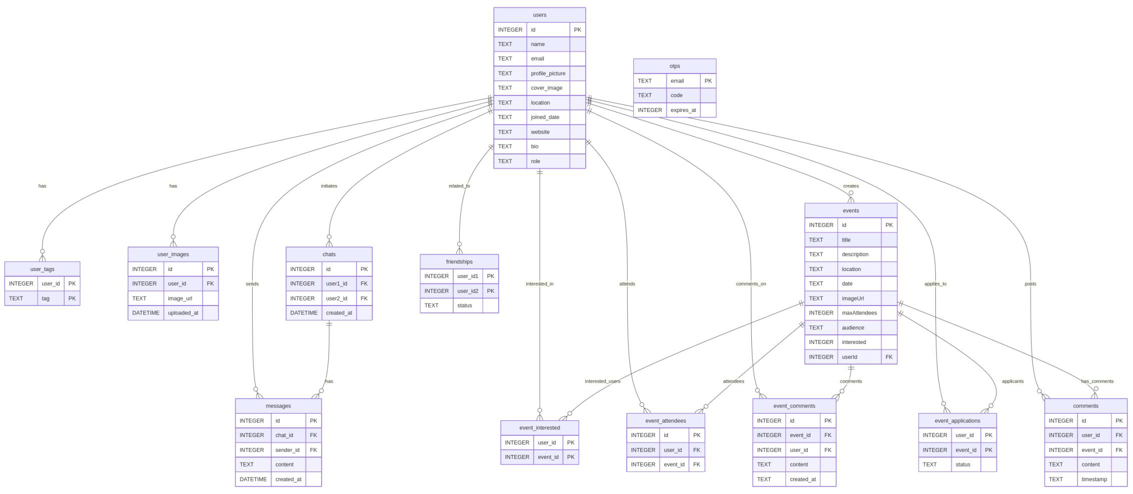


Figure 4.1: Entity Diagram for Database

IDs and a status. Instead of creating two separate queries, a single parameterized query uses a conditional expression to determine who the "other" user is: If the current user appears in the first column, the second column refers to the friend ID, and vice-versa. This method greatly simplifies the code, and only retrieves "accepted" friends in one database call.

When creating a new event record, safety and expediency are similarly achieved. A prepared insert statement lists the event parameters and each parameter has a placeholder for the title, description, date, image URL, audience public/private scope, organizer ID, location, and optionally the maximum number of participants. Before executing the prepared statement, optional fields are only bound when they are present to add to the insert statement, otherwise, a NULL is bound when absent. Once the insert statement executes successfully, the database engine returns the auto-incremented event ID, which is sent back to the client to indicate the event has been created.

Finally, before initializing a real-time chat between two users, the system check the existence of a conversation. This check is done by a prepared select query with two parameters for the communicating parties. In some implementations, a mirrored condition is added to assure that it does not matter in what order a user ID comes. After an existing chat is found, the identifier is returned by the backend, otherwise is going to create a new chat entry, taking care of possible problems like duplication and preserving a correspondence between chat sessions.

Through these examples, the design demonstrates how prepared statements, conditional SQL logic, and referential constraints combine to deliver reliable, performant, and secure data access across all features of Excursia.

### 4.4.4 Design Rationale

SQLite was chosen for its ease of use, zero-configuration setup, and reliability for small to medium-scale applications. This design is ideal for rapid development and prototyping, and can be migrated to more scalable solutions like PostgreSQL or MySQL with minimal changes to SQL syntax.

## 4.5 API Testing with Postman

To confirm that each backend endpoint does what it is meant to do, Postman collections are grouped in folders according to features: Auth, Events, Chats, Friends, Profile, and Notifications. Each folder contains requests which implement the primary functionality of that feature; for example, in Auth there is OTP generation, OTP verification, and token output, and in Events, there is a list of all events, get single event by ID, create events, update events.

The requests use a common set of environment variables, including base URL and a reusable JWT bearer token. This token is inserted into the Authorization header automatically with Postman's environment so that protected routes are called under valid authentication. For endpoints requiring path parameters, the environment variable is substituted into the URL at run time.

Although the requests are executed manually in from the Postman UI while developing the application, with the help of Newman CLI they can also be scripted, a single command running the entire suite.

This method guarantees thorough testing of every backend function with standardized environment variables and assertions using these feature folders. In addition, it acts as living documentation permitting frontend developers and QA engineers to understand the API contract and check behavior without much overhead.

# Chapter 5

# User Manual

This chapter aims to provide a guide accompanied with screenshots in order to get the most out of the event discovery application. Besides the login page layout components like the navigation bar, friend list and side-bar are omnipresent.

## 5.1 Login and Register

Before enjoying the Excursia social application, you need to create or log into an account. There are no buttons for creating one, because the account will automatically be created once you enter the OTP sent to the email. (Figure 5.1)

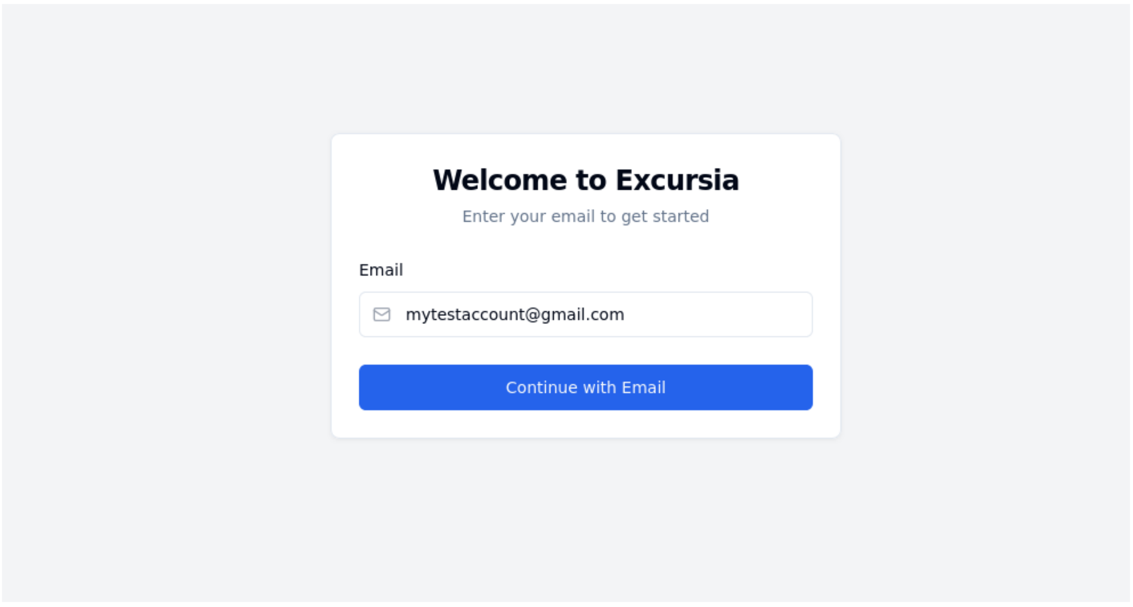
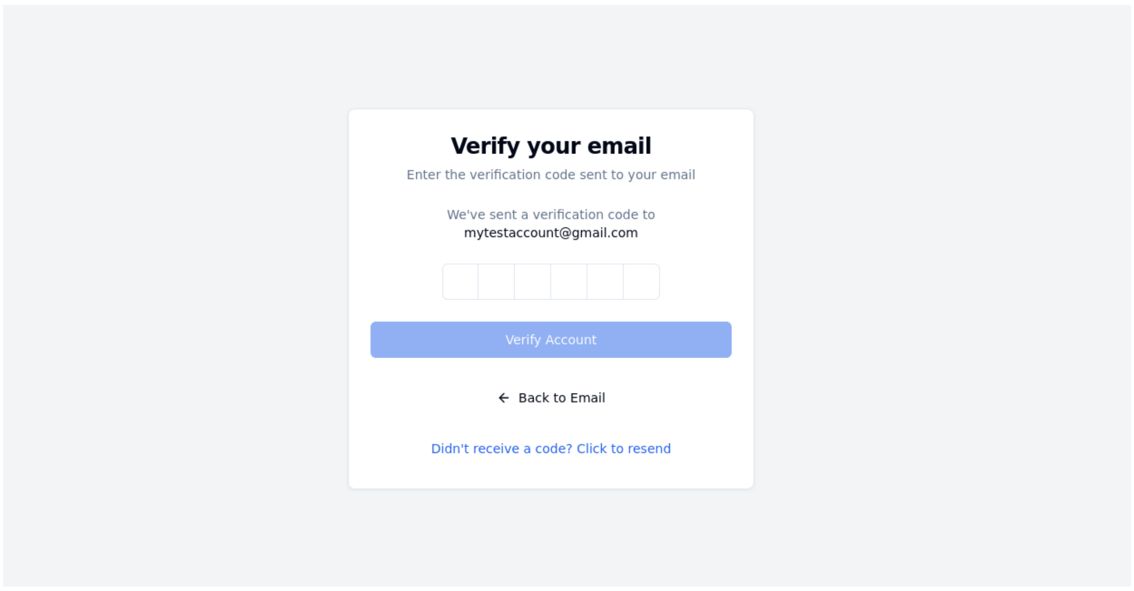


Figure 5.1: Registration and Login Screen

## 5.2 OTP Verification

After an email was provided, the user is redirected on this page. Here he is asked to enter the password received on the email. If he doesn't receive one, he can click the resend button or just return back on the login page. (Figure 5.2)


Verify your email

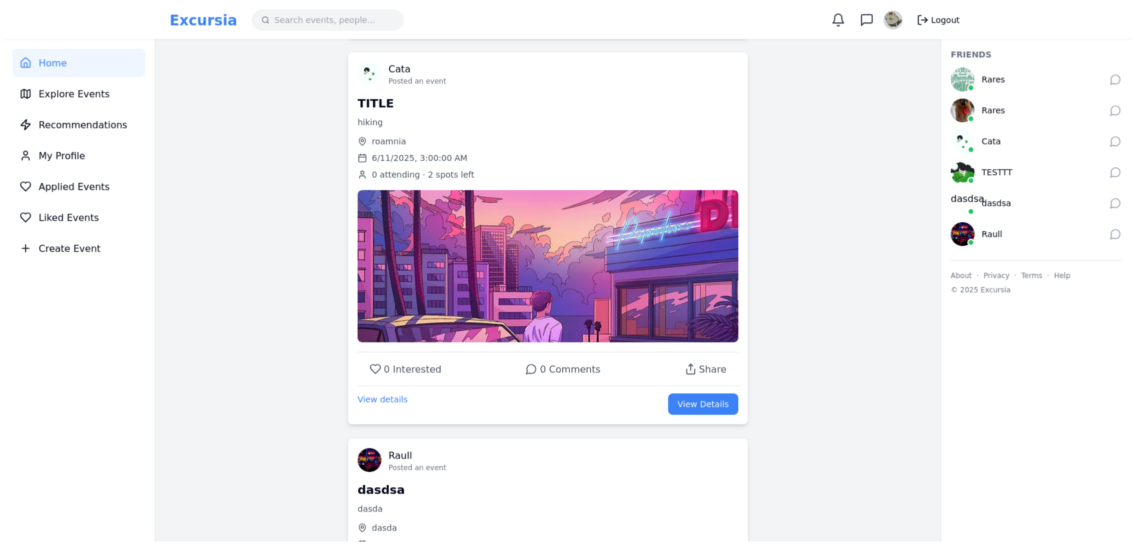
Enter the verification code sent to your email

We've sent a verification code to
mytestaccount@gmail.com

Figure 5.2: OTP Verification Page

## 5.3 Home Page

When you first log in to Excursia, you will arrive on the Home Page. This is your hub from where you can discover events, choose fast chats and navigate the app. (Figure 5.3)



Cata
Posted an event
TITLE
hiking
roamnia
6/11/2025, 3:00:00 AM
0 attending · 2 spots left

Raull
Posted an event
dasdsa
dasda
dasda

Figure 5.3: Home Page

### 5.3.1 Navigation bar

At the top of the page you will have access to global actions such as searching, viewing notifications, or accessing your account profile. The navigation bar can be found anywhere in the application itself; therefore, you are never too far away from where you can quickly jump to another area of the application using a click or tap.(Figure 5.3)

### 5.3.2 Event Card

These cards are found in the feed and are used to display compressed information about the event, such as the user name, event title, date, location, number of attendees and an appealing photo. It also has three buttons, one to show his interest in the event, one to share the event and another to apply to it. By clicking on the event card, you are redirected to the event page. (Figure 5.3)

### 5.3.3 Sidebar and Friend list

They can be found only on the desktop. On the left side there are listed links to the major sections of the app such as Home, My Events and Notifications, so you can quickly navigate to different areas without returning to the top navigation. On the right side are shown which of your friends and you can click on a friend's name to start a chat. (Figure 5.3)

### 5.3.4 Explore page

The explore page works in the same manner as the home page, it just shows events that are also public. (Figure 5.3)

## 5.4 Create Event

Just like when you decide to create a new event, an initial page will pop up for you with the heading "Create a New Event" centered on the document. The top of the panel will contain the heading so you can orient yourself within the app.

In the first box, you will enter the title of your event. By clicking into the text box, you enter a name for your event that easy to remember, and then move on to the next box. Similar to the text area in your profile, the text area is an invitation for you to describe what participants can expect to see at your event. The next step is mandatory and the user must select the privacy option for his Event. There are two radio buttons to choose from: "Public" (anyone can see and join your event) or "Friends Only" (your event will only be visible to your connections).

Below there are two columns on wider screen (and a single column on mobile). On the left side you will enter in the location by typing it. On the right side you will select the date and time, each labeled with either a calendar or clock symbol to help remember what goes where. If you have a cap on how many people can attend, there's a field for maximum attenders.

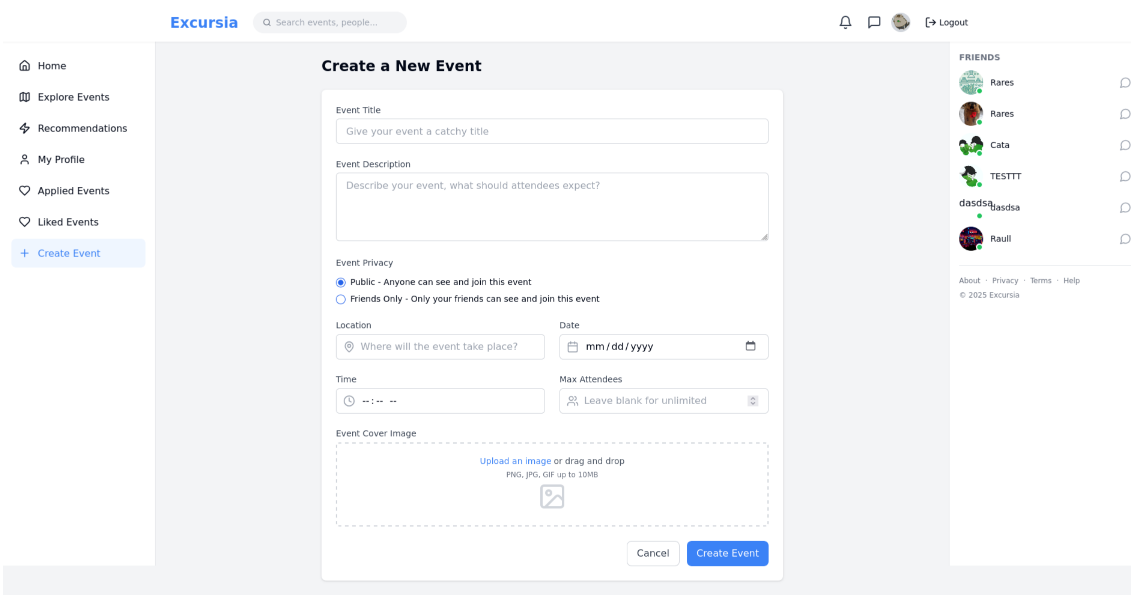
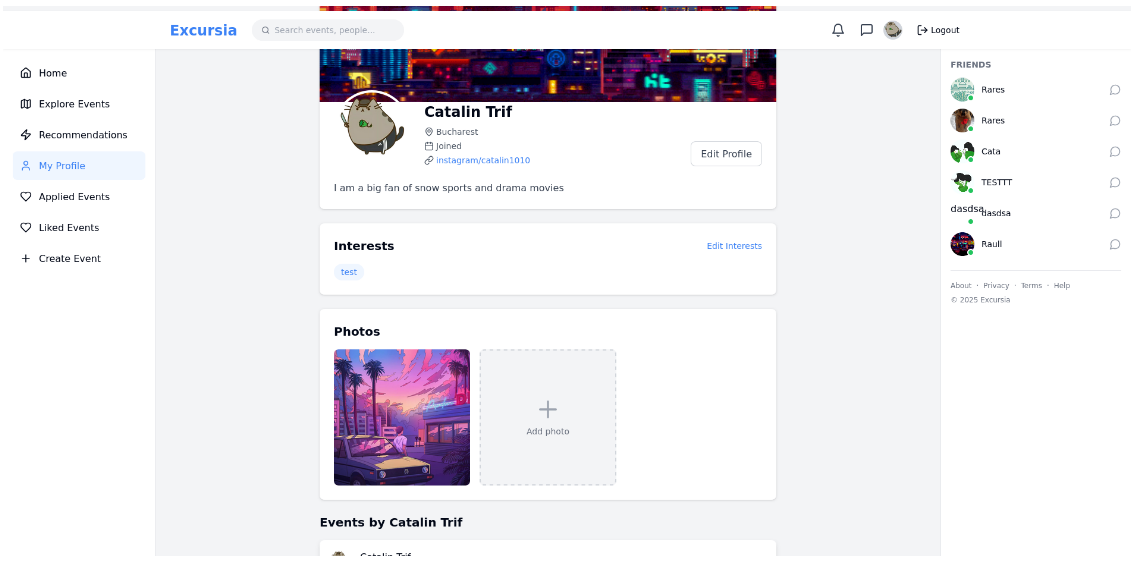


Figure 5.4: Create Event page

Lastly, you can upload a cover picture. The dashed-border dropzone gives you the option to click to select a file, or simply drag and drop a file in. Once you select a file, you will see an updated preview of the file, however you can remove it any time simply by clicking it again. At the bottom of the form are two buttons: ‘Cancel’ will allow you to go to the previous screen without saving it, and ‘Create Event’ will submit your event details to the server. If you leave a required field blank or incorrectly formatted it will provide an inline prompt reminding you to correct anything before the event goes. (Figure 5.4)

## 5.5 User Profile

Upon navigating to a user’s profile, at the very top sits the profile header component, showcasing the user’s name, avatar, and cover image. If you’re viewing your own profile, a button to edit the profile appears here, allowing you to update your personal details or change your avatar, otherwise there are two buttons, one to add a him as a friend and the other to message him. (Figure 5.5)

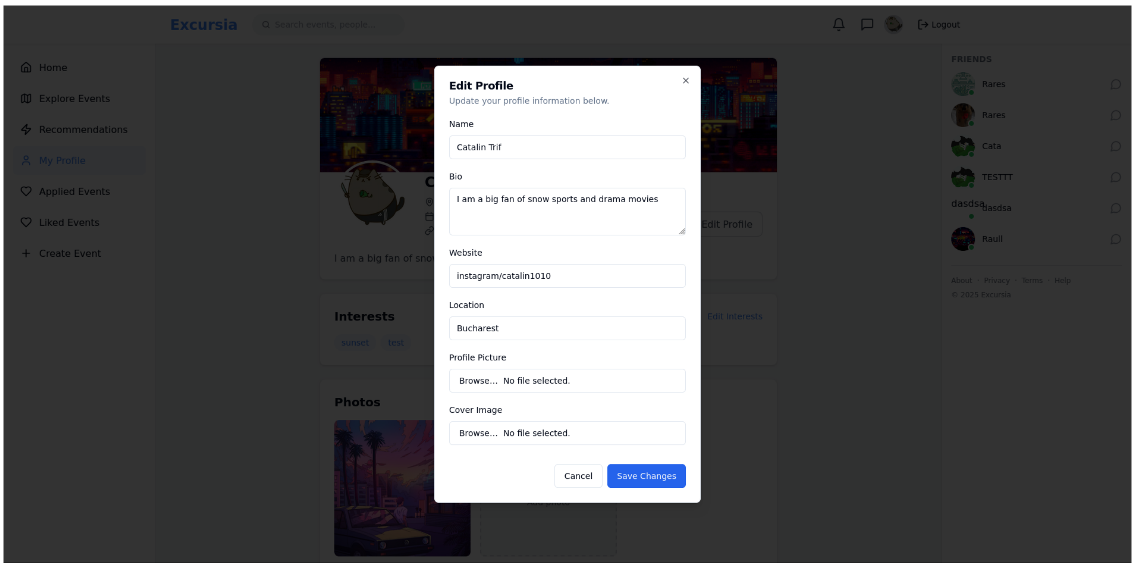


Catalin Trif
Bucharest
Joined
instagram/catalin1010

I am a big fan of snow sports and drama movies

Figure 5.5: User Profile

Below the header, a panel labeled Interests displays all of the tags that characterize this user. If it's your profile, an Edit Interests link sits to the right of the title. Tapping that link opens a dialog where you can add or remove tags, instantly refreshing the list once you save. Continuing, we have the picture section where a user can add up to six photos by tapping the upload box. If the user wants to remove a photo, a red button appears in the corner of each picture for easy deletion. Finally, the user can have quick access to the events below his photos. (Figure 5.6)

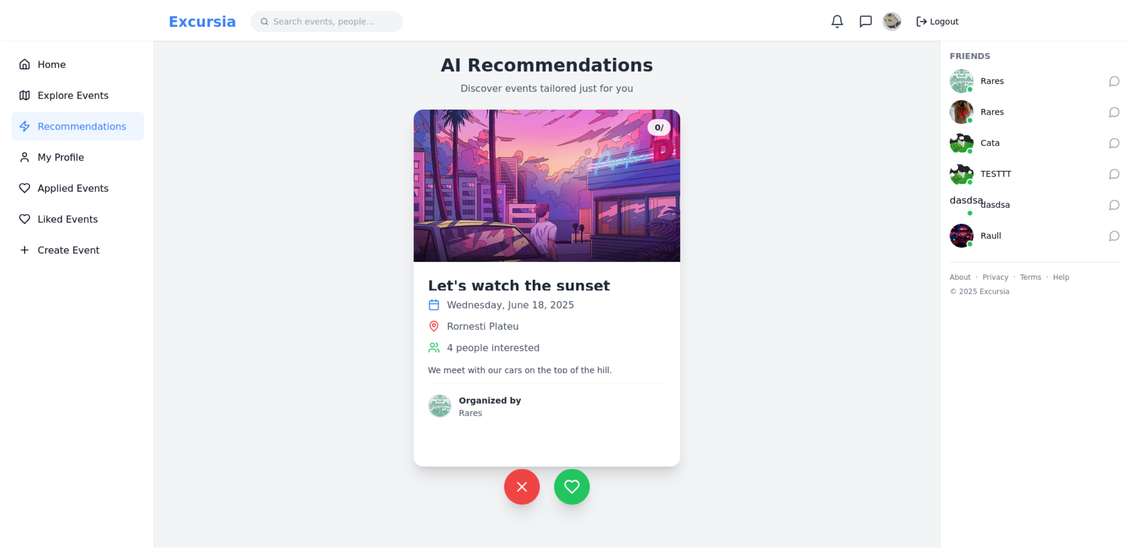


Edit Profile
Update your profile information below.

Name: Catalin Trif
Bio: I am a big fan of snow sports and drama movies
Website: instagram/catalin1010
Location: Bucharest
Profile Picture: Browse... No file selected.
Cover Image: Browse... No file selected.

Figure 5.6: User Edit Profile

## 5.6 Recommendation Page



AI Recommendations

Discover events tailored just for you

Let's watch the sunset

Wednesday, June 18, 2025

Romesti Plateu

4 people interested

We meet with our cars on the top of the hill.

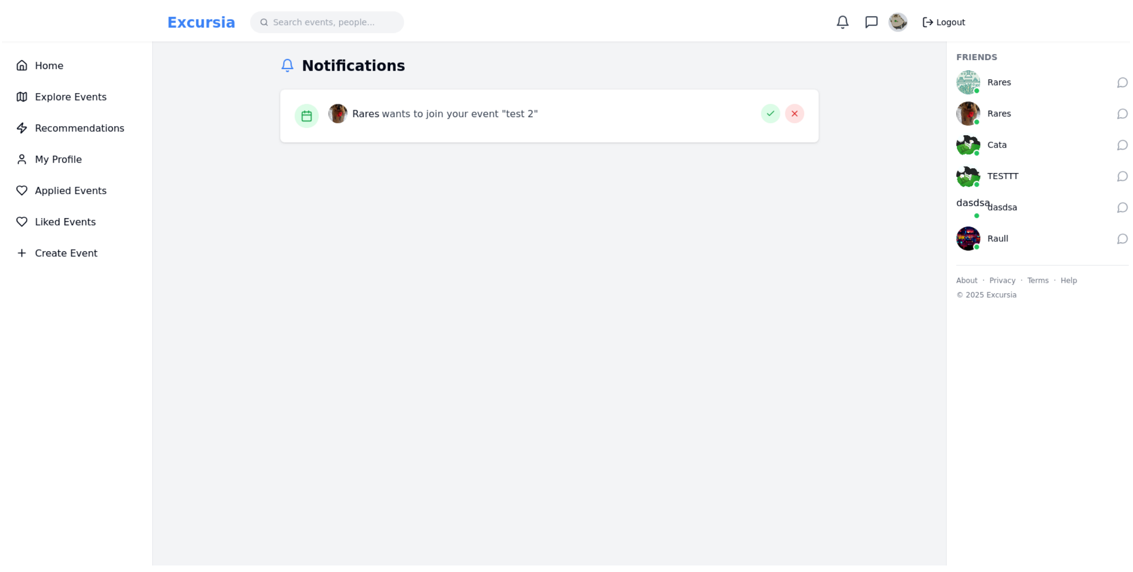
Organized by
Rares

Figure 5.7: Recommendation page

On this page we have a modern swiping card, that shows very brief details about the event such as name of the event, date, location, number of attendees and organizer. To interact, you simply click and drag (or touch and swipe) the card side to side. If you drag the card to the right past a certain threshold, you will be marked as Interested and if you swipe to the left, you will Pass. As you drag the card out of view, it tilts, fades, and, smoothly slides off the screen. If you did not drag the card to the threshold, it logs back to center with an easing animation. (Figure 5.7)

## 5.7 Notification page

This page is made to show the user different alerts. When there are no items, a centered message reassures you are all caught up. As soon as a notification arrives, it appears as its own card. Each notification is customized, since there are two types: one for friends and one for events. In the right part of the card, there are two buttons that can either accept or reject a request. (Figure 5.8)

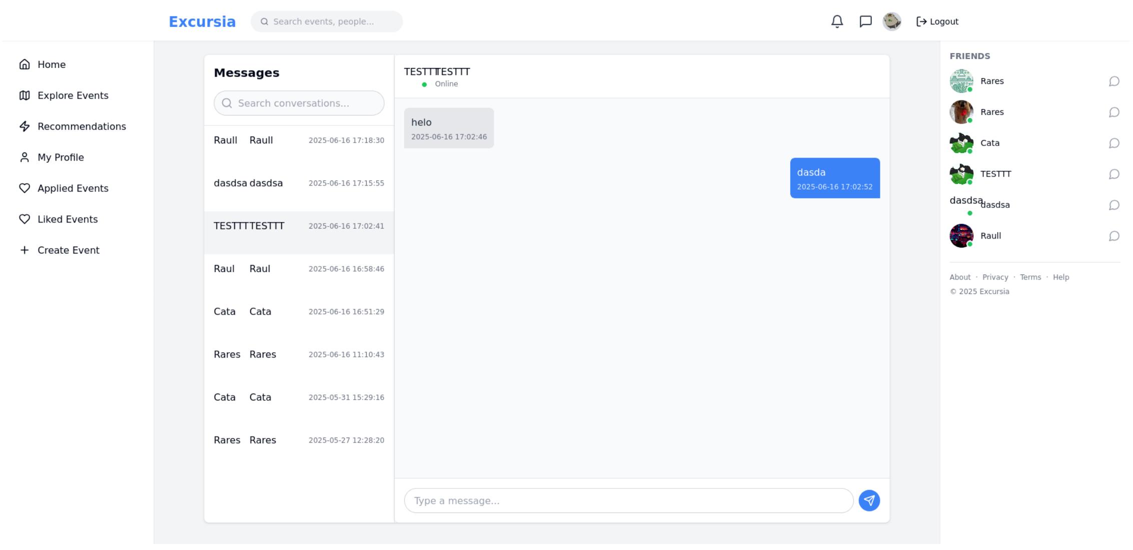


Rares wants to join your event "test 2"

Figure 5.8: Notification Page

## 5.8 Chats page

On the Messages page you'll see a split-pane layout combining your conversations list with the active chat window. The left pane is dedicated to Contacts: here you can search your existing chats by typing a name or keyword into the search field at the top. Each contact entry displays the friend's avatar (if there is one), their name, the timestamp of the most recent message, and a preview of that last message (or "No messages yet" if you haven't chatted before). Clicking on any contact highlights it and opens the chat on the right.



| | |
|---|---|
| Raull Raull | 2025-06-16 17:18:30 |
| dasdsa dasdsa | 2025-06-16 17:15:55 |
| TESTTTTESTTT | 2025-06-16 17:02:41 |
| Raul Raul | 2025-06-16 16:58:46 |
| Cata Cata | 2025-06-16 16:51:29 |
| Rares Rares | 2025-06-16 11:10:43 |
| Cata Cata | 2025-05-31 15:29:16 |
| Rares Rares | 2025-05-27 12:28:20 |

TESTTTESTTT

helo
2025-06-16 17:02:46

dasda
2025-06-16 17:02:52

Figure 5.9: Chats Page

The chat pane occupies the remainder of the view. Below, the message history scrolls automatically, with your friend's messages aligned to the left and your own responses to the right. A text input along the bottom lets you type and send new messages instantly, appearing in real time without refreshing the page. Worth mentioning is that on the mobile, the chat view modifies, first showing only your contacts, and after pressing a user takes you in the chat. (Figure 5.9)

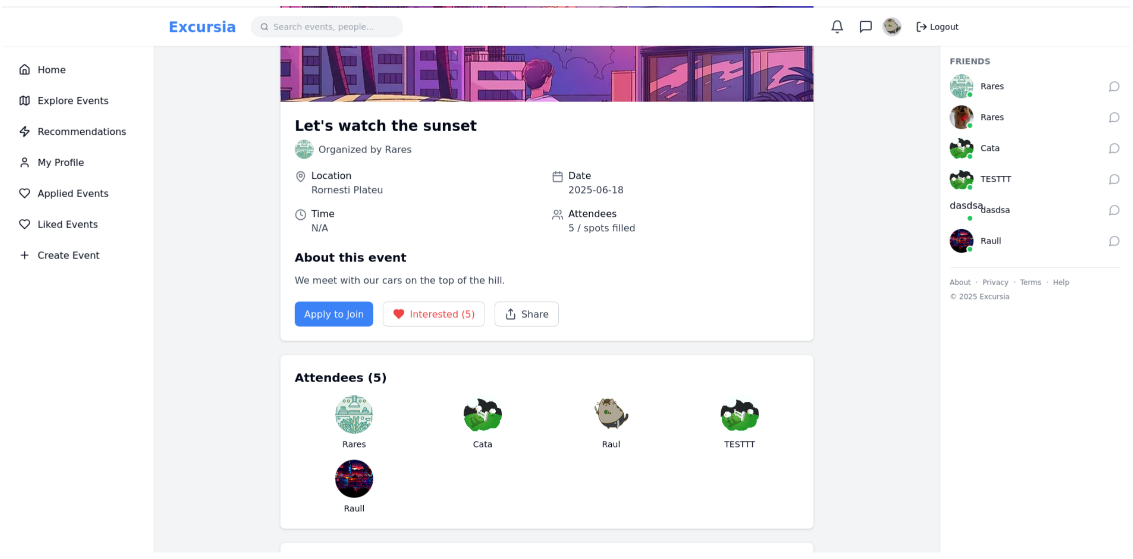
## 5.9 Event page

When you view the detail page of an event (Figure 5.10), you'll see the cover image at the top, followed by the title of the event and the name of the organizer (you can click the name of the organizer to see their profile). Following that, a simple grid provides a location, date, time (or "N/A"), and count of attendees, all made clear by icons.

In the "About this event" section, you'll find the full description of the event. Below are three buttons: apply to join or cancel your application, toggle your interest with a counter, and share the event.

Next, you will see up to eight attendee avatars, presented in a compact grid. If there are more than eight, a "Show all attendees" link will expand the grid to show a full list of attendees. Anything with an avatar, such as accounts in this case, has a link to the user's profile.

At the bottom, the "Discussion" area displays comments, if any have been posted, and a text box for you to add your own. Comments display the avatar of the commenter, their name, their message, and small action links marked relevant for liking or replying. This layout allows you to see everything from event details to event chats in a single scrollable area.



Let's watch the sunset

Organized by Rares

Location
Romesti Plateu

Date
2025-06-18

Time
N/A

Attendees
5 / spots filled

About this event

We meet with our cars on the top of the hill.

Figure 5.10: Event Page

# Chapter 6

# Conclusion and Future Directions

The following chapter aims to provide a future direction for Excursia, including new planned features and also to provide the reader with a concise conclusion summing up the contents of the thesis.

## 6.1 Future Work

Looking forward, there are several improvements that can be done to the application. On the communication side, groups can be created after an event is published and automatically adding people accepted by the organizer and a calendar feature that shows you all the planned events in a visual appealing way. To broaden accessibility and simplify login, OAuth integration for Google and Facebook accounts will be implemented, allowing users to authenticate via their existing social credentials. This addition will coexist with the existing passwordless flow.

For the recommendations, perhaps exploring hybrid model architectures might lessen reliance on calls to external GPT-4, and offer greater speed with less latency, and facilitate the main inference being done locally, in terms of costs associated with prompt injections. Lastly, operational scalability can be addressed by migrating from SQLite to a distributed database such as PostgreSQL.

## 6.2 Closing Remarks

As Excursia continues to evolve, it aims to redefine how people discover, plan, and participate in real world events. By swiping through events based on intelligent recommendations or scrolling the feed, the platform brings a new view for event planning tools and more time enjoying shared experiences. With ongoing innovation and user-centered enhancements, Excursia is setting new standards for convenience, personalization, and social engagement in the event-discovery space.

# Bibliography

[FM14] Asger Feldthaus and Anders Møller. Checking correctness of TypeScript interfaces for javascript libraries. In Andrew P. Black and Todd D. Millstein, editors, *Proceedings of the 2014 ACM International Conference on Object-Oriented Programming Systems Languages & Applications (OOPSLA 2014)*, pages 1–16. ACM, 2014.

[LdGS11] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul Kantor, editors, *Recommender Systems Handbook*, pages 73–105. Springer, 2011.

[LPW14] Chen Luo, Wei Pang, and Zhe Wang. Hete-CF: Social-based collaborative filtering recommendation using heterogeneous relations. In *Proceedings of the IEEE International Conference on Data Mining (ICDM '14)*, 2014.

[MDN25] MDN Web Docs. Cross-origin resource sharing (cors), 2025.

[Met25] Meta Open Source. React — the library for web and native user interfaces, 2025.

[Nod25] Node.js Documentation. Overview of blocking vs non-blocking, 2025.

[Ope25] OpenAI. Openai api reference, 2025.

[RIS+94] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW '94)*, pages 175–186. ACM, 1994.

[RRSK11] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul Kantor, editors. *Recommender Systems Handbook*. Springer, 2011.

[sha25] shadcn. Introduction — shadcn/ui, 2025.

[Soc25] Socket.IO Contributors. Introduction — socket.io 4.x documentation, 2025.

[Wis25] Joshua Wise. better-sqlite3 • the fastest and simplest library for sqlite in node.js, 2025.

[WZQ+24] Likang Wu, Zhi Zheng, Zhaopeng Qiu, Hao Wang, Hongchao Gu, Tingjia Shen, Chuan Qin, Chen Zhu, Hengshu Zhu, Qi Liu, Hui Xiong, and Enhong Chen. A survey on large language models for recommendation. *arXiv preprint arXiv:2305.19860*, 2024.

[YC25] Evan You and Vite Contributors. Getting started — vite documentation, 2025.