Introduction

Roamly is a mobile application born out of personal experience. The idea came to life during my Erasmus exchange period, a time in which I had the opportunity to meet dozens of international students, backpackers, and fellow solo travelers many of whom shared the same excitement, curiosity, and challenges of living temporarily in a foreign country. What united us was not only the desire to explore new cultures, but also a recurring struggle: the difficulty of creating spontaneous, meaningful social connections while on the move. As I continued to travel staying in hostels, co-living spaces, and community-driven accommodations I met a growing number of international students and **digital nomads**: people working remotely while exploring the world. From software developers and graphic designers to content creators and online entrepreneurs, each of them expressed a common need for **authentic local interactions** and a way to quickly connect with like-minded people without relying solely on long-term relationships or dating apps.

Despite the diversity of our backgrounds, everyone I met seemed to crave the same thing: a simple and effective way to meet people nearby who shared similar interests, spoke the same languages, or were just open to hanging out whether for a coffee, a walk around the city, or a spontaneous night out. Most existing apps were either too focused on dating, too impersonal, or too slow to meet the needs of short-term travelers who needed **real-time discovery** and **low-barrier socialization**. This is where **Roamly** comes in.

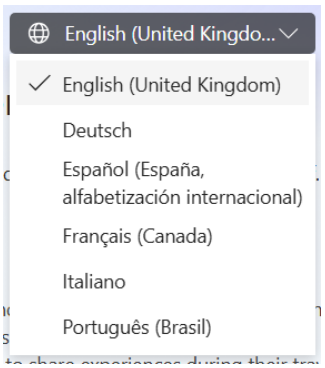
Roamly is designed to **bridge the gap between social spontaneity and location-based discovery**, offering a tool for travelers, students, and nomads to find and join local activities, meet people instantly, and build shared memories even in unfamiliar places. The app allows users to create or browse nearby meetups ranging from casual aperitifs and language exchanges to hikes and live music nights based on location, shared interests, vibe, and languages spoken. Everything is designed to be quick, intuitive, and social-first: no complex setup, no forced interactions, just genuine human connections in real time.

Roamly is the app I wish I had during my travels: something that goes beyond static forums or Facebook groups and offers a **dynamic, map-based social layer** to the places we explore. By making it easier to meet, connect, and share experiences on the go, Roamly transforms solo travel into a richer, more social, and more memorable adventure.

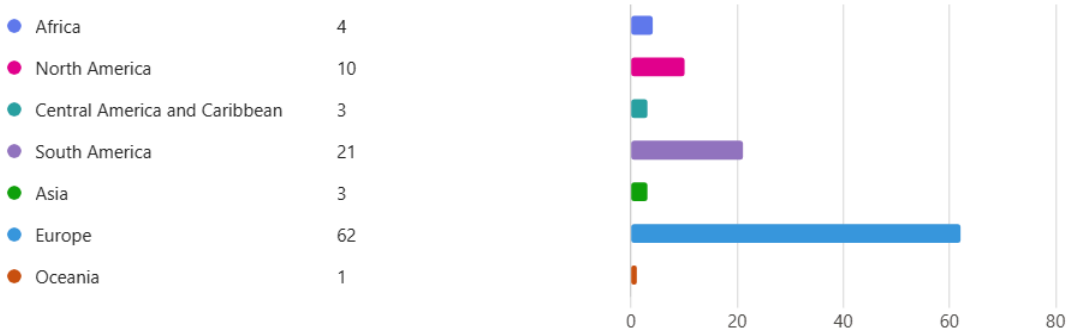
User Research

Demographics

Before starting the design phase of Roamly, I carried out a structured user research process to better understand the real needs, expectations, and behaviors of potential users. Although the app concept was inspired by my own experiences during an Erasmus exchange and frequent hostel stays, I wanted to ensure that development would be data-driven and informed by a diverse set of perspectives. To that end, I created a **32-question survey**, carefully designed to explore travel habits, social preferences, attitudes toward spontaneous meetups, language usage, and platform expectations. In order to reach a broad and multicultural audience, I **translated the questionnaire into multiple languages**, including English, Italian, Spanish, German, French, and Portuguese. This allowed me to collect feedback not only from people across Europe, but also from travelers and nomads originating from **South and North America, Africa, Asia and Oceania**.



1. Which geographical region best describes your origin?

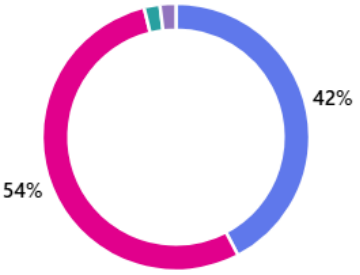


The distribution of the questionnaire was mostly organic: I shared it with people I met throughout my journeys, in hostels, coworking spaces, universities, and local gatherings. As a result, I received **more than 100 valid responses** from individuals of various age groups, backgrounds, and nationalities. The vast majority were between **18 and 34 years old**, a demographic that aligns well with the app's

intended user base. Many respondents identified as **international students and independent travelers**, reinforcing the idea that Roamly could serve a wide spectrum of mobile and socially curious users.

2. What is your gender?

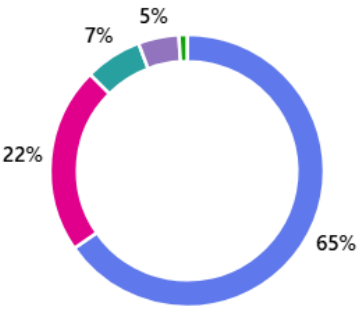
Man	44
Woman	56
Prefer not to say	2
Altro	2



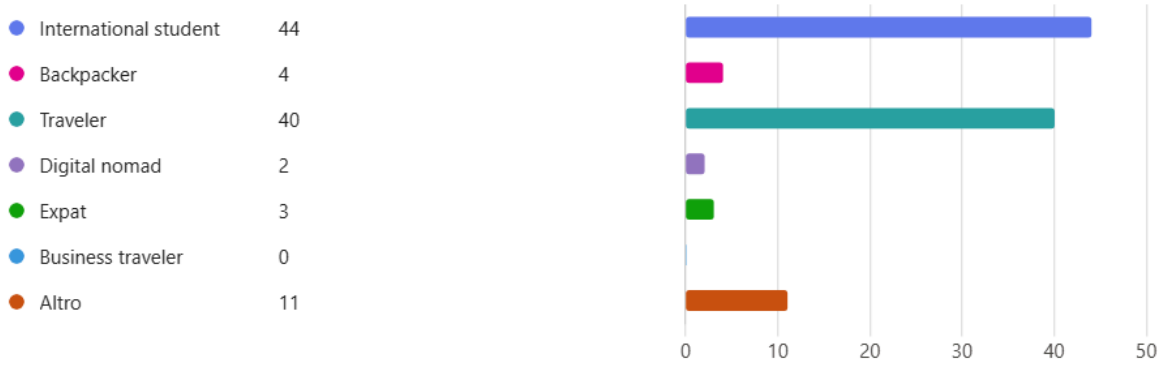
56	anonymous	18-24	English (United Kingdom)
57	anonymous	25-34	Español (España, alfabetización internacional)
58	anonymous	18-24	English (United Kingdom)
59	anonymous	18-24	Italiano
60	anonymous	18-24	Deutsch
61	anonymous	18-24	English (United Kingdom)
62	anonymous	18-24	English (United Kingdom)
63	anonymous	18-24	Português (Brasil)
64	anonymous	18-24	Italiano
65	anonymous	18-24	Português (Brasil)
66	anonymous	25-34	English (United Kingdom)
67	anonymous	18-24	English (United Kingdom)

3. What is your age range?

18-24	68
25-34	23
35-44	7
45-54	5
55+	1



4. Do you identify with any of the following categories?



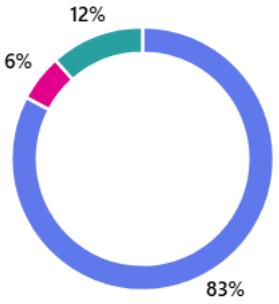
One of the most significant insights was that **over 80% of participants** reported having wanted, at least once, to meet someone to share an experience with while traveling. This highlighted a strong demand for lightweight, real-time social interaction tools that go beyond traditional dating or professional networking platforms. Respondents commonly expressed frustrations with the inefficiency of Facebook groups, the limitations of dating apps for platonic connections, and the overall lack of intuitive platforms to meet people nearby with shared interests.

8. Have you ever traveled alone?

9. Have you ever traveled with someone to a place where you knew nobody?

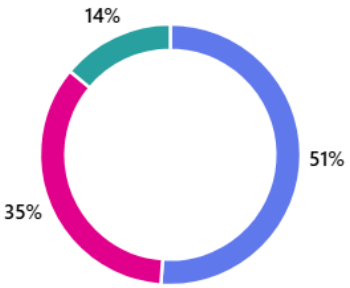
10. Have you ever wanted to find people with whom to share an experience during a trip?

Yes	86
No	6
Maybe	12



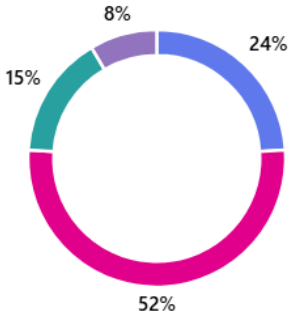
11. Did you manage to find someone to share the experience with?

Yes	44
No	30
Maybe	12



12. How did you find them?

Social media	17
In-person meeting	37
Local associations / groups	11
Altro	6

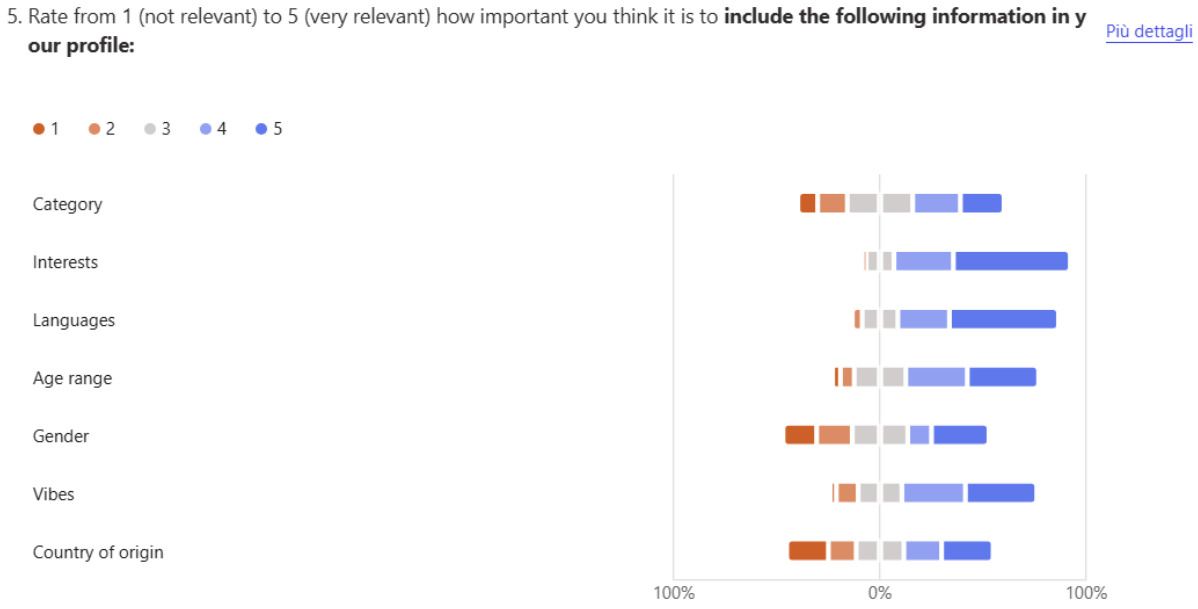


Feature Prioritization

Based on the survey data, I was able to clearly define a typical user profile and extract common behavioral patterns, motivations, and pain points. These findings were instrumental in guiding both the design of user profiles and the prioritization of features that would make the app relevant and intuitive.

One of the main areas investigated was the relevance of including specific user attributes in the profile. Respondents were asked to rate how important it would be to include each piece of information on a scale from 1 (not relevant) to 5 (very relevant). The most highly rated attributes were:

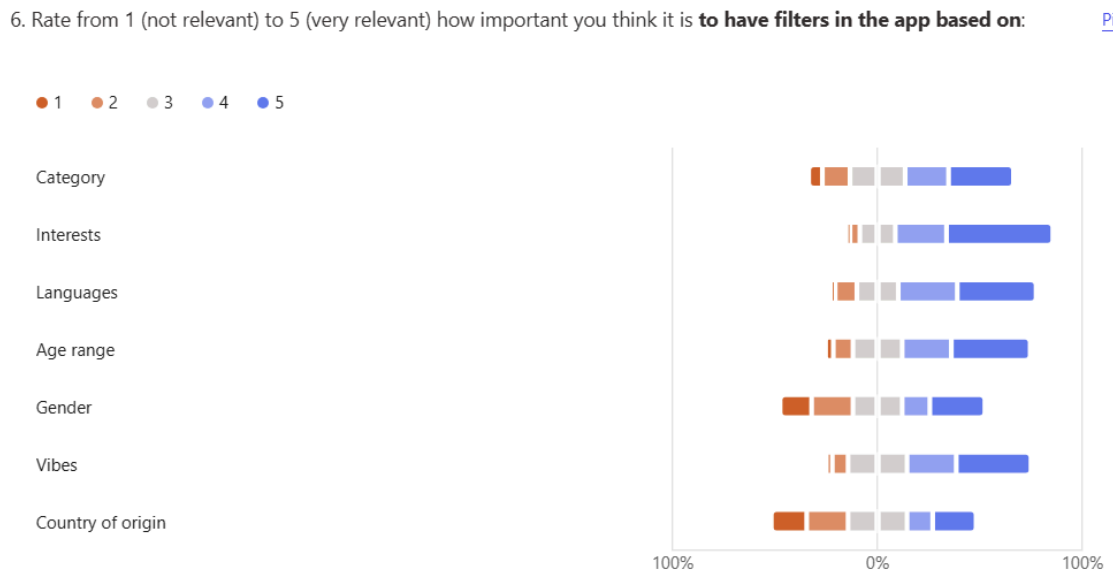
- Interests (avg. score: 4.41)
- Languages spoken (avg. score: 4.26)
- Age range (avg. score: 3.85)
- Vibes or mood status (avg. score: 3.84)



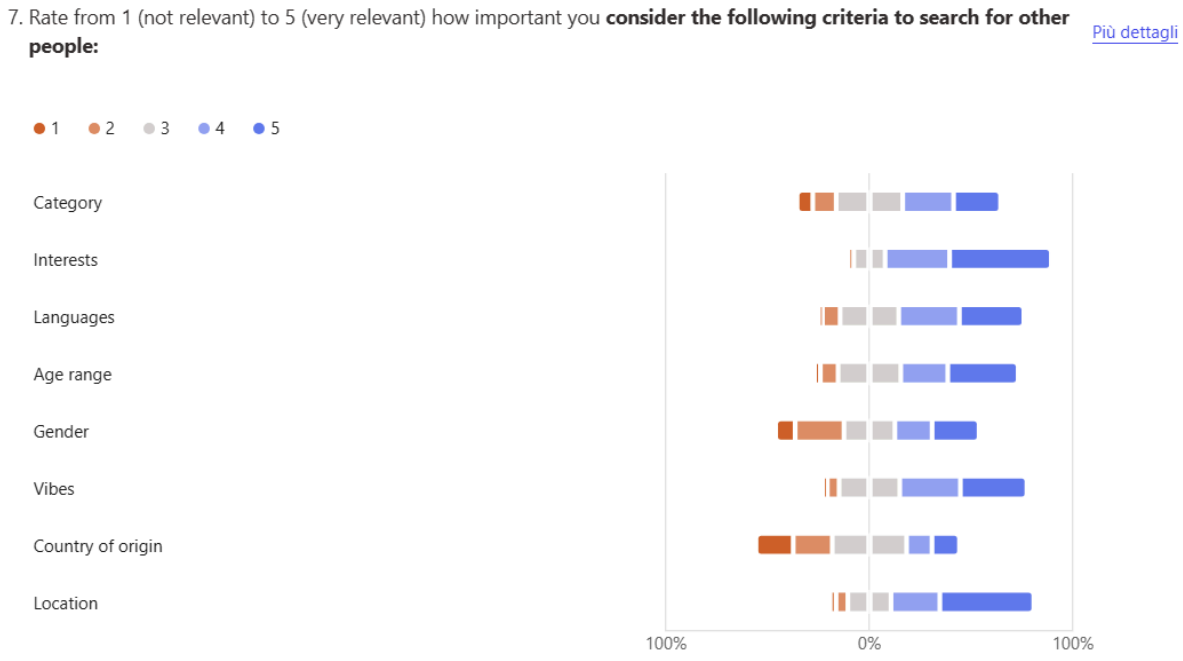
Other profile elements such as category (e.g., traveler, student, nomad), gender, and country of origin were perceived as moderately relevant (scores ranging from 3.14 to 3.32), suggesting that while useful, they should not dominate the onboarding or filtering process.

Users were also asked how important they considered various filters for navigating the app and discovering people or activities. The top-rated filters mirrored those preferred in profiles, with interests (4.22), languages (3.91), and age range (3.83)

again taking precedence. Mood-based filtering (vibes) was also strongly supported (3.83). These preferences validated the importance of offering users lightweight but meaningful discovery tools focused on shared traits and communication comfort.



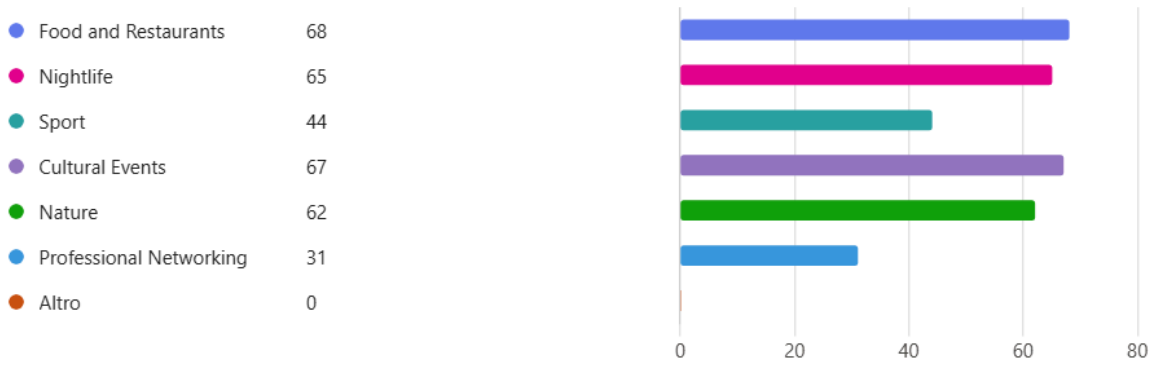
Less importance was attributed to filtering by gender (3.16) or country of origin (3.00), confirming that Roamly users are more interested in contextual compatibility than static demographics. This reinforced the idea of promoting inclusivity and avoiding segmentation based on fixed identity parameters.



Beyond user demographics and profile preferences, the survey explored participants' interest in the core functionalities that Roamly aimed to offer. This part of the research was essential to understand what users truly valued in a social app designed for short-term, real-time connections. Respondents showed strong enthusiasm for the **ability to see nearby activities created by other users**. **The idea of being able to browse a map or feed of spontaneous activities in one's immediate area resonated particularly well with travelers and students who often make plans last-minute.** In addition, there was clear interest in having filters that allow users to narrow down activities and people based on shared interests, spoken languages, and even current mood or "vibe" such as whether someone was feeling social, chill, or ready to party. These filters were seen not just as a convenience, but as a way to ensure more meaningful and comfortable interactions. Another widely appreciated feature was the **ability to chat directly with other users, either before joining an activity or after being matched through the app.** Respondents considered real-time messaging a vital element for building trust, clarifying plans, or simply getting to know each other before meeting up in person.

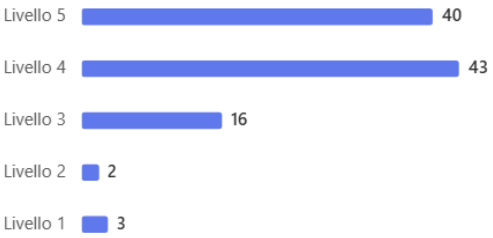
Overall, the results strongly supported the idea of a platform that enables spontaneous discovery of local plans, paired with smart filters and lightweight communication tools to facilitate real-world social connections. These features aligned perfectly with the lifestyle and preferences of the app's target users.

13. Which types of activities would you be interested in?



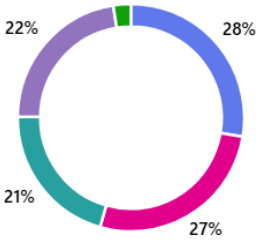
14. How relevant would a feature allowing you to propose activities be for you? Più dettagli

4.11
Valutazione media



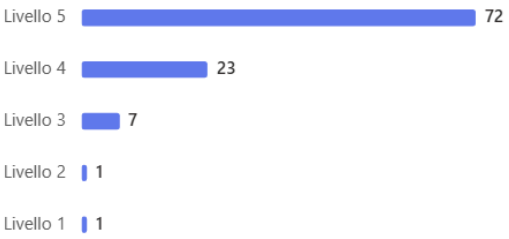
15. Which details should a proposed activity include? Più dettagli

Location	88
Related Interests	85
People Involved	66
Time	71
Altro	8



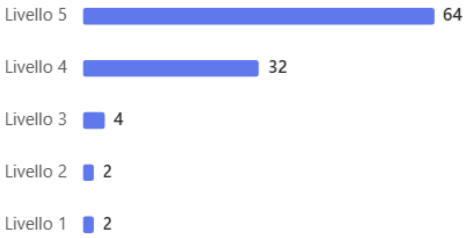
16. How relevant would a map showing the approximate locations of **nearby activities** be for you? Più dettagli

4.58
Valutazione media



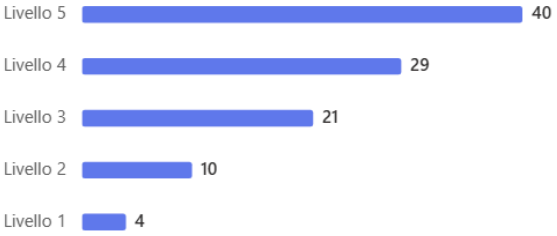
17. How relevant would a feature that allows you to **filter nearby activites** on the map be for you? Più dettagli

4.48
Valutazione media



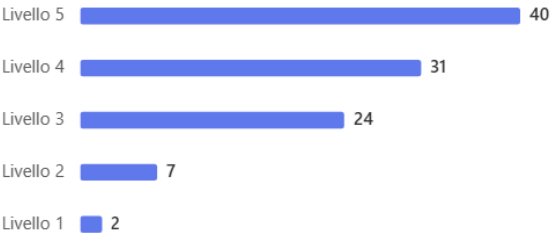
18. How relevant would a map showing the approximate locations of **nearby users** be for you? Più dettagli

3.88
Valutazione media



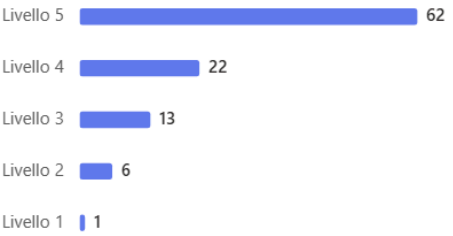
19. How relevant would a feature that allows you to **filter nearby users** on the map be for you? Più dettagli

3.96
Valutazione media



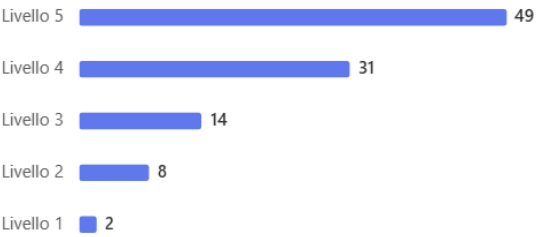
20. How important is it for you to send direct messages to other individual users? Più dettagli

4.33
Valutazione media



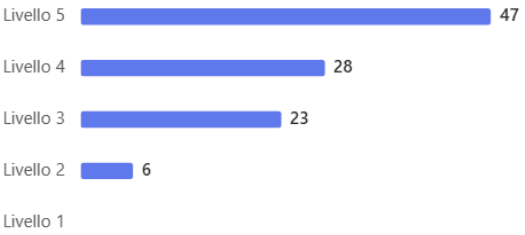
21. How relevant would a feature allowing you to add friends be for you? Più dettagli

4.13
Valutazione media



22. How important is it for you to have a group chat feature to manage activities?

Più dettagli



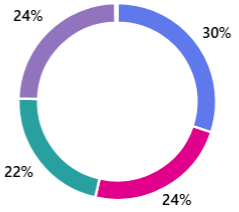
Trust, Notifications, and Community Features

The final part of the survey focused on evaluating user attitudes toward comfort, safety, and overall trust in the application environment. These questions were designed to assess how users perceive the app not only in terms of functionality, but also in relation to emotional comfort, moderation tools, and willingness to participate in a shared social space. First, participants were asked about their comfort with receiving notifications such as alerts for nearby activities, new messages, or friend requests.

23. What types of notifications would you like to receive?

Più dettagli

Activity invitations	92
People with similar interests nearby	72
Friend requests	67
Private messages	74
Altro	1

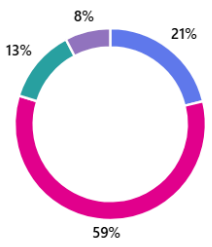


Second, the survey addressed personal safety when interacting with strangers through the app. Many users expressed the need to feel reassured about who they might be meeting. As a response to this, the idea of including features such as reporting suspicious behavior, blocking users, and even identity verification through documents or social media was widely supported. These tools were seen as important mechanisms to maintain a respectful and secure environment.

24. What would make you feel safer using the app?

Più dettagli

Ability to report users	22
Profile verification through social media or ID	61
Reviews from other users	13
Altro	8



25. How important is it for you to be able to report someone? Più dettagli

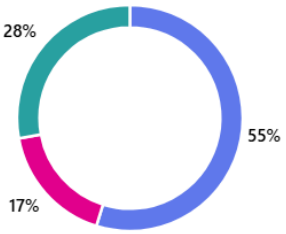
Third, opinions were collected regarding the possibility of leaving reviews, both for individual users and for the activities they host or attend. A large portion of respondents were in favor of implementing a lightweight review system, especially to build reputation and trust over time. Most users valued the option of reading feedback before deciding to join an activity or interact with someone new, especially in unfamiliar locations.

26. How important is a review system for evaluating people you've interacted with? Più dettagli

27. Would you feel comfortable if someone could leave a review about how an activity you participated in went? Più dettagli

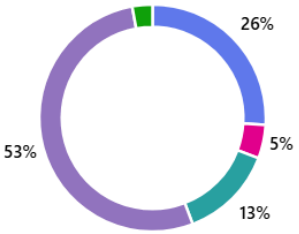
28. Would you feel comfortable if someone could leave a review about their experience interacting with you during an activity? Più dettagli

Yes	57
No	18
Maybe	29



29. If reviews were to be included, how should they be structured? Più dettagli

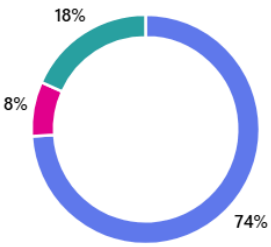
Text	27
Like / Dislike	5
Like / Dislike / No opinion	14
Mixed system	55
Altro	3



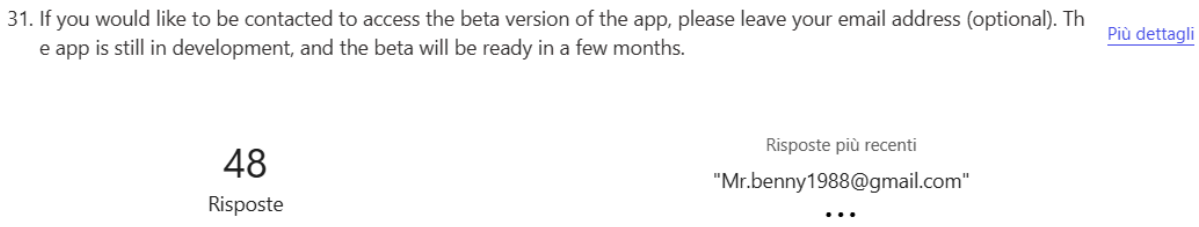
Fourth, the survey asked whether users were generally interested in using an app like Roamly. The response was strongly positive: the vast majority expressed enthusiasm or clear curiosity about the idea of discovering nearby activities and connecting with like-minded people while traveling. Many commented that they had long felt the need for such a platform but had not found any existing solution that fully addressed it.

30. Do you think you might be interested in the services offered by an app like this? Più dettagli

Yes	77
No	8
Maybe	19



Finally, the last question invited respondents to leave their contact information in case they wished to participate in the beta testing phase of the app, **as well as to share any additional suggestions**. A notable number of users volunteered, reinforcing not only the interest in the concept but also their willingness to support and engage with the product as it evolves.



10	anonymous	Che sia gratuita, sicura e affidabile. Quindi con possibilità di contattare il gestore dell'app per dare feedback e possibilità degli utenti di aggiungere informazioni utili per gli altri nei posti in cui si è stati..	Italiano

From Research to Design

The insights gathered from the questionnaire were not only valuable on a statistical level, but they also served as the foundation for shaping a user-centered design process. Specifically, the data collected was used to model a set of representative personas fictional but realistic user profiles that reflect the behaviors, goals, and pain points of our target audience. Each persona was carefully crafted to represent a specific user archetype observed in the responses such as the solo traveler, the Erasmus student, or the remote worker in a new city. These personas were enriched with details drawn directly from the survey results, including motivations, preferred features, communication styles, and concerns related to comfort and safety.

In parallel, the same data informed the creation of a navigation map that outlines the ideal structure and flow of the app. It served as a strategic tool to ensure that the application would be both intuitive and aligned with the real expectations of its intended users. Together, the personas and navigation map helped translate abstract survey data into concrete design guidelines, keeping the user at the core of every decision throughout the development process.

Personas

Amara Petrova

Amara Petrova

"Archetype"

- Age: 21
- Occupation: Master's student in Environmental Sciences
- Origin: Bulgaria
- Current Location: Amsterdam, Netherlands
- Languages: Bulgarian, English, learning Dutch
- Favorite Apps: Duolingo, WhatsApp, Instagram, Pinterest, Google Maps

"Moving abroad is exciting, but it's also overwhelming. I wish making new friends was as easy as it was back home."

Bio

Amara moved from Bulgaria to Amsterdam for her master's, excited by the adventure. But building friendships hasn't been easy. She spends time studying in cafés and visiting museums alone, wishing for people to share it with. Most student events focus on partying, which isn't her vibe, and online groups feel hard to turn into real connections. She hopes to meet other international students who also want to explore the city through shared interests, not just nightlife.

Challenges

- Most social events are party-focused, making it hard to find casual, interest-based meetups
- Traditional language classes feel too formal and intimidating for real conversation practice
- Feels isolated exploring the city alone and unsure how to find others with similar interests

Goals

- Find people to explore the city with, visit museums, or study together
- Join interest-based activities like photography walks or language exchanges
- Improve Dutch through informal conversations, not just classroom lessons
- Discover social events without relying on scattered student forums or party groups

Scenario:

It's Saturday afternoon in Amsterdam, and Amara just left a museum she explored alone again. She pulls out her phone, hoping to find something casual and not party related for the evening. She'd love to meet other international students who are also into journaling, photography, or just chilling at a café. She opens the app and sees a nearby "Slow Sunday Sketch & Language Swap" event hosted by a few other students. Dutch practice and creative vibes? Perfect. For once, she feels like she won't be the awkward outsider walking into a room full of strangers.

Wandering Thiago

🌍 Wandering Thiago

"Archetype"

- Age: 23
- Occupation: Freelance Videomaker & Content Creator
- Origin: Brazil
- Current Location: Prague, Czech Republic (passing through)
- Languages: Portuguese, English, basic Spanish
- Preferred Apps: Couchsurfing, Google Translate, Instagram, WhatsApp, YouTube

"I meet people for a day or two, then we all move on. It's exciting—but it gets lonely without a deeper connection."

Bio

Thiago left Rio to backpack across Europe, documenting his journey as a freelance videomaker. He funds his travels with small gigs and uploads content to his growing YouTube channel. In some cities, he visits distant relatives, but most of the time, he's solo. Hostel chats come and go, but meaningful connections are rare. He craves spontaneous meetups with people who share his creative spirit. The hardest part isn't the travel—it's the feeling of always being in-between.

Challenges

- Feels disconnected and anonymous in new cities
- Finds it hard to meet people without a shared language or activity
- Social apps feel superficial or geared toward dating
- Sometimes relies on extended family in certain cities, but often travels solo

Goals

- Connect with fellow travelers or creatives for deeper conversations
- Find spontaneous, low-pressure meetups—like a music jam or walking tour
- Share cultural experiences and stories over food or art
- Discover people to collaborate with or learn from along the way

Scenario:

Thiago just arrived in Prague and is checking into his hostel. He's filmed some footage for his vlog but isn't in the mood for another "pub crawl with strangers" night. He wants to connect with people on a deeper level ideally creatives, maybe other travelers with a story to tell. He opens the app and spots an open mic night with a casual pre meet at a small café. He decides to go, camera in hand, and ends up chatting with a Mexican musician and a Belgian painter about travel, culture, and content creation. For the first time that week, it doesn't feel like he's just passing through.

Nomadic Luca

Nomadic Luca

"Archetype"

- Age: 34
- Occupation: UX/UI Designer & Freelance Consultant
- Origin: Italy
- Current Location: Lisbon, Portugal
- Languages: Italian, English, basic Portuguese
- Preferred Apps: Notion, Slack, Airbnb, Strava, Spotify, Instagram

"I love the freedom of working anywhere—but it can feel like I'm starting from scratch every time I land somewhere new."

Bio

Luca left his Milan apartment three years ago to explore the world while working remotely. He's lived in Bali, Berlin, and Mexico City before settling for a few months in Lisbon. He thrives in co-working spaces, appreciates good coffee, and likes connecting with other creatives and techies. But starting over in a new place every few months can be emotionally draining. The hardest part is that every city feels like a reset button — socially, emotionally, and culturally.

Challenges

- Feels isolated at times, despite being in vibrant cities
- Has no time to scroll through random expat forums
- Wants to avoid dating apps when just seeking genuine community

Goals

- Build genuine friendships with people in similar situations
- Find buddies for workouts, hikes, or surf sessions
- Discover interest-based meetups without relying on scattered platforms like Facebook or Reddit

Scenario:

Luca finishes a client call at his coworking space in Lisbon and realizes it's Friday afternoon with no weekend plans. He scrolls through Instagram, sees beach photos from strangers, and feels the usual wave of disconnection. He doesn't want to join a massive expat Facebook group he's done that before, and it's mostly noise. What he really wants is to find someone nearby who's into design, hiking, or even just a good espresso chat. He opens the app, filters for solo creatives in the area, and finds a small Saturday morning surf & sketch meetup hosted by another remote worker like him. He finally feels a bit of that "community on the move" vibe he's been missing.

Wired Isabela

🌍 Wired Isabela

"Archetype"

- Age: 39
- Occupation: Full Stack Developer (Remote, Fintech)
- Origin: Brazil
- Current Location: Medellín, Colombia
- Languages: Portuguese, English, intermediate Spanish
- Preferred Apps: Slack, VSCode, GitHub, Telegram, Meetup, Spotify

"I love Medellín's energy—but between work calls and solo meals, it's easy to feel like I'm just passing through, even after a year."

Bio

Isabela moved from São Paulo to Medellín for her remote developer job, drawn by the lifestyle and tech scene. She works from cafés and co-working spaces, but often feels disconnected outside work hours. Most of her social life happens online or in passing at tech events. She misses speaking Portuguese and lights up when she meets other Brazilians or Mexicans. After a year, Medellín still feels temporary. She's not looking for parties—just meaningful connections with people who share her rhythm and interests.

Challenges

- Feels socially in-between—too foreign for locals, too different from short-term expats
- Works remotely, which limits her chances to meet new people organically
- Misses speaking Portuguese and connecting with people who get her cultural background

Goals

- Build friendships with other long-term expats and remote workers
- Find spaces where she can speak Portuguese or connect with other Brazilians or Mexicans
- Discover laid-back meetups around music, coding, or food
- Create a sense of home in a city she chose for its culture and cost of living

Scenario:

After another long workday full of Zoom calls, Isabela walks through Laureles looking for a reason not to head straight home. She loves Medellín, but after a year, she still hasn't found "her people." She opens the app and browses events tagged "tech," "music," and "Portuguese-speaking." There's a midweek coding + arepas night hosted by a small group of Brazilian and Mexican developers. It's nearby, it's chill, and she won't have to explain what React is or fake small talk in Spanish. Maybe tonight, it'll feel a little more like home.

Competitive Landscape

To better position Roamly within the broader market of travel-related social apps, I analyzed both direct and indirect competitors. This research served to identify functional gaps, market saturation points, and user needs that are not yet adequately addressed. Competitors were chosen based on their relevance to Roamly's core mission: enabling spontaneous meetups among travelers, international students, and digital nomads, with an emphasis on geolocation, interest-based discovery, and real-time social interaction.

Direct Competitors

Direct competitors include platforms specifically designed for people on the move such as travelers, backpackers, and nomads that offer some form of real-time social engagement

Features	Couchsurfing Hangouts	Travello	Backpackr
User profiles with interests and languages	✅	✅	✅
Post activities for interests	✅	❌	✅
Friends system	❌	✅	✅
Matching based on interests, location, and mood	❌	✅	✅
Real-time chat (DMs and groups)	✅	✅	✅
Notifications for friend requests, activities	✅	✅	✅
Geolocation-based and experiences	✅	✅	✅
Reviews on people	❌	❌	✅
Reviews on experiences	✅	✅	✅

While each of these apps attempts to foster social connection during travel, they often fall short in delivering a seamless, filtered, and real-time experience. For example, Couchsurfing Hangouts provides spontaneous meetups but lacks user matching capabilities. Backpackr and Travello support chat and profiles but are less dynamic in promoting local discovery based on user mood, language, or intent.

Indirect Competitors

Indirect competitors are platforms not specifically designed for travelers, but that share overlapping features such as social discovery, geolocation, or community building.

Features	Tinder/Bumble BFF	Meetup	Nomad List
User profiles with interests and languages	✅	✅	✅
Post activities for spontaneous meetups	❌	❌	❌
Friends system	✅	❌	❌
Matching based on interests, location, and mood	❌	❌	❌
Real-time chat (DMs and groups)	✅	❌	❌
Notifications for friend requests, activities	✅	✅	❌
Geolocation-based features	✅	❌	✅
Reviews on people	❌	❌	✅
Reviews on experiences	❌	❌	✅

These platforms confirm the growing interest in interest-based micro-sociality, but they do not fully support spontaneous, localized meetups. For instance, Bumble BFF is primarily a repurposed dating app, while Meetup is more structured and event-driven. Nomad List provides insights into nomad-friendly locations but lacks tools for real-time interpersonal connection.

Emerging Competitor: NomadTable

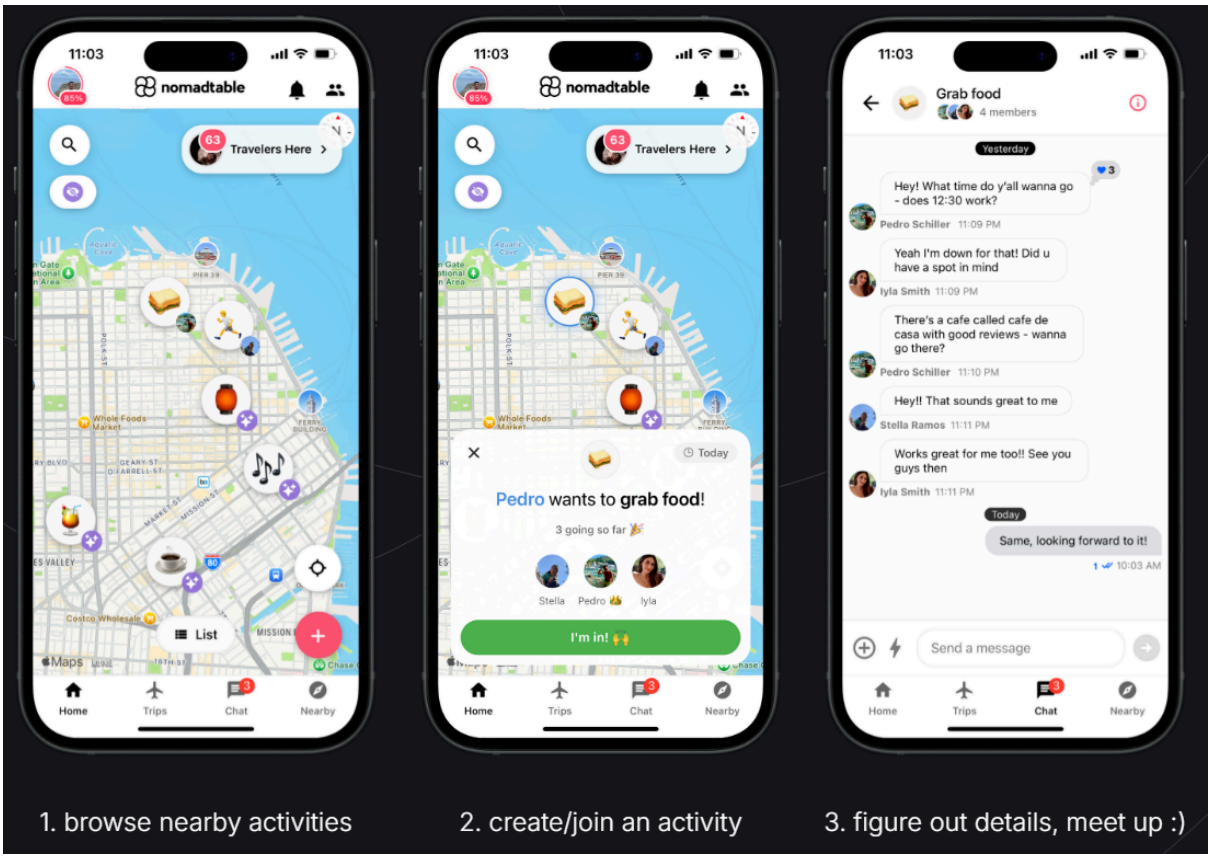
A notable recent entry in this space is NomadTable, an app officially launched in late March 2025 in Silicon Valley. The platform targets digital nomads and remote workers, aiming to facilitate spontaneous, location-based meetups among people who happen to be in the same city.

The app enables users to:

- Browse nearby activities on an interactive map;
- Create or join group meetups (e.g., for food, coworking, or local hangouts);

- Coordinate with participants through in-app group chats.

NomadTable is built around the idea of frictionless social interaction, with a minimal and intuitive interface that prioritizes speed, simplicity, and spontaneity. It caters well to users who are looking for quick plans without much planning or long-term commitment.



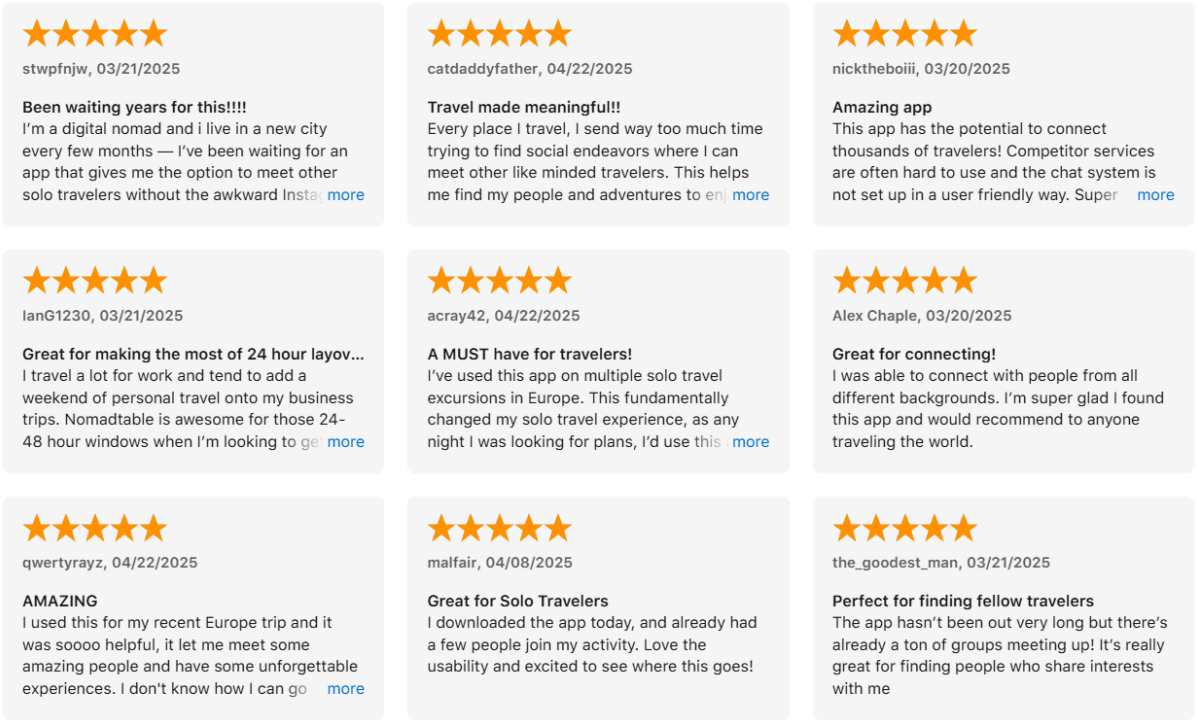
However, while NomadTable successfully implements the basic mechanics of casual meetups, it currently lacks several key features that emerged as important in our user research, such as:

- Filtering activities by spoken language, interests, and mood/vibe;
- A rich user profile that allows for more detailed self-expression and connection;
- Trust and safety mechanisms, such as profile verification and the ability to leave reviews or report inappropriate behavior.

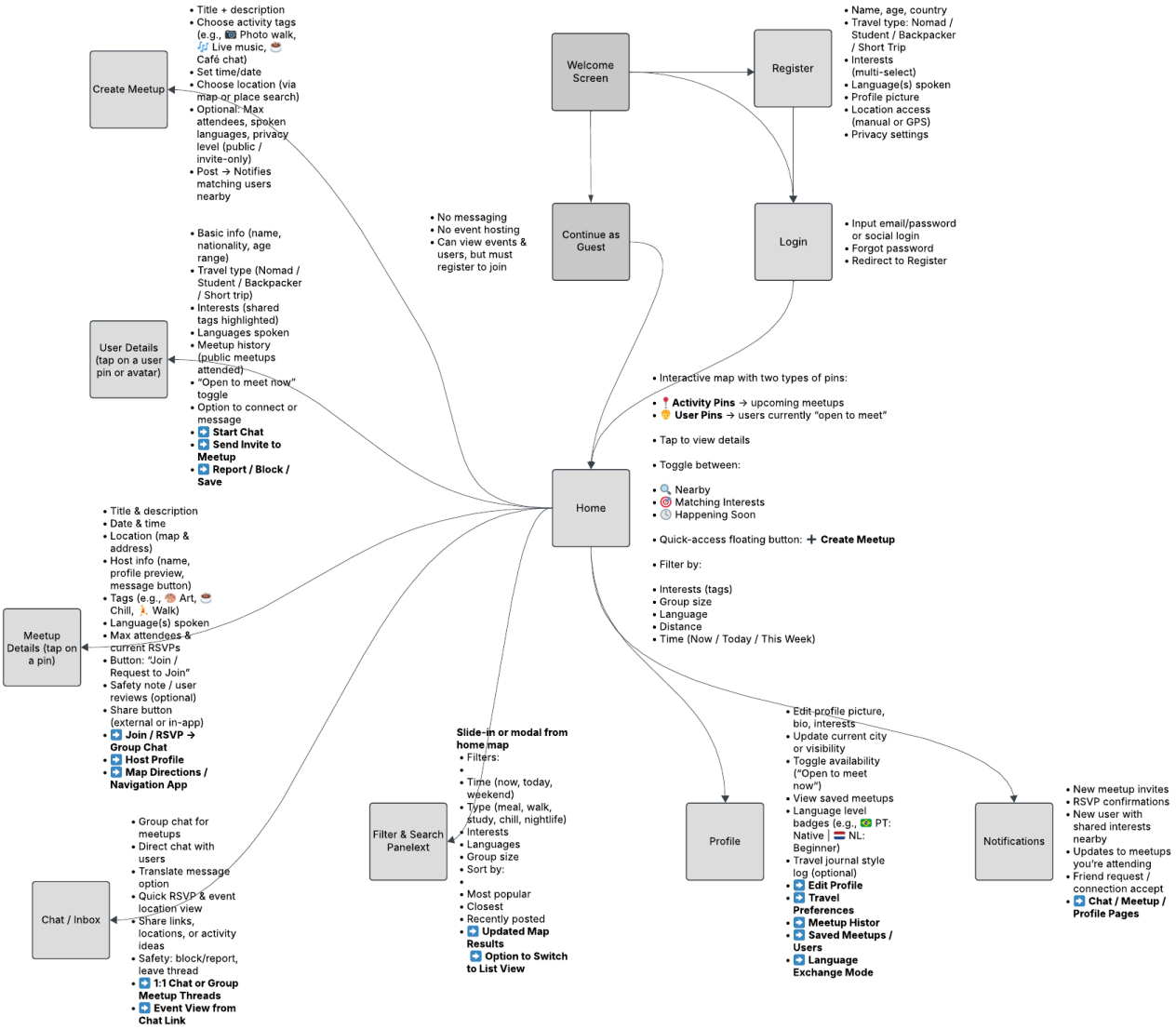
Additionally, NomadTable appears to be focused primarily on the North American digital nomad scene, whereas Roamly is strategically positioned to serve the

European market, with particular attention to 18–24-year-olds, such as Erasmus and international students. These users are often looking for casual, short-term social opportunities, and benefit from a more culturally sensitive and inclusive platform.

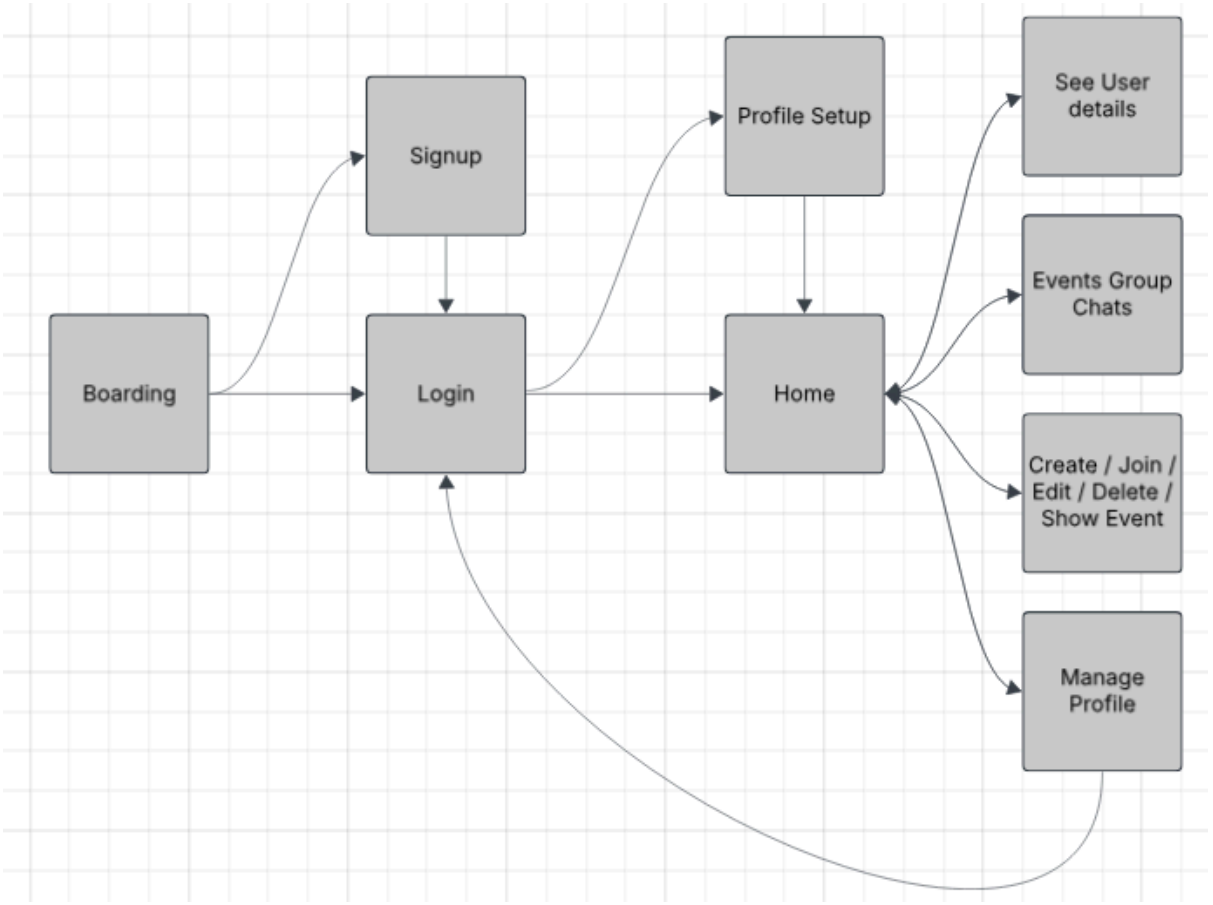
In this sense, NomadTable's emergence confirms a growing global demand for spontaneous social connection tools, but Roamly differentiates itself by offering deeper personalization, multi-language support, and features designed for young, mobile communities across Europe.



Original Navigation Map



Navigation Map Evolution and Design Constraints



The original navigation map envisioned a fully integrated ecosystem, including detailed user profiles, dynamic filtering of activities, multilayered chat systems (1:1 and group), event management and more. The goal was to create an intuitive and feature-rich experience tailored for spontaneous social interaction while traveling. However, during the development phase, several practical limitations emerged, most notably, the scope of the project relative to the available time and individual development capacity. Being the sole developer of the application, I had to balance ambition with feasibility. Advanced features such as multi-filters, complex event flows, and deeply integrated chat systems require significant backend logic, UX design, and QA time none of which were realistically achievable within the timeframe of a university project.

As a result, a simplified version of the navigation map was created. This version focuses on the core experience: user registration, profile setup, map-based user/event discovery, and basic event participation and group chat. While it lacks some of the originally planned advanced functionalities, this streamlined structure ensures a stable and usable MVP that retains the core mission of Roamly: enabling short-term social meetups for international users based on proximity and shared interests.

This decision reflects a strategic trade-off, prioritizing implementation realism over feature breadth. The simplified map still allows for future expansion, leaving room for iterative updates based on further user feedback and available resources.

Application Flow

The user journey in Roamly follows a structured and intuitive sequence designed to ensure a smooth onboarding process and an engaging user experience.

1. Onboarding
   When the app is launched for the first time, users are greeted by an onboarding screen that presents two options: Sign Up or Log In. This screen serves as the entry point for both new and returning users.

2. Sign Up / Log In

   - New users can create an account via email and password.

   - Returning users can authenticate with their existing credentials.

   - Upon successful login, if the user has never completed onboarding before, they are directed to the multi-step profile creation process.

3. Multi-Step Profile Creation
   This process is divided into four sequential activities:

   - MakeProfile1Activity: Users input their full name, age, country of origin, category (e.g., student, traveler), and upload a profile picture.

   - MakeProfile2Activity: Users select the languages they speak. These are saved and displayed using interactive chips, with automatic flag icons.

   - MakeProfile3Activity: Users choose their interests among a fixed set (e.g., food, culture, sports, networking).

   - MakeProfile4Activity: Users define their current vibe (party/chill) and visibility status. After saving, the user is redirected to the main screen of the app.

4. HomeActivity – Central Hub
   Once onboarding is complete, users land on the HomeActivity, which acts as the control center of Roamly. From here, users can:

- View and interact with a dynamic map showing other users and public events within a 10km radius.

- Tap on markers to view detailed tooltips for either user profiles or events.

- Create new events by long-pressing on the map, choosing the type, time, participant limits, and other filters.

- Join or leave existing events, see participants, and access integrated group chats tied to each event.

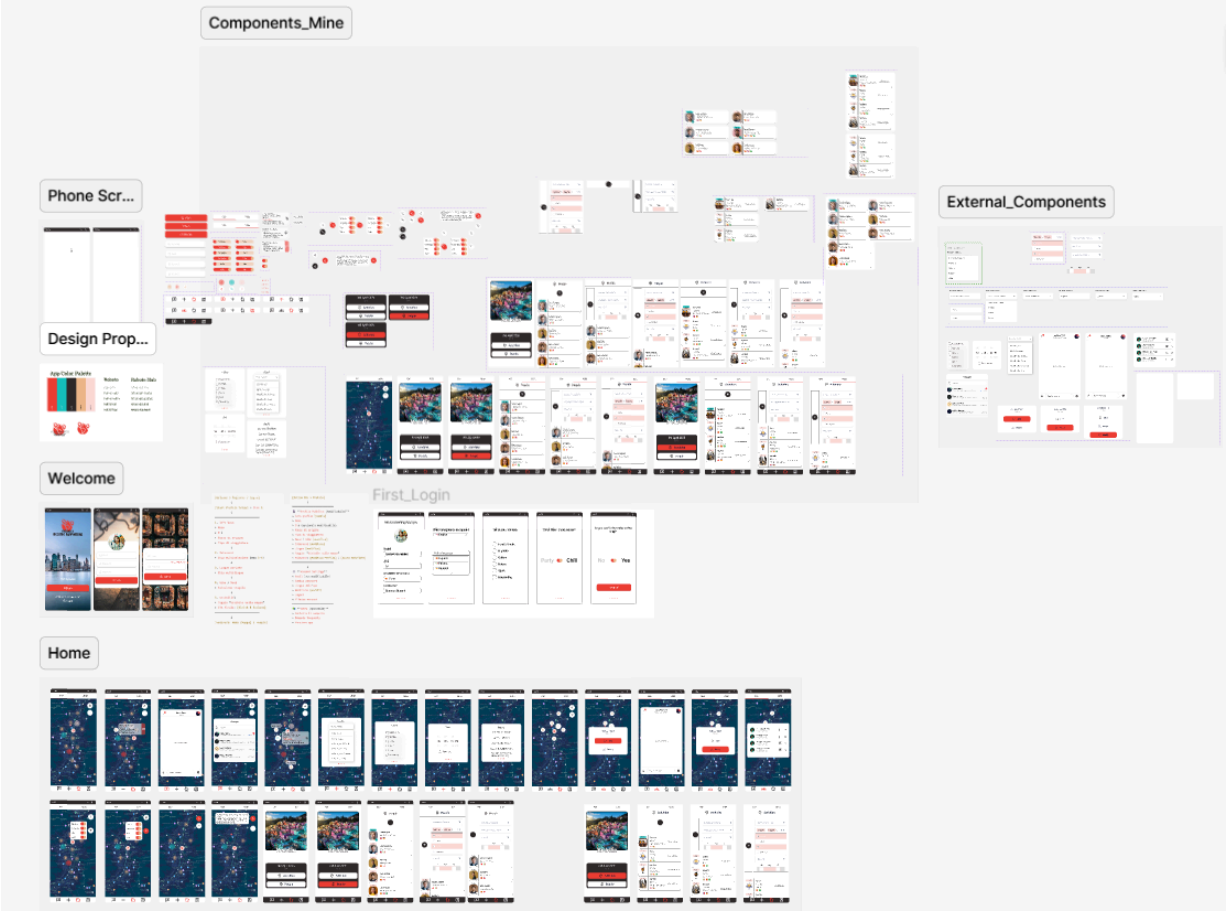
- Access the Profile section, where they can update most parameters from the onboarding process and manage account settings. A dedicated section allows the user to change their password or log out.

Figma Mockups:

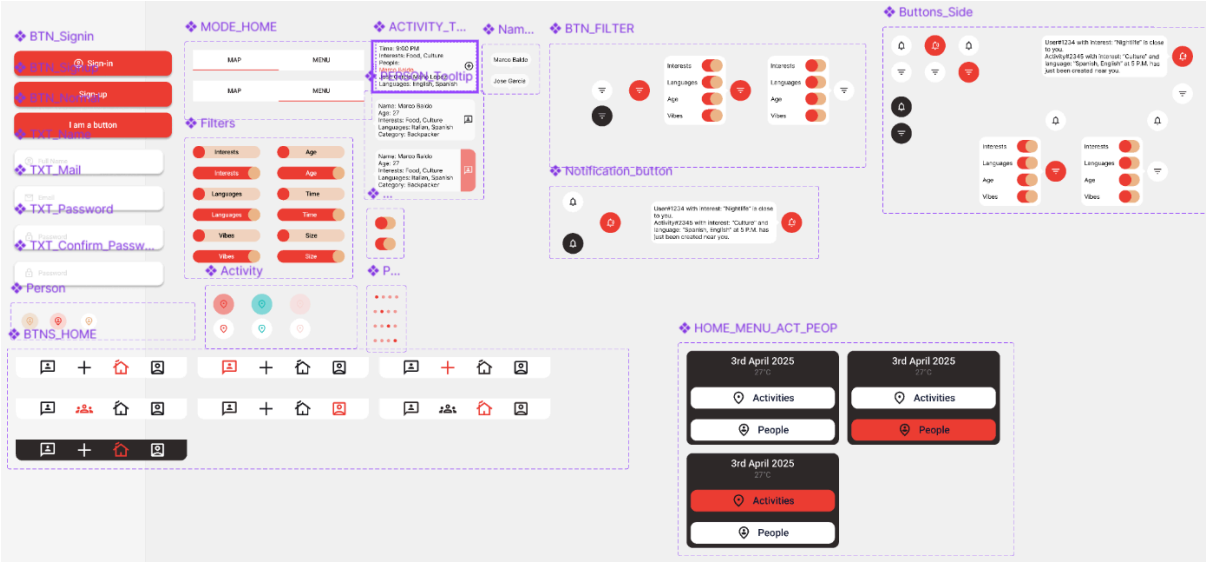
A significant effort was dedicated to designing the application's interface using Figma, with the aim of remaining faithful to the initial vision of the project. Throughout this process, a substantial amount of time was spent learning and mastering Figma's component system particularly how to structure reusable components in a scalable and maintainable way, enabling consistency and efficiency across the entire design.

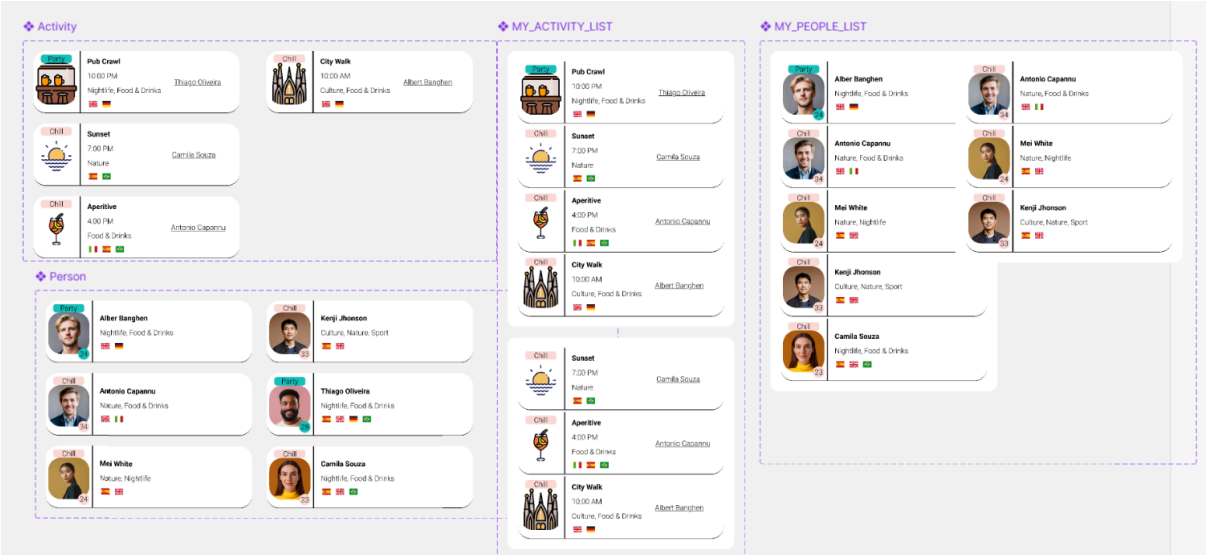
The design was articulated along two parallel conceptual directions. On one hand, a more structured and interface-driven approach was explored, featuring dedicated screens and menu-based navigation. On the other, a map-centered interface was conceived, aiming to offer a more dynamic and immersive user experience. Ultimately, this second approach based on geolocation and map interaction was selected as the foundation for the MVP implementation.

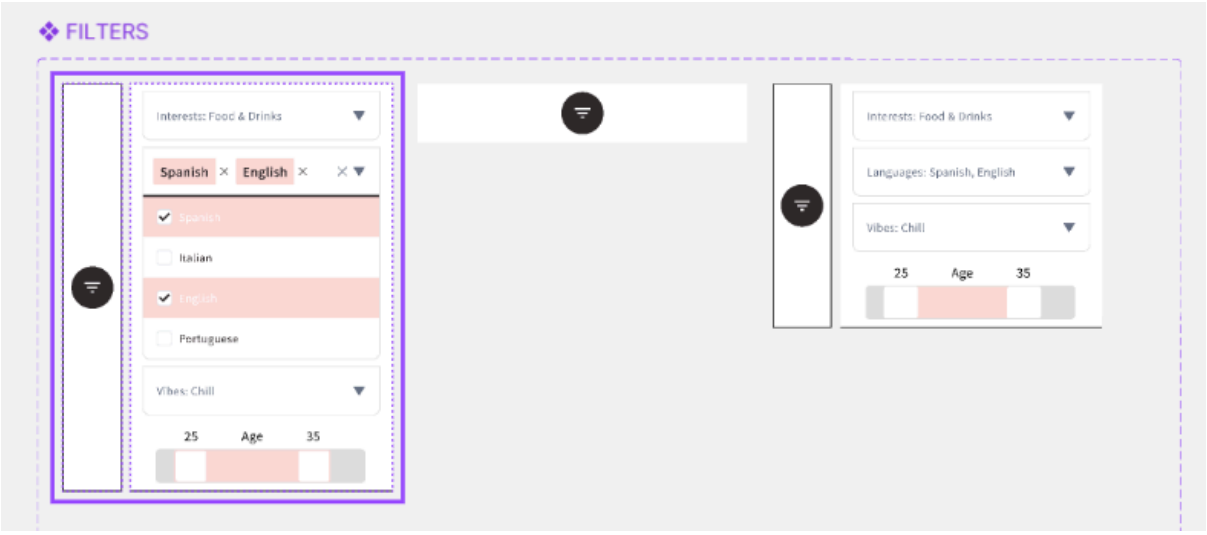
Although the idea of generating application code directly from the Figma design system was initially considered, this path was not pursued in the final development. Nevertheless, the design process was crucial in shaping the application: the visual identity, UI structure, and layout decisions of the MVP were strongly based on what had been created during this stage.



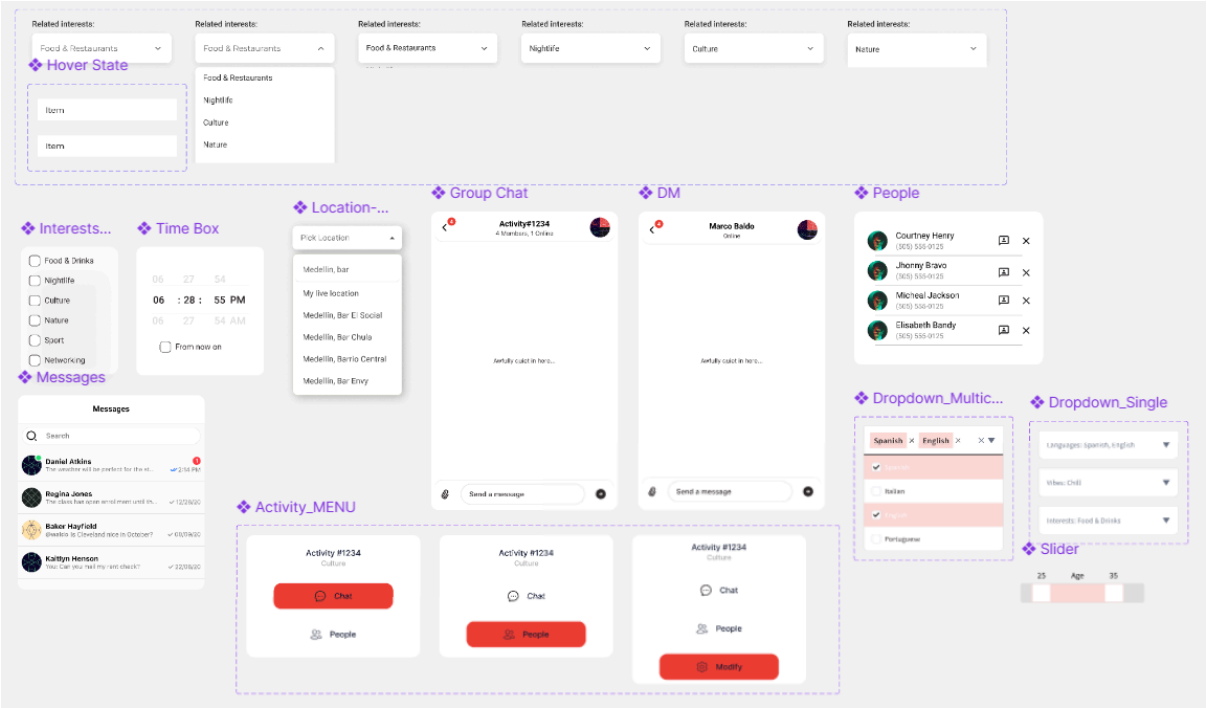
Internal Components



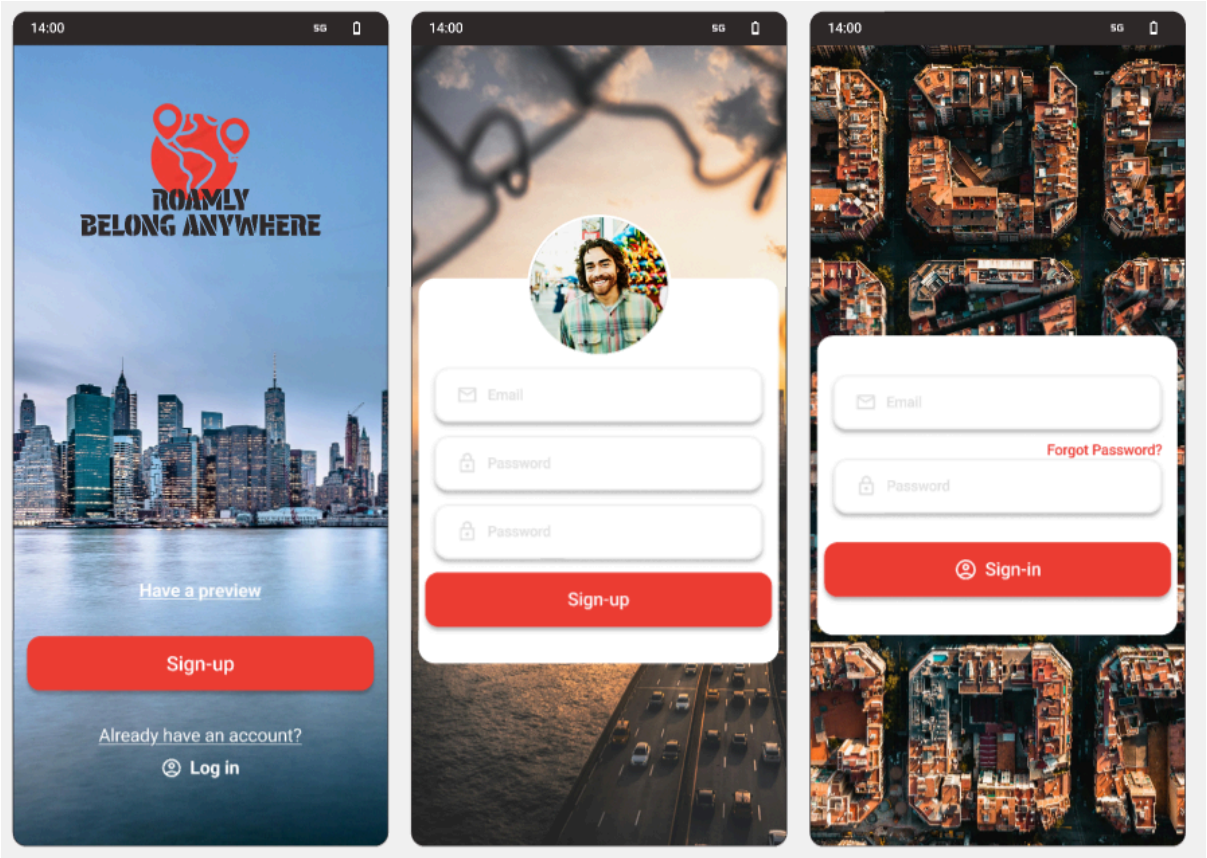




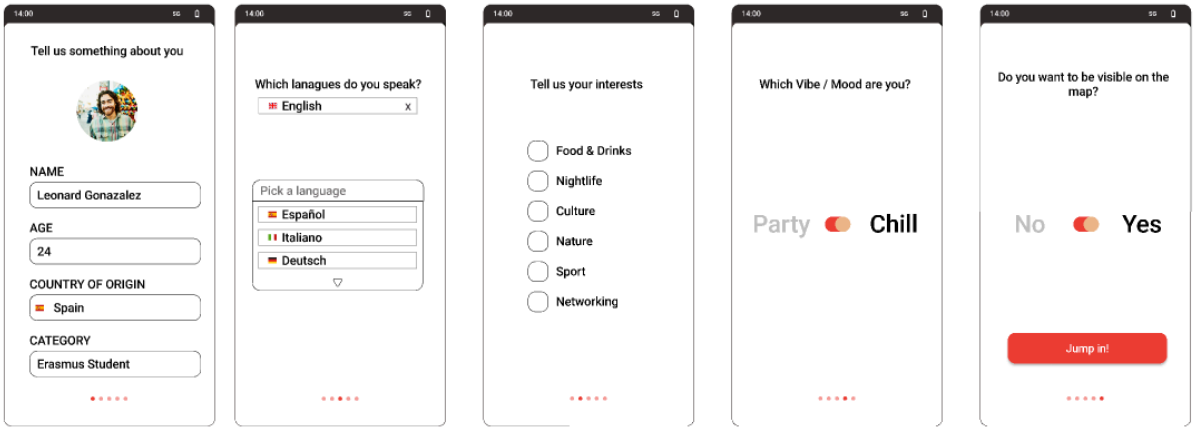
External Components



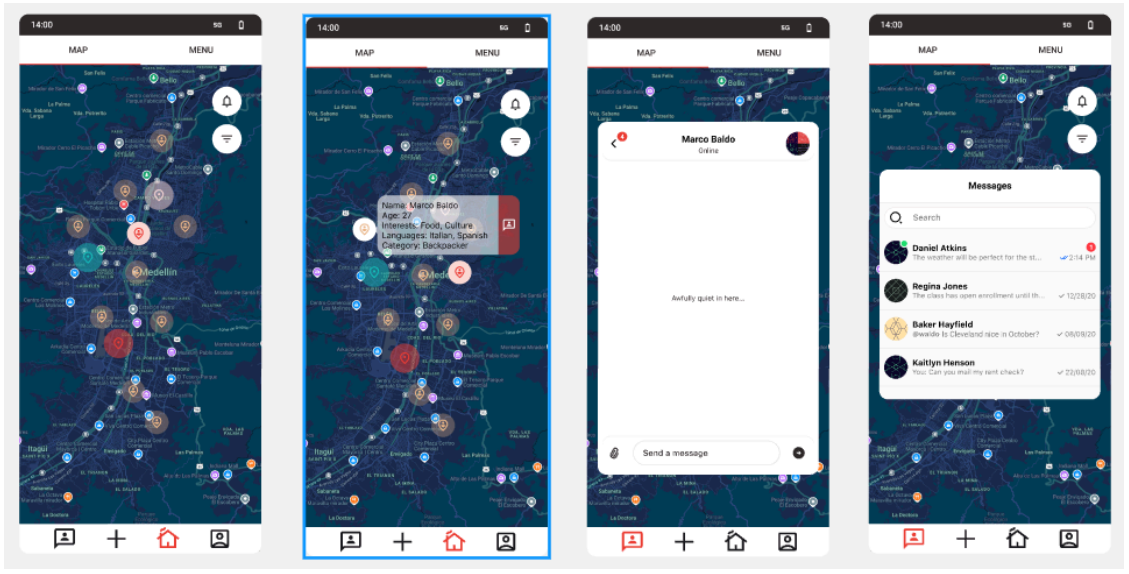
Onboarding

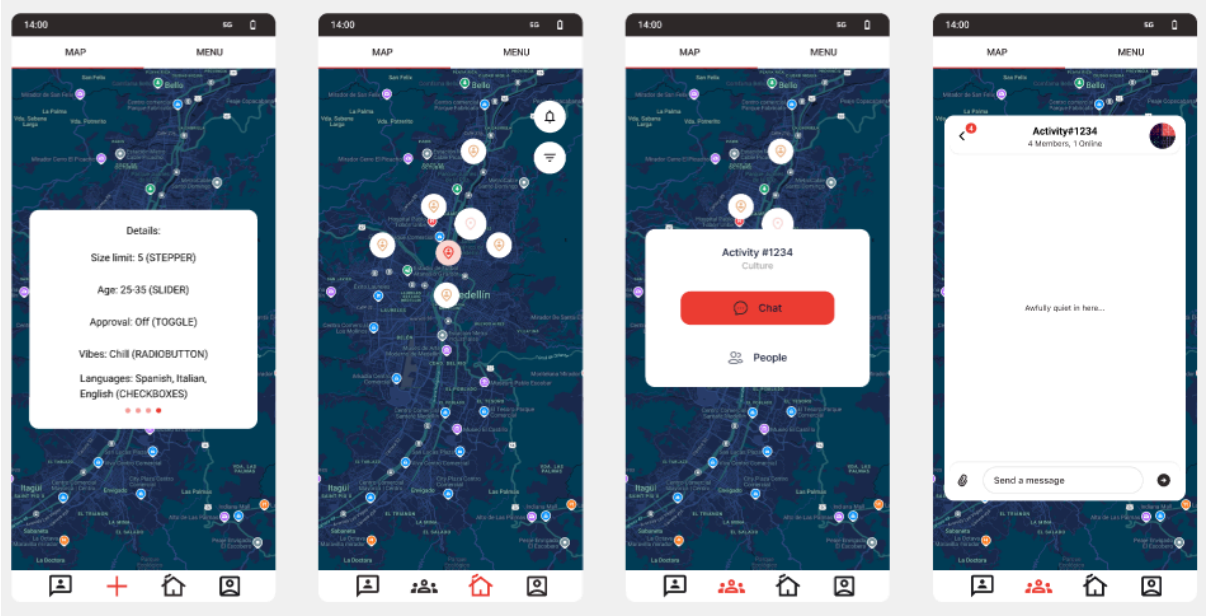


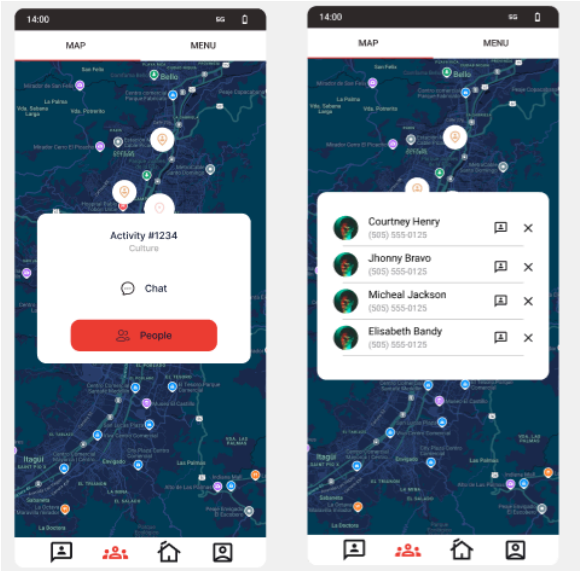
First Login

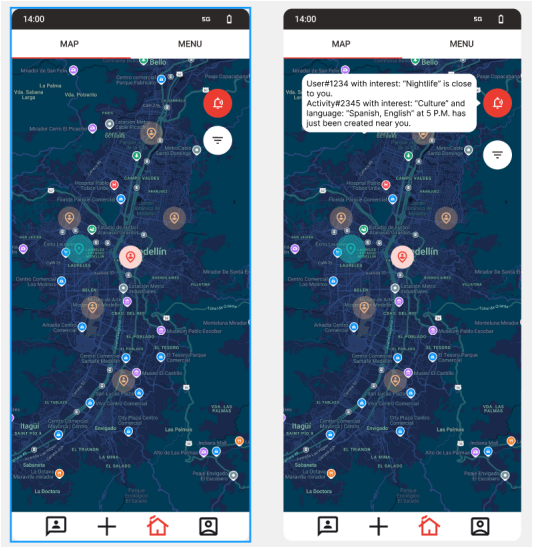


Home Map

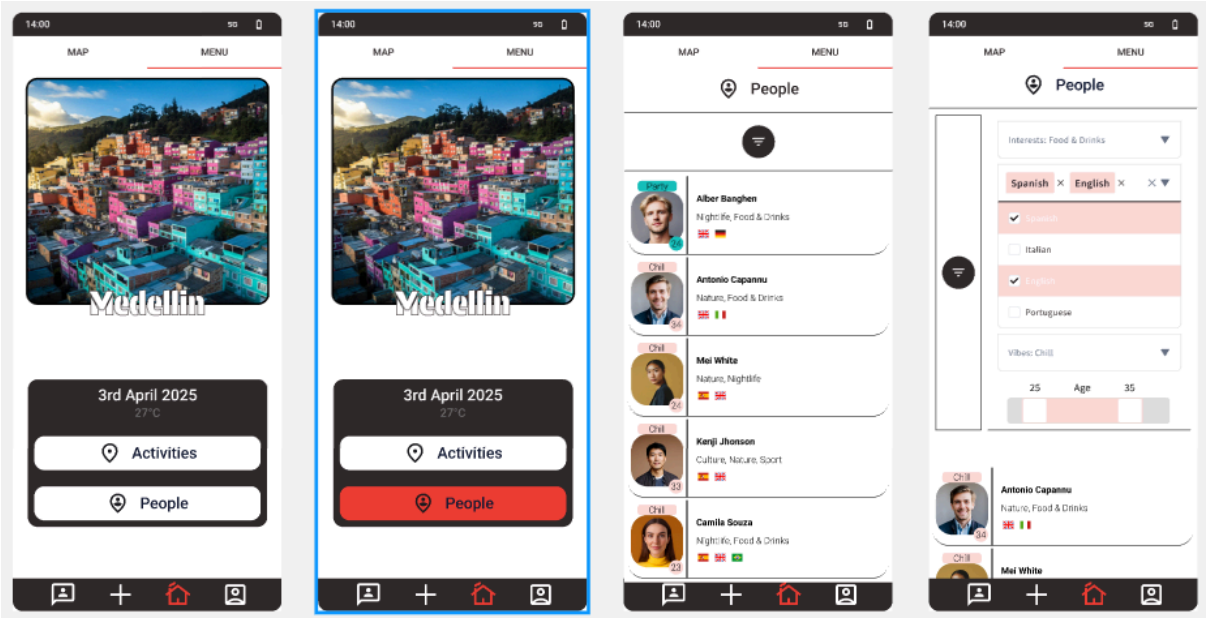


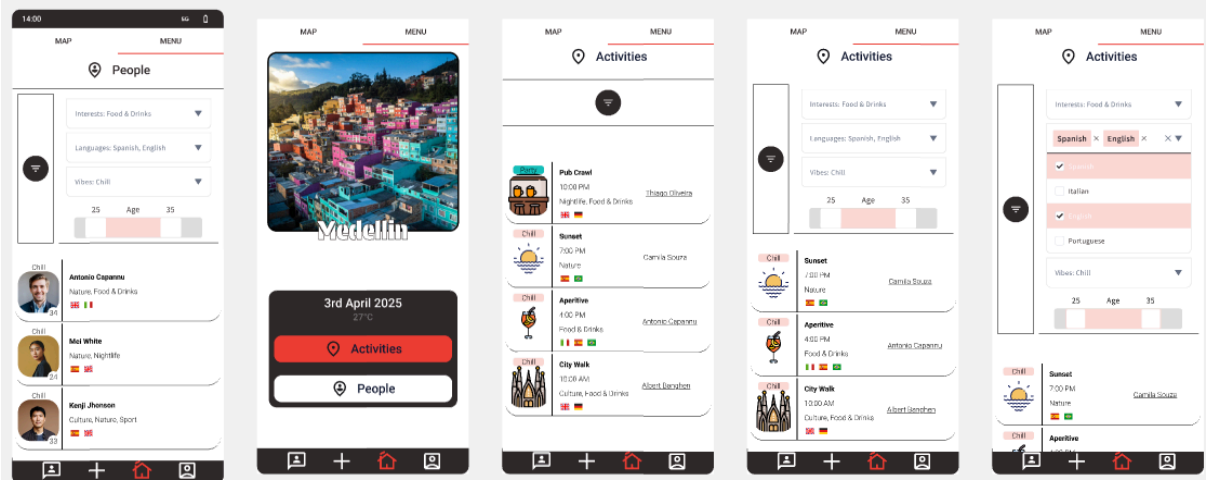

14:00
MAP
MENU
Details:
Size limit: 5 (STEPPER)
Age: 25-35 (SLIDER)
Approval: Off (TOGGLE)
Vibes: Chill (RADIOBUTTON)
Languages: Spanish, Italian, English (CHECKBOXES)
Activity #1234
Culture
Chat
People
Activity#1234
4 Members, 1 Online
Awfully quiet in here...
Send a message


14:00
MAP
MENU
Activity #1234
Culture
Chat
People
Courtney Henry
(505) 555-0125
Jhonny Bravo
(505) 555-0125
Micheal Jackson
(505) 555-0125
Elisabeth Bandy
(505) 555-0125


14:00
MAP
MENU
User#1234 with interest: "Nightlife" is close to you.
Activity#2345 with interest: "Culture" and language: "Spanish, English" at 5 P.M. has just been created near you.

Home Menu


MAP
MENU
Medellin
3rd April 2025
27°C
Activities
People
Party
Chill
Alber Banghen
Nightlife, Food & Drinks
Antonio Capannu
Nature, Food & Drinks
Mei White
Nature, Nightlife
Kenji Jhonson
Culture, Nature, Sport
Camila Souza
Nightlife, Food & Drinks
Interests: Food & Drinks
Spanish
English
Italian
Portuguese
Vibes: Chill
25
Age
35


People
Activities
Medellin
3rd April 2025
27°C
Pub Crawl
10:00 PM
Nightlife, Food & Drinks
Thiago Oliveira
Sunset
7:00 PM
Nature
Camila Souza
Aperitivo
4:00 PM
Food & Drinks
Antonio Capannu
City Walk
10:00 AM
Culture, Food & Drinks
Albert Banghen

Implementation

The development of this application has been an intense but highly rewarding journey. I'm genuinely proud of the result and fully convinced that this app has strong potential. The design created in Figma, was a central part of the process. I invested significant effort into learning how to properly manage and reuse components within the Figma design system, staying true to the original vision of the project. While I initially intended to generate code directly from the design, this was not ultimately implemented, though the Figma work proved extremely useful: the MVP's design is based on what was created in that phase.

I would love to keep working on this app and ideally form a small team to extend its features and refine the experience. Despite developing the entire MVP alone, I've managed to implement a wide range of functionalities, learning by doing and overcoming a variety of technical challenges along the way. From a backend perspective, I originally started with Firebase, but since its storage service is not free, managing images would have required integrating a third-party solution. I eventually decided to switch completely to Supabase, which provided more flexibility but also required a rework of the backend logic.

Integrating Mapbox was another complex task. I had to experiment extensively to understand how to work with markers, tooltips, interactions, and user interface overlays. Managing dynamic event and user markers, fetching and updating information from the database, and synchronizing tooltip behavior with map interactions required a lot of trial and error.

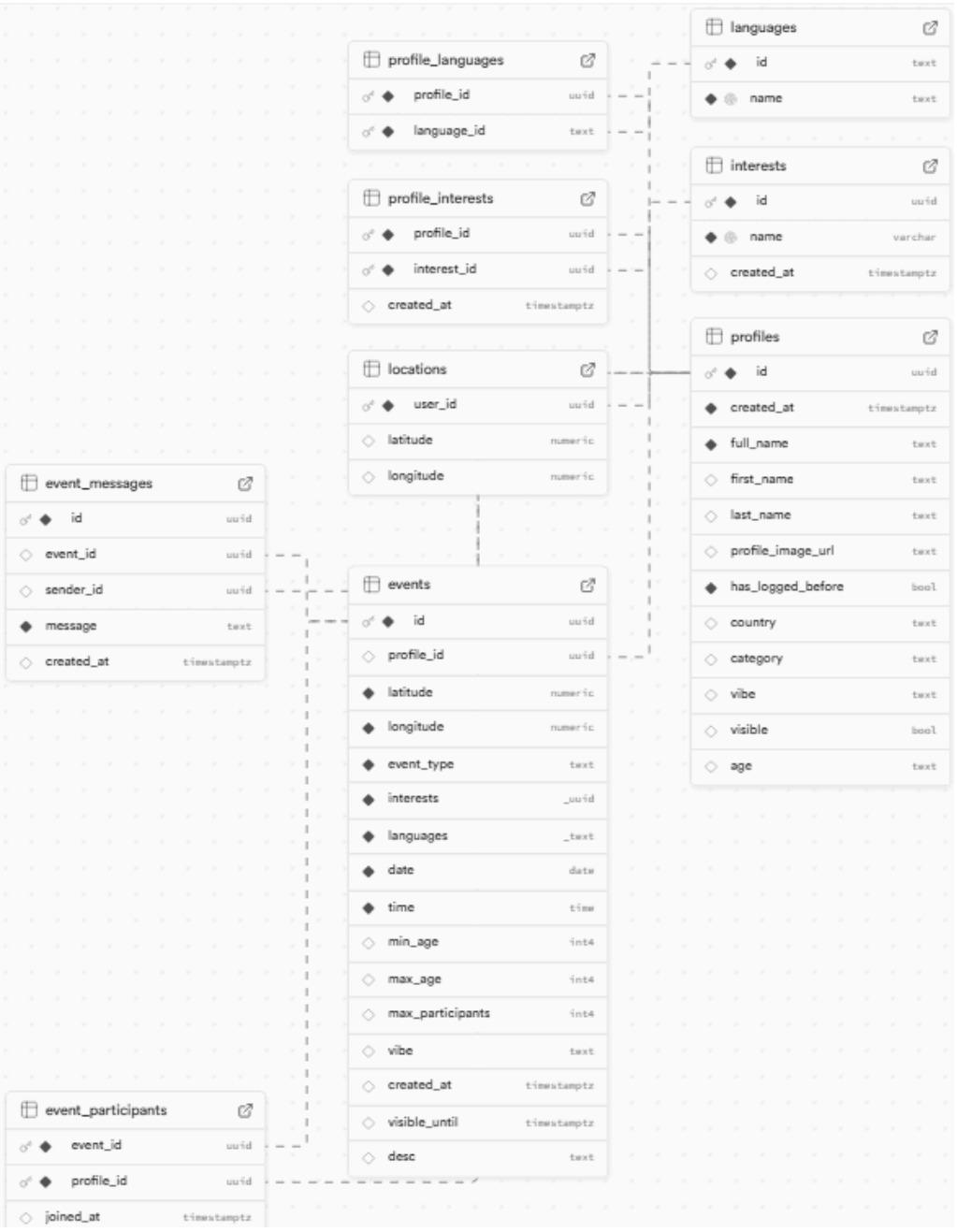
Another major challenge was correctly handling geolocation, permissions, and position updates. Ensuring that user position was updated continuously, while respecting system permissions added significant complexity to the app's lifecycle management. Finally, managing countries and languages across the UI and backend brought its own set of difficulties. ISO codes, drawable resources, and language labels often didn't align, so I had to implement logic to normalize data and map elements correctly for both display and storage.

There are still many improvements to be made and features to add, but the essential structure is in place. The MVP is fully functional and reflects a solid base for further development. This project has given me the opportunity to deepen my understanding of mobile development, backend integration, external service APIs, and design systems and it has strongly reinforced my motivation to keep growing in this field.

Supabase

Database Schema

The database schema was designed to support the core features of the application, including user profiles, geolocation-based event discovery, interest/language-based filtering, group participation, and messaging. Supabase was used as the backend platform, offering a scalable and developer-friendly environment with support for Postgres, row-level security, and simple integration via REST and client SDKs.



1. profiles

This is the main table storing user profile information.

- id: primary key (UUID), corresponds to the Supabase auth user.
- created_at: timestamp of profile creation.
- full_name, first_name, last_name: user name fields.
- profile_image_url: optional profile picture URL.
- has_logged_before: boolean used to distinguish new users at first login.
- country, category, vibe: categorical descriptors.
- visible: boolean flag to hide/show the user on the map.
- age: user's declared age.

id uuid	created_at timestamptz	full_name text	first_name text	last_name text	profile_image_url text
0b182beb-836c-4238-9e6b-bdd14ca75c9	2025-07-06 09:42:08.64843+0	Utente numerodue	Utente	numerodue	https://sfbhiqjlqkwzkeexsbif.supaba
37e3f003-cd2b-4c16-8b2f-2ef2328d06e7	2025-07-06 09:39:05.953431+0	Utente Numerouno	Utente	Numerouno	https://sfbhiqjlqkwzkeexsbif.supaba
38fa031e-22b5-437b-8644-fec4d492ec44	2025-07-06 09:47:42.614326+0	Utente numeroquattro	Utente	numeroquattro	https://sfbhiqjlqkwzkeexsbif.supaba

2. languages

Stores the list of available spoken languages.

- id: ISO code (e.g., "en", "es", "fr").
- name: full name of the language (used for UI display).

id text	name text
el	Ελληνικά
en	English
es	Español
et	Eesti
fa	فارسی

3. interests

Contains the master list of interests available for filtering and profile customization.

- id: UUID (primary key).
- name: interest name (e.g., "culture", "nightlife", “nature”).
- created_at: optional timestamp.

id uuid	name varchar	created_at timestamptz	+
13e870d5-fd5a-4daf-8793-b6057e01e0e1	Networking	2025-06-29 19:04:51.500523+0	
9fac420a-a601-46ef-af96-aeb216dc85d4	Nature	2025-06-29 19:04:51.500523+0	
aada897e-96bb-472f-81fa-35493b3eabcc	Culture	2025-06-29 19:04:51.500523+0	

4. profile_languages

Many-to-many relation linking users with the languages they speak.

- profile_id: reference to a profiles entry.
- language_id: reference to a languages entry (text).

profile_id uuid	language_id text
0b182beb-836c-4238-9e6b-bdd1...	de
0b182beb-836c-4238-9e6b-bdd1...	fr
0b182beb-836c-4238-9e6b-bdd1...	it

5. profile_interests

Many-to-many relation associating users with selected interests.

- profile_id: reference to a profiles entry.
- interest_id: reference to an interests entry (UUID).
- created_at: timestamp for when the link was created.

profile_id uuid	interest_id uuid	created_at timestamptz
0b182beb-836c-4238-9e6b-bdd1...	9fac420a-a601-46ef-af96-aeb216...	2025-07-06 10:49:38.481858+0
0b182beb-836c-4238-9e6b-bdd1...	c10080bb-746b-4118-9870-edbf1e...	2025-07-06 10:49:38.481858+0
37e3f003-cd2b-4c16-8b2f-2ef232...	d12c5da8-1120-4a8e-a164-c49ff0...	2025-07-06 09:40:11.64541+00
37e3f003-cd2b-4c16-8b2f-2ef232...	df375807-2a7e-4d38-b06d-5fb20...	2025-07-06 09:40:11.64541+00

6. locations

Stores the most recent known location of each user.

- user_id: foreign key to profiles.
- latitude, longitude: geographic coordinates (numeric).

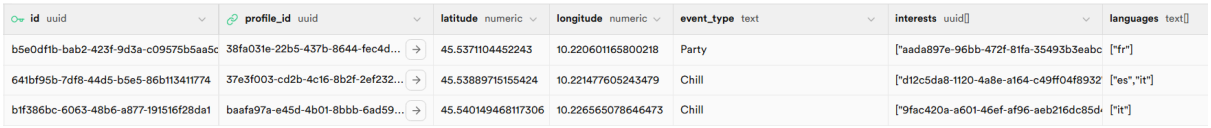
user_id uuid	latitude numeric	longitude numeric
0b182beb-836c-4238-9e6b-bdd1...	45.5392983	10.2199
37e3f003-cd2b-4c16-8b2f-2ef232...	45.5384	10.2199
38fa031e-22b5-437b-8644-fec4d...	45.5379983	10.221

7. events

Main table for storing all group events created by users.

- id: UUID (primary key).
- profile_id: reference to the user who created the event.
- latitude, longitude: event location.
- event_type: short label (e.g., "food", "walk", "drink").
- interests: serialized list of UUIDs from interests.
- languages: serialized list of language codes from languages.
- date, time: date and time of the event.
- min_age, max_age: age range restrictions.

- max_participants: limit for attendees.
- vibe: desired mood (e.g., "chill", "party").
- created_at: timestamp of creation.
- visible_until: used to remove expired events after a grace period.
- desc: textual description of the event.

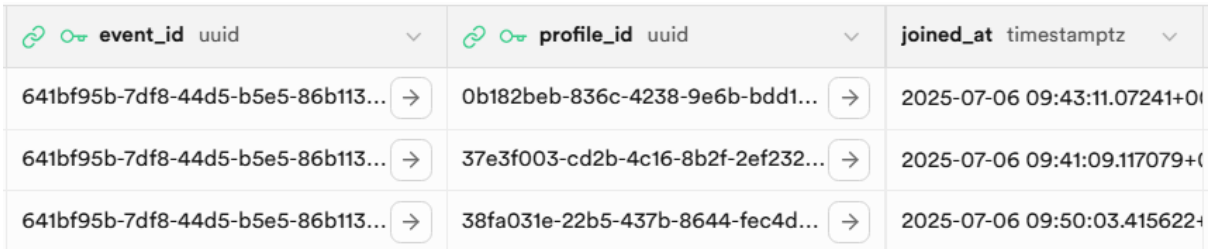


id uuid	profile_id uuid	latitude numeric	longitude numeric	event_type text	interests uuid[]	languages text[]
b5e0df1b-bab2-423f-9d3a-c09575b5aa5c	38fa031e-22b5-437b-8644-fec4d...	45.5371104452243	10.220601165800218	Party	["aada897e-96bb-472f-81fa-35493b3eabc	["fr"]
641bf95b-7df8-44d5-b5e5-86b113411774	37e3f003-cd2b-4c16-8b2f-2ef232...	45.53889715155424	10.221477605243479	Chill	["d12c5da8-1120-4a8e-a164-c49ff04f8932	["es","it"]
b1f386bc-6063-48b6-a877-191516f28da1	baafa97a-e45d-4b01-8bbb-6ad59...	45.540149468117306	10.226565078646473	Chill	["9fac420a-a601-46ef-af96-aeb216dc85d	["it"]

8. event_participants

Tracks user participation in events.

- event_id: reference to an events entry.
- profile_id: user who joined.
- joined_at: timestamp (automatically managed).



event_id uuid	profile_id uuid	joined_at timestamptz
641bf95b-7df8-44d5-b5e5-86b113...	0b182beb-836c-4238-9e6b-bdd1...	2025-07-06 09:43:11.07241+0(
641bf95b-7df8-44d5-b5e5-86b113...	37e3f003-cd2b-4c16-8b2f-2ef232...	2025-07-06 09:41:09.117079+(
641bf95b-7df8-44d5-b5e5-86b113...	38fa031e-22b5-437b-8644-fec4d...	2025-07-06 09:50:03.415622+

9. event_messages

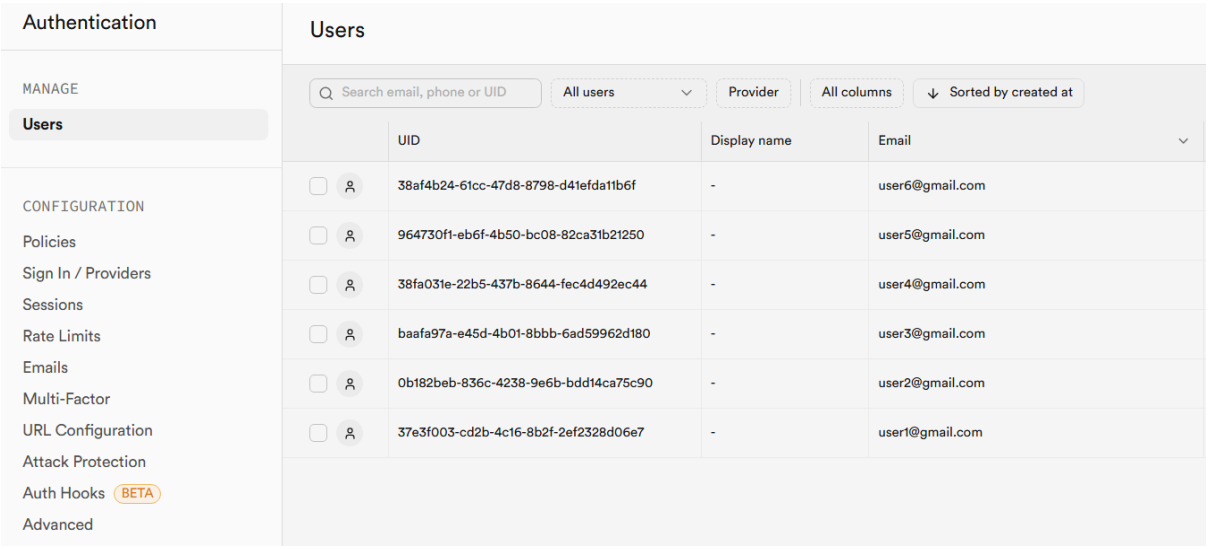
Stores messages exchanged in the group chat associated with each event.

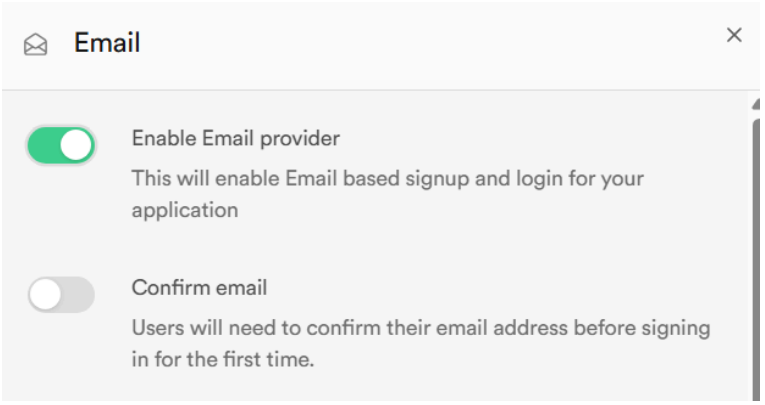
- id: UUID (primary key).
- event_id: reference to the event this message belongs to.
- sender_id: reference to profiles.
- message: text content of the message.
- created_at: timestamp.

id uuid	event_id uuid	sender_id uuid	message text	created_at timestamptz
4c0c67a0-d0a8-4362-a241-3a6385b6ab4	641bf95b-7df8-44d5-b5e5-86b113...	38fa031e-22b5-437b-8644-fec4d...	Ciao a tutti ragazzi	2025-07-06 12:24:14.684599+0
6e59c6c4-f997-4ea0-b2af-e1fa87f325c1	641bf95b-7df8-44d5-b5e5-86b113...	baafa97a-e45d-4b01-8bbb-6ad59...	hello hello	2025-07-06 12:53:32.599588+0
72289e5c-e4ec-44e6-8c2c-9877e8120791	b1f386bc-6063-48b6-a877-191516...	38fa031e-22b5-437b-8644-fec4d...	Buongiorno	2025-07-07 12:06:04.677161+00
ab38d061-ee5d-4046-9ac1-6cc6270f42a3	641bf95b-7df8-44d5-b5e5-86b113...	0b182beb-836c-4238-9e6b-bdd1...	Ciao	2025-07-07 11:40:51.097092+00
c9582acf-7dbe-4d52-b01a-5e2b389c7226	641bf95b-7df8-44d5-b5e5-86b113...	38fa031e-22b5-437b-8644-fec4d...	Mi leggete?	2025-07-06 12:24:24.961816+00
eece263c-39b7-4d3c-842e-fe61e298d1fd	641bf95b-7df8-44d5-b5e5-86b113...	37e3f003-cd2b-4c16-8b2f-2ef232...	Hello	2025-07-06 14:33:30.323935+0

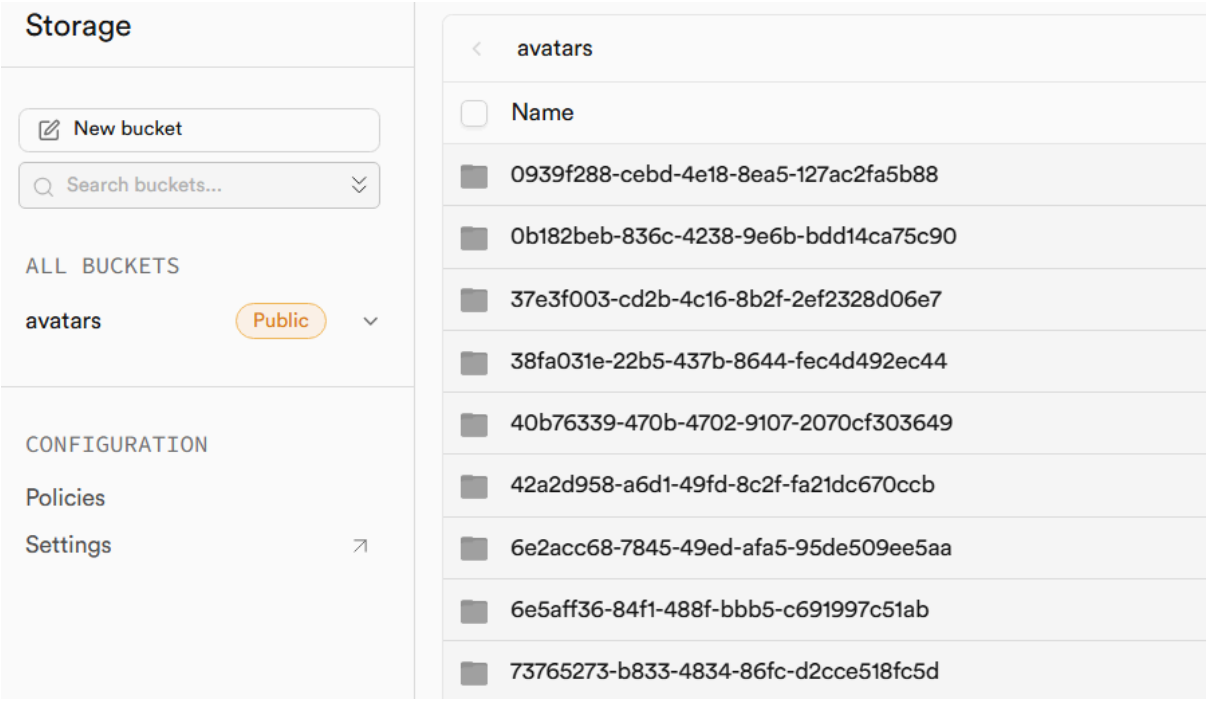
Authentication, Bucket and RLS

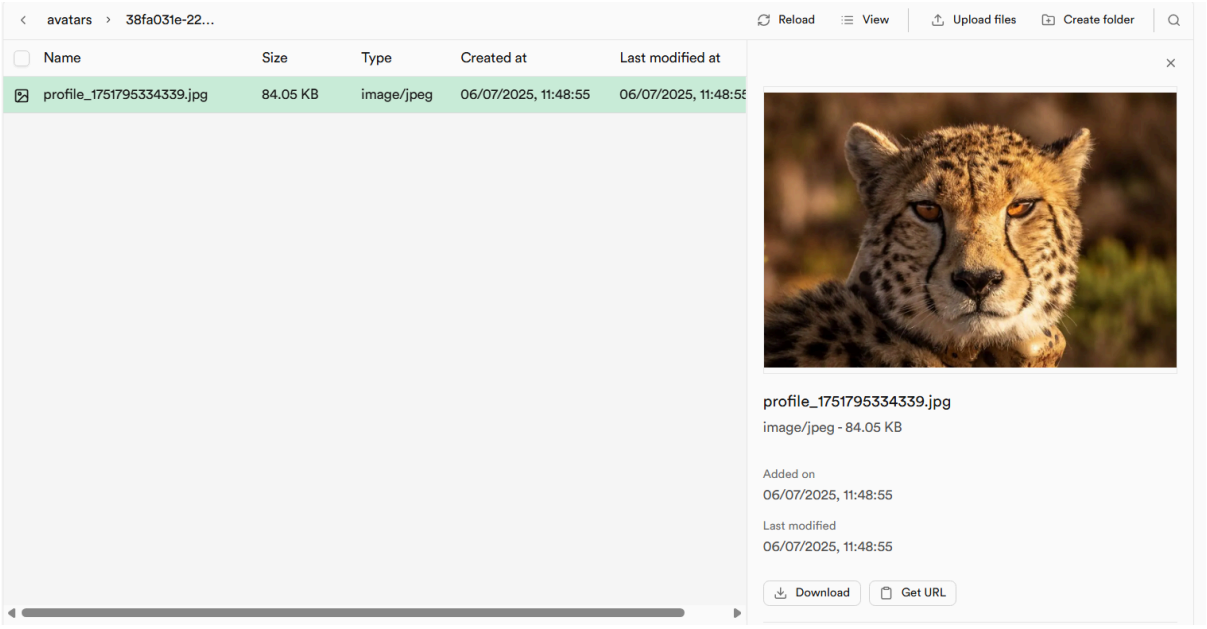
The authentication system for the application is implemented using Supabase Auth, employing a classic email and password mechanism. Each registered user is uniquely identified via a UUID, which serves as the primary foreign key across all related tables (e.g., profiles, events, event_participants, and event_messages). Users are considered authenticated only after successfully logging in, which introduces an additional layer of complexity when managing application state, particularly in areas like data fetching, conditional rendering, and access control. For simplicity, email verification has been deliberately disabled to streamline the onboarding process and avoid delays caused by email confirmation steps during development and testing.

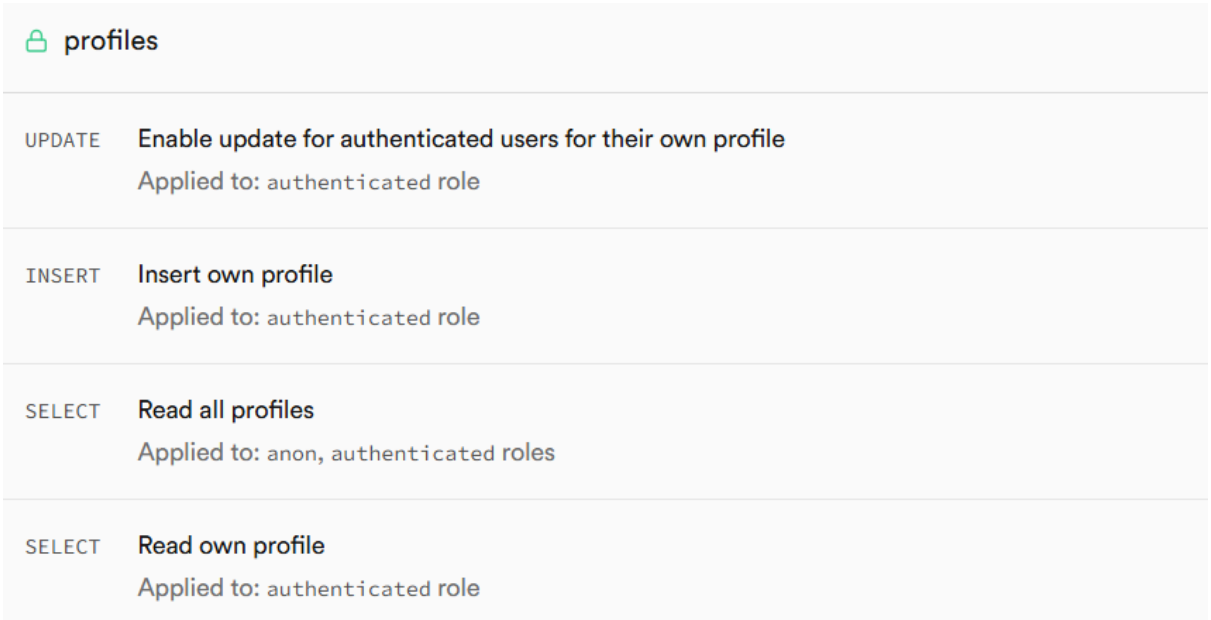




To handle media uploads, such as profile pictures, a Supabase Storage bucket named avatars has been configured. This bucket is structured with a folder hierarchy where each user's directory is named after their UUID. Within these directories, user-specific profile images are stored. The bucket is publicly accessible to allow seamless integration with image views on the client side, while still maintaining organizational structure and preventing naming conflicts.







On the security side, Row-Level Security (RLS) is enabled across all key tables to ensure access control. Custom RLS policies have been applied to manage CRUD permissions. For instance:

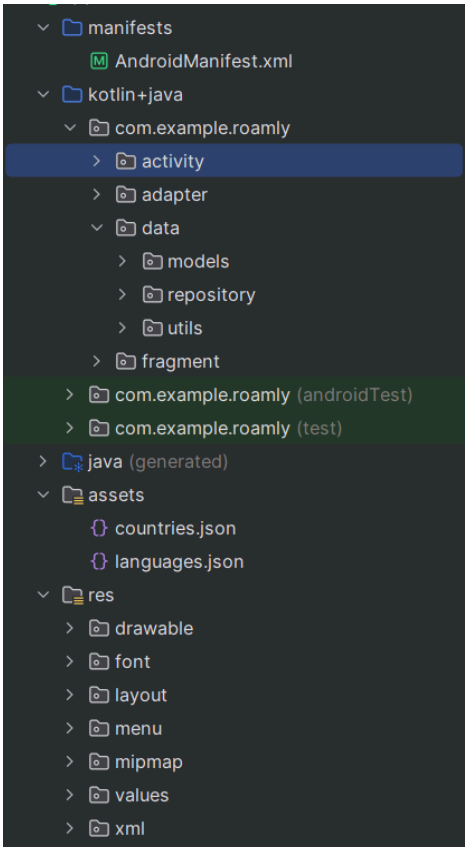
- In the event_participants table, users can only insert rows corresponding to their own participation and can delete records only if they are the participant or the event creator.

- The event_messages table enforces that only users who are participants in a given event can read or write messages related to that event.

- Tables such as profiles, profile_languages, and profile_interests have policies in place to restrict modifications to authenticated users and ensure that only users can access or edit their own data.

In certain cases more permissive policies were applied to reduce friction during development. However, the architecture is designed with full RLS coverage in mind, and policies can be progressively tightened as the system evolves toward production readiness. This layered structure of authentication, storage, and access control provides a secure foundation for managing sensitive user data and reinforces the scalability and maintainability of the application backend.

Project Structure



The project is organized according to modular and scalable principles, promoting separation of concerns across the app's components. Below is an explanation of the core packages and resources used.

activity/

This package contains all Activity classes, which serve as the entry points to major flows of the app.

Typical classes include:

- HomeActivity: Central screen that manages the map, markers, event/user logic, and bottom navigation.
- LoginActivity, SignupActivity: Handle authentication flow.
- MakeProfileActivity: Guides users through the profile creation process.

adapter/

This package holds all RecyclerView.Adapter classes and associated ViewHolders used for dynamic lists and dropdowns.

Examples:

- LanguageAdapter, InterestAdapter: For chip-based UI selections.
- EventChatListAdapter: For displaying a list of chat events.

Bind data to UI views efficiently, inflating layout resources and applying user interactions like clicks or removals. Adapters are tightly coupled with layout XML files (e.g., item_event_chat.xml).

data/

Structured into three logical sub-packages:

- models/: Kotlin data classes representing core entities like Profile, Event, Language, EventParticipant, etc.
- repository/: Responsible for data operations and interfacing with Supabase, such as ProfileRepository and EventRepository.
- utils/: Common utilities such as DateFormatter, LocationUtils, or DataCache.

The data layer is responsible for business logic, local caching, and communication with external services (e.g., Supabase). This abstraction helps decouple the UI from backend operations.

fragment/

Contains UI fragments that represent modular and reusable screens.

Examples:

- ProfileEditFragment: Lets users edit personal information.
- EventCreationFragment, EventEditFragment: Manage the creation or editing of events.
- EventChatFragment: Displays real-time group chat for a specific event.

Fragments encapsulate UI and related behavior for a specific screen or feature. They interact with repositories for data and adapters for rendering content.

assets/

Contains static JSON files bundled with the app.

Files:

- countries.json: Used to populate country selections during profile creation.
- languages.json: ISO-compliant list of supported languages.

Provides structured, static data to avoid hardcoding values. This makes the app easier to maintain and extend.

res/ (Resources)

A subset of Android's resource directory used to define UI appearance and reusable assets.

- drawable/: Custom icons, shapes, and image assets used across the app (e.g., profile icons, event type markers).
- layout/: XML layout files for fragments, adapters, tooltips, and screens. Structured consistently to separate presentation from logic.
- values/: Includes strings.xml, colors.xml, dimens.xml, and styles.xml for localization, theming, and consistency.
- font/: Custom font resources imported for branding or stylistic consistency.

Dependencies

```
implementation(libs.androidx.core.splashscreen)

// Gestirà automaticamente le versioni per i moduli Supabase
implementation(platform("io.github.jan-tennert.supabase:bom:3.1.4"))

// Moduli Supabase
implementation ("io.github.jan-tennert.supabase:auth-kt")
implementation ("io.github.jan-tennert.supabase:postgrest-kt")
implementation ("io.github.jan-tennert.supabase:storage-kt")

// Client HTTP esplicito
implementation("io.ktor:ktor-client-okhttp:3.2.0")

// Serialization JSON (necessaria per Supabase)
implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.6.3")

// Kotlin Coroutines
implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.7.3")

// Lifecycle per lifecycleScope
implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.6.2")

implementation("com.google.code.gson:gson:2.10.1")

implementation("com.mapbox.maps:android:11.13.1")

implementation ("com.google.android.gms:play-services-location:21.0.1")

implementation ("com.github.bumptech.glide:glide:4.15.1")
```

This project includes a set of external libraries to support Supabase integration, networking, JSON handling, concurrency, and essential Android services:

- Supabase SDK (v3.1.4)
  Integrates Supabase modules for authentication (auth-kt), database access (postgrest-kt), and file storage (storage-kt). A BOM (Bill of Materials) is used to manage consistent versions across all modules.

- Ktor HTTP Client (v3.2.0)
  HTTP client (ktor-client-okhttp) used for making explicit network requests,

compatible with Supabase operations.

- Kotlinx Serialization (v1.6.3)
  Provides JSON (de)serialization support required for interacting with Supabase's RESTful APIs.

- Kotlin Coroutines (v1.7.3)
  Used for asynchronous programming and concurrency, particularly to interact with external services such as Supabase in a lifecycle-safe and non-blocking way.

- AndroidX Lifecycle (v2.6.2)
  Enables lifecycle-aware coroutine scopes (lifecycleScope), ensuring that async operations are correctly managed within activity or fragment lifecycles.

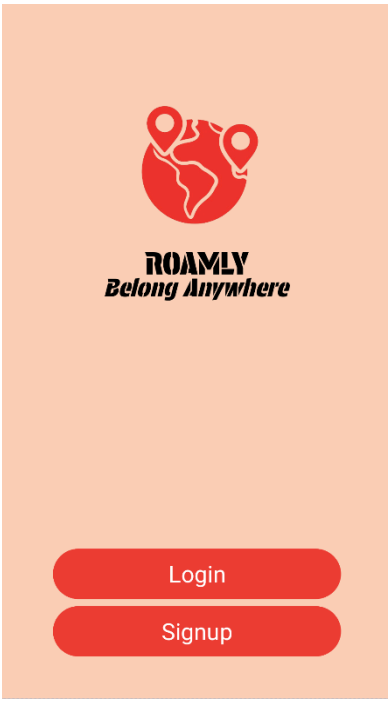
- Gson (v2.10.1)
  Primarily used for parsing and managing static or predefined JSON data, such as lists of supported languages and countries, where data models may not require @Serializable.

- Mapbox SDK (v11.13.1)
  Powers the map view and geospatial interactions, including custom markers and user/event location visualization.

- Google Play Services Location (v21.0.1)
  Provides access to the device's current location with high accuracy, necessary for location-aware features.

- Glide (v4.15.1)
  Efficient image loading and caching library used for displaying profile images, which are stored and retrieved from Supabase Storage buckets.

OnboardingActivity



This is the initial screen of the application. Its goal is to introduce the user to the Roamly app, displaying the app name and slogan, and offering two primary actions: log into an existing account (Login) or create a new one (Signup).

The interface includes:

- A centered logo at the top (ImageView), sized 150x150dp.
- The app name "ROAMLY" directly below the logo, styled with a custom font.
- The app's slogan "Belong Anywhere" in italic style, placed below the name.
- Two large red buttons (Login and Signup), centered and vertically stacked inside a LinearLayout.

Simplified XML structure of *activity_onboarding.xml*:

```
<ConstraintLayout>
    <ImageView android:id="@+id/logo" ... />
    <TextView android:id="@+id/Roamly" text="ROAMLY" ... />
    <TextView android:id="@+id/Punchline" text="Belong Anywhere" ... />
```

```
    <LinearLayout android:orientation="vertical">
        <Button android:id="@+id/btnLogin" text="Login" ... />
        <Button android:id="@+id/btnRegister" text="Signup" ... />
    </LinearLayout>
</ConstraintLayout>
```

Technical details:

- The root layout is a ConstraintLayout, ensuring responsive design across screen sizes.
- Background color: @color/roamly_blush.
- Buttons use the red accent color @color/roamly_red and a variable font called roboto_variablefont.
- The logo is loaded from @drawable/logo_no_background.
- Constraints and margins are used to ensure consistent centering of all elements.

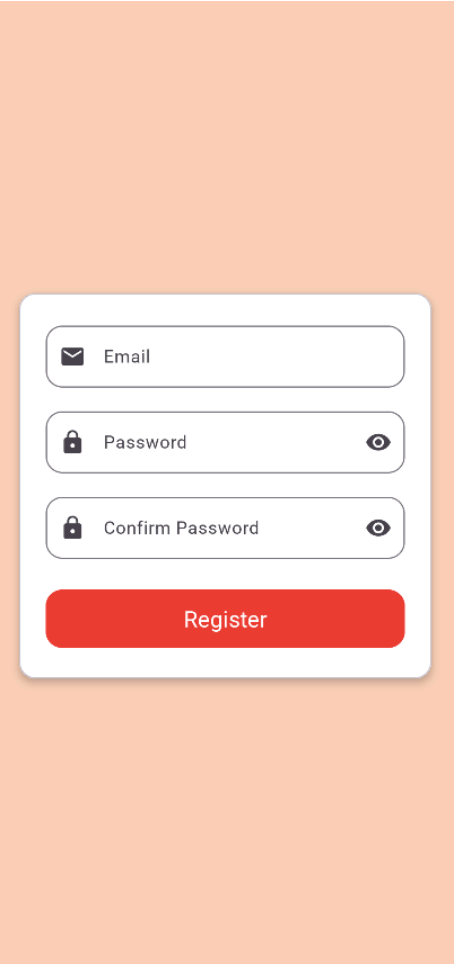
Logic and Navigation, Kotlin Class: OnboardingActivity.kt

- Sets the screen layout using activity_onboarding.xml.
- Adapts the layout to the edges of the screen (to support full-screen mode).
- Gets the two buttons from the layout: Login and Signup.
- When the Login button is clicked, it opens LoginActivity and closes the current screen.
- When the Signup button is clicked, it opens SignupActivity and closes the current screen.

```
btnLogin.setOnClickListener {
    startActivity(Intent(this, LoginActivity::class.java))
    finish()
}
```

```
btnRegister.setOnClickListener {
    startActivity(Intent(this, SignupActivity::class.java))
    finish()
}
```

SignupActivity



This screen manages the creation of a new user account via email and password. It presents a clean, accessible interface that adheres to Material Design standards and integrates seamlessly with Supabase authentication services.

Interface Overview

The screen is built around a visually centered MaterialCardView which contains the entire signup form.

The layout includes:

- A centered MaterialCardView with rounded corners and light elevation to visually separate the form from the background.
- A vertical LinearLayout inside the card that includes:
    - An email input field
    - A password input field
    - A password confirmation input field
- All inputs are wrapped in TextInputLayout components from the Material library, offering:
    - Built-in label animations
    - Icons (e.g., email, lock)
    - Password visibility toggle
- A bottom-aligned MaterialButton styled in the app's signature red for the “Register” action.

Simplified XML Structure of activity_signup.xml

```
<ConstraintLayout>
  <MaterialCardView>
    <LinearLayout orientation="vertical">
      <TextInputLayout>
        <TextInputEditText android:id="@+id/registerEmail" />
      </TextInputLayout>
      <TextInputLayout>
        <TextInputEditText android:id="@+id/registerPassword" />
      </TextInputLayout>
      <TextInputLayout>
```

```
            <TextInputEditText android:id="@+id/registerConfirmPassword" />

        </TextInputLayout>

        <MaterialButton android:id="@+id/btnRegister" text="Register" />

    </LinearLayout>

  </MaterialCardView>

</ConstraintLayout>
```

Technical Details

- The root layout is a ConstraintLayout, ensuring full responsiveness across screen sizes.
- The form is encapsulated in a MaterialCardView with cardCornerRadius and cardElevation attributes for a polished look.
- All input fields use TextInputLayout and TextInputEditText from the Material Components library, which was chosen for its flexibility and aesthetic consistency. Material components offer high customization with minimal configuration and handle accessibility, elevation, and theming out of the box.
- Font used: roboto_variablefont.
- The registration button is styled with the primary app color @color/roamly_red.

Logic and Navigation (Kotlin Class: SignupActivity.kt)

The logic is handled in the SignupActivity class, which includes both input validation and communication with Supabase Auth.

All asynchronous operations are executed within the lifecycleScope.launch coroutine block. This is necessary because Supabase operations require suspending functions, and lifecycleScope:

- Automatically respects the lifecycle of the Activity
- Ensures that ongoing coroutines are cancelled when the Activity is destroyed
- Avoids memory leaks and manual coroutine management

Signup Flow

- Input fields are validated:
    - Email format is verified using Patterns.EMAIL_ADDRESS
    - Password must be at least 6 characters
    - Password confirmation must match
- If valid, Supabase Auth is used with the Email provider:

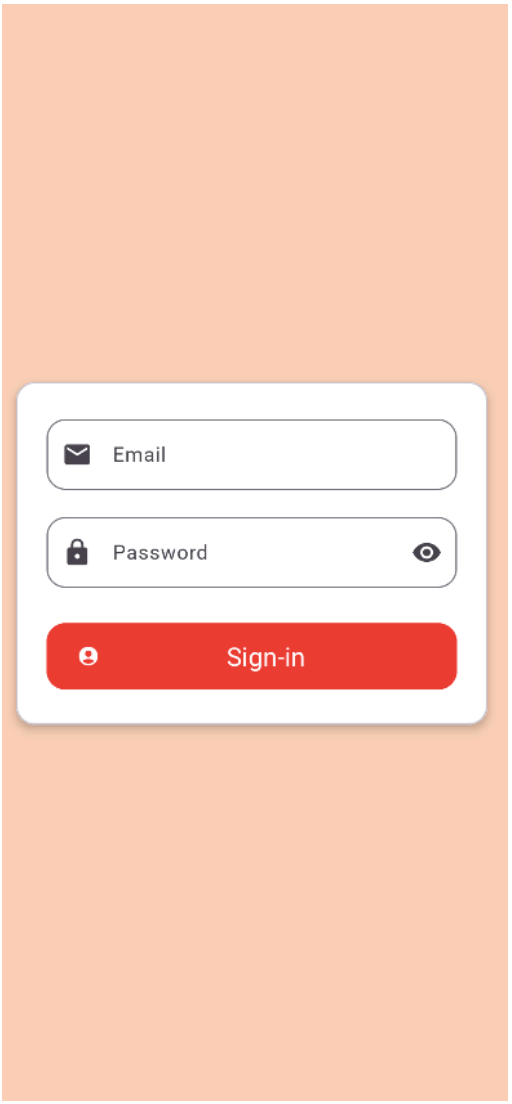
```
val session = SupabaseClientProvider.auth.signUpWith(Email) {
   this.email = email
   this.password = password
}
```

- If the user is already authenticated, the app signs them out before proceeding with the registration.
- On success:
    - A Toast confirms completion.
    - Navigation transitions to LoginActivity.

Additional Notes

- Email verification has been disabled for simplicity during the MVP phase and user testing.
- All input error messages are inline and managed automatically by TextInputLayout.
- Exception handling is implemented using try/catch with full logs and user feedback via Toast.

LoginActivity



This screen is responsible for user authentication and grants access to the application's core features. It is implemented using Material Design components to ensure a modern and intuitive user experience.

Interface Overview

The interface is structured around a MaterialCardView centered vertically and horizontally, visually isolating the login form from the background.

The layout includes:

- An email input field with a leading icon (ic_email)
- A password input field with a lock icon and toggle visibility

- A red login button labeled “Sign-in”, enhanced with a user icon and styled for visual clarity and prominence

Simplified XML Structure of activity_login.xml

```
<ConstraintLayout>
  <MaterialCardView>
    <LinearLayout orientation="vertical">
      <TextInputLayout>
        <TextInputEditText android:id="@+id/loginEmail" />
      </TextInputLayout>
      <TextInputLayout>
        <TextInputEditText android:id="@+id/loginPassword" />
      </TextInputLayout>
      <MaterialButton android:id="@+id/loginBtn" text="Sign-in" />
    </LinearLayout>
  </MaterialCardView>
</ConstraintLayout>
```

Technical Details

- Root container: ConstraintLayout, responsive and flexible for different device sizes
- Input fields are based on TextInputLayout and TextInputEditText from Material Components, chosen for:
    - Visual consistency and accessibility
    - Built-in icon support
    - Aesthetic customization with minimal code

- The MaterialButton uses the app's red accent color (@color/roamly_red) and includes an icon (ic_user) for visual context

- The form is embedded in a MaterialCardView with rounded corners (15dp) and light elevation (4dp) to enhance readability and focus

Logic and Navigation (Kotlin Class: LoginActivity.kt)

The LoginActivity class handles validation, Supabase-based login, user profile check, and navigation flow after login.

- All operations are wrapped in lifecycleScope.launch {} to ensure asynchronous execution tied to the lifecycle of the activity

- This approach eliminates manual coroutine management

Login Flow

1. Validation
    - Email must match Patterns.EMAIL_ADDRESS
    - Password field must not be empty
2. Authentication

Credentials are passed to Supabase Auth using the Email provider:

```
SupabaseClientProvider.auth.signInWith(Email) {
  this.email = email
  this.password = password
}
```

3. Post-Login Behavior
    - The current user's ID is fetched via currentUserOrNull()?.id
    - The app attempts to retrieve the corresponding row from the profiles table
    - If no profile is found:

        - A new Profile object is created with default values and inserted into Supabase
        - The user is redirected to MakeProfile1Activity to complete onboarding
    - If a profile is found:
        - If has_logged_before is false, onboarding continues

Otherwise, the user is redirected to HomeActivity with a flag indicating a fresh login:

putExtra("is_fresh_login", true)

4. Error Handling
    - Any error in the login process is caught and shown with a Toast

Additional Notes

- Email verification is disabled for simplicity in this MVP version, streamlining the onboarding and testing process
- The login process is tightly integrated with profile management, ensuring that every authenticated user has a corresponding database record in the profiles table

MakeProfile1Activity

This screen represents the first step of the user onboarding process, allowing users to define the basics of their profile. It collects profile image, full name, age, country of origin, and category, all styled using Material Design for consistency and clarity.

Interface Overview

The layout includes:

- A screen title ("Tell us something about you"), centered at the top.
- A clickable circular profile image placeholder that opens the image picker.
- A full name input field, using TextInputLayout.
- An age slider (18–99) with real-time value preview.
- A dropdown for country selection, populated from local JSON.
- A dropdown for selecting user category, with icons.
- A "Next" button styled with the app's red accent and rounded corners.

Simplified XML Structure of activity_make_profile.xml

```
<ConstraintLayout>
    <TextView android:id="@+id/titleText" />
    <MaterialCardView>
        <ScrollView>
            <LinearLayout
                android:orientation="vertical">
                <ShapeableImageView android:id="@+id/profileImageView" />
                <TextInputLayout>
                    <TextInputEditText android:id="@+id/nameField" />
                </TextInputLayout>
                <TextView android:id="@+id/ageValueText" />
                <Slider android:id="@+id/ageSlider" />

                <TextInputLayout android:id="@+id/countryDropdownLayout">
                    <MaterialAutoCompleteTextView android:id="@+id/countryDropdown" />
                </TextInputLayout>

                <TextInputLayout android:id="@+id/categoryDropdownLayout">
                 <MaterialAutoCompleteTextView android:id="@+id/categoryDropdown" />
                </TextInputLayout>

                <MaterialButton android:id="@+id/nextButton" text="Next" />
            </LinearLayout>
        </ScrollView>
    </MaterialCardView>
```

</ConstraintLayout>

- Root container: ConstraintLayout, responsive for all screen sizes.
- Form container: MaterialCardView with 15dp rounded corners and 4dp elevation.
- ImageView: ShapeableImageView, clickable, uses circular overlay and border.
- Input fields: All based on Material Components (TextInputLayout, Slider, AutoCompleteTextView) for styling, accessibility, and built-in icon support.
- Button: MaterialButton with red background (@color/roamly_red) and 15dp corner radius.

Logic and Navigation (MakeProfile1Activity.kt)

The MakeProfile1Activity class handles:

- Image selection and upload to Supabase Storage (bucket avatars).
- Dynamic population of country dropdown from local assets using CountryProvider.
- Category dropdown with icons, using a custom CategoryAdapter.
- Input handling and validation for profile fields.
- Supabase profile update, followed by navigation to MakeProfile2Activity.

Profile Creation Flow

1. Image Selection & Upload

- Uses ActivityResultContracts.GetContent() to open the image picker.
- Automatically uploads the image to Supabase using a custom path:
  user_id/profile_timestamp.jpg
- Optionally decodes the JWT to verify expiration before upload (for debugging).

2. Field Validation

- Full name must not be blank.
- Country and category must be selected.
- Full name is split into first_name and last_name automatically.

3. Supabase Update

A Profile object is built with the collected values and uploaded image URL:

```
SupabaseClientProvider.db.from("profiles").update(updatedProfileData) {
    eq("id", userId)
}
```

- On success user is redirected to MakeProfile2Activity.
- On failure error is logged and shown with a Toast.

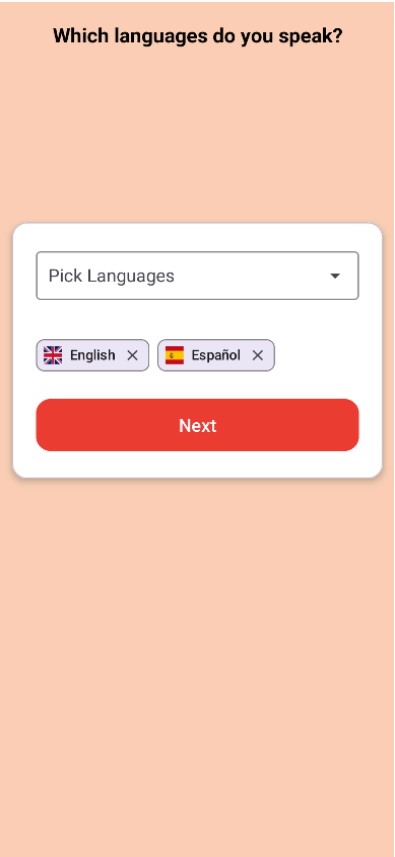
Error Handling

- Image upload failure "Upload error" toast shown to the user.
- Missing inputs Toast warnings shown for required fields.
- Database error Error logged with full context and user-facing message.

Additional Notes

- Image upload uses upsert = true to allow overwriting files with same path.
- JWT parsing is used only for debugging the session and expiration.
- The country list is loaded once from local assets (JSON).
- Category dropdown defaults to the first entry if no selection is made.

MakeProfile2Activity



This screen represents the second step of the user onboarding process, allowing users to select the languages they speak. It provides a searchable dropdown list of supported languages (with flags) and lets users add multiple entries, shown as removable chips. The selected languages are saved to Supabase and the user proceeds to the next onboarding screen.

The layout includes:

- A screen title ("Which languages do you speak?"), centered at the top.
- A MaterialCardView container with vertical scroll support.
- A dropdown (MaterialAutoCompleteTextView) to pick languages from a filtered list.
- A ChipGroup to display selected languages with flags and close icons.
- A “Next” button styled with the app's red accent and rounded corners.

Simplified XML Structure of activity_make_profile2.xml

```
<ConstraintLayout>
   <TextView android:id="@+id/titleText2" />
   <MaterialCardView>
      <ScrollView>
         <LinearLayout
            android:orientation="vertical">
            <TextInputLayout android:id="@+id/languagesDropdownLayout">
              <MaterialAutoCompleteTextView android:id="@+id/languagesDrpdwn" />
            </TextInputLayout>
            <ChipGroup android:id="@+id/selectedLanguagesChipGroup" />
            <MaterialButton android:id="@+id/nextButton2" text="Next" />
         </LinearLayout>
      </ScrollView>
   </MaterialCardView>
</ConstraintLayout>
```

- Root container: ConstraintLayout, full-screen and responsive.
- Form container: MaterialCardView with 15dp rounded corners and 4dp elevation.
- Dropdown field: TextInputLayout wrapping a MaterialAutoCompleteTextView, non-editable and styled.
- ChipGroup: Displays each selected language as a chip with icon and close functionality.
- Button: MaterialButton with @color/roamly_red and 15dp corner radius.

Logic and Navigation (MakeProfile2Activity.kt)

The MakeProfile2Activity class handles:

- Loading all available languages from languages.json via LanguageProvider.
- Displaying languages in a dropdown with a custom LanguageAdapter that shows flags.
- Adding languages as chips upon selection, and allowing removal with close icons.
- Saving selected languages to Supabase (profile_languages table), replacing any previously saved entries.
- Proceeding to MakeProfile3Activity upon successful save.

Language Selection Flow

1. Dropdown Initialization

   Loads all supported languages from assets. The list is filtered dynamically to exclude already-selected entries. Each item displays the language name and corresponding flag.

2. Chip Creation & Management

   When the user selects a language, it is added to selectedLanguages, and a new chip is created in the ChipGroup. Chips include:

   - Flag icon (via getFlagResId)
   - Language name
   - Close icon for removal

3. Supabase Update

   On pressing “Next”:

   - Clears existing profile_languages entries for the user.
   - Inserts all currently selected languages (LanguageLink instances).
   - Shows a success toast.
   - Redirects to MakeProfile3Activity.

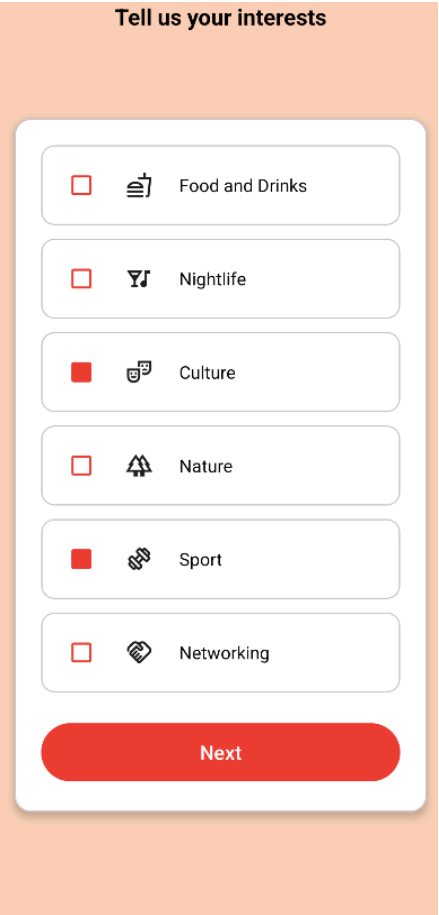
Error Handling

- User not authenticated: Shows a toast and blocks save.
- Supabase failure: Error logged and shown to the user.
- Missing flags or JSON issues: Logged with warnings, skipped silently.

Additional Notes

- The flag icons are dynamically loaded using the ISO language–country mapping (languageCodeToCountryCodeMap).
- The dropdown list updates in real time as selections change, to prevent duplicates.
- The adapter (LanguageAdapter) uses a ViewHolder pattern for performance and includes fallback for missing flags.
- The layout is scrollable to ensure compatibility with small screens.
- Language selection is optional (but encouraged), no hard validation is enforced.

MakeProfile3Activity



This screen represents the third step of the user onboarding process, allowing users to define their personal interests. It displays a scrollable list of selectable interests, each with an icon, label, and checkbox. The selected interests are saved to Supabase and associated with the user's profile before continuing to the next step.

The layout includes:

- A screen title ("Tell us your interests") centered at the top.
- A scrollable MaterialCardView containing all available interest items.
- Each interest is shown as:
    - A horizontal row with:
        - A colored checkbox (MaterialCheckBox)

        - An icon representing the interest (e.g., food, nightlife, culture, etc.)
        - A label with the interest name
- A red "Next" button styled with rounded corners to proceed.

Simplified XML Structure of activity_make_profile3.xml

```
<ConstraintLayout>
    <TextView android:id="@+id/titleText3" />
    <MaterialCardView android:id="@+id/profileCard3">
        <ScrollView>
            <LinearLayout
                android:orientation="vertical">
                <!-- Repeated for each interest -->
                <MaterialCardView>
                    <LinearLayout android:orientation="horizontal">
                        <MaterialCheckBox android:id="@+id/interestFood" />
                        <ImageView android:src="@drawable/ic_food" />
                        <TextView android:text="Food and Drinks" />
                    </LinearLayout>
                </MaterialCardView>
                <MaterialButton android:id="@+id/nextButton3" text="Next" />
            </LinearLayout>
        </ScrollView>
    </MaterialCardView>
</ConstraintLayout>
```

- Root container: ConstraintLayout, responsive to screen sizes.
- Form container: MaterialCardView with 15dp rounded corners and light elevation.
- Each interest: Displayed in its own MaterialCardView row with visual separation.
- Checkboxes: Styled with @color/roamly_red for visual consistency.
- Icons: Custom vector drawables representing each category.
- Button: MaterialButton styled with red background and 30dp corner radius.

Logic and Navigation (MakeProfile3Activity.kt)

The MakeProfile3Activity class handles:

- Dynamic interest loading from Supabase (interests table).
- Mapping interest names to IDs for later saving.
- Handling checkboxes for 6 pre-defined interests.
- Saving selected interests to Supabase in profile_interests table.
- Navigation to MakeProfile4Activity on successful save.

Interest Selection Flow

1. Load All Interests
    - At startup, all available interests are loaded from Supabase (interests table).

The result is used to populate a local map:

interestNameToId["Food and Drinks"] = "uuid-123..."

2. User Selects Interests
    - User can check any of the six interests:

        - Food and Drinks
        - Nightlife
        - Culture
        - Nature
        - Sport
        - Networking
    - Each one has a corresponding checkbox and icon.
3. Save to Supabase
    - When “Next” is pressed:
        - Authenticated user ID is retrieved from Supabase.
        - The list of checked checkboxes is collected.
        - The interest names are mapped to IDs.
        - A list of profile_id + interest_id entries is built and inserted into profile_interests.
4. On Success
    - A success toast is shown: "Interessi salvati!".
    - The user is redirected to MakeProfile4Activity.

Additional Notes

- Interest list is currently hardcoded in layout, not generated dynamically (but mapping IDs is dynamic).
- Icons like ic_food, ic_nightlife, etc., are used for each interest.
- All logs (debug, warning, error) are prefixed clearly (SupabaseLoad, SupabaseSave) for filtering in Logcat.

- Interest data is inserted in bulk using .insert(entriesToInsert) to improve efficiency.

MakeProfile4Activity

MakeProfile4Activity

This screen represents the final step of the user onboarding process, allowing users to define two key attributes of their profile: their social vibe ("Party" or "Chill") and their visibility on the map. These preferences are saved to Supabase, and upon completion the user is redirected to the HomeActivity.

The layout includes:

- A centered screen title ("Completa il tuo profilo") at the top.
- Two separate MaterialCardView blocks:
    1. Vibe Selector: Allows users to pick between a “Party” or “Chill” mood using a MaterialSwitch.
    2. Visibility Selector: Allows users to toggle their visibility on the map (Yes/No), also using a MaterialSwitch.
- A red "Jump in!" button at the bottom, which finalizes the onboarding.

Each switch is accompanied by two TextView labels. When toggled, the label colors update dynamically to indicate the current state.

Simplified XML Structure of activity_make_profile4.xml

```
<ConstraintLayout>
    <TextView android:id="@+id/titleText4" />
    <LinearLayout android:id="@+id/cardContainer">
        <MaterialCardView android:id="@+id/vibeCard">
            <LinearLayout>
                <TextView android:id="@+id/partyText" />
                <MaterialSwitch android:id="@+id/vibeSwitch" />
                <TextView android:id="@+id/chillText" />
            </LinearLayout>
        </MaterialCardView>
        <MaterialCardView android:id="@+id/visibilityCard">
            <LinearLayout>
                <TextView android:id="@+id/noText" />
                <MaterialSwitch android:id="@+id/visibilitySwitch" />
                <TextView android:id="@+id/yesText" />
            </LinearLayout>
        </MaterialCardView>
    </LinearLayout>
    <MaterialButton android:id="@+id/jumpInButton" text="Jump in!" />
</ConstraintLayout>
```

- The layout is responsive built with ConstraintLayout and nested LinearLayout containers for clear visual grouping.
- Each selector card has rounded corners and padding for better touch ergonomics.
- Switches use custom colors defined in the app's palette.
- The final action button spans full width with 30dp corner radius.

Logic and Navigation (MakeProfile4Activity.kt)

The MakeProfile4Activity class handles:

- Initializing all interactive UI components (switches, labels, button).
- Updating text color dynamically based on switch state.
- Saving the selected vibe and visibility into the user's profile.
- Redirecting to HomeActivity after a successful update.

Vibe and Visibility Flow

1. User Interaction
    - The Vibe Switch controls the selection between:
        - "Party" (when unchecked)
        - "Chill" (when checked)
    - The Visibility Switch allows users to define if they want to appear on the map:
        - "No" if unchecked
        - "Yes" if checked
    - For both sections, the corresponding labels are highlighted in black when active, and dimmed to gray when inactive.
2. Saving to Supabase

    - On pressing the "Jump in!" button:
        - The app checks if the user is authenticated.
        - It queries the profiles table from Supabase to get the current profile.
        - It updates the vibe, visible, and has_logged_before fields.
        - The updated profile is pushed to Supabase using the .update() method.
        - A confirmation message is shown, and the user is taken to HomeActivity.

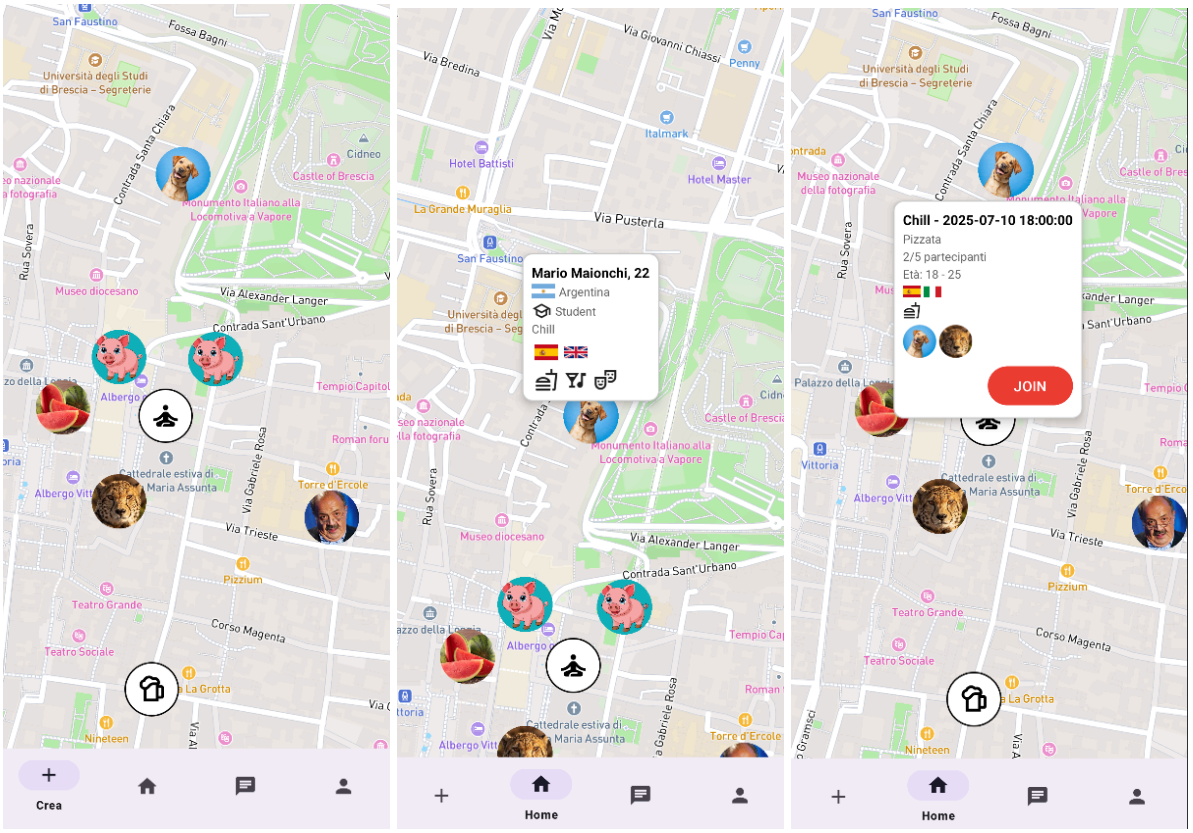
Error Handling

- If the user is not authenticated, a toast message informs them: "Utente non autenticato. Riprova il login."
- If the user's profile cannot be found in the database, a message is shown: "Profilo non trovato!"
- If the Supabase update fails for any reason (e.g., network issue, backend error), the exception is logged and the user is shown a toast with the message: "Errore aggiornamento profilo" followed by the error detail.

Additional Notes

- The field vibe is saved as a string, matching "Party" or "Chill" based on the switch state.
- The visibility status is stored as a boolean (true or false) under the field visible.
- has_logged_before is set to true to mark the completion of the onboarding flow.
- The MaterialSwitch and label pairing offers a clear and minimal UI for setting preferences.
- This screen is the last step in the profile creation flow before entering the main application.

HomeActivity



The HomeActivity serves as the central screen of the Roamly application. It integrates a full-screen interactive Mapbox map where users can view nearby user profiles and active events, interact with detailed tooltips, create new events via drag-and-drop, and navigate across the main sections of the app (Home, Profile, Chat) through a bottom navigation menu.

Simplified XML Structure of activity_home.xml

<FrameLayout>

<MapView android:id="@+id/mapView" />

<FrameLayout android:id="@+id/tooltipContainer" />

<FrameLayout "@+id/profileFragmentContainer" android:visibility="gone" />

<FrameLayout "@+id/chatFragmentContainer" android:visibility="gone" />

<FrameLayout "@+id/eventFragmentContainer" android:visibility="gone" />

<BottomNavigationView android:id="@+id/bottomNavigationView" />

<ImageView android:id="@+id/draggableEventIcon" android:visibility="gone" />

</FrameLayout>

This layout uses a full-screen FrameLayout to stack the map, overlay fragments, tooltips, and UI components. The draggable event icon floats above all other views and becomes visible during user interaction.

Interface Behavior

- When first opened, the map automatically centers on the user's current location.
- If the user is not authenticated, they are redirected to the login screen.
- The user's current location is displayed with a circular marker that includes their profile image.
- Events and visible profiles are fetched and displayed as markers on the map.
- Clicking a marker triggers a camera animation (flyTo) and opens a detailed tooltip with event or profile information.
- Markers and tooltips are cleared when the map is manually moved or a marker is clicked twice.
- A special drag-and-drop interaction allows users to create events by dragging the event icon onto the map.

Core Functionality (HomeActivity.kt)

- Checks for user authentication.
- Starts GPS tracking using FusedLocationProviderClient and updates every 20 seconds.
- Updates the user's location in Supabase and maintains a personalized marker on the map.
- Loads global data (languages, countries, interests) once on startup.
- Fetches nearby visible profiles every 20 seconds and updates markers dynamically via UserAnnotationManager.
- Fetches active events every 30 seconds and manages markers via EventAnnotationManager.

- Manages tooltips using UserTooltipManager and EventTooltipManager, depending on marker type.

- Fragment containers (profileFragmentContainer, chatFragmentContainer, eventFragmentContainer) are dynamically shown or hidden depending on bottom navigation selection.

- Handles the drag-and-drop "Create Event" interaction by showing the event icon and instantiating the EventCreationFragment at the drop point.

Supabase Integration

- The user’s geolocation is saved in the locations table with their user_id.

- Profiles are fetched from the profiles table (only those with visible = true) and joined with location data.

- Events are retrieved from the events table and filtered by distance (<10 km) and expiration (visible_until field).

- Event and profile details are stored in a shared in-memory cache (DataCache) to reduce redundant network requests.

- Event tooltips allow actions like JOIN, LEAVE, EDIT, DELETE, depending on the current user's relationship with the event. These actions trigger appropriate calls to EventRepository.

Bottom Navigation Behavior

- nav_home: Hides all fragments, resets tooltips, recenters the map on the user’s location.

- nav_profile: Opens ProfileEditFragment inside the profile container.

- nav_chat: Shows EventChatListFragment inside the chat container.

- nav_create_event: Enables drag mode; a touch event shows a draggable event icon and opens the creation form when released.

Periodic Updates

- Every 20 seconds:

    - The user’s location is updated in Supabase.
    - Nearby visible profiles are fetched and their markers updated.
- Every 30 seconds:
    - Active events are synchronized and markers updated.
- All updates are performed via coroutine jobs and lifecycle-aware components to avoid memory leaks.

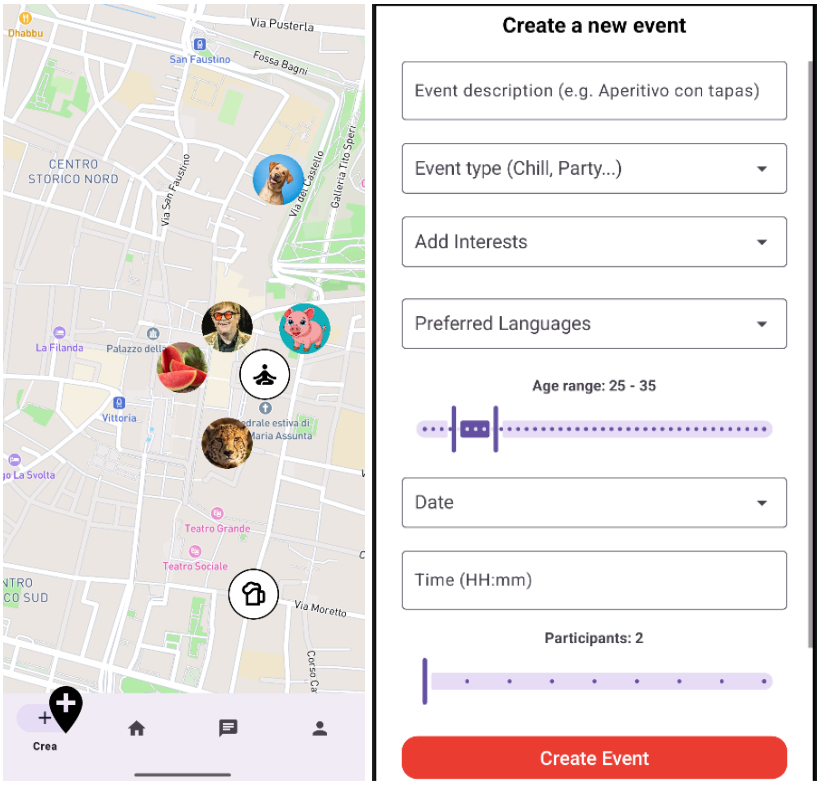
Error Handling

- If location services are disabled, a toast message is shown: "Enable location services", and the user is redirected to settings.
- If location permissions are denied, a message is shown: "Location permission denied".
- If user location is null, a fallback toast message appears: "Location unavailable".
- If Supabase operations fail (e.g., profile or event fetch), errors are logged and user feedback is provided.
- If the user is not logged in when onResume() is called, the entire map state is reset and the user is redirected.

Additional Notes

- Tooltip management uses shared logic with getCurrentShownEventId and getCurrentShownProfileId to avoid overlapping UI elements.
- Only one tooltip (event or profile) can be visible at any time.
- The resetMapState() method resets markers, tooltips, cache, and state variables (used during logout or re-authentication).
- All markers are updated or removed dynamically using distance filters and visibility expiration.
- The application avoids redundant fetches by caching user and event data between refresh cycles.

EventCreationFragment



The EventCreationFragment allows users to create a new event by selecting its type, date, time, age range, maximum number of participants, preferred languages, interests, and a textual description. Once confirmed, the event is validated and stored in the Supabase backend. A corresponding marker is then generated on the Mapbox map to represent the event location.

XML Layout (fragment_create_event.xml)

This screen is built on top of a ScrollView, which wraps a vertical LinearLayout. Inside, a series of Material components guide the user through all the required input fields for event creation. The layout includes:

- A TextInputEditText for the event description.
- Dropdowns (MaterialAutoCompleteTextView) for:
    - Event type ("Chill" or "Party")
    - Interests (populated via InterestRepository)
    - Preferred languages (loaded from app assets)

    - Date (today and tomorrow)

- ChipGroups for visualizing and removing selected interests and languages.
- A RangeSlider for age range selection, and a Slider for the number of participants.
- A non-editable time input (TextInputEditText) which opens a 24-hour MaterialTimePicker.
- A button to confirm event creation.
- Interests and languages are loaded asynchronously using a coroutine (lifecycleScope.launch).
- Once selected, each interest/language appears as a chip with icon and close functionality.
- The age and participants sliders immediately update their respective label text.
- The time input opens a dialog and formats the selected time into "HH:mm" format.
- The “Create Event” button becomes functional only after the user has filled all required fields.

Kotlin Logic (EventCreationFragment.kt)

onCreateView / onViewCreated

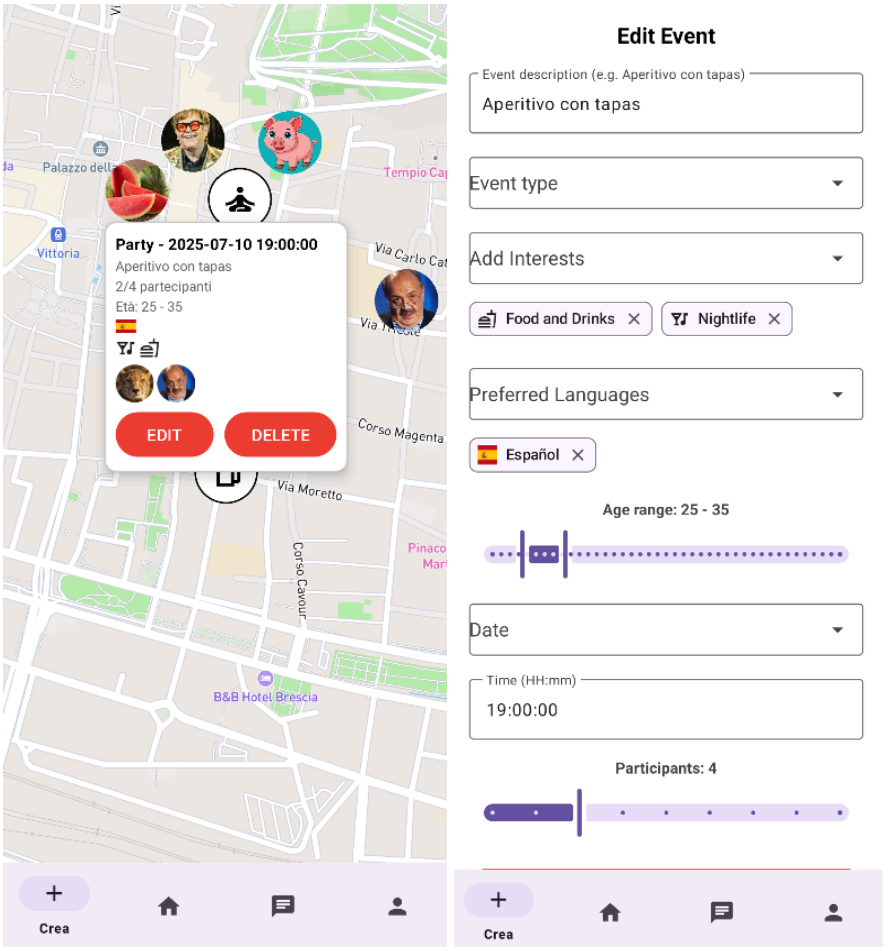
The fragment inflates its layout and sequentially initializes:

- bindViews: Connects each view to its corresponding variable.
- setupDropdowns: Loads event types, dates, languages, and interests.
- setupChipSelections: Manages chip addition and removal logic.
- setupSliders: Binds live value updates for age and participants.
- setupTimePicker: Configures the time selection dialog.
- setupCreateButton: Orchestrates the full event creation flow.

Event Creation Flow

1. Validates authentication using SupabaseClientProvider.auth.
2. Retrieves latitude and longitude from the fragment arguments.
3. Collects all user input values from the UI:
    - Type, date, time, min/max age, max participants
    - Selected interest IDs and language codes
    - Description
4. Constructs an Event object with a random UUID as ID.
5. Sends the event to Supabase via EventRepository.createEvent(...).
6. Once created, displays the new event on the map using EventAnnotationManager.createEventMarker(...), passing:
    - Context, MapView, MapboxMap, the created event and its coordinates
    - getCurrentShownEventId and onToggleEventCallout from the parent HomeActivity
7. Pops the back stack to return to the map screen.

EventEditFragment



The EventEditFragment allows a user to modify an existing event previously created on the map. The fragment preloads all event data into editable fields and enables the user to change the description, time, age range, maximum number of participants, interests, and languages. Once submitted, the changes are validated and updated on Supabase.

XML Layout Overview

The layout is built with a vertical ScrollView containing:

- A TextInputEditText for the event description.
- A non-editable dropdown for event type.
- A dropdown for selecting interests, with a corresponding ChipGroup showing selected items.
- A dropdown for selecting preferred languages, with a ChipGroup as well.

- A RangeSlider for selecting minimum and maximum participant age.
- A non-editable dropdown for the date.
- A time field (TextInputEditText) that opens a TimePickerDialog.
- A Slider for selecting the maximum number of participants.
- A button to confirm and save changes.

User Interface Behavior

- When the fragment is created, the fields are automatically filled with data from the Event object passed via the static eventToEdit field.
- Interest and language chips are rendered with icons and can be removed by tapping the close icon.
- Age and participants sliders update text labels live as the values change.
- The time field opens a 24-hour time picker dialog.
- The event type and date are not editable once the event is created.

Kotlin Logic (EventEditFragment.kt)

onViewCreated

- Initializes all view bindings.
- Sets up live update listeners for sliders.
- Displays a time picker dialog on time input click.
- Loads full language and interest lists using LanguageProvider and InterestRepository.
- Sets up dropdown adapters (LanguageAdapter, InterestAdapter) and chip logic.
- Populates all fields from the provided Event object using populateFields(...).
- Fetches the current number of participants from EventRepository.getEventParticipants(...) and adjusts the slider's minimum

accordingly.

- Handles save button logic via saveChanges().

saveChanges()

- Collects the current form data into a new Event object (using .copy() on the original).
- Sends the updated object to Supabase via EventRepository.updateEvent(...).
- If successful:
    - The map tooltip is cleared (tooltipContainer.removeAllViews()).
    - A toast confirms the update.
    - The fragment is closed (popBackStack()).
- On failure, an error toast is shown.

Supabase Integration

- Uses EventRepository.updateEvent() to persist the updated data.
- Participant count is dynamically validated to avoid setting a value below the current number of participants.
- Interest and language IDs are extracted from chip selections.
- Supabase is also queried to fetch the latest list of participants to enforce slider minimum constraints.

Supporting Utilities

- LanguageProvider: Loads all languages with localized names and dynamic flag resources.
- InterestRepository: Retrieves the full interest list from Supabase.
- LanguageAdapter and InterestAdapter: Used to display autocomplete options for selection.

- getInterestIcon(...): Returns the correct drawable resource for each interest category.

Error Handling

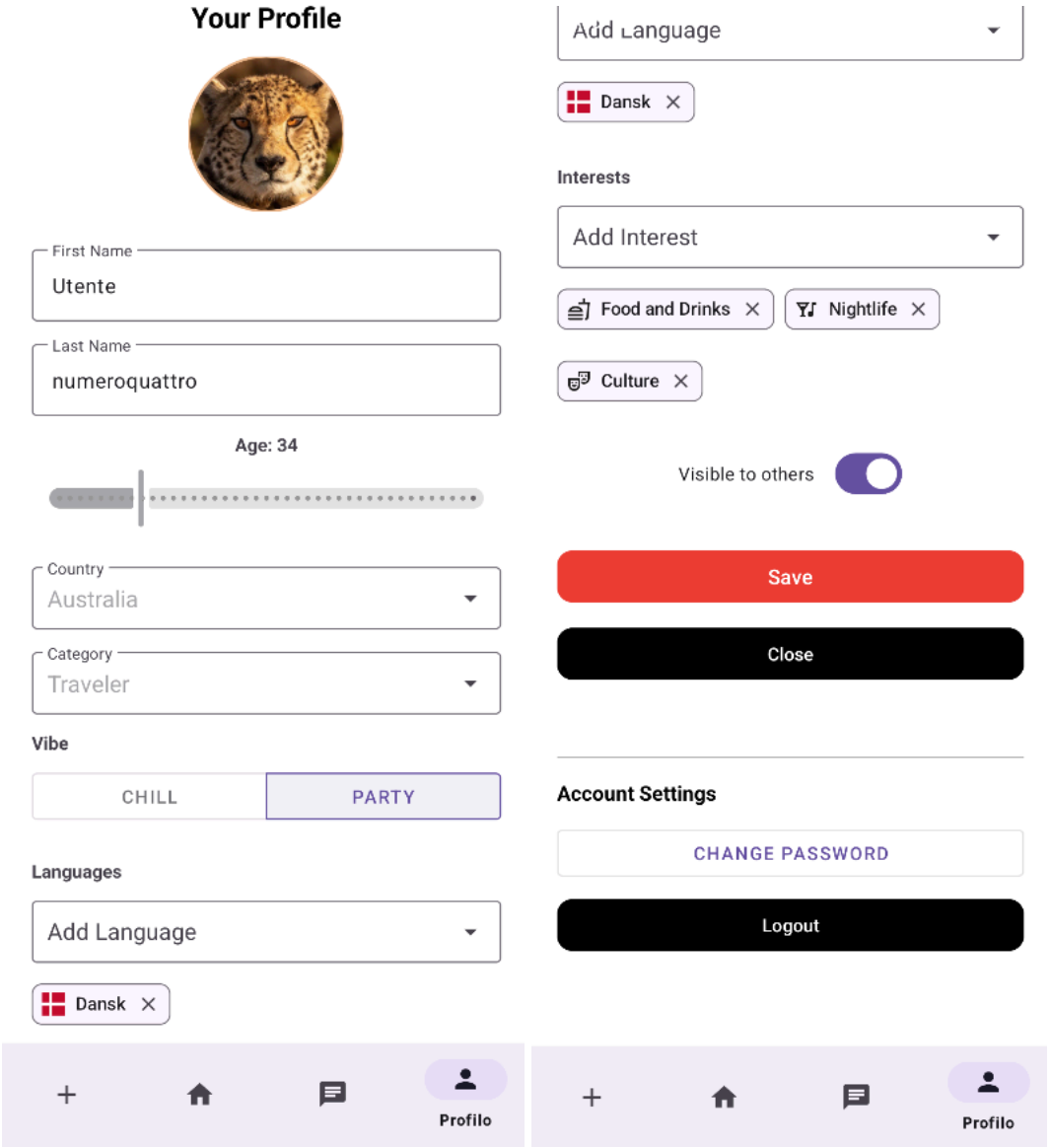
- If the eventToEdit is null, the user is notified and the fragment is immediately closed.

- If fetching languages or interests fails, a toast message is shown.

- All update operations are wrapped in coroutine blocks, and any exception triggers user feedback.

- UI prevents saving until required fields are filled.

ProfileEditFragment



The ProfileEditFragment enables users to modify their personal profile within the app. Users can update their first and last names, vibe (chill or party), interests, spoken languages, profile visibility, and profile picture. Age, country, and category fields are shown but not editable. Additionally, the fragment provides account management options, including logout and password change.

XML Layout Structure fragment_edit_profile.xml

The layout consists of a vertically scrollable interface built using NestedScrollView containing a centered LinearLayout with Material components.

Key elements include:

- A tappable profile picture preview using ShapeableImageView
- Editable fields for first name and last name
- A non-editable slider displaying the user's age
- Dropdowns for country and category (non-editable)
- A MaterialButtonToggleGroup to select vibe ("chill" or "party")
- Autocomplete dropdowns with chip selection for:
    - Spoken languages (using flags)
    - Personal interests (with icons)
- A MaterialSwitch for toggling profile visibility
- Buttons for saving changes, closing the fragment, changing the password, and logging out

User Interface Behavior

- When the fragment loads, all profile data is fetched from Supabase and displayed.
- Languages and interests can be added via dropdowns. Selected values are displayed as Chip components with a close icon for removal.
- Age is displayed using a slider but is not editable.
- The country and category fields are disabled and cannot be changed.
- The vibe is selected using a toggle button group.
- Profile visibility can be toggled with a switch.

Kotlin Logic (ProfileEditFragment.kt)

Fragment Initialization

On onViewCreated, the following steps are performed:

1. All views are bound using bindViews(view).

2. A click listener is added to the profile image to allow image selection and upload.

3. Language and interest dropdowns are populated using data from:

    - LanguageProvider.loadLanguagesFromAssets

    - InterestRepository.fetchAllInterests()

4. The user's current profile is loaded using ProfileRepo.getCompleteProfile.

5. The interface is populated with the fetched data using loadUserProfileAndPopulate().

On tapping "Save", the method saveProfileChanges() is called:

- Constructs a new Profile object by copying the existing one and updating:

    - First and last names

    - Vibe

    - Visibility status

- Collects selected interest IDs and language codes

- Calls ProfileRepository.saveCompleteProfile(...) to persist all changes to Supabase

- Displays a Toast indicating success or failure

If the user selects a new profile image:

- The image is uploaded to the avatars bucket in Supabase Storage

- The image URL is stored in the user's profile

- Glide reloads the profile image into the view

Account Management

- Logout:

    - Logs the user out via SupabaseClientProvider.auth.signOut()
    - Navigates back to OnboardingActivity
- Password Change:
    - Shows an AlertDialog with input fields for new and confirm password
    - Validates password match and length
    - Sends the update using SupabaseClientProvider.auth.updateUser

Supabase Integration

- Authentication:
    - Obtained via SupabaseClientProvider.auth.currentUserOrNull()
- Database:
    - Profile data and preferences are stored using ProfileRepository
    - Language links are fetched directly from profile_languages
- Storage:
    - Profile images are stored in the public avatars bucket

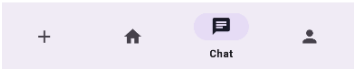
Supporting Classes

- ProfileRepository: Handles profile fetch and save operations
- InterestRepository: Retrieves available interests
- LanguageProvider: Loads all languages and provides flag resources

Error Handling

- Displays a Toast if profile image upload fails, or if saving the profile is unsuccessful
- Password change validates both matching input and minimum length
- Errors are caught and logged using Log

EventChatListFragment



The EventChatListFragment displays a list of all events that the currently authenticated user is participating in, where a group chat is available. When an item in the list is tapped, it opens the corresponding event chat screen via the EventChatFragment.

XML Layout Structure fragment_event_chat_list.xml

The layout consists of a simple vertical LinearLayout containing:

- A RecyclerView with ID recyclerViewChatList which fills the screen vertically
- A white background and top margin for spacing

UI Behavior

- The RecyclerView is initialized in onViewCreated with a vertical LinearLayoutManager.
- The EventChatListAdapter is set as the adapter and initialized with:
    - The current list of events

    - A click listener that opens the relevant chat screen when an event is selected

- The list of events is loaded asynchronously from Supabase based on the currently authenticated user's ID.

Kotlin Logic (EventChatListFragment.kt)

onCreateView

- Inflates the layout fragment_event_chat_list which contains the RecyclerView.

onViewCreated

- Finds and sets up the RecyclerView.
- Attaches the EventChatListAdapter with the eventList and a click handler.
- Calls loadUserEvents() to retrieve and display the list of relevant events.

loadUserEvents()

- Retrieves the current user ID via SupabaseClientProvider.auth.currentUserOrNull()?.id.
- Uses EventRepository.getEventsWithUserParticipation(currentUserId) to fetch the list of events the user is participating in.
- Clears and repopulates the local eventList with the fetched events.
- Notifies the adapter to refresh the displayed list.

openChat(eventId: String, eventDesc: String)

- Opens a new EventChatFragment using newInstance(...) passing the selected event's ID and description.
- Replaces the current fragment within the chatFragmentContainer.

- Adds the transaction to the back stack for navigational consistency.

Supabase Integration

- Authentication: The user's ID is retrieved from Supabase Auth using SupabaseClientProvider.auth.

- Event data is fetched using a repository method: EventRepository.getEventsWithUserParticipation(...).

- No writing operations are performed in this fragment; it is purely for data retrieval and navigation.

Supporting Components

- EventChatListAdapter: A RecyclerView.Adapter responsible for rendering individual event rows in the list. Each item includes the event's description and relevant metadata. Tapping an item triggers a callback to open the chat.

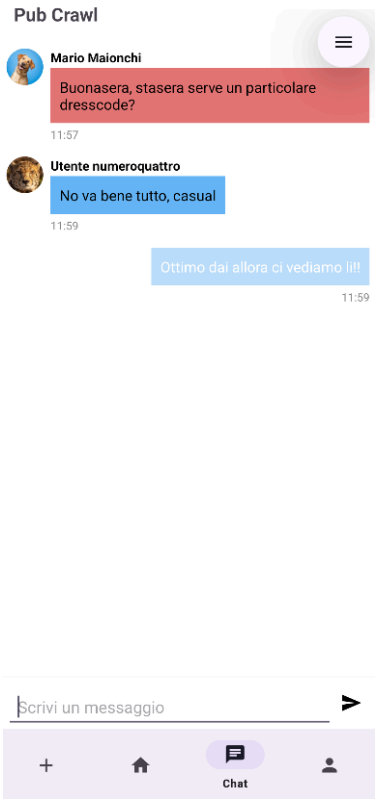
- EventRepository: Provides access to event-related queries including which events a given user is participating in.

Error Handling

- If the user is not authenticated, loadUserEvents() exits early without attempting any Supabase query.

- No additional error handling or user feedback (e.g., loading spinner, empty state) is implemented in this version.

EventChatFragment



The EventChatFragment displays the group chat associated with a specific event. It is opened when a user selects an event from the EventChatListFragment. The screen allows the user to view all messages exchanged within that event's chat and to send new messages, provided the user is a participant (or is automatically added as one when sending their first message).

XML Layout Structure fragment_event_chat.xml

The layout consists of a vertical LinearLayout containing three main components:

1. A TextView with ID textChatTitle to display the event's title.

2. A RecyclerView with ID recyclerViewEventChat that fills the remaining space and displays the list of chat messages.

3. A horizontal LinearLayout at the bottom, containing:

    - An EditText with ID editTextMessage for inputting a new message.

    - An ImageView with ID buttonSend to send the message.

The layout uses a white background and is padded to accommodate the BottomNavigationView at the bottom of the screen.

UI Behavior

The RecyclerView is initialized in onViewCreated with a vertical LinearLayoutManager.

The message list is managed by EventMessageAdapter, which displays each message either as "sent" or "received" depending on the sender.

Messages are color-coded per sender, and avatars/names are shown for received messages when the sender changes from the previous message.

The send button triggers logic that:

- Verifies that the user is a participant.
- Sends the message via Supabase.
- Reloads the message list.

Dynamic padding is applied at runtime to ensure the message input field stays visible above the BottomNavigationView.

Kotlin Logic (EventChatFragment.kt)

onCreateView

Inflates the layout fragment_event_chat.xml, which contains the event chat interface.

onViewCreated

- Retrieves eventId and eventDesc from the fragment arguments.
- Fetches the current user ID using Supabase Auth.
- Sets the chat title using the TextView.
- Sets up the RecyclerView with a LinearLayoutManager.
- Initializes listeners and UI logic:
    - Loads participants and messages via loadParticipantsAndMessages().
    - Sets up the send button to call sendMessage() when pressed.

    - Adds padding to the bottom of the RecyclerView and input box based on the height of the BottomNavigationView.

sendMessage(text: String)

- Checks if the current user is already a participant in the event by calling EventRepository.getEventParticipants(eventId).
- If not, adds the user via EventRepository.addParticipant(eventId, currentUserId).
- Sends the message to Supabase using EventRepository.sendMessageToEvent(...).
- Calls loadParticipantsAndMessages() to refresh the message list.

loadParticipantsAndMessages()

- Loads the list of participant IDs using EventRepository .getEventParticipants(eventId).
- Fetches participant profiles using ProfileRepository.getProfilesByIds(...).
- Assigns a unique color to each user (excluding the logged-in user) using assignColorsToUsers().
- Retrieves messages from Supabase using EventRepository .getMessagesForEvent(...).
- Maps each message to a EventMessageWithSender, combining message data with sender profile.
- Instantiates EventMessageAdapter with the message list, current user ID, and user color map.
- Assigns the adapter to the RecyclerView and scrolls to the latest message.

assignColorsToUsers(profiles: List<Profile>)

Assigns a unique color to each profile (excluding the logged-in user) from a predefined list of 10 colors, cycling as needed.

Supabase Integration

- The user's authentication status and ID are retrieved via SupabaseClientProvider.auth.currentUserOrNull().
- Messages and participants are fetched and updated via EventRepository.
- Profiles of participants are retrieved via ProfileRepository.

Read and write operations include:

- Fetching participants and messages.
- Sending messages to event_messages.

Supporting Components

EventMessageAdapter: A RecyclerView.Adapter responsible for rendering chat messages in the RecyclerView. It distinguishes between sent and received messages using getItemViewType().

- Sent messages use the layout item_event_message_sent.xml, aligned right.
- Received messages use item_event_message_received.xml, aligned left, and may include sender name and avatar if the message is not part of a consecutive group from the same user.

The adapter uses a color map to assign each sender a unique color, which is applied to the message bubble.

EventMessageWithSender: A wrapper class that combines a EventMessage with its corresponding Profile.

Error Handling

- If the user is not authenticated, onViewCreated throws an IllegalStateException and prevents further execution.
- No loading indicators, retry mechanisms, or user-facing error messages are implemented.
- If a message fails to send, there is no explicit feedback; the UI simply reloads the chat silently after attempting.

Improvements

1. Filters and Event Participation Consistency

Currently, events display preferences like age, languages, and interests, but these are not enforced. Users can join events even if they don't meet the specified criteria.

Improvement:

- Add client-side and server-side checks to block participation when criteria are not met.
- Show clear feedback to users if they’re not eligible to join.

2. Private Messaging Between Users

Currently, chat functionality is only available for events.

Improvement:

- Introduce a private messaging system between users.
- Add a private_messages table with RLS policies allowing access only for sender and recipient.

3. Dropdown Layout Refactoring

Dropdown layouts (e.g., for languages and interests) are repetitive and hard to maintain.

Improvement:

- Use a single reusable layout (e.g., item_dropdown.xml) with a TextView and optional ImageView.
- Create generic dropdown adapters to reduce duplicated code.

4. Row-Level Security (RLS) Management

Current RLS policies are permissive or incomplete in some tables.

Improvement:

- Harden policies to restrict access to authenticated users for their own data only.
- Test edge cases like forced join or unauthorized event deletions.

5. Supabase Realtime Integration

Event and location synchronization uses polling every 30s.

Improvement:

- Replace polling with Supabase Realtime (channels + broadcast) to:
    - update map markers live;
    - receive event messages instantly;
    - respond to join/leave events dynamically.

6. Animations and Transitions

There's little use of animations or coherent transitions.

Improvement:

- Add fade-in effects for tooltips.
- Animate marker appearance/disappearance.
- Use MotionLayout for smoother UI transitions.

7. App State and Session Management

State is managed manually via variables in activities.

Improvement:

- centralize session and app state;
- survive configuration changes;
- separate UI and business logic cleanly.

8. Atomic Profile Creation

Profile creation is split across multiple steps that are not atomic.

Improvement:

- Ensure the profile and all associated records (interests, languages, image) are saved in a single atomic transaction.
- Block navigation to Home until profile creation is complete.

9. Expanded Event Types and Separation from Vibe

The event_type field is limited and confused with user vibe.

Improvement:

- Add more event categories (e.g., hiking, work, food, sport).
- Separate vibe (user mood) from event_type (event category).

10. Localization

Many UI strings are hardcoded in the layout or Kotlin code.

Improvement:

- Move all strings to strings.xml.
- Use plurals.xml for dynamic texts (e.g., number of participants).
- Prepare for future multi-language support.

11. Push Notifications

Currently, there are no in-app or system notifications.

Improvement:

- Enable push notifications for:
    - new messages in joined events;

    - nearby event alerts;
    - invitations or private events.

12. Battery Optimization

The current location update strategy is aggressive and may drain battery.

Improvement:

- Reduce GPS update frequency using PRIORITY_BALANCED or movement-based strategies.
- Pause background tasks when app is not visible.

13. Security and Account Verification

Currently, there's no way to reset the password or verify the identity of users beyond email.

Improvement:

- Add password reset functionality.
- Introduce optional phone number verification and email confirmation during signup.
- Offer account linking with social platforms (Google, Apple, Facebook, etc.).

14. Other Improvements

- Marker clustering for dense user/event areas.
- More interactive, responsive tooltips.
- Client-side validation for max_participants limit.
- Espresso UI tests for critical flows (login, marker tap, tooltip display).
- Loading animation placeholders while waiting for data.
- Secure storage of Supabase and Mapbox tokens.