Università
degli Studi di
Messina

Software Engineering

Report

Professor: Salvatore Distefano

Students: Tairbek Akhayev, 551094

Zhalgas Abylkassymov, 547507

1. INTRODUCTION

1.1 PROJECT GOALS AND METHOD

The central objective of **Trip DVisor** is to create an intelligent, AI-driven travel planning platform that fundamentally changes the way people organize their trips. Today, preparing for a journey often means jumping between dozens of websites to compare flights, book accommodation, and look for interesting things to do. This process is not only time-consuming but also overwhelming, as information is scattered and requires constant manual filtering. Trip DVisor addresses this problem by introducing a conversational assistant that interacts with the user in a natural and intuitive way.

Instead of browsing different platforms, the traveler simply communicates their preferences—such as budget, interests, or desired pace of travel—and the system automatically produces a customized itinerary. The output is not just a list of recommendations but a carefully structured, day-by-day travel plan. Each day is divided into meaningful segments (morning, midday, afternoon, and evening), allowing users to balance sightseeing, relaxation, and personal time. Moreover, once the itinerary is ready, it can be exported as a PDF file, so that travelers can keep it on hand during their journey, even when offline.

Given the complexity of the project and the fact that requirements can change rapidly in the development of such platforms, the team chose to adopt the **Scrum methodology**. This agile approach allowed for incremental progress and constant refinement of features. Each sprint delivered a tangible part of the system, while regular feedback loops with stakeholders ensured that the project evolved in the right direction. This method not only reduced risks but also created room for experimentation and iterative improvement, which are crucial in building innovative, user-centric solutions.

1.2 PROJECT BACKGROUND

Traditional trip planning is often a fragmented and time-consuming process that can easily become stressful, especially for people who are not experienced travelers. Typically, a traveler needs to spend hours researching destinations across different websites, manually comparing hotels, flights, and local activities, and then trying to combine these pieces into a coherent schedule. Once the plan is ready, sharing it with friends or family often requires the use of external applications or messaging tools, which adds another layer of complexity.

This disjointed approach not only wastes time but also creates frustration, as most people lack the proper tools to optimize their itineraries effectively. Trip DVisor was designed to directly address these challenges by integrating all aspects of planning into a single platform. Through an AI-powered conversational interface, users can plan their trips more naturally and with far less effort. The system

automatically generates complete itineraries tailored to individual preferences, ensuring that each day is balanced and well organized.

In addition, the platform supports group collaboration by enabling features such as voting, commenting, and shared access to plans, making it easier for friends or families to coordinate. For premium subscribers, Trip DVisor also introduces advanced features, including weekly AI-suggested trip ideas, offering inspiration and convenience for frequent travelers.

1.3 THE REQUEST OF THE PROJECT

The initial request for the project was to design and implement a full-stack web application capable of supporting the complete cycle of travel planning, management, and collaboration. To achieve this goal, the platform was required to incorporate several core features.

At the center of the system is the **AI-powered conversational planner**, which allows users to input their travel details and preferences in a natural and intuitive way. Based on this input, the system generates structured itineraries that can then be saved within the platform. To ensure usability even when internet access is unavailable, the itineraries can also be exported as PDF documents for offline use.

Since travel is often a shared experience, the project also emphasized the importance of **collaboration tools**. Users should be able to share their planned trips, participate in group voting to decide between alternatives, and leave comments to coordinate more effectively with friends or family.

Equally important was the need for a **secure and reliable user management system**. The application supports different user roles—regular users, premium subscribers, and administrators—each with specific privileges. To strengthen security, the system also integrates two-factor authentication (2FA).

To maximize value for users, the project request further included **integration with external services**, particularly in the areas of booking and payments. This makes it possible to not only plan a trip but also take concrete actions such as purchasing tickets or managing subscriptions directly within the platform.

Finally, the platform had to provide **administrative tools** for overseeing users, managing stored trips, and monitoring system analytics. These tools are essential for ensuring scalability, maintaining data integrity, and supporting continuous improvement of the application.

Meeting this project request required a robust, modular, and scalable solution that brings together modern web technologies with reliable external APIs, ensuring both functionality and long-term sustainability.

1.4 WEBSITE DESCRIPTION

Trip DVisor was designed and developed as a **scalable, cloud-hosted web platform**, built with a modular architecture to ensure reliability, security, and extensibility. The system combines a modern frontend, a robust backend, a relational database, and multiple external APIs to provide an end-to-end travel planning experience.

- The **frontend** was implemented as a responsive React Single-Page Application (SPA), delivering a seamless and consistent UI/UX across desktop and mobile devices. Key libraries such as **react-icons** were used for lightweight iconography, while **html2canvas** and **jspdf** enable users to export itineraries directly into offline-ready PDF documents.
- The **backend** is powered by a Python **Flask REST API**, structured around Blueprints to separate concerns like authentication, trip management, voting, and administration. A number of extensions strengthen its functionality: **Flask-CORS** for handling cross-origin requests, **Flask-JWT-Extended** for role-based JWT authentication, **flask-dance** for OAuth integration, and **stripe** for payment workflows. Additional modules such as **python-dotenv**, **psycopg2-binary**, **requests**, and **reportlab** support configuration management, database connectivity, API interactions, and advanced PDF generation.
- Data persistence is handled by a **PostgreSQL relational database**, with a schema optimized for scalability and consistency. The main entities include users, trips, trip_group, voting_sessions, votes, notifications, and email_2fa_codes, each supporting critical features such as collaboration, secure login, and system alerts.

The platform integrates multiple **external services** to expand its capabilities:

- **Stripe API** for subscription payments and webhook-based updates.
- **SMTP** for transactional emails, including notifications and 2FA codes.
- **OAuth providers (e.g., GitHub)** for simplified and secure login.
- **Perplexity AI API** to power conversational itinerary planning.
- **Amadeus API** for flight search and booking functionality.
- **Google Maps API** for geocoding, routes, and map visualizations.
- **Hotelbeds API** for hotel search and reservations.

For **deployment**, both the backend and the PostgreSQL database are hosted on **Railway**, ensuring a cloud-native environment with automatic scaling and streamlined continuous deployment. Environment variables are securely managed to keep API keys and credentials protected.

This setup enabled the team to deliver a modern, production-ready solution that is not only functional in its current state but also extensible for future enhancements such as mobile applications, calendar integrations, and expanded booking services.

2. AGILE METHODOLOGY

2.1 Agile framework

The development of Trip DVisor followed **Agile principles** to guarantee adaptability, rapid iteration, and continuous improvement. In contrast to the traditional Waterfall model—where requirements are locked at the beginning and testing is postponed until late stages—Agile emphasizes delivering value early and often. This approach was particularly important for our project, since building an AI-powered system involves constant experimentation with prompt engineering, user interaction design, and integration of third-party services.

By adopting Agile, we were able to incrementally deliver **working features** at the end of each sprint. This not only provided the team with tangible progress but also allowed stakeholders to evaluate results in real time during sprint reviews and demos. Their feedback directly shaped subsequent iterations, ensuring that the product evolved in line with user expectations and actual needs.

A core strength of Agile lies in its ability to **embrace change**. During the development of Trip DVisor, requirements often shifted as new APIs were tested, user flows were refined, and AI-generated itineraries were adjusted for clarity and usability. Instead of treating these changes as setbacks, the Agile framework allowed us to integrate them naturally into our workflow, reducing friction and ensuring that the system stayed relevant and effective.

We directly applied the four key values of the **Agile Manifesto**:

- **Individuals and interactions over processes and tools** – team collaboration and communication were prioritized over rigid procedures.
- **Working software over comprehensive documentation** – functioning prototypes were consistently delivered, giving immediate value even before full documentation was complete.
- **Customer collaboration over contract negotiation** – regular engagement with stakeholders replaced lengthy requirement specifications.
- **Responding to change over following a fixed plan** – adaptability was embedded into our mindset, making it easier to adjust scope and direction when needed.

This mindset created a flexible environment not only for development but also for decision-making. The result was a project that could grow organically while still staying on track toward delivering a robust, user-friendly travel planning platform.

2.2 MOTIVATION TO CHOOSE SCRUM

Within the Agile family of methodologies, we chose **Scrum** as the framework best suited to the development of Trip DVisor. This decision was guided by both the nature of the project and the composition of the team.

One of the key factors was the **dynamic scope** of the system. Since the platform relies heavily on AI integration, requirements frequently shifted as prompts were refined, workflows were restructured, and output formats were adjusted to match user expectations. Scrum, with its iterative cycles, provided the flexibility needed to accommodate these changes without derailing progress.

Another crucial reason was the importance of **continuous feedback**. Features such as group voting, itinerary quality, and user collaboration tools required early validation from stakeholders. By presenting functional increments at the end of each sprint, we could quickly gather insights, test usability, and implement improvements before moving on to the next iteration.

The use of **short, two-week sprints** allowed us to maintain momentum and ensure that progress was visible and measurable. These regular checkpoints helped the team stay aligned, while also providing stakeholders with confidence that development was advancing steadily.

Scrum also contributed to **risk reduction**. By structuring the process so that every sprint ended with a potentially shippable product increment, uncertainty was minimized. Even if priorities shifted, the team always had a working version of the platform that could be deployed or demonstrated.

Finally, Scrum fit well with the **size and structure of our team**. With only three members, traditional heavy project management approaches would have been inefficient. Scrum's lightweight roles and ceremonies allowed us to rotate responsibilities flexibly, ensuring that each team member gained experience across different aspects of the project while maintaining clear accountability.

In summary, Scrum provided a framework that aligned perfectly with the evolving nature of Trip DVisor, enabling us to balance adaptability, stakeholder engagement, and steady delivery of value.

2.3 Scrum Methodology

To manage the development process, the team applied the Scrum methodology in a structured yet flexible way. The framework was tailored to fit the needs of a small team while still maintaining all the essential practices that make Scrum effective.

Roles
The three members of the team assumed rotating responsibilities across Scrum roles to ensure both accountability and shared learning.

- The **Product Owner** was responsible for keeping the product backlog aligned with the academic requirements of the course as well as the broader goals of the Trip DVisor project.
- The **Scrum Master** facilitated the ceremonies, helped maintain sprint discipline, and acted as a problem-solver whenever blockers arose.
- Meanwhile, the role of **Developers** was shared among all team members, with each person working across the frontend, backend, and database to gain full-stack exposure and ensure redundancy of skills.

Ceremonies
The team followed the standard Scrum ceremonies, adapted to the academic setting. During **Sprint**

Planning, backlog items were estimated using story points, and each sprint was given a clear, achievable goal. **Daily Standups**—kept to 15 minutes—served as a quick sync to share progress, identify blockers, and agree on the next steps. At the end of each sprint, the team conducted a **Sprint Review**, presenting live demos of completed features. For example, authentication was demonstrated in Sprint 1, the chat module in Sprint 2, and PDF export in Sprint 3. Stakeholder feedback was collected immediately after each demo and fed directly into the backlog. Finally, **Sprint Retrospectives** provided space for reflection, where the team discussed process improvements such as "increasing test coverage," "refining AI prompts," or "documenting APIs earlier."

Artifacts
Scrum artifacts were consistently used to bring transparency and structure to the project. The **Product Backlog** served as the prioritized feature list and was maintained via GitHub Projects. The **Sprint Backlog** outlined the subset of tasks chosen for each iteration, tracked visually on Kanban boards. **Burndown Charts** were generated by monitoring GitHub issue velocity, providing a clear picture of progress and potential bottlenecks. Finally, the **Definition of Done** was agreed upon as a quality checklist covering testing, code review, security validation, and deployment readiness (see Section 4.1 for details).

By adhering to this methodology, the team managed to stay aligned, deliver incremental value, and continuously refine both the product and the process.

2.4 Scrum Implementation

In practice, the Scrum framework was applied consistently and with discipline throughout the development of **Trip DVisor**. The team worked in fixed two-week sprints, a rhythm that provided enough time to deliver meaningful increments while maintaining frequent opportunities for feedback and adjustment.

Daily coordination took place through short stand-up meetings on **Discord** and **Telegram**. These quick syncs were essential for a small team, allowing members to share progress, raise blockers, and align priorities without unnecessary overhead.

At the end of every sprint, the team held **Sprint Reviews**, where completed increments were demonstrated live to stakeholders. Each review showcased functional progress:

- **Sprint 1** → User Management & Security Foundations (registration, login, session handling, 2FA, profile reset, encryption).
- **Sprint 2** → AI Trip Planning (conversational planner, questionnaire, AI itineraries, PDF export, favorites).
- **Sprint 3** → Collaboration & Notifications (trip sharing, group voting, comments, notifications, dark mode).
- **Sprint 4** → Premium & External APIs (Stripe subscriptions, weekly AI trip suggestions, flight/hotel search, Google Maps).
- **Sprint 5** → Administration, Optimization & Release (admin dashboards, monitoring, analytics, scalability).

These demos not only validated the work internally but also gave stakeholders the chance to provide early feedback, ensuring that development remained aligned with expectations.

Following each review, the team conducted **Sprint Retrospectives** to reflect on the process and identify areas for improvement. Action points were logged in a shared Google Doc to ensure accountability and follow-up. Examples of retrospective insights included:

- Sprint 1 → Sprint 2: strengthen error handling and validation in authentication flows.
- Sprint 2 → Sprint 3: improve AI prompt formatting for better itinerary generation.
- Sprint 3 → Sprint 4: increase notification coverage and refine group voting usability.
- Sprint 4 → Sprint 5: optimize API integrations and introduce performance monitoring.

This disciplined cycle of **planning, delivery, review, and reflection** enabled the team to achieve incremental delivery, gather early validation, and maintain continuous adaptation. These qualities were particularly important given the evolving requirements of an **AI-powered travel platform**, where usability, reliability, and integration with external APIs had to be constantly reassessed and refined.

3. Software Requirements

3.1 Functional Requirements

The functional requirements of Trip DVisor were organized into several major domains, reflecting the different aspects of the platform's purpose and usage. Each domain captured a distinct set of features essential to delivering a complete and user-friendly travel planning solution.

Trip Planning
At the heart of the platform lies the trip planning functionality. Users can interact with a **conversational AI planner**, providing their preferences and constraints in natural language. To ensure that essential details are captured, the system also supports a **structured questionnaire**, guiding users through key inputs such as destination, travel dates, budget, preferred activities, and overall travel style. Based on this input, the platform performs **automatic itinerary generation**, producing detailed day-by-day schedules broken down into meaningful segments (morning, midday, afternoon, and evening).

To further enhance planning, the system integrates external APIs that provide real-world travel options. The **Amadeus API** supplies suggestions for flights, enabling users to align their itineraries with available transportation routes. In parallel, the **Hotelbeds API** recommends accommodation options filtered by the budget and preferences defined during the planning process. These integrations transform the itinerary from a simple activity schedule into a comprehensive travel plan that seamlessly combines destinations, transportation, and lodging.

Once generated, itineraries can be **exported to PDF** for offline use and securely stored in the user's account for future access.

User Management
The platform provides robust user management to guarantee both accessibility and security. Users can

register and log in using **JWT-based session handling**, or opt for simplified onboarding via **OAuth authentication through GitHub**. To strengthen account protection, the system integrates **Two-Factor Authentication (2FA)** and **CAPTCHA validation during** registration and login attempts, mitigating risks of automated attacks or bot sign-ups. Profile management features allow users to reset passwords and update personal data as needed. To enforce permissions, **Role-Based Access Control (RBAC)** distinguishes between standard users, premium subscribers, and administrators.

Collaboration
Since travel is often a group activity, collaboration is a core part of the system. Users can generate **shareable trip links** to invite friends or family to participate in planning. Decision-making is facilitated through **voting sessions**, which support both anonymous and authenticated voting modes. In addition, participants can provide **comments and feedback** on proposed itineraries. To ensure timely awareness, the platform delivers **email notifications** about updates, voting results, and changes to shared trips.

Premium Features
The platform introduces a premium tier to enhance user experience and sustainability. **Stripe-powered subscription payments** enable users to upgrade their accounts seamlessly. Premium subscribers also benefit from **weekly AI-generated trip suggestions**, delivered directly by email, offering inspiration and ready-to-use plans.

Administration
Finally, a dedicated administration layer ensures effective platform governance. Administrators have access to a **management dashboard** for overseeing users and trips. They also hold **voting oversight capabilities**, enabling them to monitor group decision-making processes and intervene in case of misuse.

Together, these functional requirements define the scope of Trip DVisor as a comprehensive, AI-driven, and collaborative travel planning platform that balances usability, security, and scalability.

3.2 Non-Functional Requirements

In addition to functional capabilities, Trip DVisor was designed to meet a series of non-functional requirements. These requirements ensure that the platform is not only feature-rich but also secure, scalable, performant, and maintainable over the long term.

Security
Security is a cornerstone of the system's design. All authentication and authorization mechanisms are based on **JWT tokens**, which guarantee secure session handling and role enforcement. To protect against automated abuse, **reCAPTCHA** validation is integrated into registration and login flows. Passwords are stored only after being processed with strong hashing algorithms such as **bcrypt**, making them resistant to brute-force and dictionary attacks. Furthermore, rigorous **input validation and sanitization** mechanisms are enforced to prevent vulnerabilities such as SQL injection and cross-site scripting (XSS).

Performance
The platform was engineered to deliver a responsive and efficient user experience. **Backend APIs** are required to respond with a 95th-percentile latency of under **500 milliseconds** under normal operating loads. The **PDF itinerary generation** process completes in under **three seconds**, even for longer trips. The system architecture is designed to support at least **1,000 concurrent users**, ensuring that spikes in demand can be handled gracefully.

Reliability
Reliability is ensured through **regular automated database backups**, minimizing the risk of data loss. A comprehensive **logging infrastructure** is in place for error tracking and auditing, helping both developers and administrators monitor system health. For critical workflows—such as payment processing and trip creation—**transactional integrity** guarantees that operations are either fully completed or rolled back, preventing inconsistent states.

Usability
The platform emphasizes user-friendliness across devices. The **responsive design** ensures seamless operation on both mobile and desktop environments. Accessibility guidelines, specifically **WCAG 2.1 AA**, were followed during UI development to ensure inclusivity for users with different needs. The interface was structured around **intuitive workflows and simple navigation**, reducing the learning curve and enhancing adoption.

Scalability
To handle growth, the system was designed with scalability in mind. The backend services are **stateless**, making horizontal scaling straightforward. **Database connection pooling and query optimization** improve efficiency and resource utilization. Moreover, the architecture was built to run on **scalable cloud platforms** such as Railway and Vercel, ensuring flexibility in deployment and future expansion.

Maintainability
Maintainability is supported by a **modular codebase**, with a clear separation of concerns between the frontend, backend, and service layers. This structure simplifies debugging, testing, and future development. APIs are **well documented**, both through inline developer comments and formal documentation, ensuring that new contributors can quickly onboard. Finally, **CI/CD pipelines** enforce testing and coding standards, automating much of the quality assurance process and reducing the risk of regressions.

Collectively, these non-functional requirements ensure that Trip DVisor is not only functional at launch but also sustainable, adaptable, and robust in the face of evolving user demands.

4. Before Sprints

4.1 Definition of Done

Before beginning development, the team established a clear and shared **Definition of Done (DoD)**. This served as a quality benchmark to ensure that every user story or feature met the same standard of

completeness before being marked as finished. The DoD acted as both a checklist and a contract between developers, testers, and stakeholders, eliminating ambiguity around what “done” truly meant.

A feature was only considered complete if it satisfied the following criteria:

- **Code implementation**: The functionality had to be fully implemented, peer-reviewed, and merged into the main branch of the repository. This step ensured code quality and consistency across contributors.
- **Testing**: Both unit and integration tests were written for the new functionality, and all tests had to pass successfully within the **CI pipeline**. This guaranteed reliability and early detection of potential regressions.
- **Validation and security**: Proper input validation, authentication checks, and security verifications (such as protection against XSS and SQL injection) were required before release. This reduced risks of vulnerabilities and enhanced system robustness.
- **Documentation**: Every feature had to be accompanied by updated **code comments, API documentation, and user guides**, ensuring that knowledge was properly recorded for developers and end users alike.
- **Deployment**: The feature needed to be deployed into the **staging environment** without errors. A review by the entire team during the sprint review confirmed that the deployment process worked as intended.
- **Stakeholder acceptance**: Finally, the feature was demoed live to the **Product Owner**, and only after their approval could the story be marked as complete.

By adhering to this definition, the team maintained strong alignment throughout development. It prevented misunderstandings, improved quality assurance, and created a consistent rhythm where “done” meant not just coded, but tested, documented, secure, deployed, and accepted.

4.2 Sprint Calendar

Breakdown of hours

- **Total hours:** 600 hours
- **Number of developers:** 2
- **Hours per developer:** 200 hours (600 ÷ 2 = 300)
- **Number of sprints:** 5
- **Sprint duration:** 2 weeks
- **Hours per sprint:**
    - Total per sprint: 600 ÷ 5 = **120 hours**
    - Per developer per sprint: 75 ÷ 2 = **37.5 hours**

Calendar plan of the sprints

- **Sprint 1:** April 15 – April 25, 2025
    - Planning & Implementation: April 15 – April 25, 2025
    - Review & Retrospective: April 25, 2025

- **Sprint 2:** April 30 – May 14, 2025
    - Planning & Implementation: April 30 – May 15, 2025
    - Review & Retrospective: May 14, 2025
- **Sprint 3:** May 15 – May 29, 2025
    - Planning & Implementation: May 15 – May 29, 2025
    - Review & Retrospective: May 29, 2025
- **Sprint 4:** June 1st – June 15th, 2025
    - Planning & Implementation: June 1st – June 15, 2025
    - Review & Retrospective: June 15, 2025
- **Sprint 5:** June 15 – July 1, 2025
    - Planning & Implementation: June 15 – July 1, 2025
    - Review & Retrospective: July 1, 2025
    - Deployment checkpoint: July 1, 2025

4.3 Product Backlog

The Product Backlog is a prioritized and evolving list of user stories and features required to deliver Trip DVisor. It serves as the foundation for sprint planning and ensures alignment between stakeholders and the development team.

ID	User Story	Estimation (SP)	Priority
1	**As a user, I want to register an account securely (email + password + CAPTCHA), so that I can access the platform safely.**	5	1
2	**As a user, I want to log in securely (email + password + CAPTCHA, or GitHub OAuth without CAPTCHA), so that I can use my account.**	5	2
4	**As a user, I want to enable or disable two-factor authentication (2FA), so that I can protect my account with an extra layer of security.**	5	3
5	**As a user, I want to reset my password, so that I can regain access if I forget or want to change it.**	5	4
24	**As a user, I want my data to be encrypted in the database, so that my personal information is always protected.**	5	5
6	**As a user, I want to interact with an AI assistant in a chat interface, so that I can plan my trip in natural language.**	8	6

7	**As a user, I want to fill in a structured questionnaire, so that the system captures all my trip details.**	5	7
8	**As a user, I want the AI to generate a day-by-day itinerary (morning, midday, afternoon, evening), so that I have a complete plan.**	8	8
9	**As a user, I want to export itineraries to PDF, so that I can access them offline.**	5	9
10	**As a user, I want to store itineraries in my favorites, so that I can reuse and manage them later.**	5	10
11	**As a user, I want to create polls with settings (title, description, deadline, vote limit, anonymity), so that my group can decide on a trip democratically.**	5	11
12	**As a user, I want to share my trip via a link, so that others can join the planning process.**	8	12
13	**As a user, I want to comment on trips and polls, so that I can collaborate with friends.**	5	13
14	**As a user, I want to receive email notifications (2FA toggle, password reset, subscription, poll results), so that I stay informed.**	5	14
15	**As a premium user, I want to subscribe via Stripe, so that I can unlock premium features.**	8	15
16	**As a premium user, I want to receive weekly AI trip suggestions by email, so that I get inspiration.**	5	16
17	**As a user, I want to see flight options via the Amadeus API, so that I can align my trip with available transport.**	8	17
18	**As a user, I want to see hotel options via the Hotelbeds API, so that I can choose accommodations within my budget.**	8	18
19	**As a user, I want Google Maps integration, so that I can visualize routes and destinations.**	5	19

19 a	**As a user, I want to switch to dark mode, so that I can personalize my interface and reduce eye strain.**	3	20
20	**As an admin, I want a dashboard to manage users, trips, and voting sessions, so that I can oversee the system.**	8	21
21	**As an admin, I want a dashboard with key analytics and charts, so that I can understand usage trends and platform health.**	5	22
22	**As a product owner, I want the platform to handle 1000+ concurrent users, so that it performs well under load.**	8	24
23	**As a developer, I want clean, modular, and documented code, so that the system is easy to maintain and extend.**	13	25

5. Actual sprints development and results

5.1. Sprint 1 – User Management & Security Foundations

5.1.1. Sprint Planning

Sprint Planning Meeting: April 15th
Duration: 2.5 hours
Sprint Goal: Build the secure foundation of the platform with registration, login, session handling, password reset, 2FA, and database encryption.
Sprint Duration: April 15th – April 29th (2 weeks)

Sprint Backlog

- **User Story 1 (ID 1)** – Secure registration (email, password, CAPTCHA)
    - Tair: Frontend – Design registration form UI,Implement validation (email format, password strength
    - Zhalgas: Backend – Develop /register API + DB schema integration
- **User Story 2 (ID 2)** – Secure login (email, password, CAPTCHA, or GitHub OAuth)
    - Tair: Frontend – Design login page UI,Implement login form logic + 2FA handoff

    - Zhalgas: Backend – Develop /login API with JWT, rate limiting, account state checks
- **User Story 4 (ID 4)** – Two-Factor Authentication (enable/disable, email codes)
    - Tair: Frontend – Create UI for entering 2FA codes, toggle in account settings,Client-side validation of 2FA flow
    - Zhalgas: Backend – Generate, send (via email), verify codes; log changes
- **User Story 5 (ID 5)** – Profile management (password reset only + 2FA toggle)
    - Tair: Frontend – Build profile settings page (reset password, enable/disable 2FA),Add password strength validation & 2FA state UI
    - Zhalgas: Backend – Implement /profile/reset-password and /profile/2fa APIs, send confirmation emails
- **User Story 24 (ID 24)** – Data encryption in DB
    - Zhalgas: Backend – Apply bcrypt hashing for passwords and encrypt sensitive data
    - Tair: Test encryption flows through frontend API calls

Estimation: ~25 story points

Roles:

- Tair – Frontend Developer (UI/UX consistency, API integration,validation, session handling, secure flows))
- Zhalgas – Backend Developer (DB schema, APIs, encryption, security)
  Scrum Master: Tair

Responsibilities

- **Tair**: Oversees overall UI/UX consistency, ensures frontend components are aligned with backend APIs, and manages session handling in the client, focuses on implementing form validation, client-side session flows, and secure input handling (sanitization, error states).
- **Zhalgas**: Responsible for backend logic, including authentication APIs, database schema design, encryption of sensitive data, and security enforcement.

Scrum Master (Tair)

- Facilitates the Sprint Planning meeting and ensures all tasks are clearly understood and assigned.
- Monitors team progress daily and actively resolves blockers.
- Enforces adherence to Agile principles, team discipline, and the agreed **Definition of Done**.

Next Steps

- Development work begins immediately after Sprint Planning.
- **Daily stand-up meetings** are held on Discord/Telegram to synchronize tasks and resolve issues quickly.
- A **Sprint Review** and **Retrospective** are scheduled for **April 30th**, where completed features will be demoed, feedback will be gathered, and process improvements discussed.

The Sprint 1 backlog is derived from the Product Backlog (Section 4.3). Given the sprint goal *User Management & Security Foundations*, the following user stories were selected: US1, US2, US4, US5, US24. These items constitute the Sprint Backlog for Sprint 1.

ID	Priority	User Story	Task
1	1	As a user, I want to register an account securely (email + password + CAPTCHA), so that I can access the platform safely.	Implement secure registration system
2	2	As a user, I want to log in securely (email + password + CAPTCHA or GitHub OAuth), so that I can use my account.	Implement secure login system + session handling
4	3	As a user, I want to enable or disable two-factor authentication, so that I can protect my account with an extra layer of security.	Implement 2FA service
5	4	As a user, I want to manage my profile (reset password, manage 2FA), so that I can keep my account secure and up to date.	Implement profile management (password reset + 2FA toggle)
24	5	As a user, I want my personal data to be encrypted in the database, so that it remains protected from breaches.	Implement secure data encryption in DB

Tasks generated from User Stories of the product backlog for Sprint 1

User Story 1 (ID 1): Implement secure registration system

- **Task 1.1** – Tair: Design registration form UI (email, password, CAPTCHA).
- **Task 1.2** – Tair: Implement frontend validation (email format, password strength, CAPTCHA handling).
- **Task 1.3** – Zhalgas: Develop backend API for user registration (POST /register) and database integration.
- **Task 1.4** – Zhalgas: Integrate email verification for account activation.

User Story 2 (ID 2): Implement secure login system

- **Task 2.1** – Tair: Design login form UI (email, password, CAPTCHA, GitHub OAuth option).
- **Task 2.2** – Tair: Implement frontend login form logic (form validation, error handling, session tokens).
- **Task 2.3** – Zhalgas: Develop backend API for login authentication with JWT (POST /login), including session management.
- **Task 2.4** – Zhalgas: Enable bcrypt hashing & salting in password verification; configure timing-safe comparison.

User Story 4 (ID 4): Implement Two-Factor Authentication (2FA)

- **Task 4.1** – Tair: Create UI for entering 2FA codes and toggle in profile settings.
- **Task 4.2** – Tair: Implement client-side logic for sending and validating 2FA codes.
- **Task 4.3** – Zhalgas: Implement backend logic for generating, sending (via email), and verifying 2FA codes.

User Story 5 (ID 5): Implement profile management (password reset + 2FA toggle)

- **Task 5.1** – Tair: Build profile settings page UI (password reset form, 2FA toggle).
- **Task 5.2** – Tair: Implement frontend validation for password reset and 2FA state changes.
- **Task 5.3** – Zhalgas: Develop backend APIs for password reset and 2FA management (/profile/reset-password, /profile/2fa).

User Story 24 (ID 24): Implement data encryption in DB

- **Task 6.1** – Zhalgas: Configure bcrypt hashing for password storage.
- **Task 6.2** – Zhalgas: Encrypt sensitive user data (e.g., 2FA codes, notifications).
- **Task 6.3** – Tair: Test encrypted/decrypted flows via frontend requests to backend APIs.

5.1.2. Sprint Development

User Story 1 (ID 1) – Secure User Registration

Title: Secure Account Registration

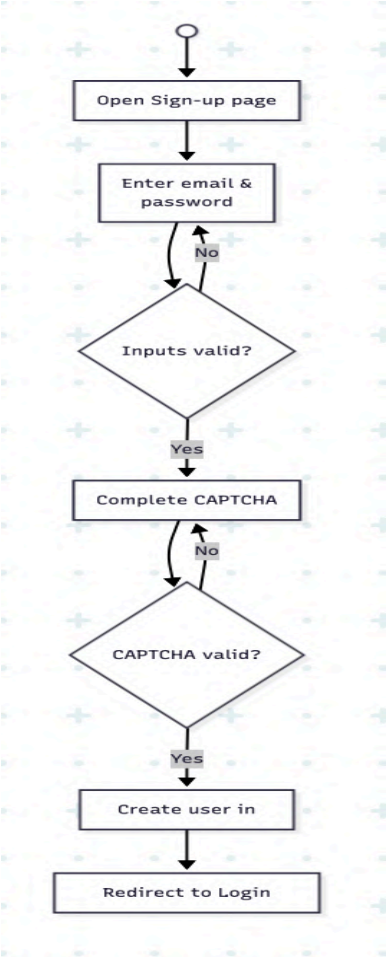
As a user
I want to securely create an account
So that I can access personalized trip-planning features and keep my personal data protected.

Acceptance Criteria

Registration Flow

1. Users can register using an email address and password.
2. The system verifies that the email address is unique before creating the account.
3. Passwords must meet minimum security requirements (length, complexity rules).
4. CAPTCHA validation is required to ensure the registration request is not automated.
5. Upon successful registration, the user redirects to login page.

Security & Data Protection
7. Passwords are stored securely using bcrypt hashing with salt.
8. Sensitive data related to the account is encrypted in the database.
9. All communication between client and server (including registration requests) is protected via HTTPS.



User Story 2 (ID 2) – Secure User Login

Title: Secure User Login
As a user
I want to log in securely (email + password + CAPTCHA or GitHub OAuth)
So that I can access my personal account and its features.

Acceptance Criteria

Security & Transport

1. All login requests and responses are transmitted over **HTTPS**.
2. Credentials are never logged; sensitive values are **redacted** in logs.

Input & Validation
3. The client validates email format; the server re-validates both email and password.
4. For invalid credentials, the system returns a **generic error** (no sensitive details leaked).

Account State
5. Only accounts with **ACTIVE** status can log in.

6. If the account is **DISABLED/BLOCKED**, the response informs the user that access is restricted (no additional details).

CAPTCHA & OAuth
7. Login via **email + password** requires **CAPTCHA** validation.
8. Login via **GitHub OAuth** is available and does **not require CAPTCHA**.

Password Verification
9. Passwords are verified using **bcrypt** (hash + salt) with a **timing-safe comparison**.

Tokens & Session
10. Successful login issues a **JWT access token** (short TTL) and a **refresh token** (longer TTL).
11. Token storage follows the app's policy (e.g., access token in client storage; refresh token via secure cookie/endpoint).
12. The client must handle **token expiration** and **logout flows**.

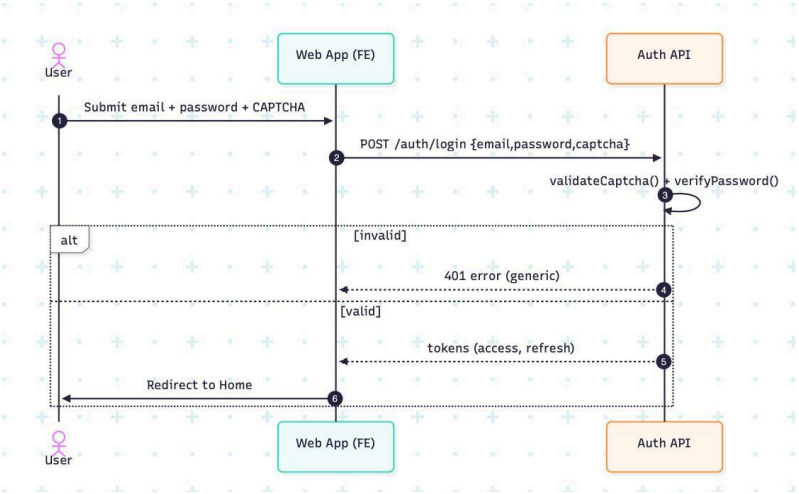
Abuse Prevention & Audit
13. Login attempts are **rate-limited** per IP/email. After repeated failures, the system enforces throttling or additional protection.
14. All authentication events (success/failure) are **audited with timestamps** (no plaintext credentials stored).

UX
15. Clear, localized feedback for errors: *"Incorrect email or password"*, *"Account disabled"*.
16. On successful login, the user is redirected to the **Home/Dashboard**.

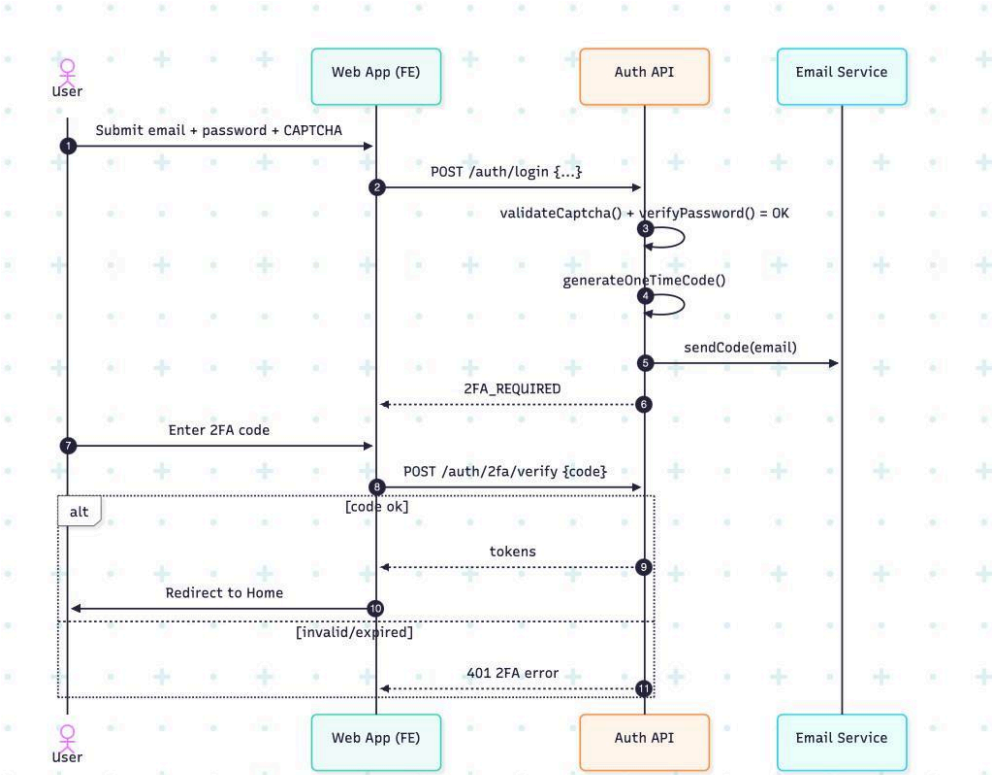
Tasks (from Sprint Backlog)

- **Task 2.1 — Tair:** Design the login form UI (email, password, error states, "Sign in with GitHub" button).
- **Task 2.2 — Tair:** Implement frontend login logic (form validation, POST /auth/login, error handling, token management, redirect on success).
- **Task 2.3 — Zhalgas:** Develop backend API (POST /auth/login) with JWT, account state checks, CAPTCHA verification, and rate limiting.
- **Task 2.4 — Zhalgas:** Ensure bcrypt password verification with timing-safe comparison; implement /auth/refresh and /auth/logout.

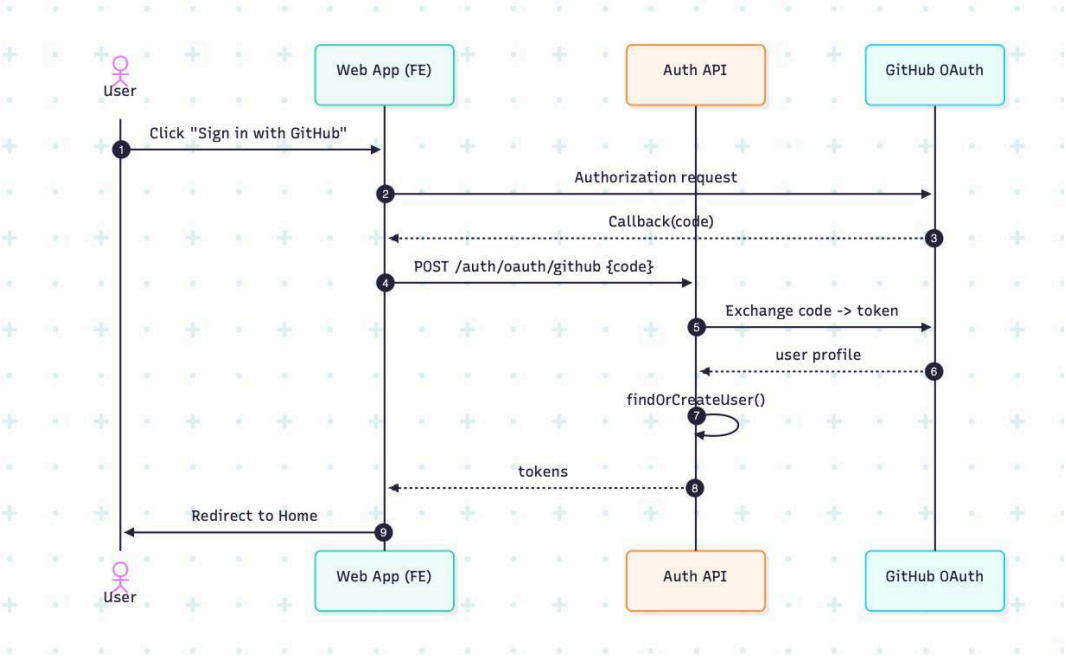
Login — Email & Password (2FA OFF)



Login — Email & Password (2FA ON)



Login — GitHub OAuth



User Story 3 (ID 4) – Two-Factor Authentication (2FA)

Title: Enable or Disable Two-Factor Authentication
As a user
I want to enable or disable two-factor authentication
So that I can protect my account with an extra layer of security.

Acceptance Criteria

A. Enabling / Disabling 2FA (Account page)

- The Account page shows current 2FA status (on/off) and a toggle.
- When the user enables 2FA, the system confirms by email that 2FA was turned on.
- When the user disables 2FA, the system confirms by email that 2FA was turned off.
- Changing 2FA state requires an authenticated session (valid JWT).
- All 2FA state changes are audited with timestamp and user id.

B. Login flow with 2FA (when enabled)

- After a successful email+password login (and CAPTCHA), if 2FA is enabled, the server responds with 2FA_REQUIRED (no long-lived tokens yet).
- The system sends a one-time code to the user's registered email.

- The user must submit the correct code within a short time window (for example, 5 minutes).
- If the code is valid and not expired, the system issues the access and refresh tokens and completes login.
- If the code is invalid or expired, the user gets a clear error and can retry.
- The user can request a new code with a resend action, subject to rate limits.

C. Codes, delivery, and security

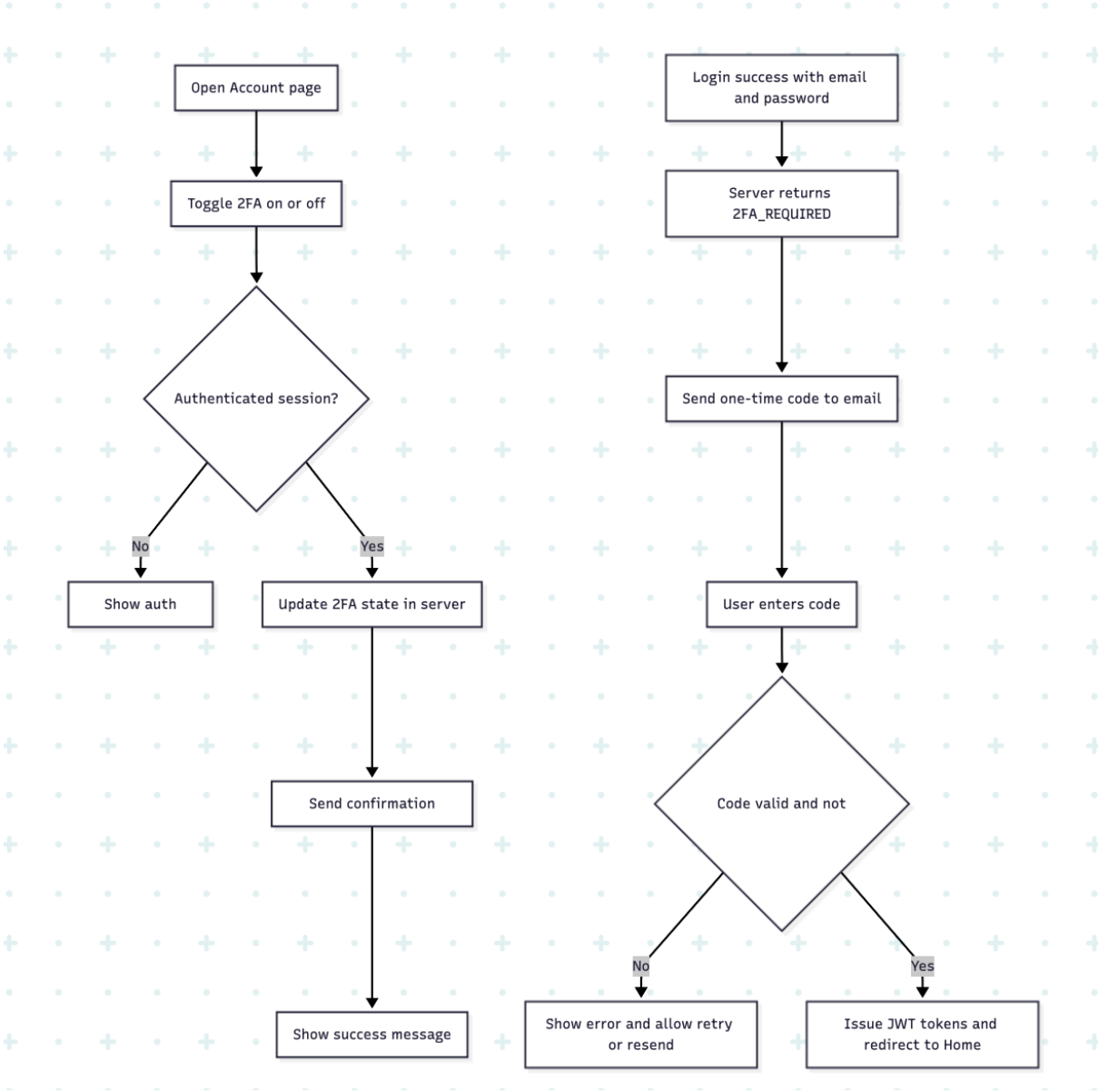
- One-time codes are random, single-use, and stored server-side with expiry and attempt counters.
- Resend is rate-limited (for example, max 3 per hour per user).
- All 2FA endpoints are protected against brute-force (rate limiting and lockouts after repeated failures).
- No secrets or full codes appear in logs; only hashed or masked values.
- All communication occurs over HTTPS.

D. UX

- Clear messages are shown for: code sent, code invalid, code expired, resend limit reached, 2FA enabled, 2FA disabled.
- If the user abandons the 2FA step, they remain logged out.

Tasks (from Sprint Backlog)

- **Task 4.1 — Tair (Frontend):**
  Build the 2FA UI: toggle on the Account page, code input screen after login, resend link, error and success states.
- **Task 4.2 — Zhalgas (Frontend):**
  Implement client logic: call /profile/2fa/enable and /profile/2fa/disable, handle 2FA_REQUIRED, submit /auth/2fa/verify, handle resend and rate-limit errors.
- **Task 4.3 — Zhalgas (Backend):**
  Implement endpoints and logic:
  - **/profile/2fa/enable, /profile/2fa/disable (with email notifications)**
  - **On login, return 2FA_REQUIRED and create a one-time code (persist in email_2fa_codes)**
  - **/auth/2fa/verify (validate code, TTL, attempts, then issue tokens)**
  - **/auth/2fa/resend (rate-limited)**
  - **Audit events and secure logging**



User Story 4 (ID 5) – Profile Management (Password Reset + 2FA Toggle)

Title: Manage Profile Security
As a user
I want to reset my password and manage 2FA from the Account page
So that I can keep my account secure and up to date.

Scope note

- 2FA enable/disable, code delivery, verification, and rate limits are **implemented in US3**.
- This story focuses on the **Account page UI/UX**, **password reset flow**, and **integration** with the existing 2FA endpoints from US3.

Acceptance Criteria

A. Account Page

- The **Account** page shows read-only **email/username**, and editable security controls:
    - **Change password** form (current password, new password, confirm).
    - **2FA toggle** showing real-time status (On/Off).
- Actions require a valid **JWT session**; expired/invalid sessions prompt re-login.
- All requests happen over **HTTPS**; sensitive fields are never logged.

B. Password Reset (Authenticated)

- User must provide a current **password** and a **new password** that meets strength policy.
- Backend verifies the current password with **bcrypt** (timing-safe compare).
- On success, backend stores the **new bcrypt hash** and **invalidates active refresh tokens**.
- The user is **logged out** (access token cleared) and must re-authenticate.
- A **confirmation email** ("Your password was changed") is sent.
- Friendly error states cover: incorrect current password, weak new password, rate-limited attempts.

C. 2FA Management (Integration with US3)

- The toggle **calls existing endpoints** from US3: /profile/2fa/enable and /profile/2fa/disable.
- After a successful change, the page reflects **updated 2FA status** without reload.
- A **notification email** is sent on 2FA state change (done in US3).
- Error states from US3 (rate limit, resend rules, etc.) are surfaced to the UI.

D. Security & Audit

- Inputs are sanitized server-side; error messages are generic (no leakage).
- All profile changes (password updates, 2FA toggles) are **audited** with timestamp and user id (no plaintext secrets).
- Brute-force mitigation: rate-limit password-change attempts per user/IP.

E. UX

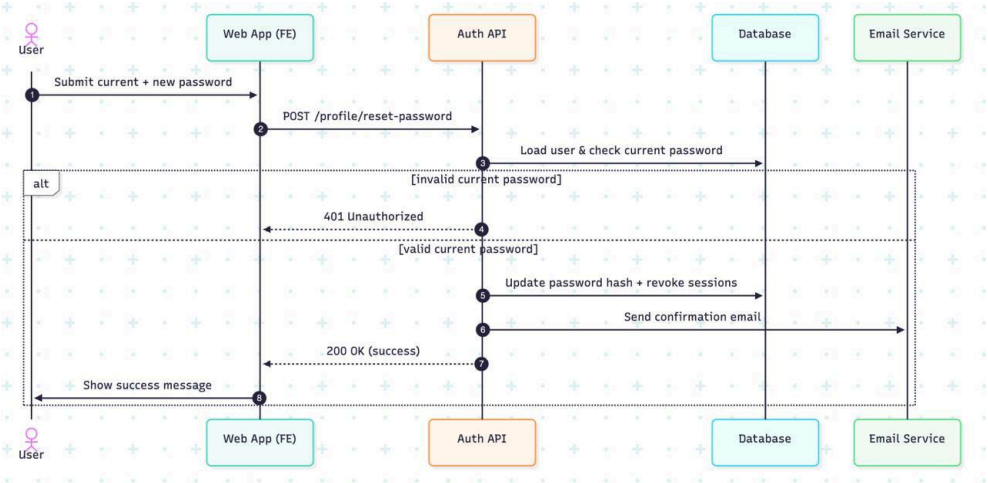
- Clear success and error banners; password strength meter on the client.
- Accessibility: keyboard navigation, ARIA labels for inputs, visible focus states.

Tasks (from Sprint Backlog)

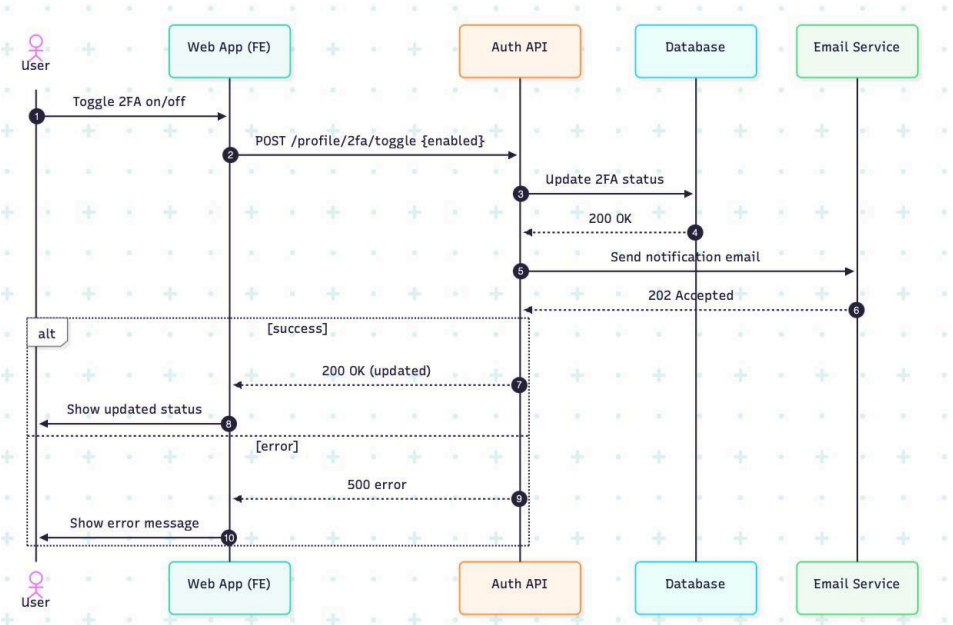
- **Task 5.1 — Tair (Frontend):**
  Build **Account** page UI: read-only email/username, password change form, 2FA toggle with live status; success/error banners.
- **Task 5.2 — Tair (Frontend):**
  Implement client validation (password strength, form states), call APIs, handle JWT expiration, show results of 2FA toggle (integrating US3 responses).

- **Task 5.3 — Zhalgas (Backend):**
  Implement /profile/reset-password (verify current password, update hash, invalidate sessions, send confirmation email); expose 2FA status endpoint if needed (read-only), reuse US3 endpoints for toggle.

Reset password



Toggle 2FA



2FA toggle flow (enable/disable) is covered in **US3**; in US4 we only **call** those endpoints and reflect the result in the UI.

User Story 5 (ID 24) – Secure Data Encryption in DB

Title: Protect Personal Data at Rest
As a user
I want my personal data to be encrypted in the database
So that it remains protected from breaches.

Scope note

- Passwords are **never encrypted**—they are **hashed** with bcrypt (one-way).
- Email is **not encrypted** (needed for login/OAuth and lookups).
- Sensitive values (e.g., 2FA codes, recovery tokens, any PII beyond email) must not be stored in plaintext.

Acceptance Criteria

A. Data Classification & Policy

- A **data classification** document exists (public vs. confidential vs. secret) with mapping to tables/fields in: users, email_2fa_codes, trips, trip_group, voting_sessions, votes, notifications.
- **Passwords** in users are stored with **bcrypt** (salted), cost factor agreed (e.g., 12), and timing-safe verification.
- **2FA codes / one-time codes** in email_2fa_codes are **not stored in plaintext**: store **hashed (HMAC or bcrypt)**plus expiry/attempt counters.
- Any **sensitive tokens/secrets** (e.g., password reset tokens) are stored as **hashes**; plaintext tokens are never persisted.

B. Encryption at Rest

- For selected sensitive columns (e.g., optional PII, user preferences that may be personal, trip notes with personal details), implement **AES-256-GCM** encryption at rest (via a server-side crypto service or pgcrypto).
- Encryption keys are **not** hard-coded; they are loaded from environment (e.g., APP_DATA_KEY) and rotated via a documented procedure.
- Each encrypted value uses a **unique IV/nonce**; ciphertext includes an **auth tag** (GCM) for integrity.
- No plaintext sensitive data appears in **DB dumps/backups**.

C. Application Integration

- A single **encryption service** layer is used by the backend (Flask) for all encrypt/decrypt operations; direct SQL must not handle plaintext.
  10. Read/write code paths are covered by **unit tests** to ensure correct encryption/decryption and backward compatibility.

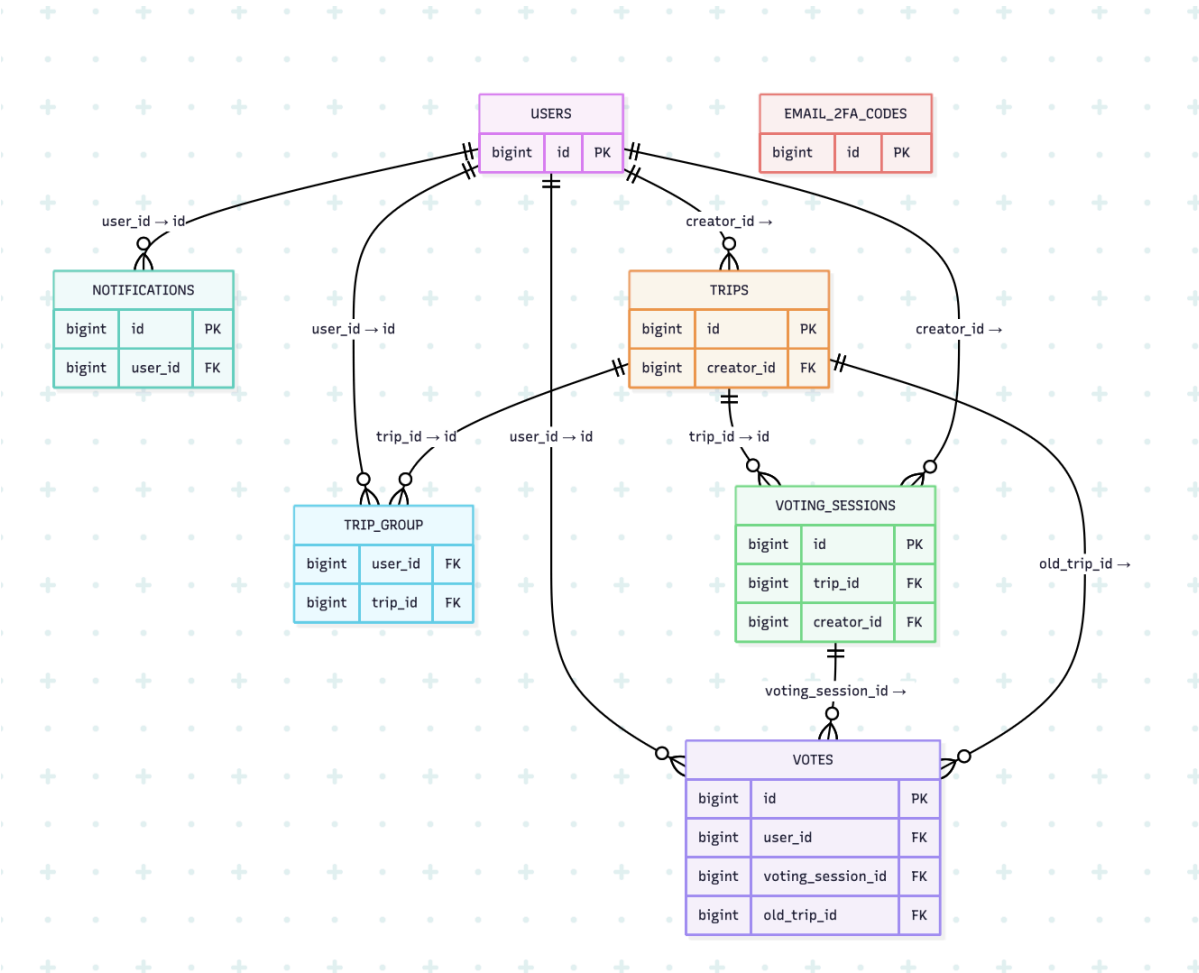
- API responses must **never** return raw secrets or decrypted values unless strictly required by UX (and then only to the owner).

D. Migration & Ops

- A **one-time migration** encrypts existing sensitive rows; migration is idempotent and logged.
- A **key rotation** runbook exists (generate new key, re-encrypt, rollback plan).
- Observability: logs include **no plaintext** secrets; only masked values.
- Performance: p95 for read/write of encrypted fields stays within **≤ 500 ms** for API endpoints (under normal load).

Tasks (from Sprint Backlog)

- **Task 6.1 — Zhalgas (Backend):**
  Define **data classification** for each table/column; decide which fields are hashed vs. encrypted vs. plaintext (email stays plaintext).
- **Task 6.2 — Zhalgas (Backend):**
  Implement a **CryptoService** (AES-256-GCM using cryptography or pgcrypto) with per-value IV and auth tag; centralize calls in the data access layer.
- **Task 6.3 — Zhalgas (Backend):**
  Ensure users.password uses **bcrypt** (cost factor agreed), timing-safe comparison in auth path.
- **Task 6.4 — Zhalgas (Backend):**
  Store **2FA codes** as **hashed** values in email_2fa_codes with TTL and attempt counters; adjust verify/resend logic accordingly.
- **Task 6.5 — Zhalgas (Backend):**
  Write a **DB migration** to encrypt existing sensitive fields; add scripts and logging; verify backups contain ciphertext only.
- **Task 6.6 — Tair(Frontend):**
  Run **end-to-end tests** of registration/login/profile flows to confirm no UX regressions and that decrypted values are displayed only where intended (never exposing secrets).
- **Task 6.7 — Zhalgas (Backend):**
  Produce a **Key Rotation runbook** (generation, activation, re-encryption, rollback) and add monitoring for crypto errors.



Entities

1. **Users**
    - **Attributes:**
        - id (Primary Key): unique identifier for each user.
        - *(other fields may include account and contact details depending on implementation).*
    - **Comments:**
        - Core entity of the system, referenced by many other tables.
        - A user can create trips, initiate voting sessions, receive notifications, and cast votes.
2. **Trips**
    - **Attributes:**
        - id (Primary Key): unique identifier for each trip.
        - creator_id (Foreign Key → Users.id): user who created the trip.
    - **Comments:**
        - Represents trips that are organized within the system.
        - Serves as the base for trip groups and voting sessions.

3. **Trip_Group**
    - **Attributes:**
        - user_id (Foreign Key → Users.id): participant in the trip.
        - trip_id (Foreign Key → Trips.id): the corresponding trip.
    - **Comments:**
        - Junction table modeling a many-to-many relationship between users and trips.
        - Allows multiple users to join the same trip.
4. **Voting_Sessions**
    - **Attributes:**
        - id (Primary Key): unique identifier for the voting session.
        - trip_id (Foreign Key → Trips.id): the trip being voted on.
        - creator_id (Foreign Key → Users.id): user who initiated the session.
    - **Comments:**
        - Represents the voting process tied to a specific trip.
        - Each session belongs to one trip and has one creator.
5. **Votes**
    - **Attributes:**
        - id (Primary Key): unique identifier for each vote.
        - user_id (Foreign Key → Users.id): user who cast the vote.
        - voting_session_id (Foreign Key → Voting_Sessions.id): the session in which the vote was cast.
        - old_trip_id (Foreign Key → Trips.id): references a trip being voted on (often an alternative/previous trip).
    - **Comments:**
        - Records voting activity of users.
        - Connects users, voting sessions, and trips.
6. **Notifications**
    - **Attributes:**
        - id (Primary Key): unique identifier for each notification.
        - user_id (Foreign Key → Users.id): recipient of the notification.
    - **Comments:**
        - Stores notifications for users (e.g., about new trips, voting results, or updates).
7. **Email_2FA_Codes**
    - **Attributes:**
        - id (Primary Key): unique identifier for each code entry.
        - *(usually includes fields like code, email/username or user_ref, expires_at, consumed_at, depending on implementation).*
    - **Comments:**
        - Handles temporary codes for two-factor authentication.
        - Not directly linked via foreign key to Users in the current schema; the relationship is logical (via email or identifier).

Relationships

- **Users → Trips**: one user can create many trips (Trips.creator_id → Users.id).
- **Users ↔ Trip_Group ↔ Trips**: many-to-many relationship between users and trips (participation).
- **Trips → Voting_Sessions**: one trip can have multiple voting sessions (Voting_Sessions.trip_id).
- **Users → Voting_Sessions**: one user can initiate multiple sessions (Voting_Sessions.creator_id).
- **Voting_Sessions → Votes**: one session can collect multiple votes (Votes.voting_session_id).
- **Users → Votes**: one user can cast multiple votes (Votes.user_id).
- **Trips → Votes**: votes can reference trips, including older or alternative ones (Votes.old_trip_id).
- **Users → Notifications**: one user can receive multiple notifications (Notifications.user_id).
- **Email_2FA_Codes**: independent entity for security, logically tied to users.

Sprint 1 – Testing Document
Test Summary

During Sprint 1, testing activities focused on validating the core security and authentication features of the Trip DVisor platform. The goal was to ensure that user accounts can be created, accessed, and protected reliably. The scope included registration, login, two-factor authentication (2FA), profile management, and database encryption. All test cases were executed, results documented, and recommendations for future iterations identified.

Test Activities

1. User Authentication and Authorization

- **Test Case 1 (TC001)**: Verify secure registration (email + password + CAPTCHA).
    - **Result**: Registration successfully creates new user accounts; CAPTCHA validation works as intended.
    - **Testing Method**: Manual UI testing.
- **Test Case 2 (TC002)**: Verify secure login (email + password + CAPTCHA).
    - **Result**: Login process works correctly; incorrect credentials are rejected.
    - **Testing Method**: Manual UI testing.
- **Test Case 3 (TC003)**: Validate GitHub OAuth login.
    - **Result**: OAuth integration works; users can log in using GitHub accounts.
    - **Testing Method**: Manual UI + API monitoring.
- **Test Case 4 (TC004)**: Validate session management with JWT tokens.
    - **Result**: Sessions are created securely and expire after timeout.
    - **Testing Method**: Automated API testing (Postman + JWT decoding).

2. Two-Factor Authentication (2FA)

- **Test Case 5 (TC005)**: Enable 2FA via email code.
    - **Result**: Users receive a code via email; code expires after set time.
    - **Testing Method**: Manual + automated email trigger check.
- **Test Case 6 (TC006)**: Disable 2FA.
    - **Result**: 2FA toggle updates user account settings successfully.
    - **Testing Method**: Manual UI testing.

3. Profile Management

- **Test Case 7 (TC007)**: Reset password workflow.
    - **Result**: Password reset email is sent, link expires as expected.
    - **Testing Method**: Manual testing with email logs.
- **Test Case 8 (TC008)**: Manage 2FA toggle in profile.
    - **Result**: 2FA can be enabled/disabled from the profile page.
    - **Testing Method**: Manual UI testing.

4. Database Security

- **Test Case 9 (TC009)**: Verify personal data encryption in DB.
    - **Result**: Sensitive fields (e.g., passwords, preferences) are stored in encrypted format (bcrypt/hashed or AES for preferences).
    - **Testing Method**: Automated DB inspection (psql queries).

Test Results

- All **critical features** (registration, login, 2FA, profile management, encryption) passed testing successfully.
- No blocking defects were identified.
- Minor issues:
    - Expired 2FA codes were not always cleared immediately (fix scheduled for Sprint 2).
    - OAuth error messages could be more descriptive for failed logins.

5.1.3. Sprint Review

Sprint Review Meeting: April 25th
Duration: 2 hours
Attendees:

- Tair (Scrum Master)
- Tair (Frontend Developer)
- Zhalgas (Backend Developer)
- Stakeholders (Clients / End Users)

Agenda:

1. Presentation of Completed User Stories and Deliverables
2. Demonstration of Implemented Functionalities
3. Feedback and Evaluation from Stakeholders
4. Discussion of Achievements and Challenges
5. Confirmation of Sprint Goals and Objectives

Outcome:

- Successfully implemented a secure **user registration system** (email + password + CAPTCHA).
- Demonstrated **secure login functionality**, including GitHub OAuth integration and session handling with JWT.
- **Two-Factor Authentication (2FA)** was introduced with email-based codes, including enable/disable toggle in profile.
- Stakeholders highlighted the importance of **profile management features**, especially password reset and 2FA settings.
- **Data encryption in the database** was verified, ensuring sensitive fields are securely stored.
- Minor challenges with 2FA code expiration logic were resolved collaboratively.
- Sprint goals were achieved, aligning with project milestones and stakeholder expectations.

Backlog Refinement

This backlog refinement reflects only the user stories addressed during Sprint 1:

ID	Priority	User Story	Task
1	1	As a user, I want to register an account securely (email + password + CAPTCHA), so that I can access the platform safely.	Implement secure registration system
2	2	As a user, I want to log in securely (email + password + CAPTCHA or GitHub OAuth), so that I can use my account.	Implement secure login system + session handling
4	3	As a user, I want to enable or disable two-factor authentication, so that I can protect my account with	Implement 2FA service

		an extra layer of security.	
5	4	As a user, I want to manage my profile (reset password, manage 2FA), so that I can keep my account secure and up to date.	Implement profile management (password reset + 2FA toggle)
24	5	As a user, I want my personal data to be encrypted in the database, so that it remains protected from breaches.	Implement secure data encryption in DB

5.1.4. Sprint Retrospective

Sprint Retrospective Meeting: April 25th
Duration: 1.5 hours
Attendees:

- Tair (Scrum Master)
- Zhalgas (Backend Developer)
- Tair (Frontend Developer)

Agenda:

1. Reflections on Achievements and Challenges
2. Identification of Action Items for Process Improvement

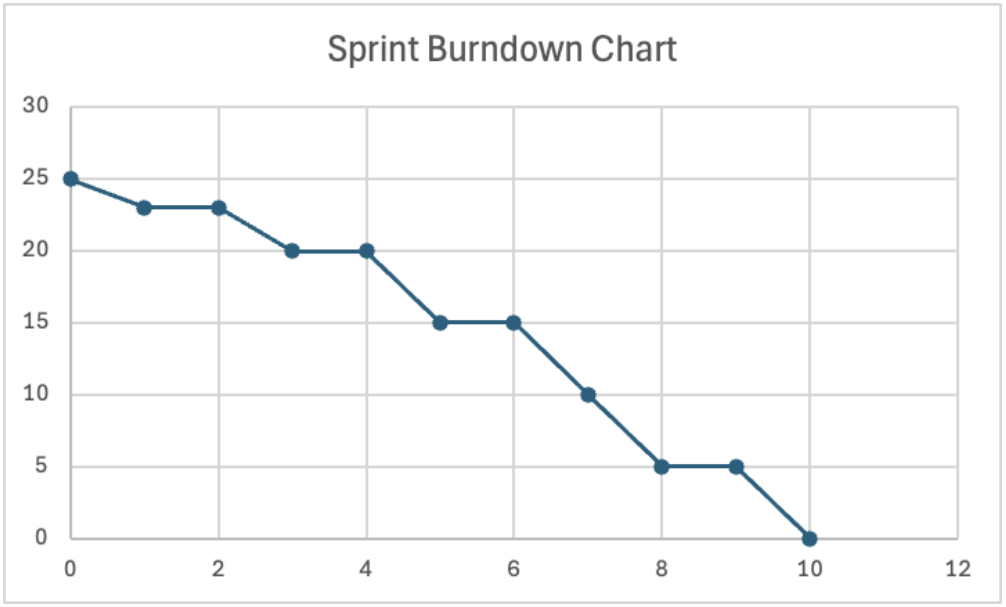
Outcome:

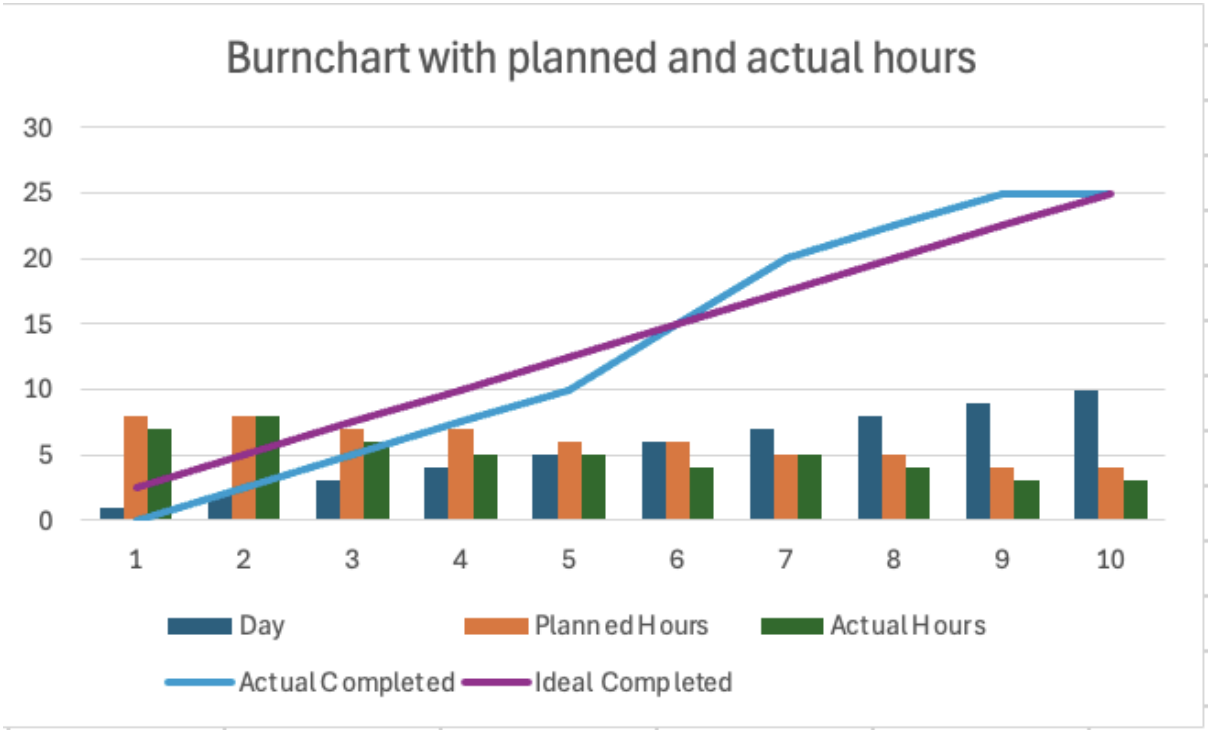
- Challenges faced during 2FA code expiration logic and OAuth error handling were discussed, emphasizing the importance of collaborative debugging.
- Action items identified included:
    - Improving communication channels between frontend and backend teams.
    - Conducting more frequent code reviews to ensure early detection of issues.
    - Expanding automated tests for authentication and security flows.

Next Steps:

- Implement the identified action items to strengthen collaboration and streamline development.
- Continuously monitor progress and adapt strategies based on feedback to enhance overall efficiency.

Burndown chart





Explanation

- **Day 1–2** → Planning and setup, work not yet closed (25 SP remain).
- **Day 3** → User Story 1 (Register securely, 5 SP) completed → 20 SP left.
- **Day 5** → User Story 2 (Login securely, 5 SP) completed → 15 SP left.
- **Day 7** → User Story 4 (2FA enable/disable, 5 SP) completed → 10 SP left.
- **Day 8** → User Story 5 (Profile management – password reset, 5 SP) completed → 5 SP left.
- **Day 10** → User Story 24 (Data encryption, 5 SP) completed → 0 SP left.

5.2. Sprint 2 – AI Trip Planning (Core Feature)

5.2.1. Sprint Planning

Sprint Planning Meeting: April 30th
Duration: 2.5 hours

Sprint Goal: Implement the core travel planning workflow with AI assistance and persistent storage.

Sprint Backlog:

1. **Conversational AI planner (User Story 6, 8 SP)**
    - Tair: Frontend – Design and implement chat interface UI,Build client-side conversation handling.
    - Zhalgas: Backend – Integrate AI backend (API/NLP pipeline).
2. **Structured questionnaire for trip details (User Story 7, 5 SP)**
    - Tair: Frontend – Design questionnaire form UI,Implement validation and data submission logic.
    - Zhalgas: Backend – Develop endpoints for structured trip input.
3. **Generate itineraries (day-by-day with flights/hotels) (User Story 8, 8 SP)**
    - Tair: Frontend – Display generated itineraries in UI,Manage itinerary state and formatting.
    - Zhalgas: Backend – Implement AI itinerary generation and external API integration.
4. **Export itineraries to PDF (User Story 9, 5 SP)**
    - Tair: Frontend – Add export button and user feedback modal,Implement client-side trigger and download handling.
    - Zhalgas: Backend – Develop PDF generation service.
5. **Store itineraries in favorites (User Story 10, 5 SP)**
    - Tair: Frontend – Favorites section in user profile UI,Implement local caching + sync with backend.
    - Zhalgas: Backend – Create DB schema and API for storing/retrieving itineraries.

Estimation: ~31 Story Points

Roles:

- **Tair**: Scrum Master & Frontend Developer – Facilitates sprint planning, oversees frontend tasks, ensures UI/UX alignment with project goals. Focuses on frontend implementation, feature development, and user interaction.
- **Zhalgas**: Backend Developer – Manages backend development, API integration, AI logic, and data persistence.

Responsibilities:

- Tair: Ensure consistency in UI/UX, lead sprint ceremonies, support team coordination.Deliver client-side features and maintain seamless interactions.
- Zhalgas: Build and secure backend services, connect external APIs, support AI pipeline.

Sprint Duration: April 30th – May 14th (2 weeks)

Next Steps:

- Development work begins immediately after sprint planning.
- Daily stand-up meetings will be held on Discord/Telegram to track progress and address blockers.
- Sprint Review and Sprint Retrospective meetings scheduled for **May 14th**.

Sprint 2 backlog originated from the main product backlog

It is important to note that some user stories were changed in priority based on the feedback from stakeholders, and this can be seen when comparing the sprint backlog to the original product backlog.

The Sprint 2 backlog is derived from the Product Backlog (Section 4.3). Given the sprint goal *AI-assisted Trip Planning & Itinerary Generation*, the following user stories were selected: US6, US7, US8, US9, US10.

These items constitute the Sprint Backlog for Sprint 2.

ID	Priority	User Story	Task
6	1	As a user, I want to plan my trip using a conversational AI chat interface so that I can receive interactive travel recommendations.	Implement conversational AI planner (chat interface)
7	2	As a user, I want to provide structured trip details through a questionnaire so that the platform can generate accurate itineraries.	Develop structured questionnaire form
8	3	As a user, I want the system to generate day-by-day itineraries (with flights and hotels) so that I can have a complete travel plan.	Implement AI itinerary generation (day-by-day plan)
9	4	As a user, I want to export my itineraries to PDF so that I can share and keep them offline.	Implement PDF export service
10	5	As a user, I want to store itineraries in my favorites so that I can easily access them later.	Implement favorites storage system

Total: 31 Story Points

User Story → Task Breakdown

- **User Story 6: Implement conversational AI planner**
    - Task 6.1: Design and build chat interface (frontend).
    - Task 6.2: Implement client-side conversation handling.
    - Task 6.3: Integrate backend AI service (API/NLP pipeline).
    - Task 6.4: Test conversation flows with sample user inputs.
- **User Story 7: Develop structured questionnaire for trip details**
    - Task 7.1: Design questionnaire form UI.
    - Task 7.2: Add validation for budget, dates, preferences.
    - Task 7.3: Connect form submission to backend endpoint.
    - Task 7.4: Ensure questionnaire integrates with AI planner.
- **User Story 8: Implement AI itinerary generation**
    - Task 8.1: Define itinerary structure (day-by-day).
    - Task 8.2: Integrate external APIs (flights, hotels).
    - Task 8.3: Implement AI logic to combine user inputs + APIs.
    - Task 8.4: Display generated itineraries in UI.
- **User Story 9: Implement PDF export service**
    - Task 9.1: Design PDF export button and feedback modal.
    - Task 9.2: Implement backend PDF generation (with itinerary data).
    - Task 9.3: Connect frontend trigger to backend export service.
    - Task 9.4: Test PDF output formatting and download.
- **User Story 10: Implement favorites storage system**
    - Task 10.1: Design Favorites section in UI.
    - Task 10.2: Implement API for saving and retrieving trips.
    - Task 10.3: Connect Save Trip button in planner to Favorites.
    - Task 10.4: Ensure trips saved in favorites can be reloaded or exported.

5.2.2. Sprint Development

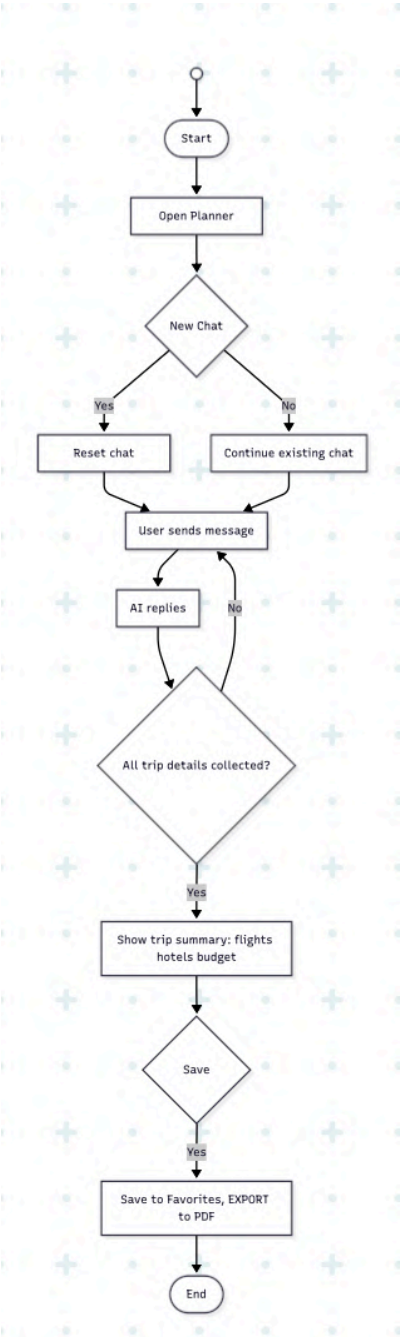
User Story 6:
Title: Conversational AI Planner
As a user, I want to plan my trip using a conversational AI chat interface,
so that I can receive interactive travel recommendations.

Acceptance Criteria:

1. **Chat Interface:**
    - Provide a clean and intuitive chat UI for the user.
    - Allow users to start a new chat session (e.g., “New Chat” button).
    - Display user inputs and AI responses clearly in conversation format.
2. **AI Integration:**
    - Connect the chat interface to the backend AI service.
    - Ensure responses are context-aware and tailored to user questions.
    - Validate that the AI asks follow-up questions when details are missing.
3. **Conversation Flow Management:**
    - Store conversation state throughout the session.
    - Support both structured questionnaire inputs and free-form chat.
    - Ensure the chat can be reset at any time without errors.

4. **Backend Logic:**
    - Implement API endpoints for sending and receiving chat messages.
    - Integrate external services (flights, hotels) for itinerary enrichment.
    - Ensure proper error handling when APIs fail.
5. **User Experience:**
    - Provide loading indicators while AI is generating responses.
    - Summarize collected trip details at the end of the conversation.
    - Allow the summary to be saved or passed to itinerary generation.

Planner



User Story 7
Title: Structured Trip Questionnaire
As a user, I want to provide structured trip details through a questionnaire,
so that the platform can generate accurate itineraries.

Acceptance Criteria

1. **AI-driven structured prompt**
    - A predefined prompt guides the AI to ask a fixed sequence of questions: destination, dates, travelers, budget, transport, accommodation level, interests, constraints.
    - The AI enforces completeness: if an answer is unclear or missing, it re-asks or reframes the question.
    - The AI keeps concise context and avoids off-topic replies during the questionnaire phase.
2. **Guided flow and state**
    - The flow is step by step with visible progress e.g. Step 1 of 6.
    - Each answer is stored in the session context immediately after the user replies.
    - The user can restart the questionnaire at any time via New questionnaire.
3. **Trip summary generation**
    - Once mandatory fields are collected, the AI produces a trip summary panel next to the chat: destination, dates, travelers, budget band, transport preference, accommodation range, interests highlights.
    - The AI confirms a short recap e.g. Please confirm the summary before proceeding.
    - The summary output is structured for downstream use by itinerary generation in US8.
4. **Hand-off to itinerary generation**
    - The collected values are passed as a normalized payload to the itinerary generator.
    - If conflicts exist e.g. dates invalid or budget too low, the AI requests corrections before hand-off.
5. **UX and accessibility**
    - Keyboard accessible flow with visible focus states and clear labels.
    - Mobile responsive layout for chat and summary panel.
    - Clear actions: Save trip, Export PDF, Restart questionnaire.
6. **Reliability and performance**
    - Questionnaire steps render instantly and responses appear under 1.5s p95.
    - Graceful error messages on network issues with retry options.
    - No itinerary generation is triggered until required fields are confirmed.

Done Criteria

- Structured AI prompt implemented and enforced in the chat flow.
- Session stores each answer and composes a validated trip summary.
- Summary appears in the side panel and can be confirmed or edited.
- Normalized payload ready for US8 itinerary generation.
- Manual tests pass for desktop and mobile; key paths covered by API tests.

User Story 8
Title: Generate Itineraries (day-by-day with flights/hotels)

As a user, I want the system to generate a structured itinerary (day-by-day with flights and hotels), **so that** I can have a complete and actionable travel plan.

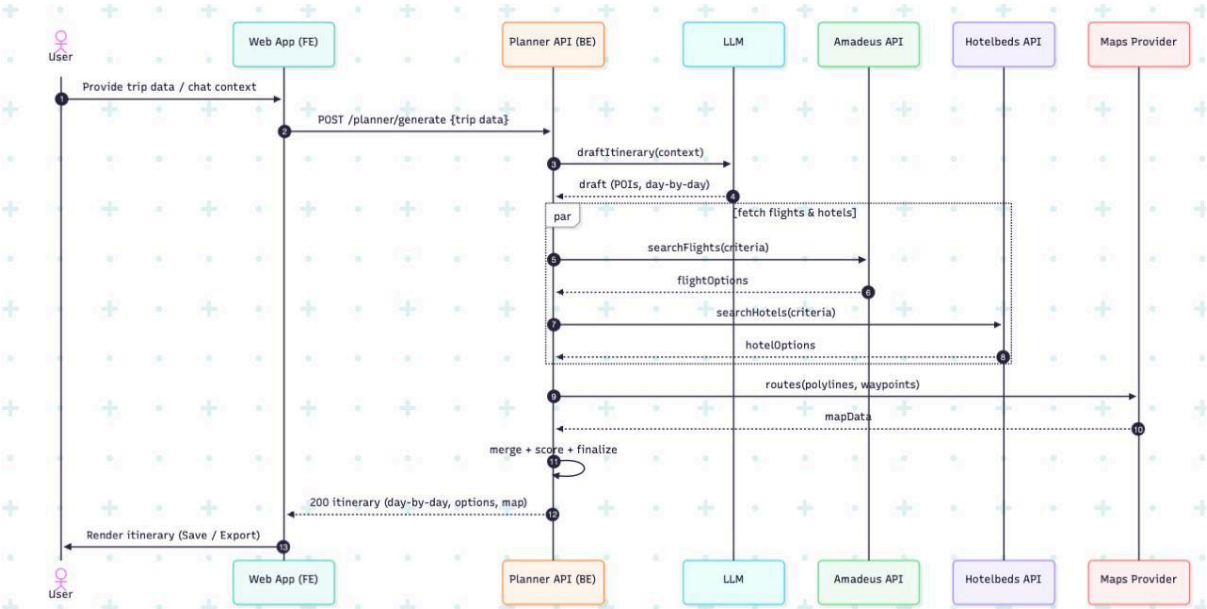
Acceptance Criteria

1. **Itinerary generation logic**
    - The system takes the **trip summary** from US7 (questionnaire) or US6 (chat).
    - A **mock itinerary generator** composes a day-by-day plan (Day 1: arrival + hotel, Day 2: sightseeing, Day 3: excursions…).
    - Flights and hotels are **mocked using static/fake datasets** to simulate real options.
    - Each activity block contains clear labels (transport, accommodation, daily activities).
2. **External API placeholders**
    - Instead of live APIs, the system uses **mock services** that return sample flight and hotel data.
    - The interfaces are designed so that in **Sprint 4** (Amadeus + Hotelbeds integration) the mocks can be swapped with real APIs.
    - Error-handling and data structures mirror the real integration format.
3. **User output (frontend)**
    - The itinerary is displayed in a **side panel next to the chat/summary**.
    - Each day is clearly separated with headings (Day 1, Day 2, …).
    - Mocked flights/hotels appear as if they were real search results, with prices and times.
    - Users can scroll, review, and confirm before saving/exporting.
4. **Persistence**
    - Users can save the itinerary to **Favorites** (US10).
    - Users can export the itinerary to **PDF** (US9).
5. **Validation**
    - The generated itinerary must match collected inputs (budget, dates, destination).
    - If input is invalid or missing, the system stops generation and shows an error message.
6. **Performance**
    - Itinerary generation under 2–3 seconds (mock data is lightweight).
    - Handles multiple concurrent requests by simulating API calls with delays.

Done Criteria

- Mock itinerary generator implemented and integrated with trip summary from US7.
- Day-by-day itinerary displayed in UI with mocked flights/hotels.
- Ability to save to Favorites (US10) or export to PDF (US9).
- Code structured to swap mock services for real APIs in Sprint 4.
- Manual and automated tests confirm itinerary matches inputs.

AI Itinerary Generation



User Story 9

Title: Save Trip → Generate & Attach PDF
As a user, I want my itinerary to be saved and exported to PDF with one action,
so that I can later access it from Favorites and download/share the file.

Acceptance Criteria

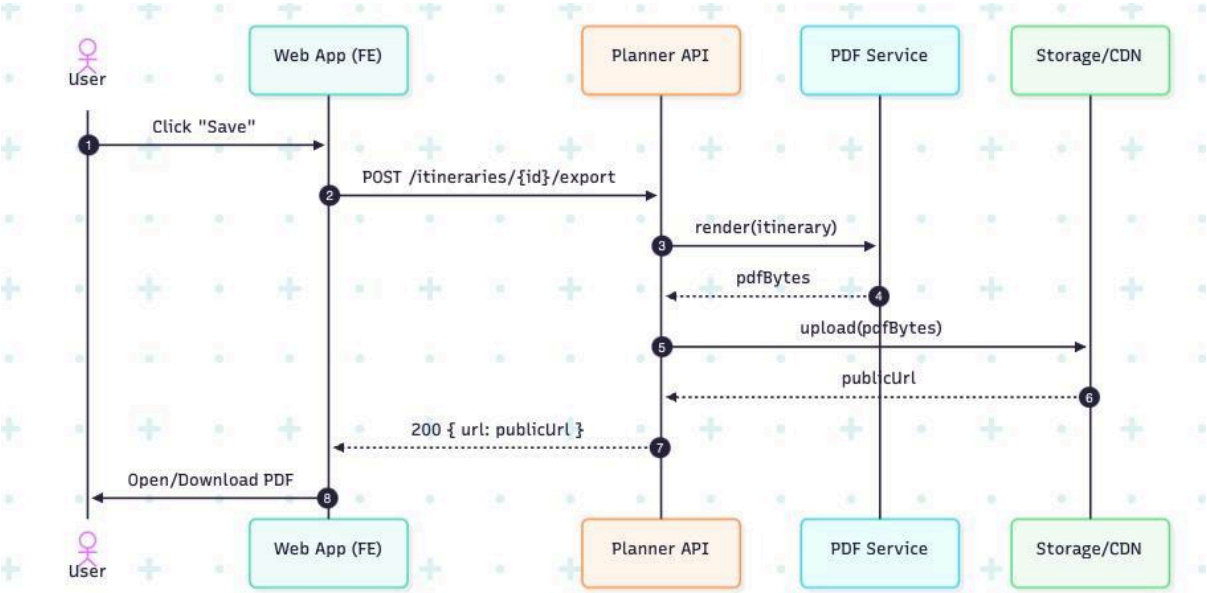
1. **Single Action (Save Trip)**
    - On the trip summary/itinerary panel there is one button: **Save Trip**.
    - Clicking it triggers two processes in parallel:
      a) Save the itinerary data into the database (in trips table).
      b) Generate a PDF from the current itinerary (in Sprint 2 this uses **mock data** from US8).
2. **Persistence & Linkage**
    - A new record is created in the trips table with fields like: id, title, created_at, owner_id, itinerary_payload, pdf_url.
    - The generated PDF is stored and linked to that trip record.
    - In the **Favorites UI**, saved trips will appear by fetching this data from trips.
    - Re-saving the same itinerary updates the existing trip record and regenerates the PDF.
3. **PDF Content & Layout**
    - PDF includes: trip title, dates, destination, budget, traveler count, transport preference, accommodation level, and **day-by-day itinerary** (with mocked flights/hotels).
    - Proper pagination, page numbers, generation date, and locale formatting (dates, currency).
    - Styled with app branding for consistency.

4. **UX & Feedback**
    - After clicking Save Trip, show a non-blocking indicator: **“Saving trip…”**.
    - On success: **“Trip saved”** + CTA **Open Favorites**.
    - In Favorites, trip cards display PDF status.
5. **Security**
    - Only the authenticated owner can save or download their PDFs.
    - Server validates the itinerary payload and sanitizes inputs.
    - Rate limiting is applied to prevent abuse.
6. **Performance & Reliability**
    - End-to-end Save + PDF p95 ≤ **3s** for a 7-day itinerary (mock).
    - Larger itineraries (up to 14 days) still complete successfully.
    - If PDF generation takes longer, the trip is still saved immediately, and the PDF link is updated when ready.
7. **Observability**
    - Log save and PDF generation events with anonymized metadata (duration, file size).
    - Export service emits traces for monitoring.
    -

Done Criteria

- One button **Save Trip** saves to Favorites **and** attaches a generated PDF.
- PDF is generated from normalized itinerary payload (mock in Sprint 2).
- Handled edge cases: success, partial success (trip saved but PDF failed), re-generation.
- Covered by contract tests, e2e flows, and manual verification (desktop + mobile).

Save Trip → Generate & Attach PDF



User Story 10

Title: Favorites Storage & Management
As a user, I want to view and manage my saved trips,
so that I can easily access, download, or delete them later.

Acceptance Criteria

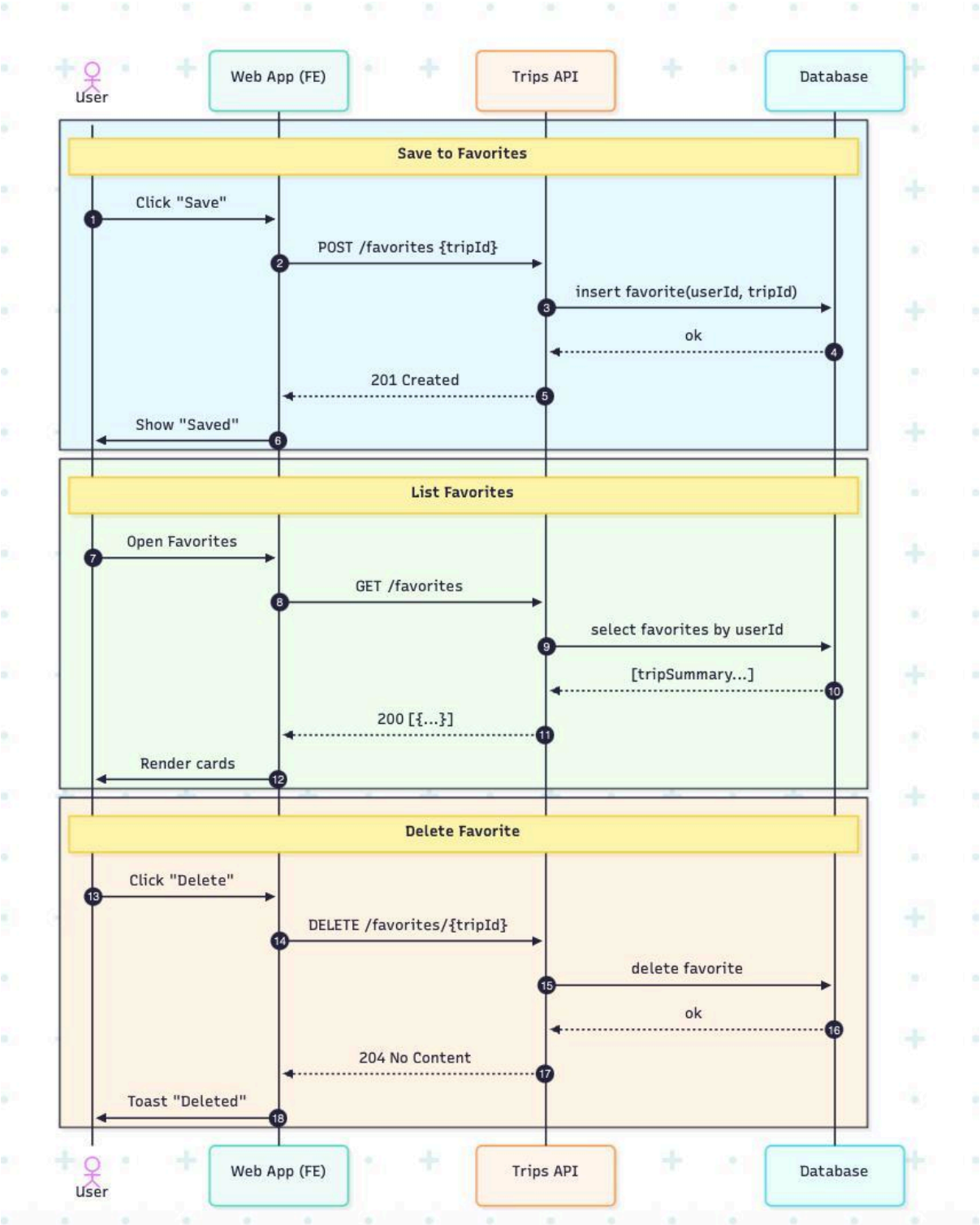
1. **Favorites Section in UI**
    - Accessible from the top navigation bar (**Favorites button**).
    - Shows all saved trips (records from trips where owner_id = current_user).
    - Each trip is displayed as a card with:
        - Title (e.g., "Summer in Rome")
        - Created date
        - Actions: **View**, **Download PDF**, **Delete**, **Create Poll**
2. **Create Poll Button (Future Sprint)**
    - Each trip card includes a **Create Poll** button.
    - In **Sprint 2**, this button is visible but not functional (disabled or showing "Coming soon").
    - Full poll creation workflow will be implemented in **Sprint 3**.
3. **Viewing a Trip**
    - Clicking **View** opens the full trip summary (restored itinerary payload).
    - The summary is identical to what was generated in Planner.
4. **Deleting a Trip**
    - Clicking **Delete** removes the trip from the trips table.
    - Confirmation dialog required to avoid accidental deletion.
    - Toast feedback: **"Trip deleted."**
5. **Persistence**
    - Saved trips remain available across sessions.
    - Trips are tied to the user via owner_id.
    - Premium users (future Sprint 4) will also see a **"Trip of the Week"** block above their personal trips.
6. **UX & Accessibility**
    - Responsive layout (desktop & mobile).
    - Keyboard navigation and ARIA labels supported.
    - Clear empty state: "You have no saved trips yet. Plan a trip in the Planner and save it here."
7. **Performance & Reliability**
    - Loading Favorites list ≤ **1s** for up to 50 trips.
    - Database queries optimized by indexing owner_id.
    - If PDF is missing (from US9 error), trip still loads with a status message.

Done Criteria

- Favorites page accessible from navigation.
- Trips are pulled from the trips table and shown in UI.
- Actions work: **View**, **Delete**.

- **Create Poll button is present in UI but disabled until Sprint 3.**
- Empty state displayed when no trips exist.
- Verified with manual e2e tests and API contract tests.

Favorites



Sprint 2 – Testing Document

Test Summary
During Sprint 2, testing activities focused on validating the AI trip planning features of the Trip DVisor platform. The goal was to ensure that users can interact with the AI assistant, provide structured trip details, generate mock itineraries, export trips to PDF, and store them in Favorites. The scope included conversational AI, structured questionnaire, itinerary generation with mock data, PDF export, and favorites storage. All test cases were executed, results documented, and recommendations for future iterations identified.

Test Activities

1. Conversational AI Planner

- **Test Case 1 (TC010):** Verify AI chat interface.
    - **Result:** Chat UI works; user messages are stored; AI responses are displayed.
    - **Testing Method:** Manual UI testing.
- **Test Case 2 (TC011):** Validate AI assistant answers user questions contextually.
    - **Result:** AI responds with relevant travel questions/recommendations.
    - **Testing Method:** Manual + mock AI backend logs.
- **Test Case 3 (TC012):** Reset chat history.
    - **Result:** Clicking "New Chat" clears history successfully.
    - **Testing Method:** Manual UI testing.

2. Structured Questionnaire

- **Test Case 4 (TC013):** Complete questionnaire flow.
    - **Result:** AI asks step-by-step questions (destination, dates, budget, transport, accommodation, interests).
    - **Testing Method:** Manual UI testing.
- **Test Case 5 (TC014):** Validate required fields.
    - **Result:** Missing mandatory answers (e.g., budget, dates) prompt AI to re-ask.
    - **Testing Method:** Manual testing.
- **Test Case 6 (TC015):** Trip summary generation.
    - **Result:** After completing questionnaire, a trip summary is displayed in the side panel.
    - **Testing Method:** Manual UI + mock backend response validation.

3. Itinerary Generation (Mock Data)

- **Test Case 7 (TC016):** Generate day-by-day itinerary.
    - **Result:** Mock itinerary is generated with daily activities, flights, and hotels.
    - **Testing Method:** Automated API mock tests + UI validation.
- **Test Case 8 (TC017):** Validate itinerary matches user input.
    - **Result:** Budget and dates from questionnaire are respected in the mock itinerary.
    - **Testing Method:** Manual testing with varied inputs.

4. PDF Export (Integrated into Save Trip)

- **Test Case 9 (TC018):** Save Trip generates PDF.
    - **Result:** Clicking Save Trip stores trip and downloads PDF file.
    - **Testing Method:** Manual UI testing + backend log inspection.
- **Test Case 10 (TC019):** PDF content validation.
    - **Result:** PDF contains trip summary + mock itinerary, formatted correctly.
    - **Testing Method:** Manual check of generated PDF files.

5. Favorites Storage

- **Test Case 11 (TC020):** Save trip to Favorites.
    - **Result:** Saved trips appear in Favorites section with title and date.
    - **Testing Method:** Manual UI testing.
- **Test Case 12 (TC021):** View trip from Favorites.
    - **Result:** Trip summary is restored correctly from stored data.
    - **Testing Method:** Manual UI testing.
- **Test Case 13 (TC022):** Delete trip from Favorites.
    - **Result:** Trip is removed; confirmation prompt works.
    - **Testing Method:** Manual UI testing.
- **Test Case 14 (TC023):** Download PDF from Favorites.
    - **Result:** User can download attached PDF file for saved trips.
    - **Testing Method:** Manual UI testing.

Test Results
All key features (AI chat, structured questionnaire, mock itinerary generation, Save Trip with PDF, Favorites storage) passed testing successfully.
No blocking defects were identified.

Minor issues:

- Occasionally, AI repeated a question twice if user input was unclear (to be refined in Sprint 3).
- PDF layout broke for very long trip descriptions (scheduled for Sprint 3 polish).
- “Create Poll” button is present in Favorites but disabled (functionality deferred to Sprint 3).

5.2.3. Sprint Review

Sprint Review Meeting: May 14th
Duration: 2 hours
Attendees:

- Tair (Scrum Master & Frontend Developer)
- Zhalgas (Backend Developer)
- Stakeholders (Clients / End Users)

Agenda:

1. Presentation of Completed User Stories and Deliverables
2. Demonstration of Implemented Functionalities
3. Feedback and Evaluation from Stakeholders
4. Discussion of Achievements and Challenges
5. Confirmation of Sprint Goals and Objectives

Outcome:

- Successfully implemented **Conversational AI Planner (chat interface)** for interactive trip planning.
- Demonstrated **Structured Questionnaire** that guides users step by step and generates a trip summary.
- Delivered **Mock Itinerary Generation** (day-by-day trips with flights and hotels simulated).
- Integrated **Save Trip → PDF Export**, allowing users to save itineraries and download them in PDF format.
- Introduced **Favorites Section**, where saved trips can be reloaded, viewed, and managed.
- Stakeholders praised the intuitive chat flow and structured questionnaire for clarity.
- Minor issues with repeated AI questions were noted and scheduled for refinement in Sprint 3.
- Sprint goals were fully achieved, ensuring alignment with milestones and stakeholder expectations.

Backlog Refinement
This backlog refinement reflects only the user stories addressed during Sprint 2:

ID	Priority	User Story	Task
6	1	As a user, I want to plan my trip using a conversational AI chat interface so that I can receive interactive travel recommendations.	Implement conversational AI planner (chat interface)
7	2	As a user, I want to provide structured trip details through a questionnaire so that the platform can generate accurate itineraries.	Develop structured questionnaire form
8	3	As a user, I want the system to generate day-by-day itineraries (with flights and hotels) so that I can have a complete travel plan.	Implement AI itinerary generation (day-by-day plan)
9	4	As a user, I want to export my itineraries to PDF so that I can share and keep them offline.	Implement PDF export service
10	5	As a user, I want to store itineraries in my favorites so that I can easily access them later.	Implement favorites storage system

5.2.4. Sprint Retrospective

Sprint Retrospective Meeting: May 14th
Duration: 1.5 hours
Attendees:

- Tair (Scrum Master & Frontend Developer)
- Zhalgas (Backend Developer)

Agenda:

1. Reflections on Achievements and Challenges
2. Identification of Action Items for Process Improvement

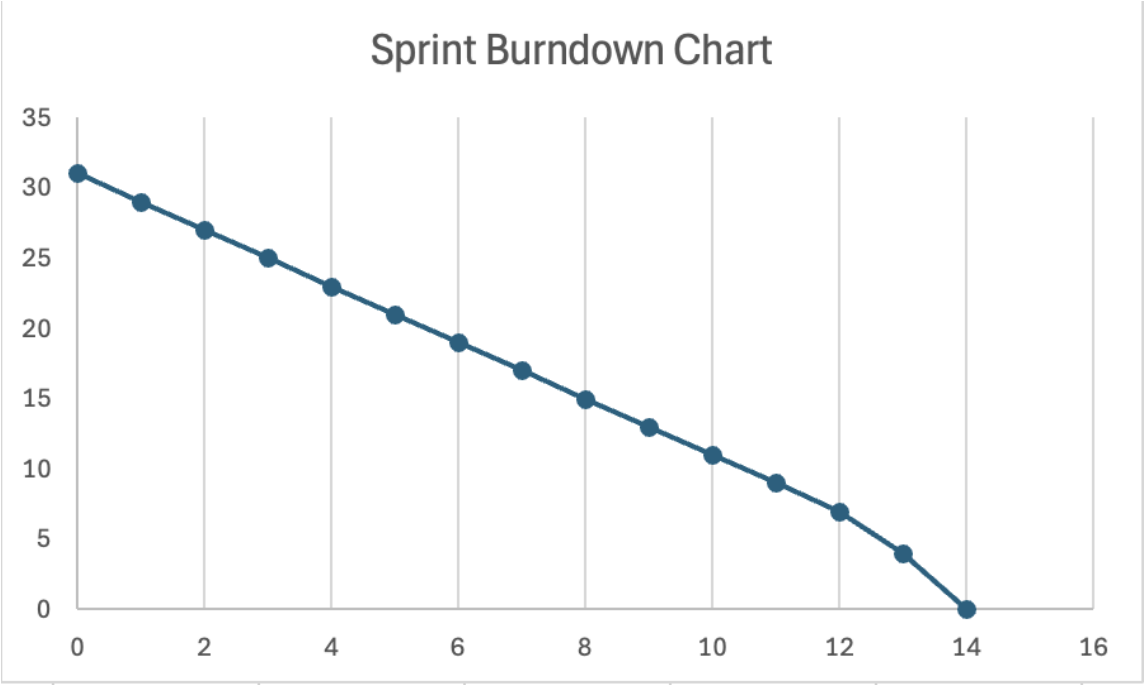
Outcome:

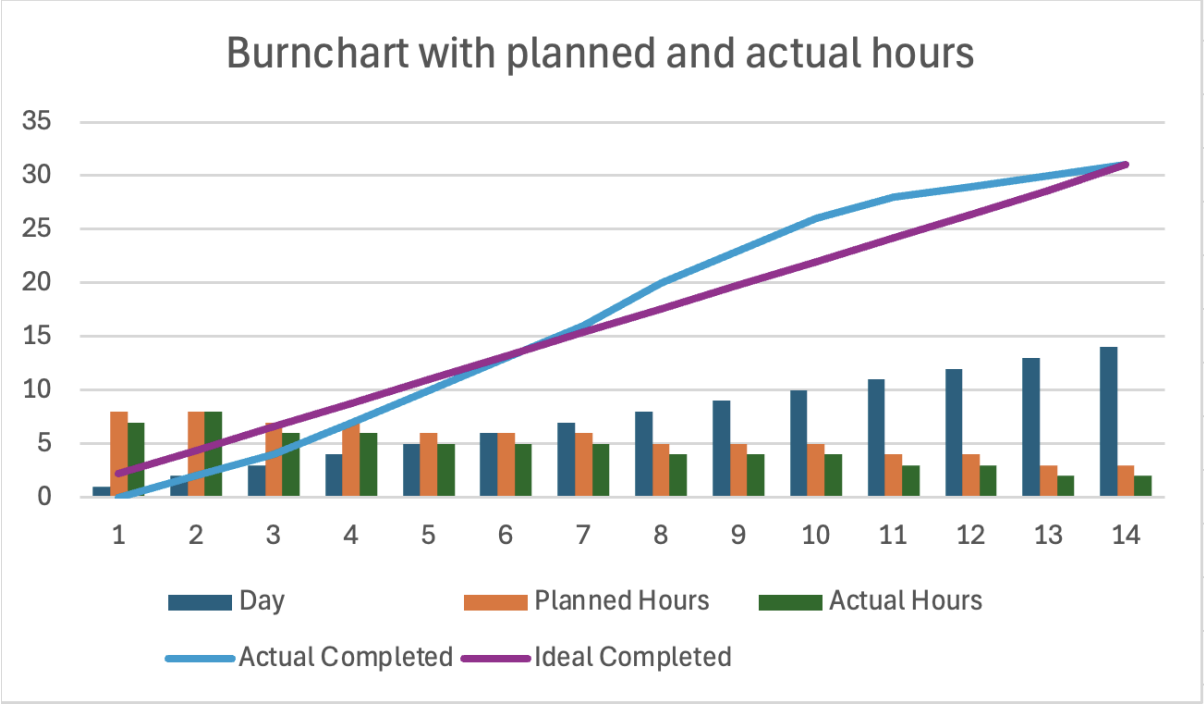
- The team successfully delivered the planned features: conversational AI, structured questionnaire, mock itinerary generation, Save Trip with PDF, and Favorites.
- Minor challenges arose with repeated AI questions during the questionnaire flow; this highlighted the need for refining prompt logic.
- PDF export formatting issues were identified when handling longer trip descriptions.
- The Create Poll button was added to Favorites but deferred to Sprint 3, which required clear communication with stakeholders.
- Collaboration between frontend and backend improved compared to Sprint 1, though testing handoffs still caused slight delays.

Next Steps:

- Refine AI prompt structure to reduce redundant questions in conversations.
- Improve automated testing for PDF generation to catch formatting issues earlier.
- Prepare implementation plan for Create Poll feature (Sprint 3) with clear acceptance criteria.
- Increase frequency of cross-team syncs to smooth frontend-backend integration.
- Continue gathering stakeholder feedback early to validate new AI features before finalizing.

Burndown chart
Total of 31 points from user stories.


Sprint Burndown Chart
35
30
25
20
15
10
5
0
0
2
4
6
8
10
12
14
16


Burnchart with planned and actual hours
35
30
25
20
15
10
5
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Day
Planned Hours
Actual Hours
Actual Completed
Ideal Completed

5.3. Sprint 3 – Collaboration & Notifications

5.3.1. Sprint planning

Sprint Planning Meeting: May 15th
Duration: 2.5 hours

Sprint Goal: Add group planning and communication features, including trip sharing, collaborative voting, comments, notifications, and UI enhancements.

Sprint Duration: May 15th – May 29th (2 weeks)

Sprint Backlog

- **User Story 11 (ID 11) – Voting (create poll, vote, comments, poll settings, email results) (8 SP)**
    - Tair: Frontend – Build poll creation modal (title, description, deadline, settings). Implement poll results UI (live stats, comments section)
    - Zhalgas: Backend – Implement poll storage, voting logic, comments API, email notifications on results
- **User Story 12 (ID 12) – Share poll via link – 5 SP**
    - After a poll is created (US11), the system generates a secure, unique share link.
    - The link leads to a **public voting page** with split view:
        - Left: voting stats (votes for/against, time remaining, voters list if non-anonymous).
        - Right: trip summary.
    - Link is copyable and shareable with group participants.
- **User Story 13 (ID 13) – Comment on trips (5 SP)**
    - Tair: Frontend – Add comment form in trip view. Display comment list with timestamps
    - Zhalgas: Backend – Store and retrieve comments linked to trip_id
- **User Story 14 (ID 14) – Notifications (2FA toggle, password reset, subscription, poll results) (5 SP)**
    - Tair: Frontend – Design notifications UI in account and trip pages. Integrate notification display (alerts/toasts)
    - Zhalgas: Backend – Trigger email notifications (2FA changes, password reset, subscription status, poll results)
- **User Story 19a (ID 19a) – Dark Mode toggle (3 SP)**
    - Tair: Frontend – Add dark mode toggle in header and persist preference in local storage. Style adjustments for dark mode consistency
    - Zhalgas: Backend – (Optional) API support for storing user preference

Estimation: ~26 Story Points

Roles:

- **Tair: Scrum Master & Frontend Developer** – Facilitates sprint planning, oversees frontend tasks, ensures UI/UX alignment with collaborative features. Focuses on implementing polls, comments, dark mode, and enhancing user interactions.
- **Zhalgas: Backend Developer** – Manages backend development for polls, notifications, and comment storage; ensures data integrity and email delivery.

Responsibilities:

- **Tair:** Lead sprint ceremonies, ensure UI/UX consistency across new collaborative features, and coordinate frontend integration. Implement client-side features (poll creation modal, results UI, notifications, dark mode toggle) and validate usability.
- **Zhalgas:** Develop backend services (poll storage, voting logic, comments API, notifications), secure shareable links, and manage email workflows.

Sprint Duration: May 15th – May 29th (2 weeks)

Next Steps:

- Development work begins immediately after sprint planning.
- Daily stand-up meetings will be held on Discord/Telegram to track progress and resolve blockers quickly.
- Sprint Review and Sprint Retrospective meetings are scheduled for May 29th, where collaborative features (polls, sharing, comments, notifications, dark mode) will be demonstrated and evaluated.

Backlog Origin:
Sprint 3 backlog originated from the main product backlog.
It is important to note that priorities were adjusted based on stakeholder feedback. For example, voting (US12) was given priority over shareable links (US11), since a link can only be generated after a poll exists.

The Sprint 3 backlog is derived from the Product Backlog (Section 4.3). Given the sprint goal *Collaboration & Notifications*, the following user stories were selected: US11, US12, US13, US14, US19a.

These items constitute the Sprint Backlog for Sprint 3.

ID	User Story	Estimation (SP)	Priority
11	**As a user, I want to create polls with settings (title, description, deadline, vote limit, anonymity), so that my group can decide on a trip democratically.**	5	11

12	**As a user, I want to share my trip via a link, so that others can join the planning process.**	8	12
13	**As a user, I want to comment on trips and polls, so that I can collaborate with friends.**	5	13
14	**As a user, I want to receive email notifications (2FA toggle, password reset, subscription, poll results), so that I stay informed.**	5	14
15	**As a user, I want to switch to dark mode, so that I can personalize my interface and reduce eye strain.**	3	20

Tasks (from Sprint Backlog)

User Story 11 (ID 11): Implement Voting (create poll, vote, comments, poll settings, email results)

- **Task 12.1 – Tair:** Design and build poll creation modal (title, description, deadline, vote threshold, anonymity toggle).
- **Task 12.2 – Tair:** Implement frontend logic for poll setup form (validation, error handling).
- **Task 12.3 – Zhalgas:** Develop backend API for poll creation (POST /polls) and store poll metadata in DB.
- **Task 12.4 – Tair:** Implement UI for live voting results (for/against counts, remaining time, comments section).
- **Task 12.5 – Zhalgas:** Implement backend voting logic (POST /polls/{id}/vote), prevent duplicate votes, enforce anonymity rules.
- **Task 12.6 – Zhalgas:** Add backend logic to close poll on deadline or when threshold reached; send results email to creator.

User Story 12 (ID 12): Implement Share poll via link

- **Task 11.1 – Tair:** Add UI to display shareable link after poll creation.
- **Task 11.2 – Tair:** Implement copy-to-clipboard functionality for share link.
- **Task 11.3 – Zhalgas:** Generate secure, unique poll links (UUID-based) and validate access when opening.
- **Task 11.4 – Tair:** Build public voting page with split view (left: stats/comments, right: trip summary).

User Story 13 (ID 13): Implement comments on trips

- **Task 13.1 – Tair:** Add comment input form on voting/trip detail page.

- **Task 13.2 – Tair:** Implement frontend logic to display comments list with author and timestamp.
- **Task 13.3 – Zhalgas:** Develop backend API (POST /comments, GET /comments/{trip_id}) to store and retrieve comments.

User Story 14 (ID 14): Implement notifications (2FA toggle, password reset, subscription, poll results)

- **Task 14.1 – Tair:** Design notification UI (toasts/banners) for account and poll pages.
- **Task 14.2 – Tair:** Implement frontend notification system (display, dismiss, error handling).
- **Task 14.3 – Zhalgas:** Develop backend triggers for email notifications (2FA changes, password reset, subscription updates, poll results).

User Story 19a (ID 19a): Implement Dark Mode toggle

- **Task 19a.1 – Tair:** Add Dark Mode toggle button in header.
- **Task 19a.2 – Tair:** Apply consistent dark mode styling across all UI components.
- **Task 19a.3 – Zhalgas:** (Optional) Extend backend API to persist Dark Mode preference per user.

5.3.2. Sprint Development

User Story 11
Title: Voting (Create Poll → Vote → Comments → Results Email)

As a user, I want to create a poll for a saved trip and let participants vote and comment,
so that we can make a group decision and receive results when the poll closes.

Acceptance Criteria

A. Create Poll

- Poll is created **from a trip card** (Favorites → *Create Poll*).
- Modal fields (required unless noted):
    - Title
    - Description (optional)
    - Deadline (date & time)
    - Vote threshold (required votes to close)
    - Anonymous voting (toggle)
- Validation: title required, deadline in the future, threshold ≥ 1.
- On successful creation → system persists poll details in DB and shows a confirmation.

B. Voting & Comments

- Voting page UI provides **For** / **Against** buttons.
- One vote per participant (duplicate prevention enforced).
- Comments section allows adding, listing, and displaying timestamps.
- If anonymous mode = ON → voter identities hidden.

C. Poll Lifecycle

- Poll closes automatically when:
    - Deadline is reached, OR
    - Required threshold of votes is met.
- Once closed: voting & commenting disabled; results frozen.

D. Results & Notifications

- Creator receives an **email with final results** (totals and outcome).
- Poll summary remains viewable in read-only mode.
- Clear **Closed** status shown on UI.

E. Error Handling

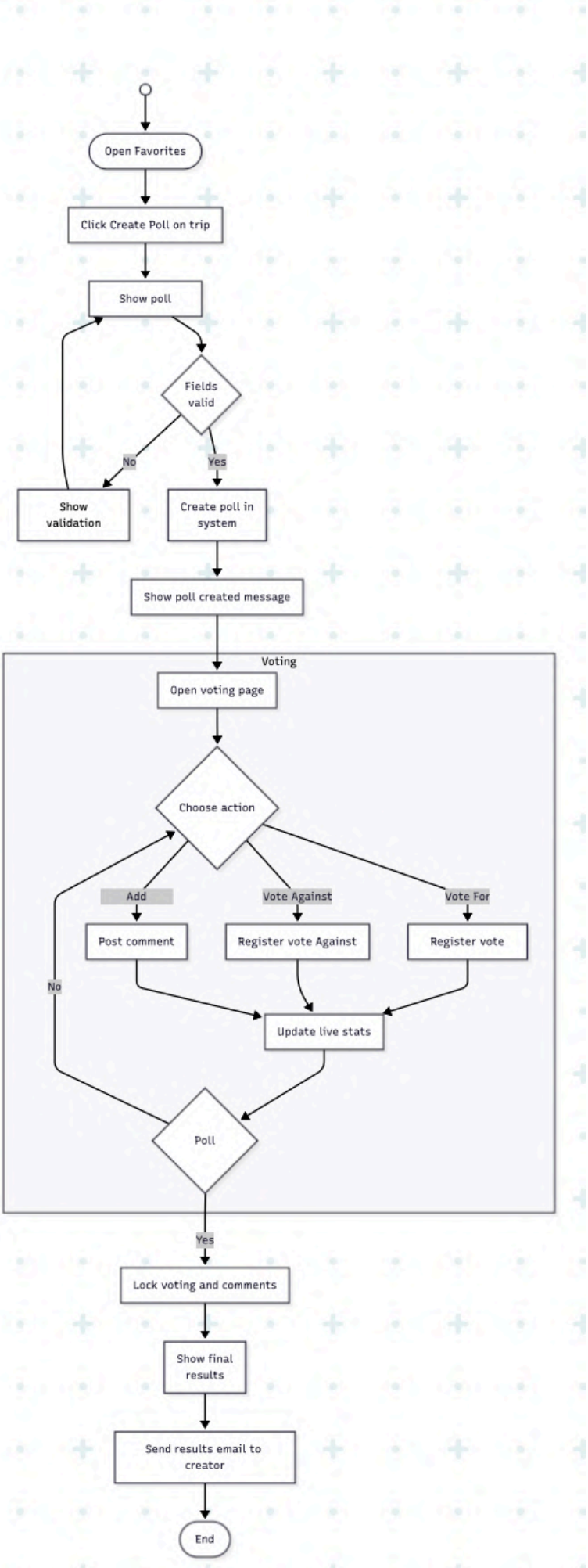
- Invalid or duplicate votes rejected with proper error messages.
- Expired/closed polls return informative status.

User Experience

- After poll creation: confirmation toast → *"Poll created successfully."*
- Voting page: clear layout with voting buttons, live stats, and comments.
- Closed state: interactive controls disabled; results visible.

Done Criteria

- Polls can be created with validated parameters.
- Participants can vote once and leave comments.
- Poll auto-closes by deadline or threshold.
- Creator receives results by email.
- Covered by unit tests (validation, voting rules) and e2e tests (create → vote → close → notify).


Open Favorites
Click Create Poll on trip
Show poll
Fields valid
No
Yes
Show validation
Create poll in system
Show poll created message
Voting
Open voting page
Choose action
Add
Vote Against
Vote For
Post comment
Register vote Against
Register vote
No
Update live stats
Poll
Yes
Lock voting and comments
Show final results
Send results email to creator
End

User Story 12 (ID 12)
Title: Share Poll via Link

As a user, I want to generate and share a secure link to a poll,
so that other participants can access the voting page without needing a full account.

Acceptance Criteria

Link Generation

- After successful poll creation (**US11**), the system automatically generates a **unique, secure link** (UUID or token-based).
- The link is tied to a specific poll_id and remains valid until the poll closes.
- UI provides a **Copy Link** button with confirmation toast: *"Poll link copied to clipboard."*

Public Voting Page

- Opening the link shows the **poll voting page**, split into two panels:
    - **Left panel:** live voting stats (for/against counters, time remaining, voter list if poll is non-anonymous), comments feed, voting actions.
    - **Right panel:** trip summary (title, destination, itinerary details, and PDF if available).
- Page is accessible to external participants without login, unless poll is configured as "authenticated only."

Voting & Commenting from Shared Link

- Participants opening the link can vote once (For/Against) and leave comments.
- Real-time updates: new votes and comments refresh without page reload.
- Duplicate voting prevention enforced even via shared links.

Security & Validation

- Links must be **unguessable** (UUIDs or signed tokens).
- Invalid or expired links redirect to an error page (*"This poll is closed or does not exist."*).
- Server validates poll state (open/closed) before accepting votes.

User Experience

- After poll creation: dialog shows "Poll created" + **Share link** button.
- When copied: confirmation message appears.
- Public page is responsive and mobile-friendly.

Done Criteria

- After poll creation, a secure share link is generated and displayed in UI.
- Copy-to-clipboard works correctly.
- Public link opens the voting page with trip summary and voting/comments panel.
- Invalid/expired links handled gracefully.

- Covered by frontend unit tests (UI copy, rendering) and backend tests (token validation, poll state checks).

User Story 13 (ID 13)
Title: Comments on Trips

As a user, I want to add and read comments on a trip (including from the public voting page),
so that participants can discuss and align before making decisions.

Acceptance Criteria

A. Comment Creation

- Users can submit a comment from:
    - Trip detail view (signed-in owners/participants).
    - Public voting page (via share link); behavior respects poll settings (anonymous vs authenticated).
- Required fields: comment text (non-empty, max length enforced).
- On submit, the comment appears instantly in the list (optimistic UI), or error is shown.

B. Comment Display

- Comments render in reverse chronological order.
- Each comment shows: author display name (or "Anonymous" per poll settings), timestamp (relative + tooltip with full datetime), and text.
- Long comments are clamped with "Show more / Show less".

C. Permissions & Modes

- If poll is **anonymous**, commenter identity is hidden on the voting page.
- If poll requires authentication, posting is blocked until the user signs in.
- Owners/admins can see moderation controls (future Sprint; out of scope here).

D. Reliability & Moderation Basics

- Server-side validation: length limits, basic profanity/HTML sanitization.
- Rate limiting (e.g., max N comments per minute per IP/user) to prevent spam.
- Errors are user-friendly (network issue, too frequent, content invalid).

E. Performance & Realtime

- New comments appear without page reload (polling or lightweight SSE/WebSocket optional).
- List virtualization for large threads (optional if >100 comments).

Done Criteria

- Users can **add** comments and **see** the updated list on public voting pages.
- Anonymity/auth rules enforced according to poll settings.
- Server validates/sanitizes input; rate limiting in place.

- Pagination works; timestamps and author (or “Anonymous”) display correctly.
- Covered by:
    - API tests: create, list, invalid input, unauthorized, rate limit.
    - E2E tests: add comment from both views; optimistic UI; error path.

User Story 14 (ID 14)
Title: Email Notifications (2FA toggle, password reset, subscription, poll results)

As a user, I want to receive email notifications for key account and collaboration events,
so that I’m informed about security changes, subscription status, and poll outcomes.

Acceptance Criteria

A. Events

- **2FA state changed** (enabled/disabled).
- **Password reset** requested and password changed confirmation.
- **Subscription** status updated (activated, renewed, canceled).
- **Poll results** when a poll closes (deadline reached or threshold met).

B. Email Content & Templates

- Localized subject/body (use current UI locale and formats for dates/currency).
- Templates:
    - **2FA Changed:** previous → new state, timestamp, location/device if available, link to account.
    - **Password Reset Requested:** secure, time-limited link; support contact hint.
    - **Password Changed:** confirmation + “not you?” security guidance.
    - **Subscription Updated:** plan, next renewal date or cancellation effective date; manage billing link.
    - **Poll Results:** trip name, vote totals, outcome, link to view closed poll.
- Plain-text fallback for all emails; brand header/footer in HTML variant.

C. Delivery & Reliability

- Send immediately after the triggering action succeeds.
- Queue with retry (exponential backoff) on provider failure; dead-letter after N attempts.
- DKIM/SPF/DMARC aligned sender; track delivery status (accepted/bounced).

D. Security & Privacy

- No secrets or full 2FA codes in emails.
- All links are signed and **time-limited**.
- Rate limit per user/event type to avoid spam; audit log for sensitive events (2FA, password).

E. Triggers

- 2FA toggle saved → send “2FA Changed”.
- Password reset requested; password changed → send “Password Changed”.

- Subscription webhook processed → send “Subscription Updated”.
- Poll closed → send “Poll Results” to creator.

Done Criteria

- Emails are sent for all four event types with correct localized templates.
- Links are signed, time-limited; no sensitive data leaked.
- Retries on failure; delivery/bounce logged.
- Unit tests cover template rendering and event→email mapping.
- Integration/E2E tests simulate each event and verify that emails are sent.

User Story 19a (ID 19a)
Title: Dark Mode Toggle

As a user, I want to switch between light mode and dark mode,
so that I can adjust the interface to my visual preference and improve readability.

Acceptance Criteria

A. Toggle Control

- A **Dark Mode toggle button** is visible in the header.
- Default mode = Light.
- Clicking the toggle instantly switches the UI theme.

B. Styling

- Dark Mode applies consistently across all pages: Home, Planner, Favorites, Account, Voting page.
- Elements (buttons, forms, modals, tables, chat panels) adapt colors for readability.
- Contrast ratios meet accessibility standards (WCAG AA).

C. Persistence

- Preference stored in **local storage** on the client.
- Optional: if user is authenticated, preference is synced with backend (so the mode is restored across devices).

D. UX & Accessibility

- Smooth transition between modes (CSS variables or theme classes).
- Icons (e.g., sun/moon) visually indicate current mode.
- Toggle is keyboard-accessible and labeled for screen readers.

E. Error Handling

- If backend persistence fails (optional feature), fall back to local storage only.

- No breaking UI states allowed during mode switch.

Done Criteria

- Dark Mode toggle appears in header and switches the theme instantly.
- Theme applies consistently across all pages and components.
- Preference is persisted in local storage (and backend if implemented).
- Covered by:
    - Unit tests for toggle component (UI state switch).
    - Integration tests ensuring theme persists across reloads.
    - Manual accessibility check for color contrast.

Sprint 3 - Testing document

Test Summary

During Sprint 3, testing activities focused on validating the collaboration and communication features of the Trip DVisor platform. The goal was to ensure that users can create polls, share them via secure links, vote and comment on trips, receive email notifications, and switch between light and dark modes. The scope included voting, poll sharing, comments, email notifications, and Dark Mode. All test cases were executed, results documented, and recommendations for future iterations identified.

Test Activities

1. Voting (Polls)

- **Test Case 1 (TC024): Create Poll**
    - **Result:** Poll creation modal validates fields (title, deadline, threshold, anonymity); successful creation persists poll.
    - **Testing Method:** Manual UI testing + API inspection.
- **Test Case 2 (TC025): Vote in poll**
    - **Result:** Users can vote For/Against once; duplicate votes blocked.
    - **Testing Method:** Manual UI testing.
- **Test Case 3 (TC026): Auto-close poll**
    - **Result:** Poll closes automatically at deadline or when threshold met.
    - **Testing Method:** Automated backend test (mock clock).
- **Test Case 4 (TC027): Results email**
    - **Result:** Creator receives final results via email when poll closes.
    - **Testing Method:** Manual email log inspection.

2. Share Poll via Link

- **Test Case 5 (TC028): Generate share link**
    - **Result:** After poll creation, secure share link is generated; “Copy Link” works.
    - **Testing Method:** Manual UI testing.
- **Test Case 6 (TC029): Access voting page via link**
    - **Result:** Opening link loads public voting page (stats/comments on left, trip summary on right).

    - **Testing Method:** Manual UI testing.
- **Test Case 7 (TC030): Invalid/expired link**
    - **Result:** Shows error page “Poll closed or does not exist.”
    - **Testing Method:** Manual backend mock + UI check.

3. Comments

- **Test Case 8 (TC031): Add comment**
    - **Result:** User can add a comment; appears instantly with author and timestamp.
    - **Testing Method:** Manual UI + API log check.
- **Test Case 9 (TC032): Anonymous poll comments**
    - **Result:** Author identity hidden if poll set to anonymous.
    - **Testing Method:** Manual UI testing.
- **Test Case 10 (TC033): Comment list retrieval**
    - **Result:** Comments load in reverse chronological order; pagination works.
    - **Testing Method:** Automated API tests.

4. Notifications (Email)

- **Test Case 11 (TC034): 2FA toggle email**
    - **Result:** User receives email confirmation when enabling/disabling 2FA.
    - **Testing Method:** Manual email log inspection.
- **Test Case 12 (TC035): Password reset email**
    - **Result:** Reset link + confirmation emails sent correctly.
    - **Testing Method:** Manual testing.
- **Test Case 13 (TC036): Subscription status email**
    - **Result:** Email sent on subscription activation/cancellation.
    - **Testing Method:** Simulated webhook test.
- **Test Case 14 (TC037): Poll results email**
    - **Result:** Final poll results emailed to creator on closure.
    - **Testing Method:** Manual test with live poll.

5. Dark Mode

- **Test Case 15 (TC038): Toggle Dark Mode**
    - **Result:** Clicking toggle switches between light/dark instantly.
    - **Testing Method:** Manual UI testing.
- **Test Case 16 (TC039): Persistence across reloads**
    - **Result:** Dark Mode preference stored in local storage; restored on refresh.
    - **Testing Method:** Manual browser test.

Test Results

All key features (polls, share link, comments, notifications, Dark Mode) passed testing successfully. No blocking defects were identified.

Minor issues:

- Occasionally, comments took ~2–3 seconds to appear due to polling delay (to be optimized in Sprint 4 with websockets).
- Dark Mode contrast on some secondary buttons did not fully meet WCAG AA (UI fix planned for Sprint 4).
- Some notification emails were flagged as spam by Gmail during testing (SPF/DKIM configuration to be refined in Sprint 4)

5.3.3. Sprint Review

Sprint Review Meeting: May 29th
Duration: 2 hours

Attendees:

- Tair (Scrum Master & Frontend Developer)
- Zhalgas (Backend Developer)
- Stakeholders (Clients / End Users)

Agenda:

1. Presentation of Completed User Stories and Deliverables
2. Demonstration of Implemented Functionalities
3. Feedback and Evaluation from Stakeholders
4. Discussion of Achievements and Challenges
5. Confirmation of Sprint Goals and Objectives

Outcome:

- Successfully implemented **Voting system** (poll creation, threshold/deadline closure, comments, email results).
- Delivered **Share poll via link**, providing secure public access to voting pages with split view (stats/comments + trip summary).
- Implemented **Comments** feature, allowing participants to discuss trips both in Favorites and on the voting page.
- Added **Email Notifications** for 2FA changes, password resets, subscription updates, and poll results.
- Introduced **Dark Mode toggle**, allowing users to switch between light and dark themes across the platform.
- Stakeholders praised the group collaboration features, especially the voting and sharing flows.
- Minor issues were noted:
    - Comments are occasionally displayed with delay due to a polling mechanism (to be improved in Sprint 4).
    - Some notification emails were marked as spam in Gmail (SPF/DKIM adjustments planned).
- Sprint goals were achieved, aligning with project milestones and stakeholder expectations.

Backlog Refinement

This backlog refinement reflects only the user stories addressed during Sprint 3:

ID	User Story	Estimation (SP)	Priority
11	**As a user, I want to create polls with settings (title, description, deadline, vote limit, anonymity), so that my group can decide on a trip democratically.**	5	11
12	**As a user, I want to share my trip via a link, so that others can join the planning process.**	8	12
13	**As a user, I want to comment on trips and polls, so that I can collaborate with friends.**	5	13
14	**As a user, I want to receive email notifications (2FA toggle, password reset, subscription, poll results), so that I stay informed.**	5	14
15	**As a user, I want to switch to dark mode, so that I can personalize my interface and reduce eye strain.**	3	20

5.3.4. Sprint Retrospective

Sprint Retrospective Meeting: May 29th
Duration: 1.5 hours

Attendees:

- Tair (Scrum Master & Frontend Developer)
- Zhalgas (Backend Developer)

Agenda:

1. Reflections on Achievements and Challenges
2. Identification of Action Items for Process Improvement

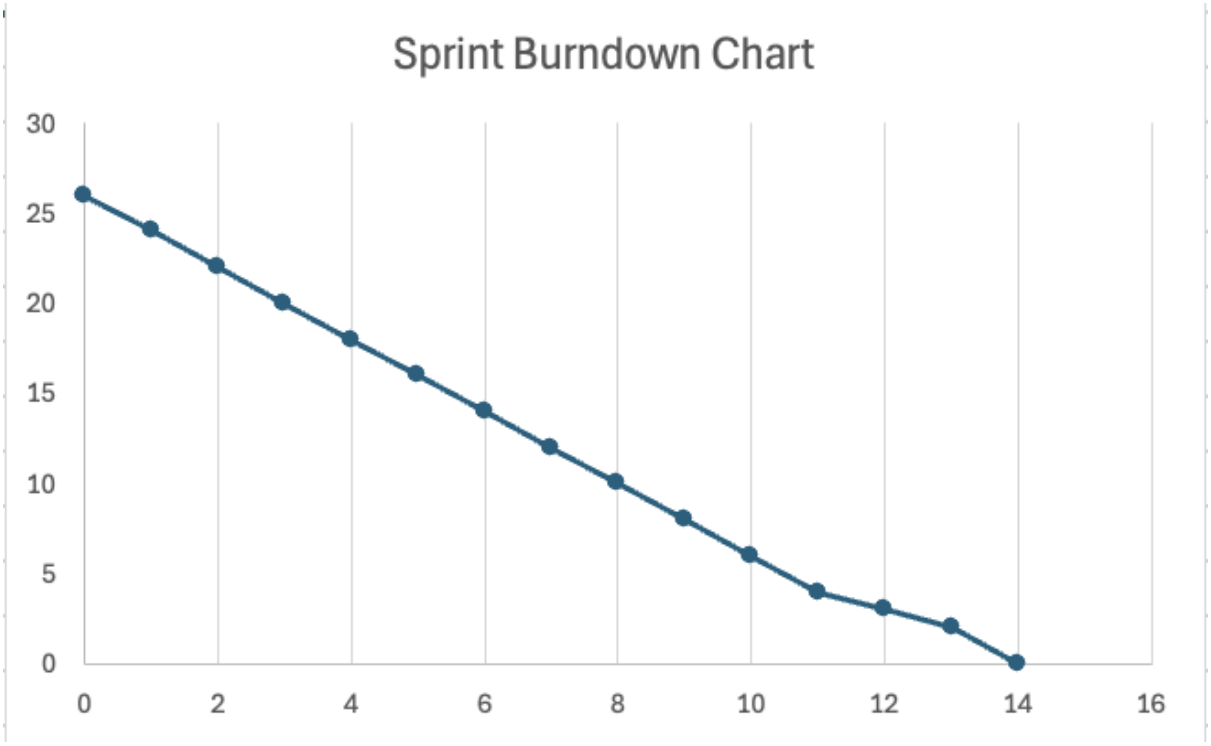
Outcome:

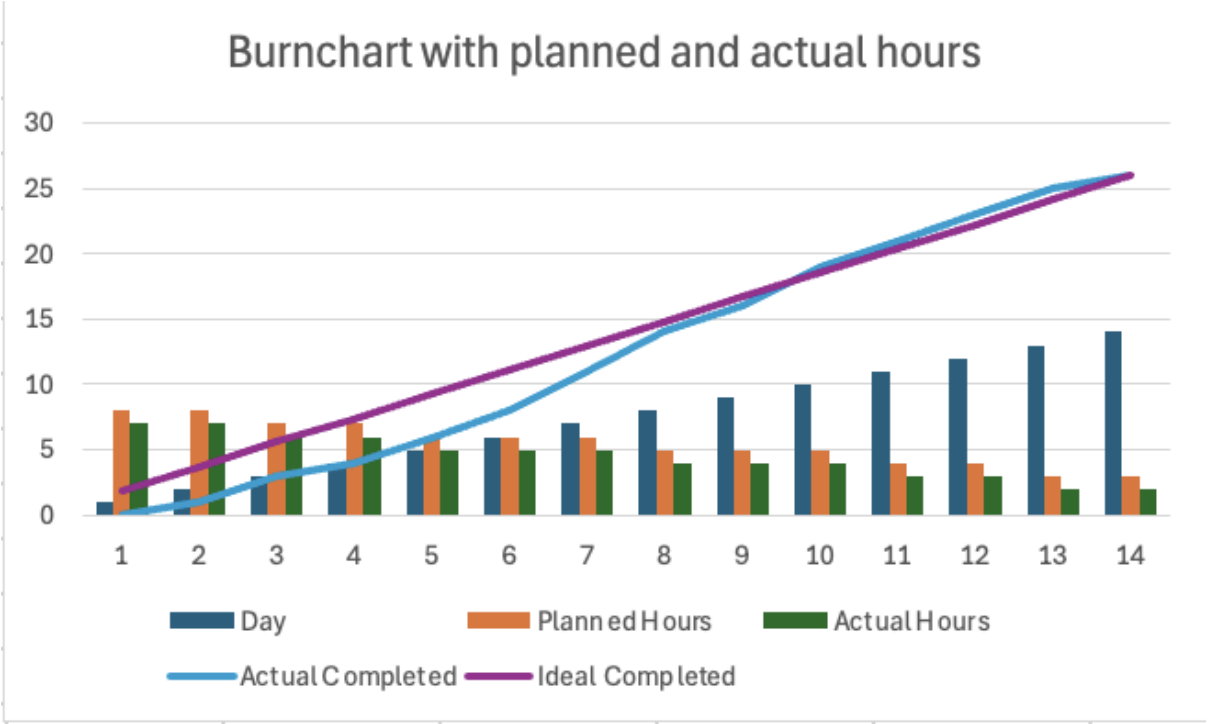
- The team successfully delivered all planned features: Voting system, Share Poll via Link, Comments, Email Notifications, and Dark Mode toggle.

- Stakeholders highlighted the collaborative voting flow as a strong addition, though some UX refinements were suggested.
- Minor challenges were encountered with comment delays (due to polling refresh cycle), email notifications being flagged as spam, and UI inconsistencies in Dark Mode styling.
- Collaboration between frontend and backend continued to improve, but integration testing still revealed mismatched assumptions in poll closure logic.

Burndown chart

Total of 26 points from user stories.





5.4. Sprint 4 – Premium & External APIs

5.4.1. Sprint planning

Sprint Planning Meeting: June 1st
Duration: 2.5 hours

Sprint Goal: Implement premium subscription functionality and integrate external APIs (Stripe, AI suggestions, Amadeus, Hotelbeds, Google Maps).

Sprint Duration: June 1st – June 15th (2 weeks)

Sprint Backlog

User Story 15 (ID 15) – Premium subscription via Stripe (checkout + success email)

- **Tair:** Frontend – Build checkout UI and integrate Stripe widget. Handle subscription state in UI, redirect flows.
- **Zhalgas:** Backend – Implement Stripe Checkout session creation and webhook for subscription activation; send confirmation email.

User Story 16 (ID 16) – Weekly AI trip suggestions (auto-enabled for premium)

- **Tair:** Frontend – UI for displaying AI trip suggestions. Client-side handling (show/hide based on premium status).
- **Zhalgas:** Backend – Implement weekly cron job/service that generates AI trip suggestions; link with premium users.

User Story 17 (ID 17) – Flight search (Amadeus API integration)

- **Tair:** Frontend – UI for searching flights (origin, destination, dates). Implement validation of search inputs and render results.
- **Zhalgas:** Backend – Integrate Amadeus API, manage OAuth token, implement /flights/search endpoint, handle errors.

User Story 18 (ID 18) – Hotel search (Hotelbeds API integration)

- **Tair:** Frontend – UI for searching hotels (destination, dates, guests). Client-side validation and rendering hotel results.
- **Zhalgas:** Backend – Integrate Hotelbeds API, implement /hotels/search endpoint with error handling.

User Story 19 (ID 19) – Google Maps integration

- **Tair:** Frontend – Add map view with trip markers and routes. Ensure correct rendering of dynamic markers and routes.
- **Zhalgas:** Backend – Provide location data API endpoints for frontend integration.

Estimation: ~34 Story Points

Roles

- **Tair –** Frontend Developer (UI/UX consistency, integration with maps & Stripe,validation, premium state handling, results rendering)
- **Zhalgas –** Backend Developer (API integrations, subscription logic, automation**)**

Scrum Master: Tair

Responsibilities

- **Tair:** Oversees UI/UX consistency, Stripe/Maps integration, ensures flows are aligned with backend APIs. Focuses on input validation, handling subscription state in UI, managing secure flows for external API results.
- **Zhalgas:** Responsible for backend integrations with Stripe, Amadeus, Hotelbeds; webhook handling; automation of AI trip suggestions.

Scrum Master (Tair):

- Facilitates Sprint Planning meetings and ensures all tasks are clearly assigned.
- Monitors progress daily and resolves blockers.
- Enforces Agile principles and Definition of Done.

Next Steps

- Development starts immediately after Sprint Planning.
- Daily stand-ups on Discord/Telegram.
- Sprint Review & Retrospective scheduled for June 4th with demo of Stripe subscription flow, AI trip suggestions, flight/hotel search, and Google Maps integration.

The Sprint 4 backlog is derived from the Product Backlog (Section 4.3). Given the sprint goal *Premium Features & External Integrations*, the following user stories were selected: US15, US16, US17, US18, US19.

These items constitute the Sprint Backlog for Sprint 4.

ID	User Story	Estimation (SP)	Priority
15	**As a premium user, I want to subscribe via Stripe, so that I can unlock premium features.**	8	15
16	**As a premium user, I want to receive weekly AI trip suggestions by email, so that I get inspiration.**	5	16
17	**As a user, I want to see flight options via the Amadeus API, so that I can align my trip with available transport.**	8	17
18	**As a user, I want to see hotel options via the Hotelbeds API, so that I can choose accommodations within my budget.**	8	18
19	**As a user, I want Google Maps integration, so that I can visualize routes and destinations.**	5	19

Tasks from User Stories

User Story 15 (ID 15): Implement premium subscription via Stripe

- Task 15.1 – Tair: Build Stripe checkout UI.
- Task 15.2 – Tair: Handle subscription state changes in frontend.
- Task 15.3 – Zhalgas: Implement backend endpoint to create Stripe session.
- Task 15.4 – Zhalgas: Implement Stripe webhook to activate subscription.
- Task 15.5 – Zhalgas: Send success email after payment.

User Story 16 (ID 16): Weekly AI trip suggestions (premium only)

- Task 16.1 – Tair: Design UI for AI trip suggestion display.
- Task 16.2 – Tair: Implement frontend toggle for premium vs free users.
- Task 16.3 – Zhalgas: Implement backend cron job to generate weekly suggestions.
- Task 16.4 – Zhalgas: API endpoint to fetch AI trip suggestions.

User Story 17 (ID 17): Flight search (Amadeus API)

- Task 17.1 – Tair: Build flight search UI.
- Task 17.2 – Tair: Validate inputs and render results.
- Task 17.3 – Zhalgas: Integrate Amadeus API (token handling, query building).
- Task 17.4 – Zhalgas: Implement /flights/search endpoint with error handling.

User Story 18 (ID 18): Hotel search (Hotelbeds API)

- Task 18.1 – Tair: Build hotel search UI.
- Task 18.2 – Tair: Validate hotel search inputs and render results.
- Task 18.3 – Zhalgas: Integrate Hotelbeds API.
- Task 18.4 – Zhalgas: Implement /hotels/search endpoint with error handling.

User Story 19 (ID 19): Google Maps integration

- Task 19.1 – Tair: Add Google Maps component with markers.
- Task 19.2 – Tair: Ensure proper rendering of routes and dynamic markers.
- Task 19.3 – Zhalgas: Provide backend endpoints with trip location data.

5.4.2. Sprint Development

User Story 15 (ID 15): Implement premium subscription via Stripe

Title: Premium Subscription (Checkout → Activation → Success Email)

As a user, I want to purchase a premium subscription via Stripe checkout,
so that I can unlock premium features and automatically receive confirmation.

Acceptance Criteria

A. Checkout Flow

- User clicks **Get Premium** on Account page → redirected to Stripe Checkout.
- Payment form securely handled by Stripe (no sensitive data stored locally).
- On successful payment → user redirected back to /success.

B. Subscription Activation

- Backend receives Stripe **webhook** confirming payment.
- System validates webhook signature to prevent tampering.
- User record updated in DB: is_subscribed = true, role = premium.
- Subscription state available immediately in frontend (premium features unlocked).

C. Success Notification

- Confirmation email sent automatically after successful payment.
- Email includes: payment confirmation, subscription start date, list of premium features.
- Account page displays success → “Subscription activated successfully.”

D. Error Handling

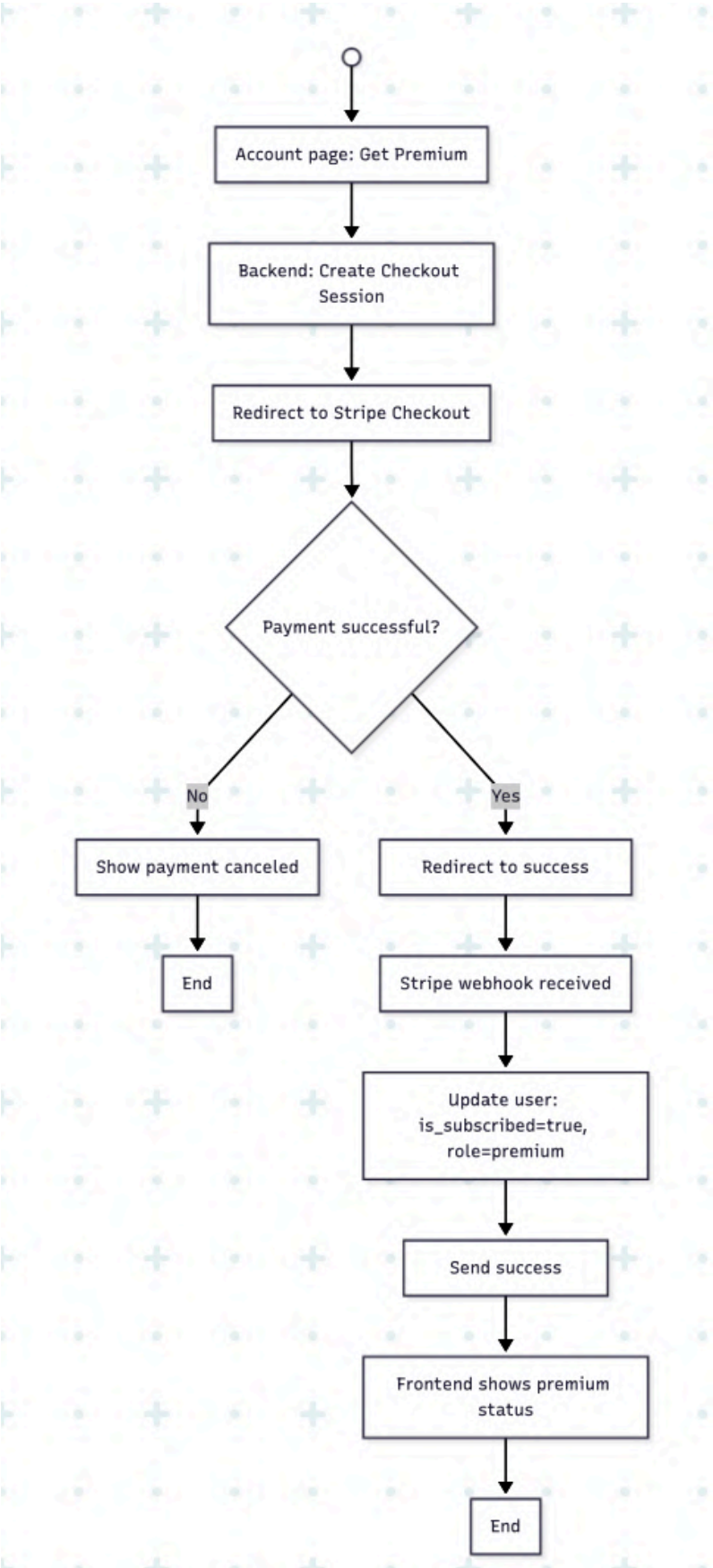
- Failed or canceled payments return proper error on frontend.
- If user exits checkout without completing payment → no DB changes applied.
- Invalid/missing webhook signature requests rejected with security error.

User Experience

- /success page clearly shows active premium status.
- Account page updates dynamically to reflect premium badge or similar visual marker.
- Stripe flow fully integrated without breaking existing login/session.

Done Criteria

- Stripe Checkout integrated end-to-end.
- Webhook handling secured, tested, and verified.
- Subscription state persisted in DB and reflected in frontend.
- Confirmation email reliably sent on success.
- Covered by unit tests (checkout creation, webhook handling, email sending) and e2e tests (payment → account upgrade).


Account page: Get Premium
Backend: Create Checkout Session
Redirect to Stripe Checkout
Payment successful?
No
Yes
Show payment canceled
End
Redirect to success
Stripe webhook received
Update user: is_subscribed=true, role=premium
Send success
Frontend shows premium status
End

User Story 16 (ID 16): Weekly AI Trip Suggestions (Premium only)
Title: Weekly AI Trip Suggestions (Automatic → Email → Favorites Integration)

As a premium user, I want to receive weekly AI-generated trip suggestions,
so that I can discover new travel ideas and save them directly to my account.

Acceptance Criteria

A. Eligibility & Access

- Only users with an active premium subscription have access.
- A weekly AI trip plan is automatically enabled when a user activates premium.
- Users can manually enable or disable the feature in the Account page.
- On enable/disable → a confirmation email is sent.

B. Delivery & Display

- A **Trip of the Week** block is displayed at the top of the Favorites page for premium users.
- The Planner sidebar can also show the most recent weekly suggestion.
- An email is sent weekly with a summary of the suggested trip and a link to view it in Favorites.
- Users can click **Save Trip** to store the suggestion in Favorites and download a PDF summary.

C. Error Handling

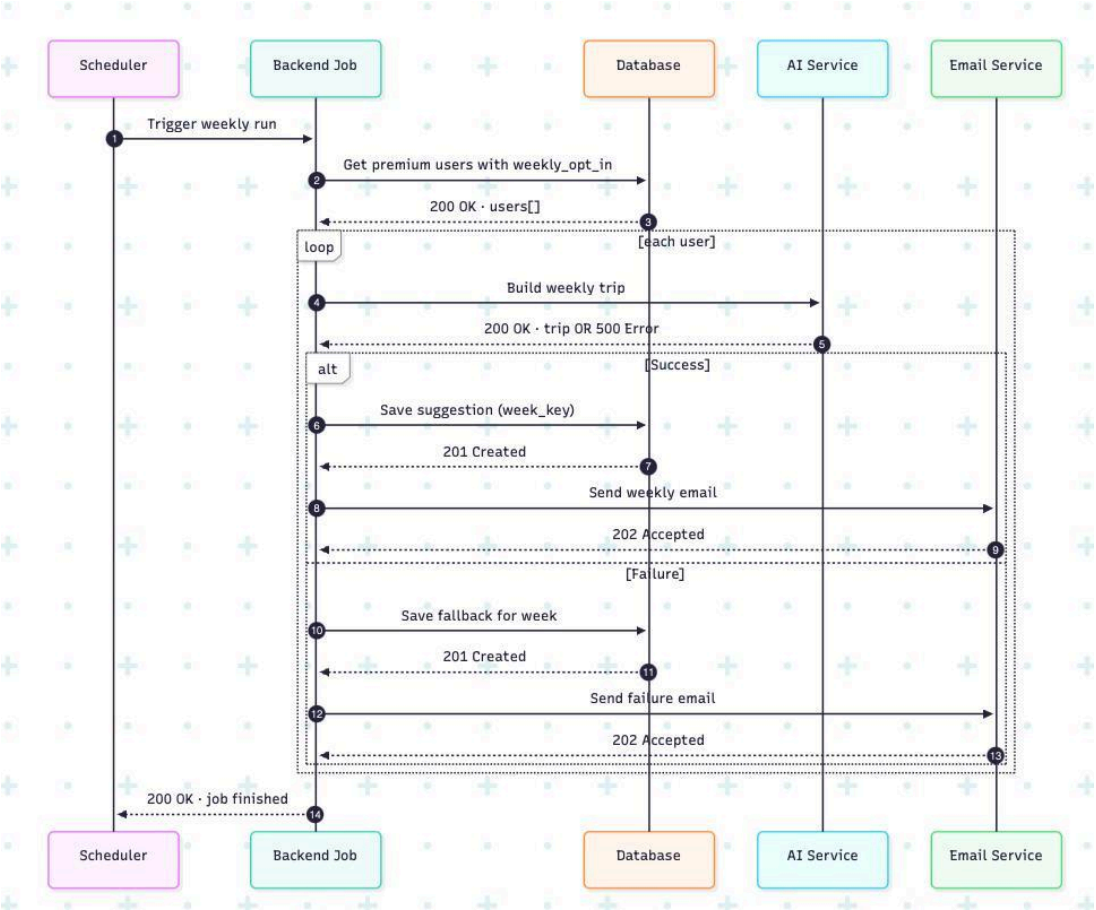
- Non-premium users or those with the feature disabled receive a proper status when calling the API.
- If the job fails → users receive a fallback email explaining that no trip was generated this week.
- Only one valid trip suggestion is stored per user per week; duplicate generation is prevented.

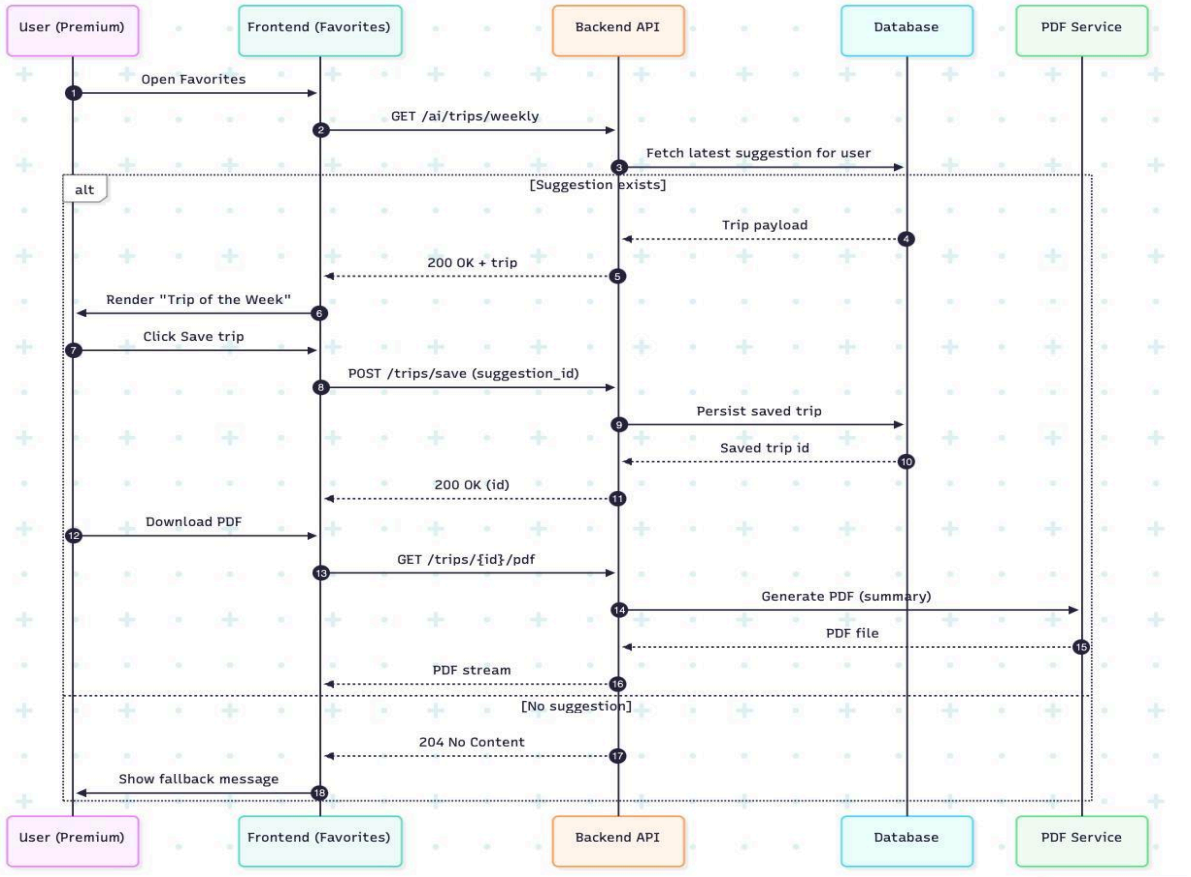
User Experience

- In **Account**: toggle for Weekly AI trip plan (on/off), visible only for premium users.
- In **Favorites**: “Trip of the Week” card with options to View, Save, and Download PDF.
- In **Email**: clear summary of the weekly trip with a link to the full details.

Done Criteria

- Weekly cron job successfully generates suggestions for all eligible premium users.
- API endpoint /ai/trips/weekly returns the latest valid suggestion with proper access control.
- Favorites page displays the weekly suggestion for premium users.
- Save Trip functionality works consistently with weekly suggestions.
- Emails are reliably sent when: feature is toggled on/off, and when new weekly suggestions are generated.
- Covered by unit tests (eligibility, scheduling, content validation) and end-to-end tests (premium activation → trip generated → displayed in UI → saved to Favorites).


Scheduler
Backend Job
Database
AI Service
Email Service
1
Trigger weekly run
2
Get premium users with weekly_opt_in
3
200 OK · users[]
loop
[each user]
4
Build weekly trip
5
200 OK · trip OR 500 Error
alt
[Success]
6
Save suggestion (week_key)
7
201 Created
8
Send weekly email
9
202 Accepted
[Failure]
10
Save fallback for week
11
201 Created
12
Send failure email
13
202 Accepted
14
200 OK · job finished
Scheduler
Backend Job
Database
AI Service
Email Service



User Story 17 (ID 17): Flight Search (Amadeus API)

Title: Flight Search (Real API Integration with Amadeus)

As a user, I want the system to automatically fetch flight options for the destination extracted from my AI trip summary, **so that** I can compare actual flight options and plan my trip.

Context

In previous sprints, the flight search feature used **mock stubs** to simulate results.
In this sprint, we connect to the **real Amadeus API** (OAuth2 + Flight Offers endpoint) so users see **live flight data instead** of mocked responses.

Acceptance Criteria

A. Search Inputs

- Origin and Destination are **parsed automatically from AI Trip Summary**.
- No dates or extra filters required at this stage.

B. Backend Integration

- Backend retrieves **OAuth2 token** from Amadeus (client credentials).

- Endpoint /flights/search?origin=X&destination=Y queries Amadeus Flight Offers API.
- Token cached and refreshed when expired.
- Results normalized into consistent format (carrier, flight number, departure & arrival, duration, price if available).

C. Search Results

- For each route:
    - Airline + flight number
    - Departure airport + time
    - Arrival airport + time
    - Duration and number of stops
    - Price if provided by Amadeus
- If multiple options exist, return top results sorted by lowest price (if available).

D. Error Handling

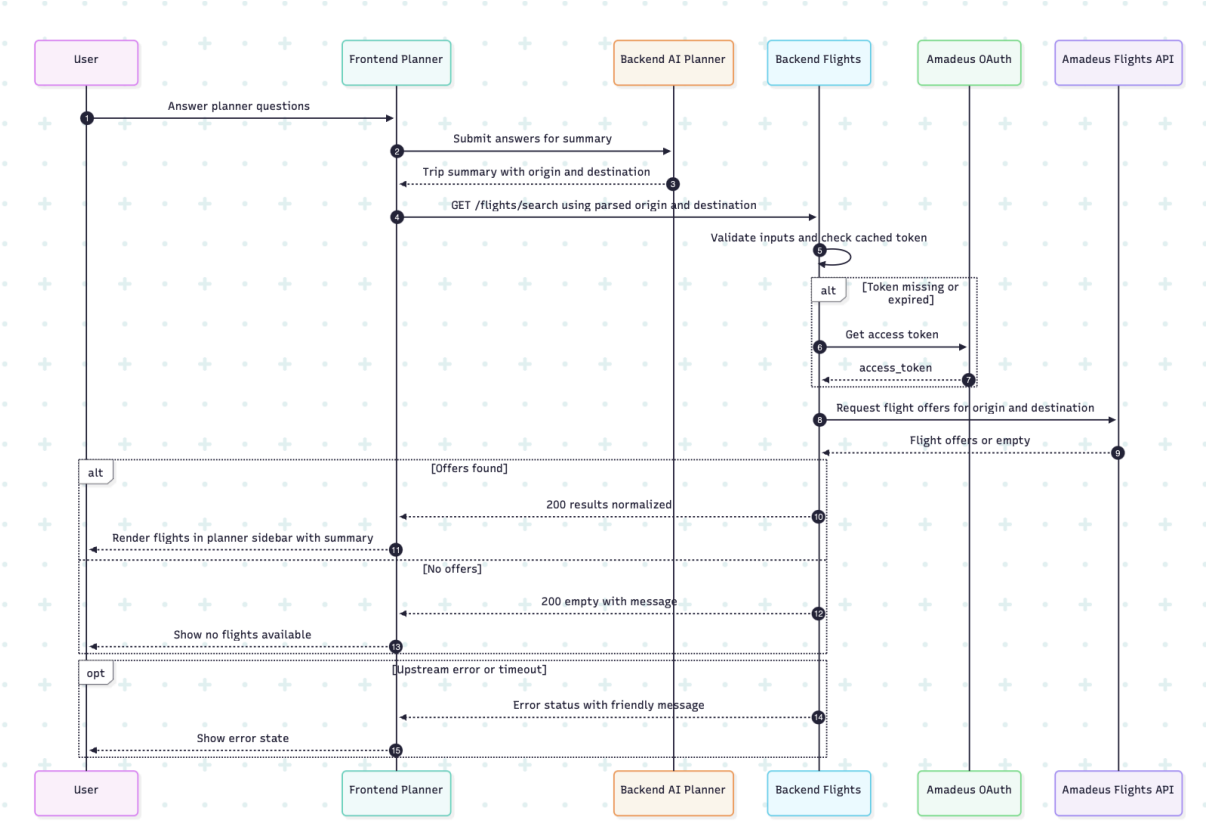
- Missing origin/destination → 400 with clear error.
- Amadeus 401 → refresh token, retry once; if still failing → 502.
- No routes found → 200 with empty array and message “No flights available.”
- Network/timeout → 504 with user-friendly message.

User Experience

- User enters **origin and destination** in a simple form.
- On submit, the results list is shown (flights grouped by airline or sorted by price).
- Loading state during search; proper error/empty messages.

Done Criteria

- /flights/search works end-to-end with Amadeus.
- Only origin and destination are required inputs.
- Real data replaces mock stubs.
- Results displayed correctly in the frontend.
- Covered by unit tests (validation, token refresh) and e2e tests (search → results).



User Story 18 (ID 18): Hotel Search (Hotelbeds API)

Title: Hotel Search (Destination from AI Trip Summary + Real API Integration)

As a user, I want the system to automatically fetch hotel options for the destination extracted from my AI trip summary,
so that I can see real accommodations matching my planned trip.

Context

In previous sprints, hotel search was implemented with **mock stubs**.
In this sprint, we integrate with the **real Hotelbeds API** so users see **live hotel data** instead of simulated results.

Acceptance Criteria

A. Search Inputs

- Destination (city or location code) is **parsed automatically from AI Trip Summary**.
- Default dates and budget are inferred from Trip Summary preferences (if available).

B. Backend Integration

- Backend connects to **Hotelbeds API** using secure credentials.
- Endpoint /hotels/search?destination=XXX queries Hotelbeds with parsed destination.
- API keys and credentials stored securely server-side, never exposed to frontend.
- Results normalized into consistent format (hotel name, rating, location, price).

C. Search Results

- For each hotel result:
    - Hotel name
    - Star rating (if available)
    - Location (city, address, geocoordinates)
    - Price per night (currency + total)
    - Room options summary (basic info if provided by Hotelbeds)
- Results sorted by lowest price by default; alternative sort by rating available.

D. Error Handling

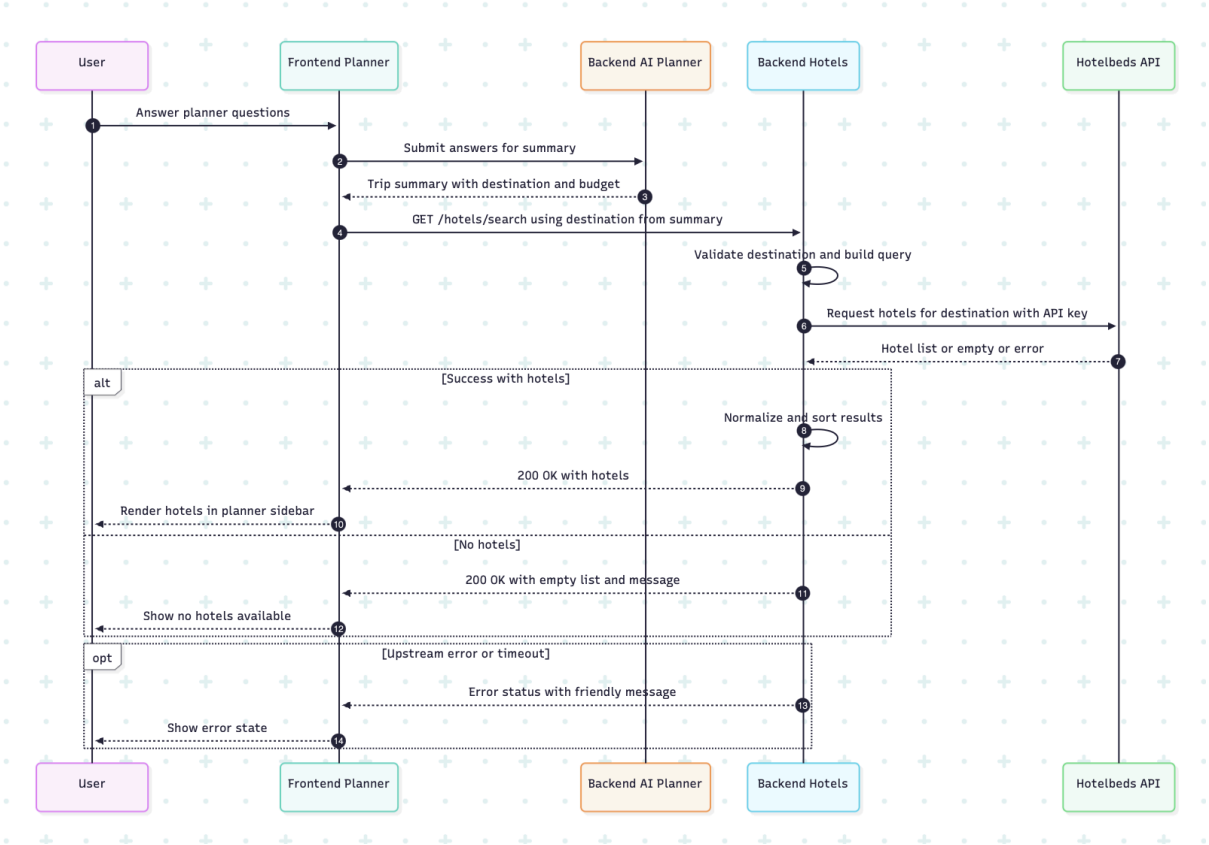
- Missing destination in trip summary → 400 with clear error.
- Hotelbeds 401/403 → backend retries once; if still failing → 502.
- No hotels found → 200 with empty list and message "No hotels available."
- Network/timeout → 504 with user-friendly message.

User Experience

- After finishing AI Planner Q&A, the Trip Summary auto-triggers a hotel search along with flights.
- Results list shown in the Planner sidebar under the "Hotels" section.
- Clear loading state while fetching; proper error/empty messages if nothing is found.
- User can click a hotel to expand details (rating, location, price breakdown).

Done Criteria

- /hotels/search works end-to-end with Hotelbeds.
- Destination extracted automatically from AI Trip Summary.
- Real data replaces mock stubs.
- Results displayed correctly in frontend Planner sidebar.
- Covered by unit tests (validation, API integration, error handling) and e2e tests (summary → search → results).



User Story 19 (ID 19): Google Maps Integration

Title: Interactive Map in Planner (Markers, Routes, Dynamic Updates)

As a user, I want to see my trip locations and routes on an interactive Google Map in the Planner, **so that** I can understand the itinerary visually while planning my trip.

Context

Google Maps integration is added only to the **Planner**, where the AI builds the Trip Summary. The map is dynamic (markers + routes) and cannot be stored statically, so it is **not included in Favorites or Saved Trips**.

Acceptance Criteria

A. Map Rendering (Planner only)

- A Google Maps component is displayed in the Planner, next to the Trip Summary.
- Map initializes to show all trip points (origin, destination, POIs) with fit-to-bounds.
- Markers are drawn for each point; clicking a marker shows an info window with name/notes.

B. Routes & Polylines

- If Trip Summary has ordered stops, draw a polyline in that order.

- Driving route by default; if type = flight, show dashed straight line.
- Bounds updated to include all points.

D. Backend Endpoint

- /summary/{id}/locations returns normalized location data: name, lat, lng, type (origin, poi, destination), order.
- Used by Planner frontend to feed Google Maps.

E. Error Handling

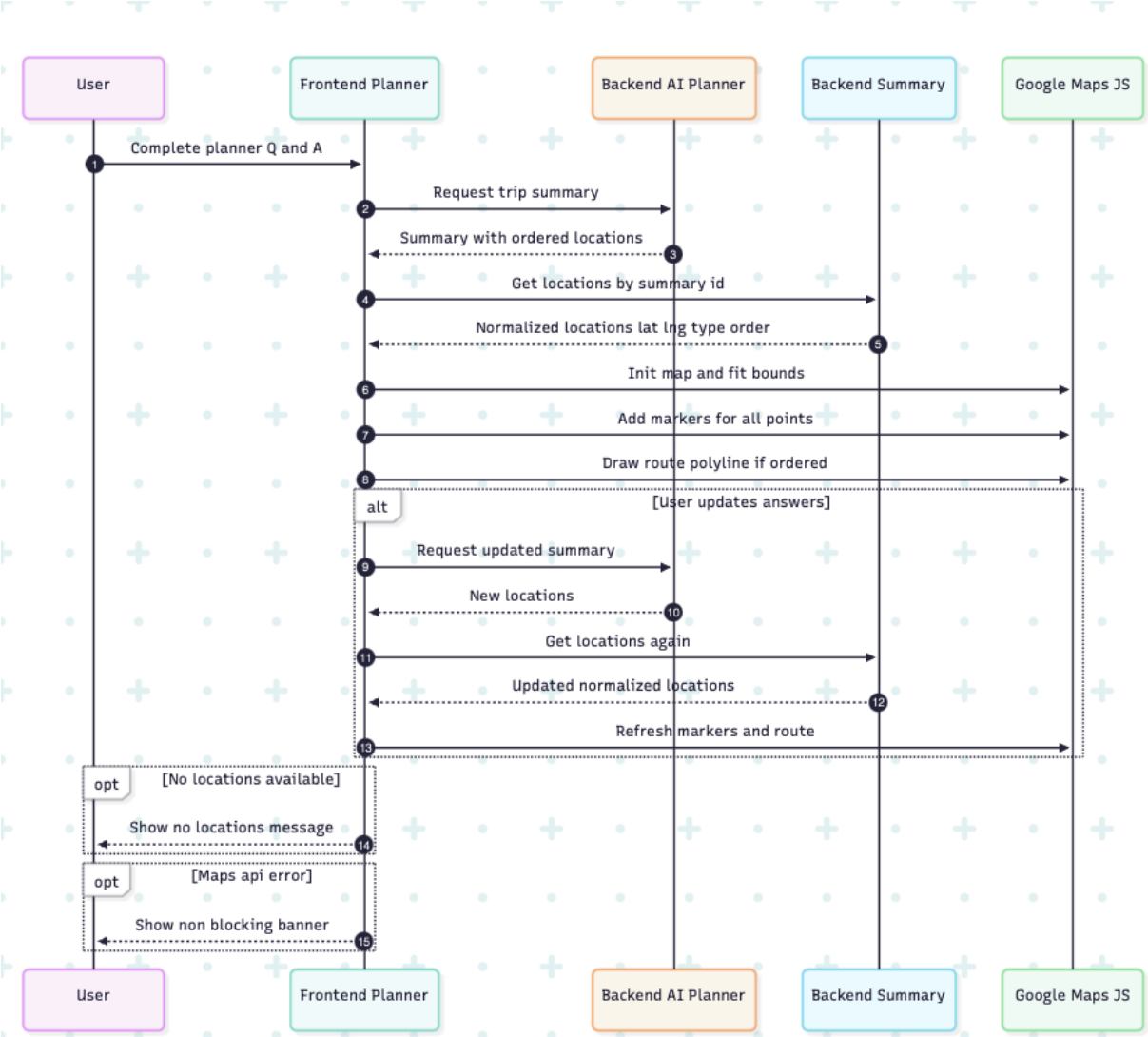
- If no locations are available → map area shows “No locations to display.”
- Invalid coordinates ignored gracefully.
- If Google Maps API fails (e.g., key error) → show a non-blocking banner, but Planner still works.

User Experience

- After finishing AI Q&A, Trip Summary is shown + interactive map with points and route.

Done Criteria

- Map renders correctly in Planner with dynamic markers and routes.
- Covered by unit tests (data normalization) and e2e tests (summary → map render → update).



Sprint 4 – Testing Document
Test Summary

During **Sprint 4**, testing activities focused on validating the **premium subscription flow and external API integrations of** the Trip DVisor platform.
The goal was to ensure that users can:

- successfully purchase a premium subscription via Stripe,
- automatically receive and manage weekly AI trip suggestions,
- retrieve real flight offers through the Amadeus API,
- search for hotels through the Hotelbeds API,
- and view trip locations dynamically on Google Maps in the Planner.

Scope included subscription activation, email notifications, cron-based weekly AI suggestions, live external data integrations (flights, hotels), and dynamic maps rendering.
All planned test cases were executed, results documented, and recommendations for improvements identified.

Test Activities

1. Premium Subscription (Stripe)

- **TC030:** Verify Stripe checkout redirection.
    - **Result:** Clicking "Get Premium" opens a secure Stripe checkout page.
    - **Method:** Manual UI testing.
- **TC031:** Confirm subscription activation via webhook.
    - **Result:** After successful payment, user status updates (is_subscribed=true, role=premium).
    - **Method:** Automated webhook tests + DB inspection.
- **TC032:** Success email delivery.
    - **Result:** User receives confirmation email with subscription details.
    - **Method:** Manual email check + backend log inspection.
- **TC033:** Cancel/failed payment handling.
    - **Result:** No DB changes applied, frontend shows "Payment canceled."
    - **Method:** Manual UI testing.

2. Weekly AI Trip Suggestions (Premium Only)

- **TC034:** Enable/disable weekly AI plan in Account.
    - **Result:** Toggle works; confirmation email sent on change.
    - **Method:** Manual UI testing.
- **TC035:** Weekly cron job execution.
    - **Result:** AI trip suggestion generated and stored for each premium user.
    - **Method:** Simulated cron trigger + DB verification.
- **TC036:** Display Trip of the Week in Planner/Favorites.
    - **Result:** Premium users see weekly suggestions; Save Trip works.
    - **Method:** Manual UI testing.
- **TC037:** Fallback handling when API data is unavailable.
    - **Result:** User receives "no trip generated" email; system stores fallback.
    - **Method:** Automated mock failure injection.

3. Flight Search (Amadeus API)

- **TC038:** Auto-extraction of origin/destination from AI summary.
    - **Result:** Flights queried based on Trip Summary, no manual input required.
    - **Method:** Automated + manual validation.
- **TC039:** Successful API call returns real flight offers.
    - **Result:** List includes airline, flight number, times, duration, price.
    - **Method:** API integration tests + UI display check.
- **TC040:** Error handling on API failure.

    - **Result:** Friendly error message shown; retry logic works.
    - **Method:** Simulated API errors.

4. Hotel Search (Hotelbeds API)

- **TC041:** Destination parsing from Trip Summary.
    - **Result:** Hotels queried automatically for destination city.
    - **Method:** Automated validation.
- **TC042:** Successful API call returns hotel options.
    - **Result:** List includes hotel name, rating, location, price.
    - **Method:** API integration tests + UI rendering.
- **TC043:** Empty/no results handling.
    - **Result:** Message "No hotels available" displayed.
    - **Method:** Manual testing with small destinations.

5. Google Maps Integration

- **TC044:** Render map with origin/destination markers.
    - **Result:** Map displays points from Trip Summary.
    - **Method:** Manual UI testing.
- **TC045:** Route polyline drawing.
    - **Result:** Ordered stops connected; flights shown as dashed lines.
    - **Method:** Manual UI validation.
- **TC046:** Dynamic updates on summary change.
    - **Result:** Map refreshes markers/routes when Trip Summary updates.
    - **Method:** Manual UI testing.

Test Results

- All **core features** (Stripe subscription, weekly AI trips, flight search, hotel search, Google Maps integration) passed functional testing.
- **No blocking defects** identified.
- **Minor issues:**
    - Stripe webhook had occasional duplicate event calls → mitigated by idempotency check (to refine further).
    - Google Maps performance slowed with >15 POI markers → optimization scheduled for Sprint 5.
    - Weekly AI trip email formatting inconsistent on mobile clients → polish planned for Sprint 5.

5.4.3. Sprint Review

Sprint Review Meeting: June 15th
Duration: 2 hours
Attendees:

- Tair (Scrum Master & Frontend Developer)

- Zhalgas (Backend Developer)
- Stakeholders (Clients / End Users)

Agenda

- Presentation of Completed User Stories and Deliverables
- Demonstration of Implemented Functionalities (Stripe, AI suggestions, Flights, Hotels, Google Maps)
- Feedback and Evaluation from Stakeholders
- Discussion of Achievements and Challenges
- Confirmation of Sprint Goals and Objectives

Outcome

- Successfully implemented **Premium Subscription via Stripe**, including secure checkout, webhook handling, and confirmation emails.
- Delivered **Weekly AI Trip Suggestions**, automatically generated for premium users and displayed as "Trip of the Week" in Planner/Favorites, with opt-in/out in Account.
- Integrated **Amadeus API** for live flight search based on origin/destination parsed from AI Trip Summary.
- Integrated **Hotelbeds API** for live hotel options based on destination parsed from AI Trip Summary.
- Implemented **Google Maps integration** in Planner, showing dynamic markers and routes from AI Trip Summary.

Minor Issues:

- Stripe webhook occasionally triggered duplicate events → added idempotency check (further refinement planned).
- Weekly AI trip email layout had formatting inconsistencies on mobile clients (to be polished in Sprint 5).
- Google Maps performance slowed with more than 15 markers (optimization scheduled for Sprint 5).

Backlog Refinement

This backlog refinement reflects only the user stories addressed during **Sprint 4**:

ID	User Story	Estimation (SP)	Priority
15	**As a premium user, I want to subscribe via Stripe, so that I can unlock premium features.**	8	15
16	**As a premium user, I want to receive weekly AI trip suggestions by email, so that I get inspiration.**	5	16

17	**As a user, I want to see flight options via the Amadeus API, so that I can align my trip with available transport.**	8	17
18	**As a user, I want to see hotel options via the Hotelbeds API, so that I can choose accommodations within my budget.**	8	18
19	**As a user, I want Google Maps integration, so that I can visualize routes and destinations.**	5	19

5.4.4. Sprint Retrospective

Sprint Retrospective Meeting: June 15th
Duration: 1.5 hours
Attendees:

- Tair (Scrum Master & Frontend Developer)
- Zhalgas (Backend Developer)

Agenda

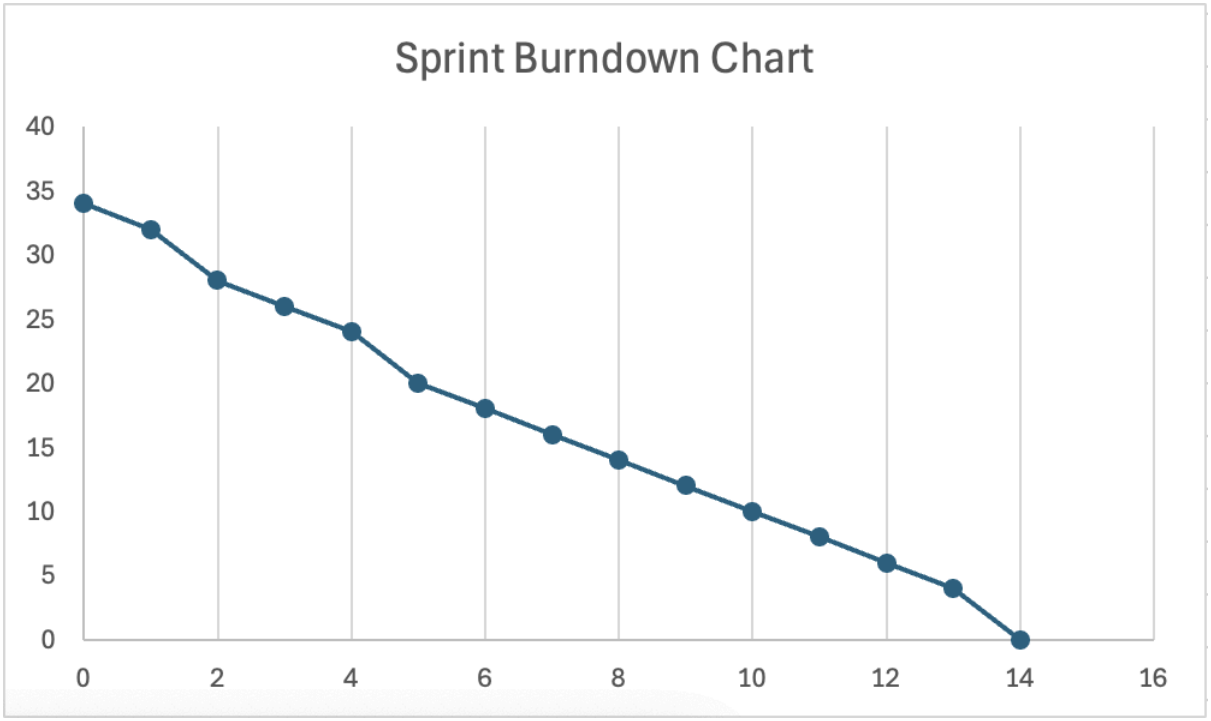
- Reflections on Achievements and Challenges
- Identification of Action Items for Process Improvement

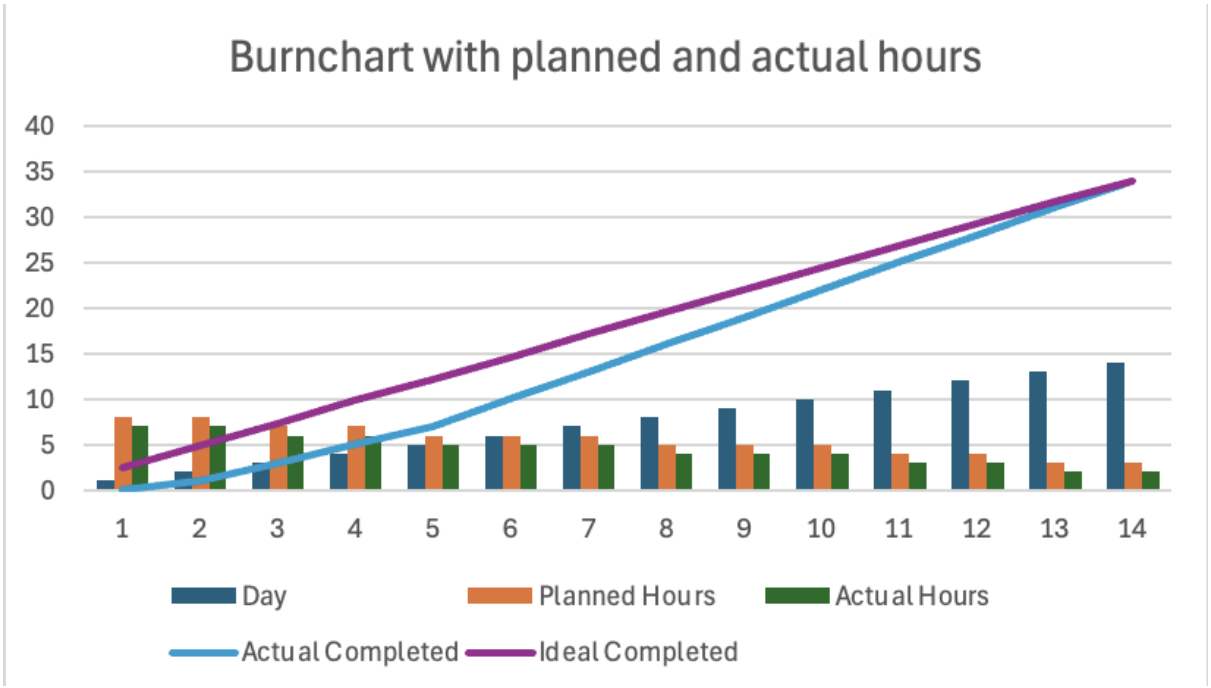
Outcome

Achievements:

- All planned features were successfully delivered:
    - Premium subscription via **Stripe** (checkout, webhook, confirmation email).
    - **Weekly AI Trip Suggestions** for premium users (cron job, email delivery, Trip of the Week).
    - Live **Flight Search** integration with Amadeus API.
    - Live **Hotel Search** integration with Hotelbeds API.
    - **Google Maps integration** in the Planner for dynamic route visualization.
- The transition from mock data to real APIs significantly increased the platform's value.
- Collaboration between frontend and backend improved, particularly in handling external APIs.

Burndown chart
Total of 35 points from user stories.





5.5 Sprint 5 – Administration, Optimization & Release

5.5.1. Sprint Planning

Sprint Planning Meeting: June 15th
Duration: 2.5 hours

Sprint Goal: Finalize administration features, add monitoring tools, and optimize performance to complete the project

Sprint Duration: June 15th – July 1st (2 weeks)

Sprint Backlog

User Story 20 (ID 20): Admin Dashboard (users, trips, votes, sessions) – 8 SP

- **Tair (Frontend):** Build Admin Dashboard UI (tables, filters, management actions). Implement role-based access and secure navigation for admin-only features.
- **Zhalgas (Backend):** Develop backend endpoints for managing users, trips, votes, and sessions.

User Story 21 (ID 22): Analytics Dashboard

- Task 22.1 – Tair: Design analytics UI (charts/graphs).
- Task 22.2 – Tair: Connect frontend visualizations to backend APIs.
- Task 22.3 – Zhalgas: Implement analytics endpoints with aggregated data.

User Story 22 (ID 23): Scalability (support 1000+ concurrent users) – 8 SP

- **Zhalgas:** Perform backend load testing, DB optimization (indexes, connection pooling).
- **Tair:** Optimize frontend requests (debouncing, pagination, lazy loading).
- **Tair:** Test dashboard performance with large datasets.

Estimation: ~21 Story Points

Roles

- **Tair – Frontend Developer** (UI/UX consistency, dashboard design,real-time session monitoring, optimizations, secure flows)
- **Zhalgas – Backend Developer** (admin APIs, monitoring, scalability, DB optimization)

Scrum Master: Tair

Responsibilities

- **Tair:** Ensure Admin Dashboard UI is clear, consistent, and performant. Implement secure flows, optimize frontend data handling, and real-time updates.
- **Zhalgas:** Deliver backend APIs for admin and monitoring, ensure DB scalability and performance improvements.

Scrum Master (Tair):

- Facilitates Sprint Planning meetings and ensures clear task ownership.
- Tracks progress daily and resolves blockers.
- Enforces Agile practices and the Definition of Done.

Next Steps

- Development begins immediately after Sprint Planning.
- Daily stand-ups on Discord/Telegram.
- Sprint Review & Retrospective scheduled for June 18th with demonstration of admin dashboard, monitoring, and scalability testing results.

The Sprint 5 backlog is derived from the Product Backlog (Section 4.3). Use Given the sprint goal *Administration & Scalability*, the following user stories were selected: US20, US21, US22.

These items constitute the Sprint Backlog for Sprint 5.

ID	User Story	Estimation (SP)	Priority
20	**As an admin, I want a dashboard to manage users, trips, and voting sessions, so that I can oversee the system.**	8	21
21	**As an admin**, I want a dashboard with key analytics and charts, **so that** I can understand usage trends and platform health.	5	22
22	**As a product owner, I want the platform to handle 1000+ concurrent users, so that it performs well under load.**	8	24

Tasks from User Stories

User Story 20 (ID 20): Admin Dashboard

- Task 20.1 – Tair: Build Admin Dashboard UI (tables for users, trips, votes, sessions).
- Task 20.2 – Tair: Implement role-based access on frontend.
- Task 20.3 – Zhalgas: Implement backend endpoints for admin CRUD operations.

User Story 21 (ID 22): Analytics Dashboard

- Task 22.1 – Tair: Design analytics UI (charts/graphs).
- Task 22.2 – Tair: Connect frontend visualizations to backend APIs.
- Task 22.3 – Zhalgas: Implement analytics endpoints with aggregated data.

User Story 23 (ID 23): Scalability

- Task 23.1 – Zhalgas: Run load tests, optimize DB (indexes, connection pooling, caching).
- Task 23.2 – Tair: Optimize frontend data handling (pagination, lazy loading).
- Task 23.3 – Tair: Validate UI performance with large datasets.

5.5.2. Sprint Development

User Story 20 (ID 20): Admin Dashboard

Title: Admin Dashboard (Users, Trips, Votes, Sessions)

As an admin, I want to access a dedicated dashboard to view and manage users, trips, votes, and sessions,
so that I can oversee the system and maintain control over platform content.

Acceptance Criteria

A. Access & Security

- Admin Dashboard accessible only to users with role = admin.
- Non-admin users attempting access are redirected to “403 Forbidden” page.
- Secure navigation in frontend (menu link visible only for admins).

B. User Management

- List of all registered users displayed (username, email, subscription status, 2FA enabled).
- Admin can delete a user → cascade deletes associated trips and votes.
- Confirmation prompt shown before deletion.

C. Trip Management

- List of all saved trips (title, user, created date).
- Admin can delete trips → results reflected in Favorites.
- Confirmation required.

D. Voting Sessions Management

- List of all voting sessions (title, creator, status: active/closed, votes count).
- Admin can close or delete sessions.
- Actions logged in backend for auditing.

E. Sessions (Auth) Overview

- Display current active sessions (user, login time, last activity).
- Admin can revoke a session → user forced to re-login.

F. Error Handling

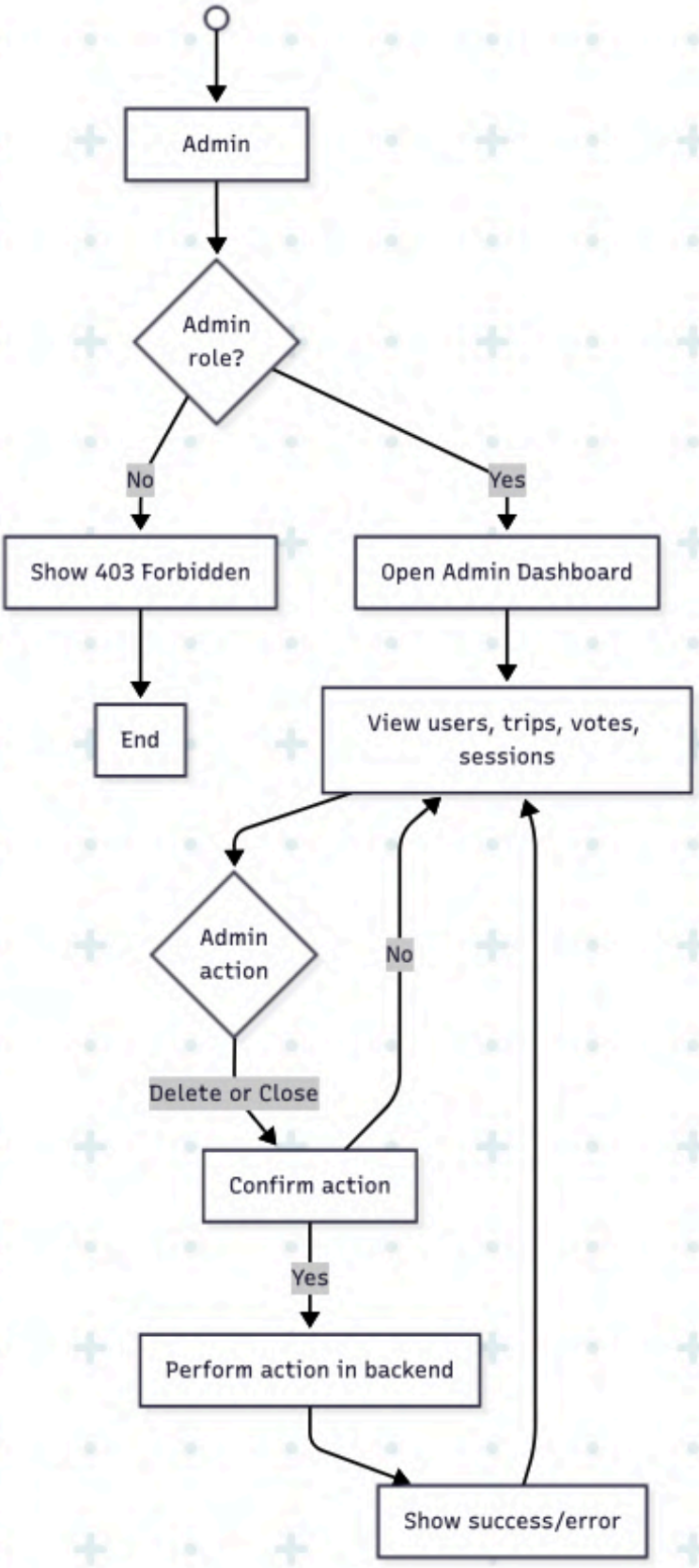
- Invalid admin requests → 403 Forbidden.
- Deletion of non-existing entity → 404 Not Found.
- Any DB errors → user-friendly errors shown in UI.

User Experience

- Admin Dashboard button visible in header only for admins.
- Dashboard uses tabbed navigation: **Users | Trips | Votes | Sessions**.
- Each tab displays a table with sorting/filtering controls.
- Actions (Delete, Close) available via buttons in each row.
- Success/failure actions show confirmation toasts.

Done Criteria

- Admin Dashboard available only to role=admin.
- UI shows four tabs: Users, Trips, Votes, Sessions.
- Admin can delete users, trips, votes, and revoke sessions.
- Confirmation prompts and notifications implemented.
- Backend endpoints secured and tested.
- Covered by unit tests (role checks, CRUD operations) and e2e tests (admin login → manage users/trips/votes/sessions).


Admin
Admin role?
No
Yes
Show 403 Forbidden
Open Admin Dashboard
End
View users, trips, votes, sessions
Admin action
No
Delete or Close
Confirm action
Yes
Perform action in backend
Show success/error

User Story 22 (ID 22): Analytics Dashboard

Title: Analytics Dashboard (KPIs, Charts, Filters)

As an admin, I want a dashboard with key analytics and charts,
so that I can understand usage trends and platform health.

Acceptance Criteria

A. Access & Security

- Dashboard is **admin-only** (role = admin).
- Non-admin access returns **403** (and UI hides the entry point).

B. KPIs & Metrics (MVP set)

- **Users:** total users, premium users, new users last 7/30 days.
- **Trips:** total saved trips, saves last 7/30 days, average trips per user.
- **Votes:** total voting sessions, active vs closed, average votes per session.
- **Email/Notification health:** sent count, bounce/error rate (if available).

C. Visualizations

- **Time series** (line/area): new users per day, saved trips per day.
- **Bar/column:** votes per session (top N), premium adoption by week.
- **Pie/donut:** premium vs free share.
- All charts support tooltips and legend toggle.

F. Performance & UX

- Dashboard initial load < **2s** on warmed cache; chart interactions < **300ms**.
- Empty states and skeleton loaders while fetching.
- No blocking errors—widgets fail independently with inline messages.

G. Error Handling

- Invalid date ranges → 400 with clear message.
- DB or aggregation failure → 502 with retry hint; widget shows recoverable error.
- Unauthorized → 401/403 and redirect from UI.

User Experience

- Entry: **Admin Dashboard → Analytics** tab.
- Layout: top row **KPI cards**, below **charts grid** (time series left, categorical right).
- Sticky filter bar (date range, plan type).
- Export buttons: **CSV** for each chart/table (client-side from payload).

Done Criteria

- Admin-only Analytics tab visible and accessible.
- KPIs and charts render correctly with filters.
- Backend aggregation endpoints implemented, secured, and cached.
- Reasonable performance under sample production-like data.
- Robust empty/error states.
- Covered by unit tests (aggregation, access control) and e2e tests (admin login → filter → charts update).

User Story 23 (ID 23): Scalability

Title: Scale to 1000+ concurrent users (perf, stability, observability)

As a platform owner, I want the system to handle high concurrent usage,
so that users experience fast, reliable performance under peak load.

Acceptance Criteria

A. Targets & SLOs

- **Throughput:** sustain **1000+ concurrent users** (simulated) with steady behavior.
- **Latency (p95):**
  - Read APIs (summary, flights, hotels, favorites): **≤ 400 ms**
  - Write APIs (save trip, vote, create poll, account ops): **≤ 700 ms**
- **Error rate:** p95 **HTTP 5xx < 1%**, timeouts < 0.5%.
- **Availability during test window:** ≥ **99.5%**.

B. Backend Performance

- **DB optimization:** add missing **indexes** on created_at, user_id, trip_id, session_id; verify query plans.
- **Connection pooling:** pool sized per CPU and DB limits; no connection exhaustion.
- **Caching:** enable short-TTL cache (60–120s) for hot reads (e.g., weekly trip, AI summary fetch, locations, admin lists).
- **N+1 elimination:** audit endpoints; replace with joins or batched queries.
- **Pagination & limits:** all list endpoints capped (e.g., 50–100 items per page).
- **Compression:** gzip/br for JSON responses over 1 KB.

C. Frontend Performance

- **Data usage:** list views use **pagination**, **infinite scroll** or **load more**; drop full-table renders.
- **Lazy loading:** defer non-critical panels (maps, charts, large tables).
- **Request hygiene: debounce** search/filters; cancel stale requests; avoid duplicate fetches via memoization/cache.
- **Bundle size:** main bundle **< 300 KB gz**; code-split admin views.

D. Load & Stress Testing

- **Scenarios:** authenticated browsing (Planner summary view), saving trip, opening Favorites, admin lists, creating a poll.
- **Profiles:** ramp to 1000 VUs, hold 15–20 min; spike test 0→1000 in <60s; soak 2h at 300 VUs.
- **Tooling:** k6/JMeter with realistic think time and mixed R/W ratios.
- **Pass criteria:** meet SLOs for p95 latency and error budget under all scenarios.

E. Observability & Safeguards

- **Metrics:** request rate, latency (p50/p95/p99), errors, DB qps, slow queries, cache hit ratio, pool usage, GC.
- **Tracing:** distributed traces for key flows (Planner → Summary → Flights/Hotels).
- **Alerts:** latency p95 breach, error rate >1%, DB CPU >80%, cache hit <70%.
- **Circuit breakers & retries:** graceful degradation on external APIs; bounded retries with backoff.
- **Rate limiting:** per-IP or per-user to protect hot endpoints.

F. Security & Reliability

- No secrets in client; env secrets rotated and scoped.
- Timeouts set per upstream; fail fast with user-friendly messages.
- Idempotency for webhook-like or repeatable writes.

User Experience

- Lists load progressively (first page fast, subsequent on demand).
- No long UI freezes; skeleton loaders indicate progress.
- Errors show inline, not blocking the whole page.

Done Criteria

- Load tests demonstrate meeting or exceeding SLOs at **1000+ concurrent users**.
- DB optimized (indexes, plans verified), connection pooling tuned, caches active with acceptable hit rate.
- Frontend implements pagination, lazy loading, debouncing across heavy views.
- Observability in place with actionable dashboards and alerts.
- Documented test results and tuning changes (before/after metrics).
- Covered by automated checks: basic perf regression test in CI (smaller profile), plus k6/JMeter scripts stored in repo.

5.5.3. Sprint Review

Sprint Review Meeting: July 1st
Duration: 2 hours
Attendees:

- Tair (Scrum Master & Frontend Developer)

- Zhalgas (Backend Developer)
- Stakeholders (Clients / End Users)

Agenda

- Presentation of Completed User Stories and Deliverables
- Demonstration of Implemented Functionalities (Admin Dashboard, Voting Sessions Monitoring, Analytics Dashboard, Scalability Tests)
- Feedback and Evaluation from Stakeholders
- Discussion of Achievements and Challenges
- Confirmation of Sprint Goals and Objectives

Outcome

- Successfully implemented **Admin Dashboard**, providing admins with visibility into users, trips, votes, and sessions, with the ability to manage and delete data.
- Delivered **Voting Sessions Monitoring**, allowing real-time observation of active and closed polls with updated statistics.
- Developed **Analytics Dashboard**, offering aggregated metrics (users, trips, votes, premium adoption) with visualizations.
- Completed **Scalability work**, including DB optimizations, caching, connection pooling, frontend pagination/lazy loading, and load testing to support 1000+ concurrent users.

Minor Issues:

- Some admin actions (bulk deletions) caused temporary UI lag → scheduled for optimization in next iteration.
- Analytics charts with large datasets loaded slower than expected → additional caching and aggregation planned.
- Load testing revealed spikes in API latency during concurrent poll creation → further DB tuning proposed.

Conclusion:
Sprint goals were achieved, completing the planned functionality and ensuring system readiness for sustained use.

Backlog Refinement

Sprint 5 backlog originated from the main product backlog

ID	User Story	Estimation (SP)	Priority
20	**As an admin, I want a dashboard to manage users, trips, and voting sessions, so that I can oversee the system.**	8	21

21	**As an admin**, I want a dashboard with key analytics and charts, **so that** I can understand usage trends and platform health.	5	22
22	**As a product owner, I want the platform to handle 1000+ concurrent users, so that it performs well under load.**	8	24

5.5.4. Sprint Retrospective

Sprint Retrospective Meeting: July 1st
Duration: 1.5 hours
Attendees:

- Tair (Scrum Master & Frontend Developer)
- Zhalgas (Backend Developer)

Agenda

- Reflections on Achievements and Challenges
- Identification of Action Items for Process Improvement

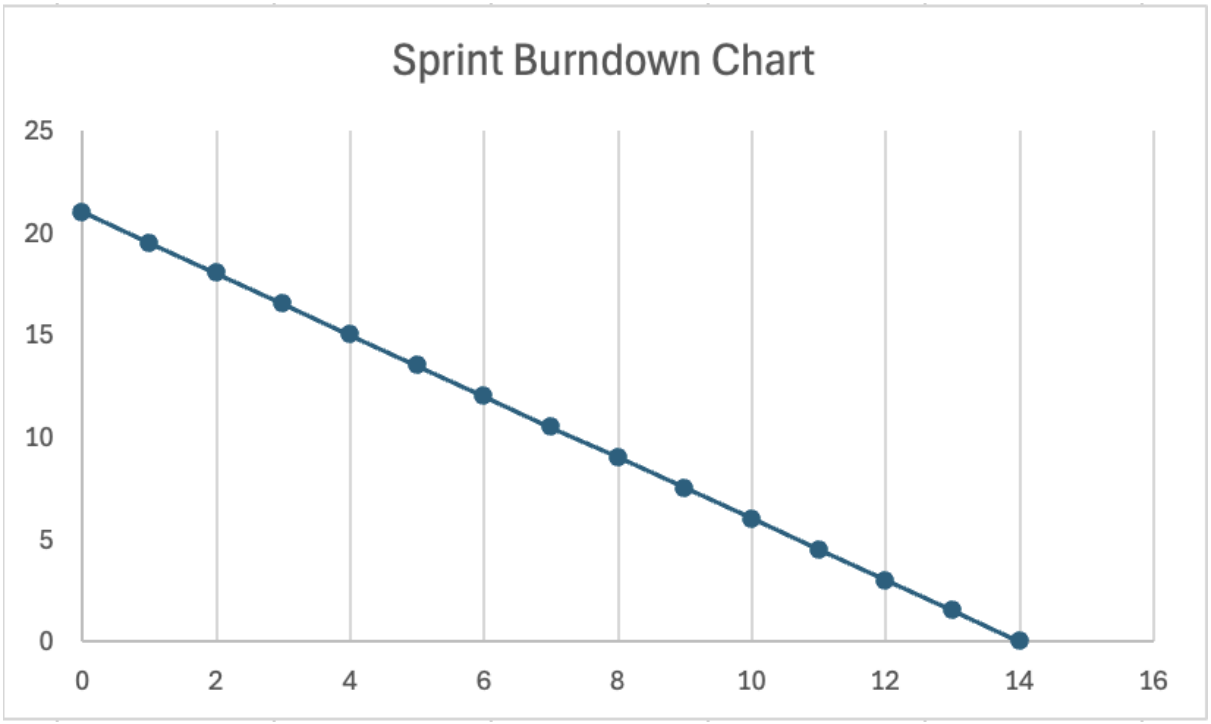
Outcome

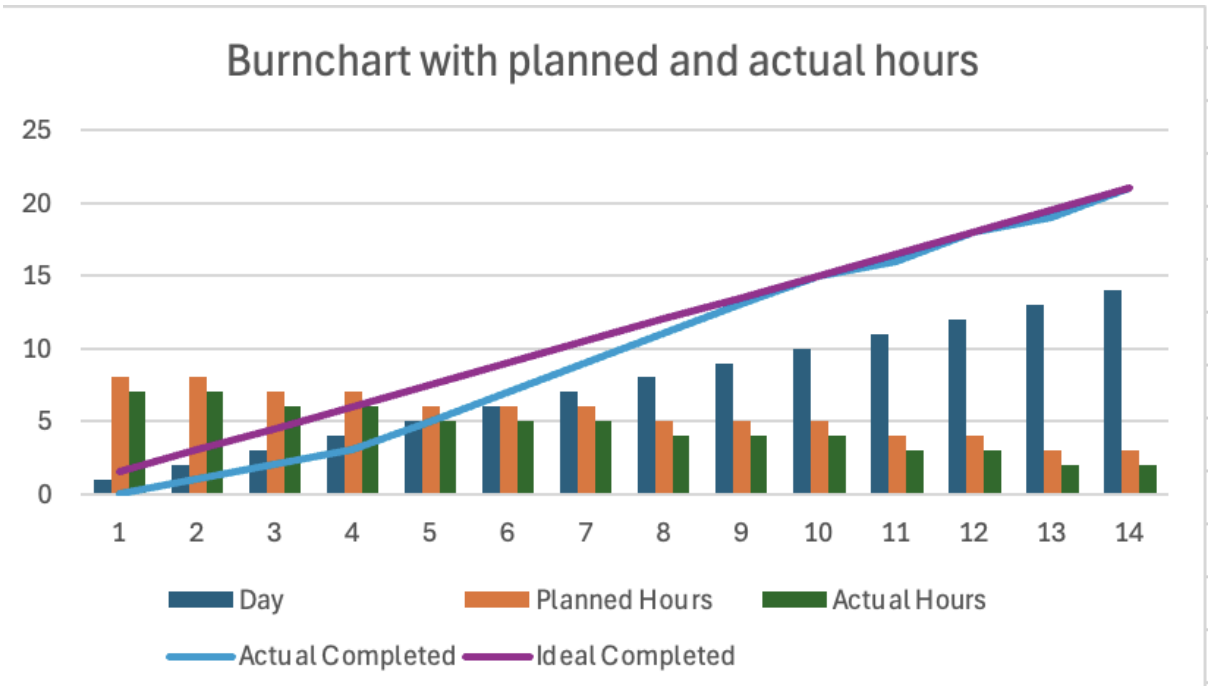
Achievements:

- All planned features were successfully delivered:
    - **Admin Dashboard** with management of users, trips, votes, and sessions.
    - **Voting Sessions Monitoring** with real-time updates for active and closed polls.
    - **Analytics Dashboard** with KPIs, charts.
    - **Scalability improvements**, including DB indexes, caching, connection pooling, frontend pagination, lazy loading, and successful load testing for 1000+ concurrent users.
- The system achieved stability and completeness, marking the finalization of major features.
- Collaboration between frontend and backend continued to improve, especially in integration and performance testing.

Burndown chart

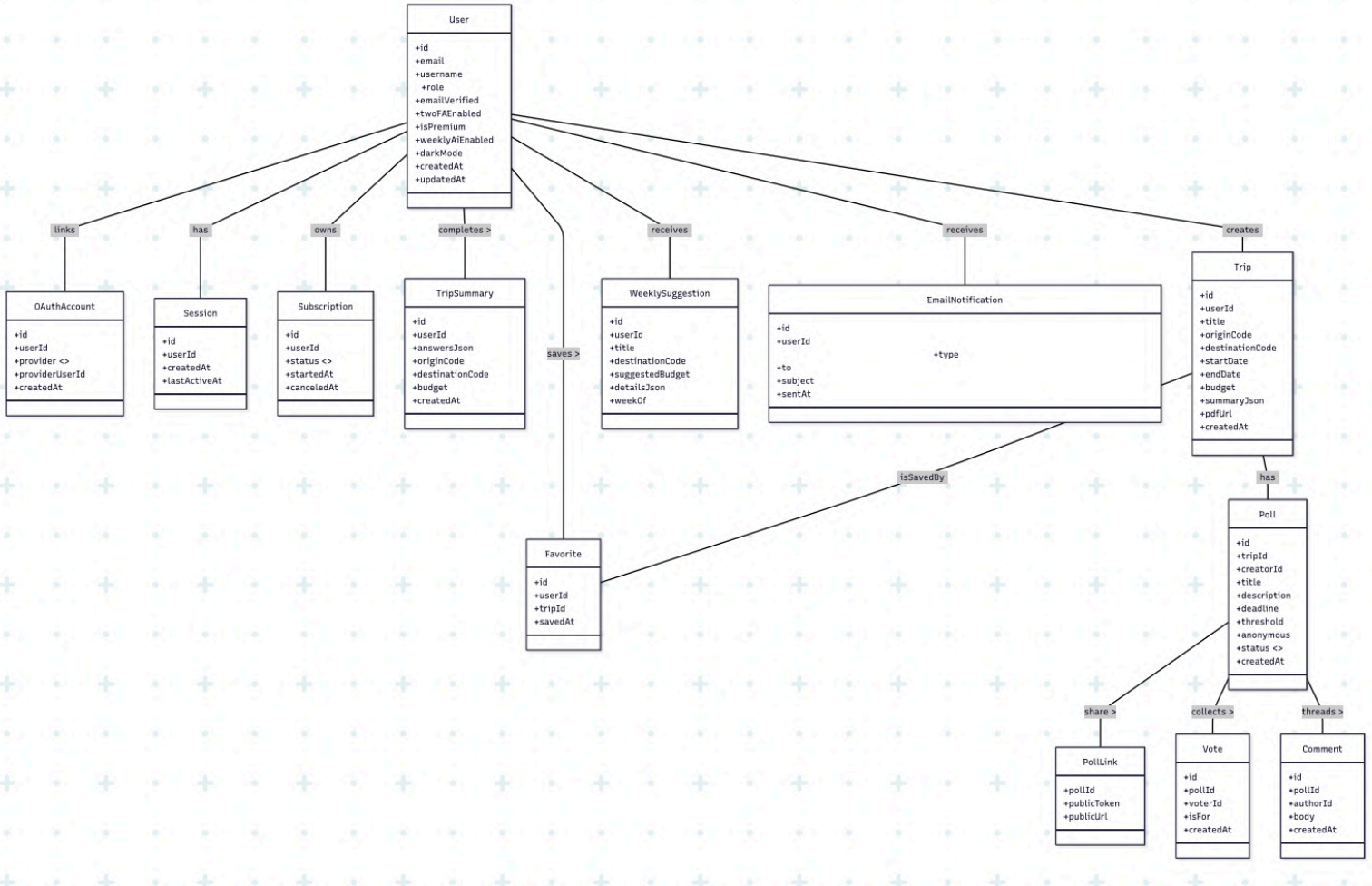
Total of 21 points from user stories.



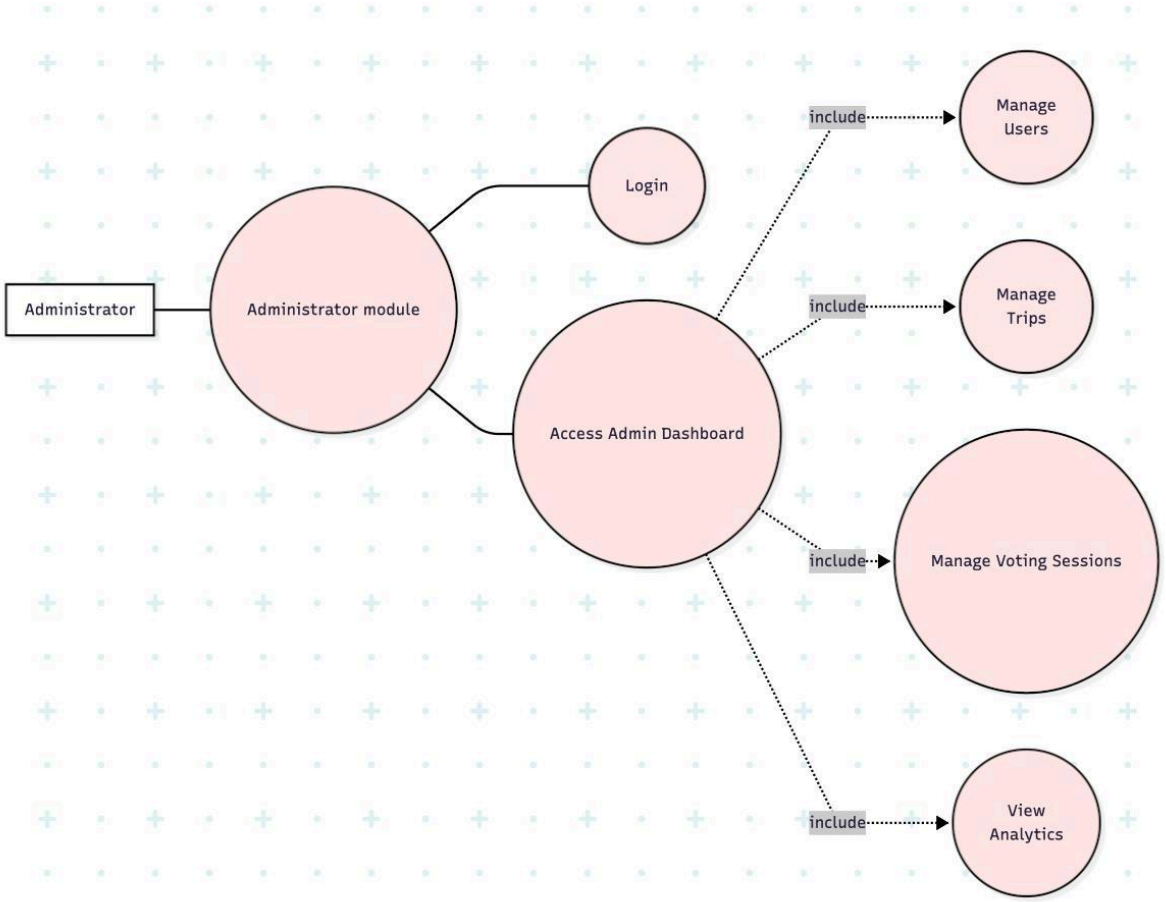


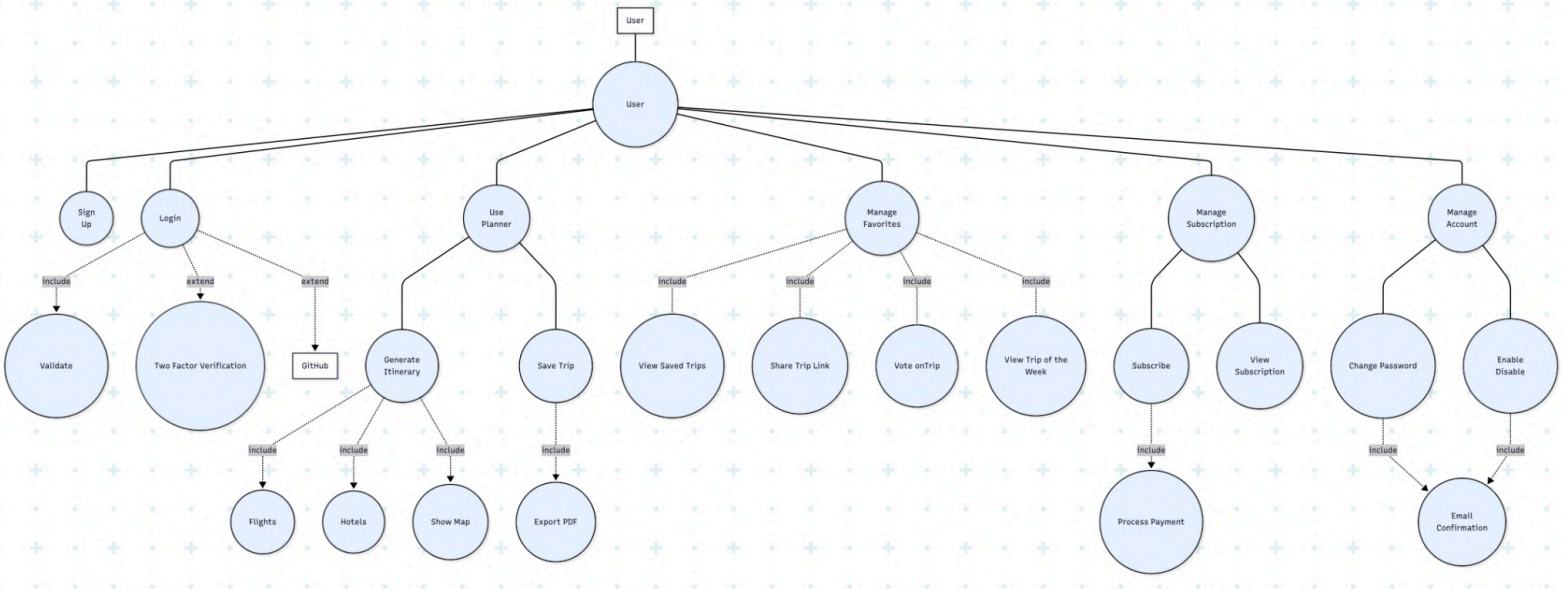
6. Project and code architecture

6.1. Class Diagram



6.2. Use Case Diagram





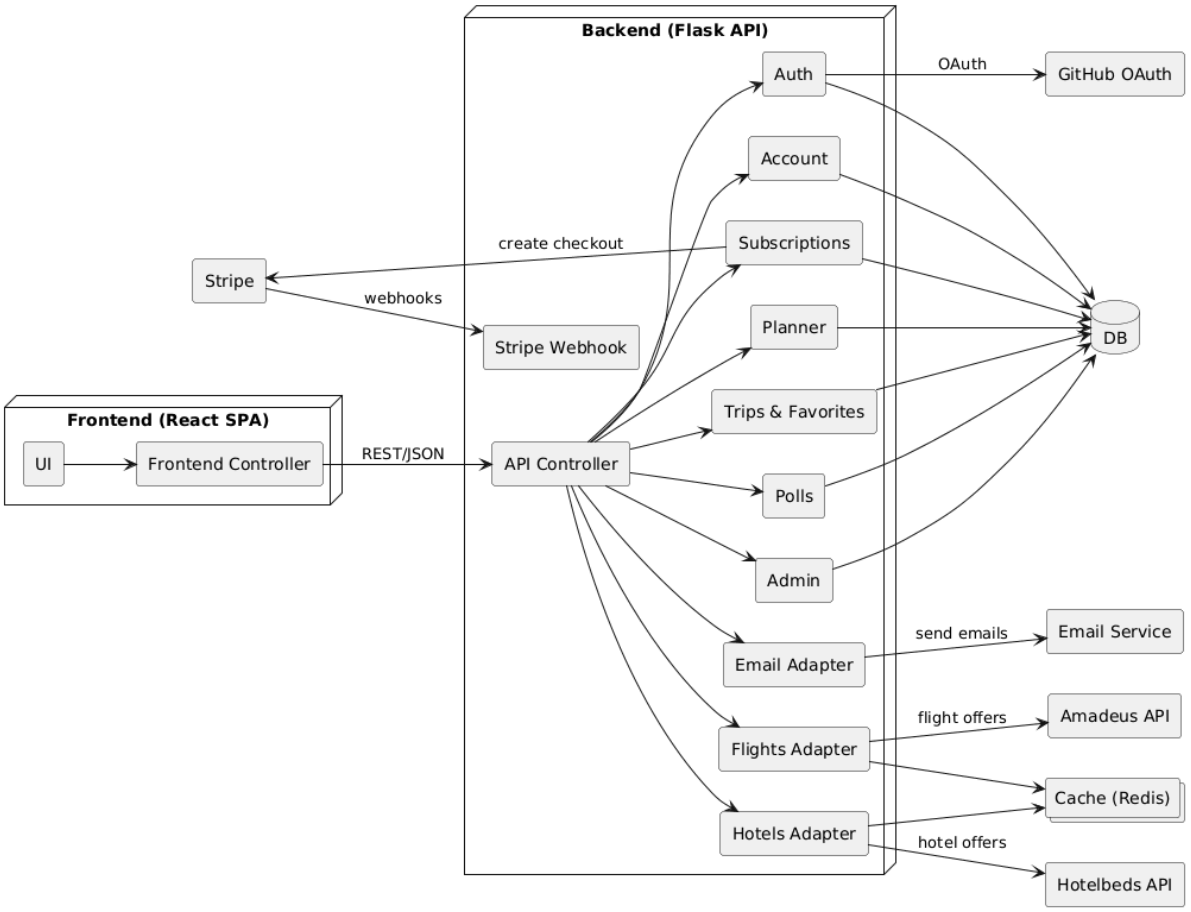
User/Customer Use Cases

- **Authentication System**: central node that includes login and registration.
- **Sign Up**: allows new users to create an account.
- **Login**: allows existing users to log in with email and password.
- **Login with GitHub**: allows users to authenticate via GitHub.
- **Planner**: access the trip planner module.
- **Create Trip**: create a new trip inside the planner.
- **New Chat**: start a new conversation with the planner assistant.
- **Save Trip**: save a created trip.
- **Export PDF**: export a trip as a PDF document.
- **Favorites**: access the favorites section.
- **View / Manage Trips**: view and manage saved trips.
- **Create Poll**: create a voting session for a trip.
- **Share Link & Vote**: generate a shareable link and allow voting.
- **Subscription**: access subscription features.
- **Payment**: perform a subscription payment.
- **View Subscription**: view the status of the current subscription.
- **Trip of the Week**: access the featured trip of the week.
- **Account**: access account settings.
- **Enable / Disable 2FA**: turn two-factor authentication on or off.
- **Change Password**: change the account password.

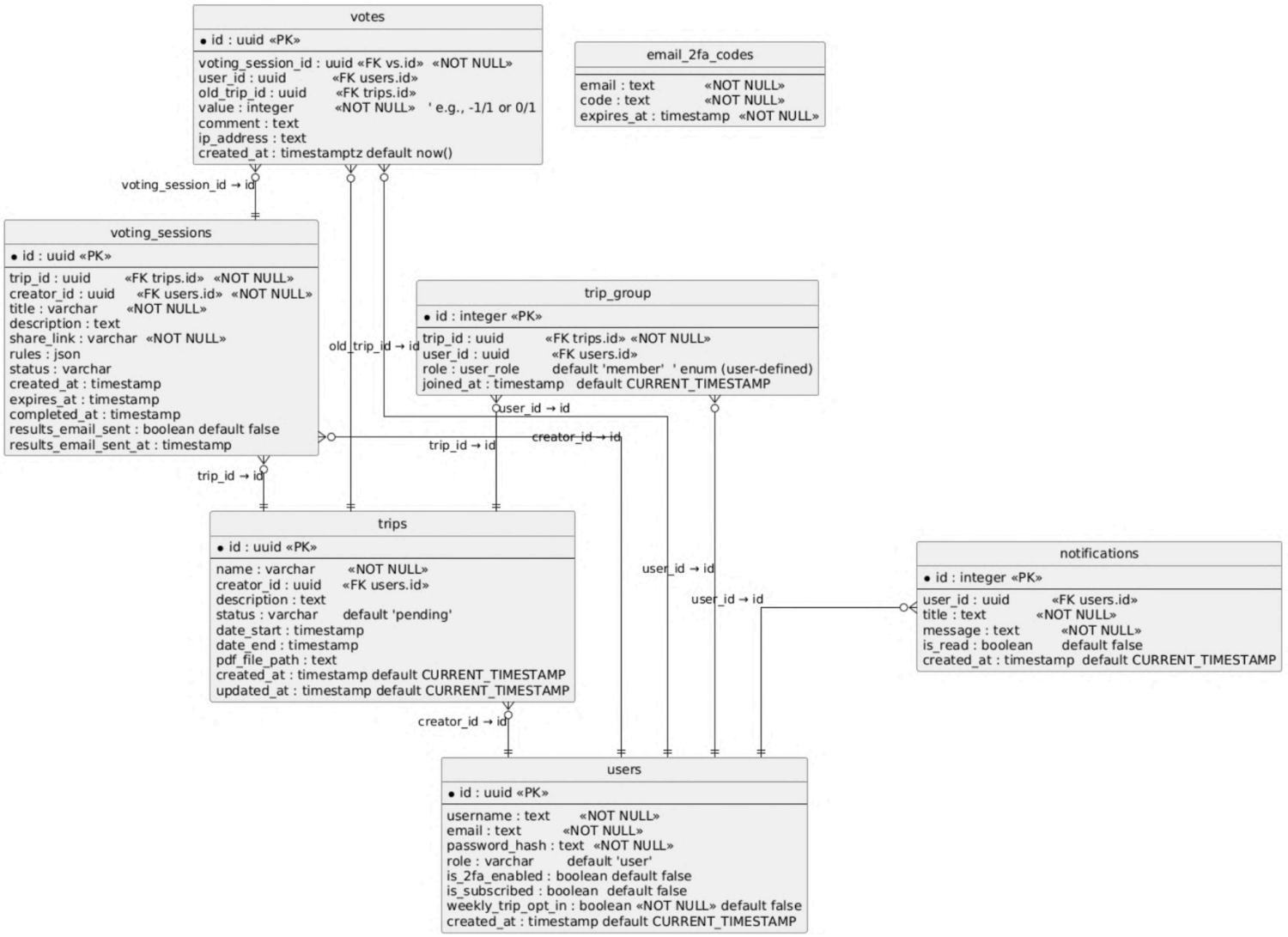
Administrator Use Cases

- **Manage Users**: access user management.
- **Add/Edit/Delete/Update Users**: perform CRUD operations on users.
- **Dashboard**: access the administrator dashboard.
- **Statistics Panel**: view system statistics.

6.3 Component diagram



6.4 Database diagram



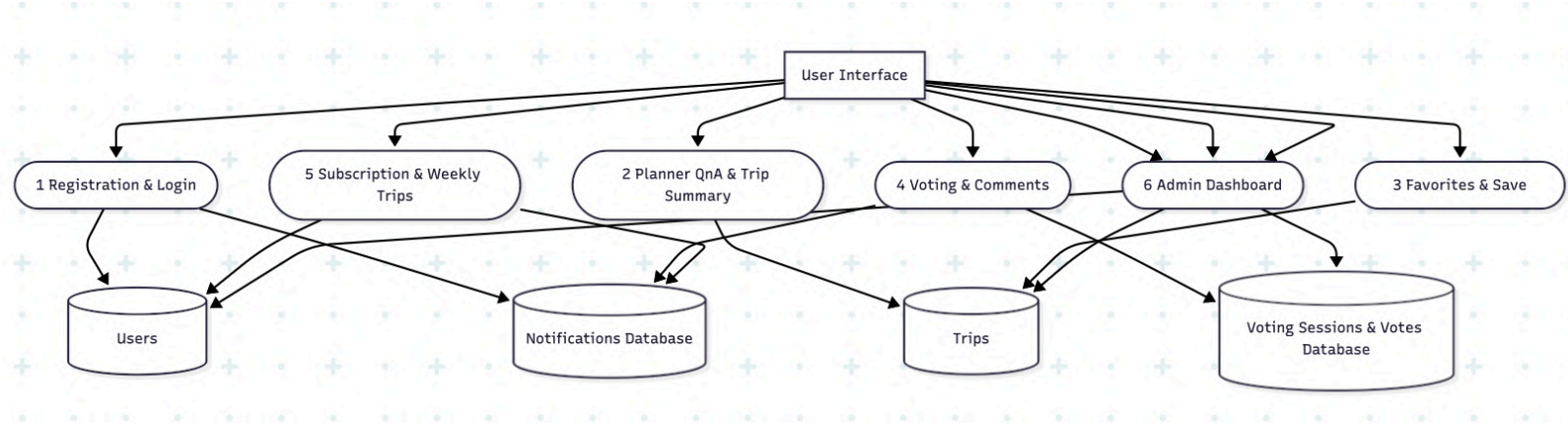
The database schema consists of seven tables: **Users, Trips, Voting_Sessions, Votes, Trip_Group, Notifications, and Email_2FA_Codes**.

- The **Users** table stores core account data such as id, username, email, password_hash, roles and subscription flags, and profile settings (e.g., 2FA, weekly trip opt-in).
- The **Trips** table holds trip details, including id, name, description, status, dates, references to the trip creator (creator_id), and optional exported PDF path.
- The **Voting_Sessions** table manages group decision-making on trips, with fields for id, trip_id, creator_id, title, description, voting rules, status, expiration, and result email tracking.

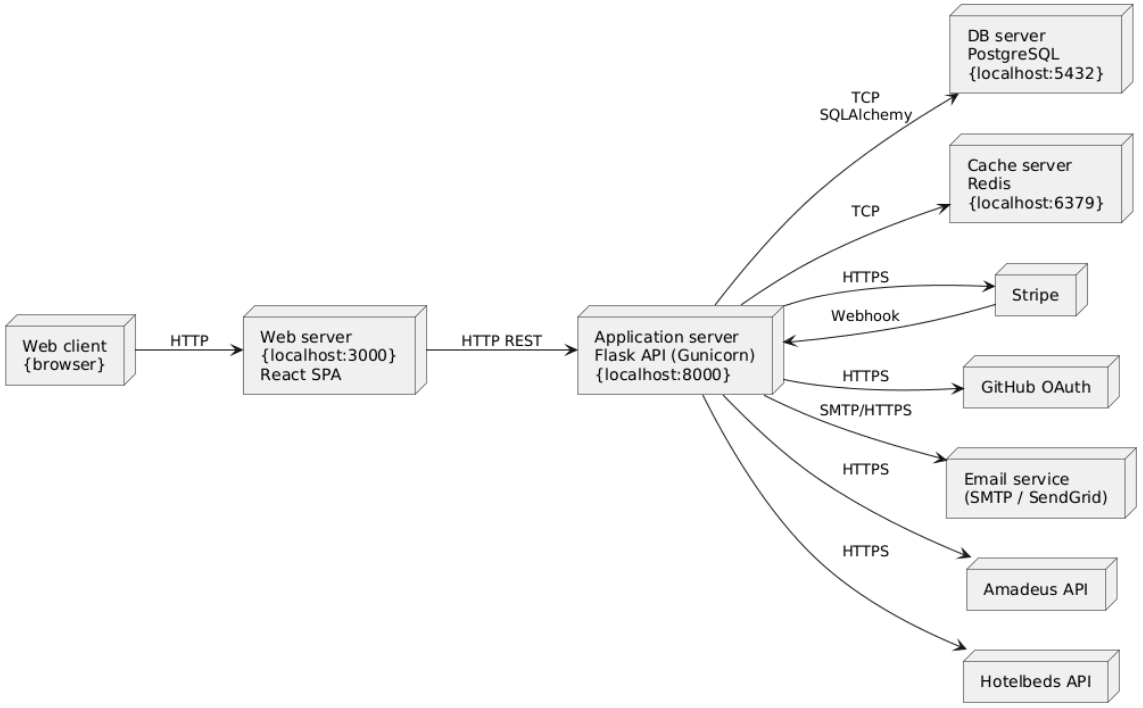
- The **Votes** table records individual voting actions, linking back to both a voting_session_id and optionally a user_id or old_trip_id, and includes fields for value, comment, ip_address, and timestamps.
- The **Trip_Group** table models membership of users in trips, with fields for id, trip_id, user_id, role, and join date.
- The **Notifications** table stores system messages for users, including title, message, read state, and creation time.
- The **Email_2FA_Codes** table provides temporary records of email-based two-factor authentication codes, including email, code, and expires_at.

This schema supports user management, trip planning, collaborative voting with comments, notifications, and secure authentication flows.

6.5 Data flow diagram



6.6 Deployment diagram



7. Tools and Technologies Used

The development of *Trip DVisor* relied on a modern, scalable, and maintainable technology stack. Each tool and framework was selected to meet specific requirements for performance, security, usability, and extensibility.

On the **frontend**, the application was built with **React**, supported by **HTML, CSS, and JavaScript**. React enabled the creation of a dynamic single-page application with reusable components, while CSS provided responsive layouts optimized for both desktop and mobile devices. JavaScript handled interactivity, event management, and communication with backend APIs. React was chosen because

of its modular structure and rich ecosystem of libraries, which accelerated feature development and testing.

The **backend** was implemented with **Python Flask**, which served as the REST API layer connecting the frontend, database, and external services. Flask was used to implement business logic such as itinerary generation, user authentication, and voting mechanisms. Its lightweight design and flexibility, combined with smooth integration with third-party APIs like Stripe and Perplexity, made it the most suitable choice for this project.

For persistent storage, the project used **PostgreSQL**. The database stored essential entities such as users, trips, votes, subscriptions, and authentication tokens, and also supported JSON fields for AI-generated itineraries. PostgreSQL was selected for its reliability, scalability, and strong support for relational queries, which were crucial for managing complex features like shared trips and voting analytics.

Stripe API was integrated to handle premium subscriptions and payments. It managed the entire subscription lifecycle, from activation to renewal and cancellation, ensuring secure transactions and PCI compliance. Stripe was chosen because it offered a reliable, ready-made solution for payment processing without requiring custom handling of sensitive financial data.

For the AI-powered trip planner, the system integrated the **Perplexity API**, which provided natural language processing capabilities for generating itineraries and travel suggestions. Leveraging an external AI service enabled the project to deliver robust conversational features without the need to train or host models in-house.

In terms of **hosting and deployment**, the backend (Flask and PostgreSQL) was deployed on **Railway**, while the frontend was hosted on **Vercel/Netlify**. These platforms were chosen for their scalability, global content delivery, and minimal DevOps overhead, allowing the team to focus on development while ensuring stable deployments.

Git and GitHub were used for version control, enabling distributed collaboration, branching strategies, and pull requests that supported smooth teamwork. To ensure consistent quality, the project adopted **GitHub Actions** for CI/CD pipelines, which automated testing, linting, and deployments. This setup reduced manual errors, improved reliability, and ensured that new features were thoroughly validated before release.

Overall, this technology stack enabled the development of a secure, scalable, and user-friendly platform while maintaining efficient workflows for the team.

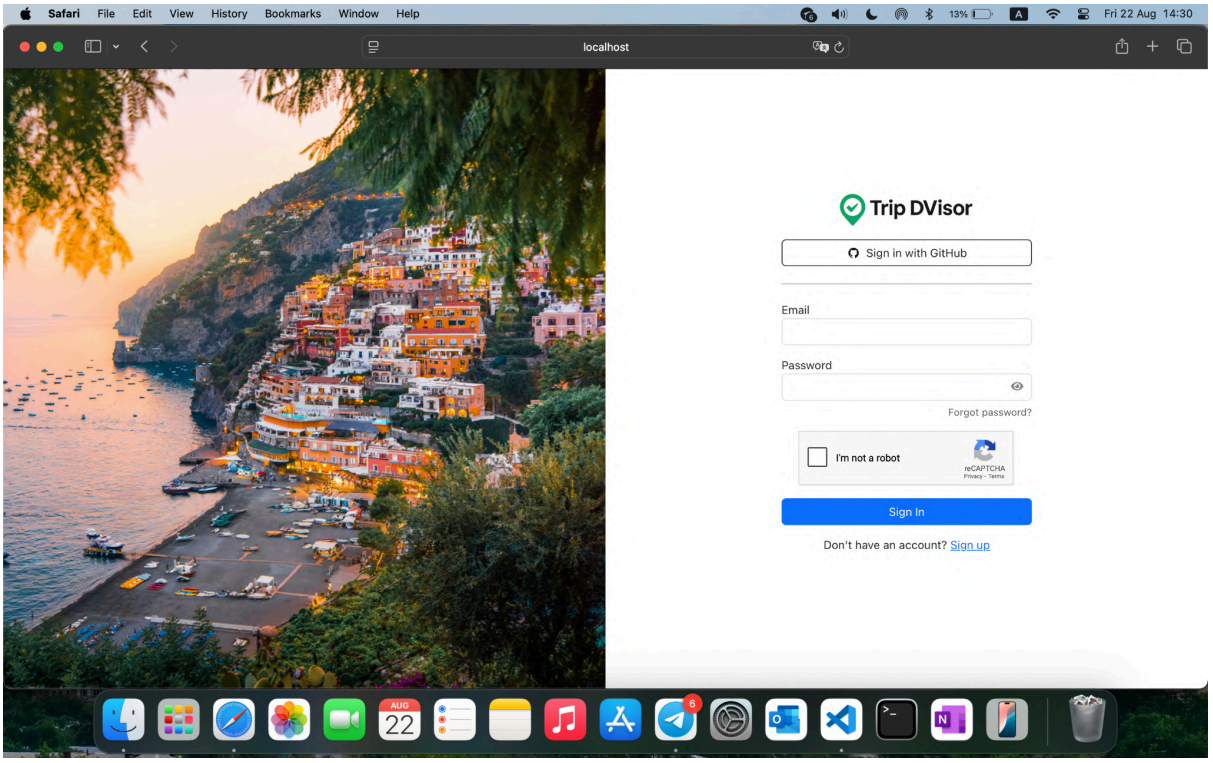
8. Features Implemented

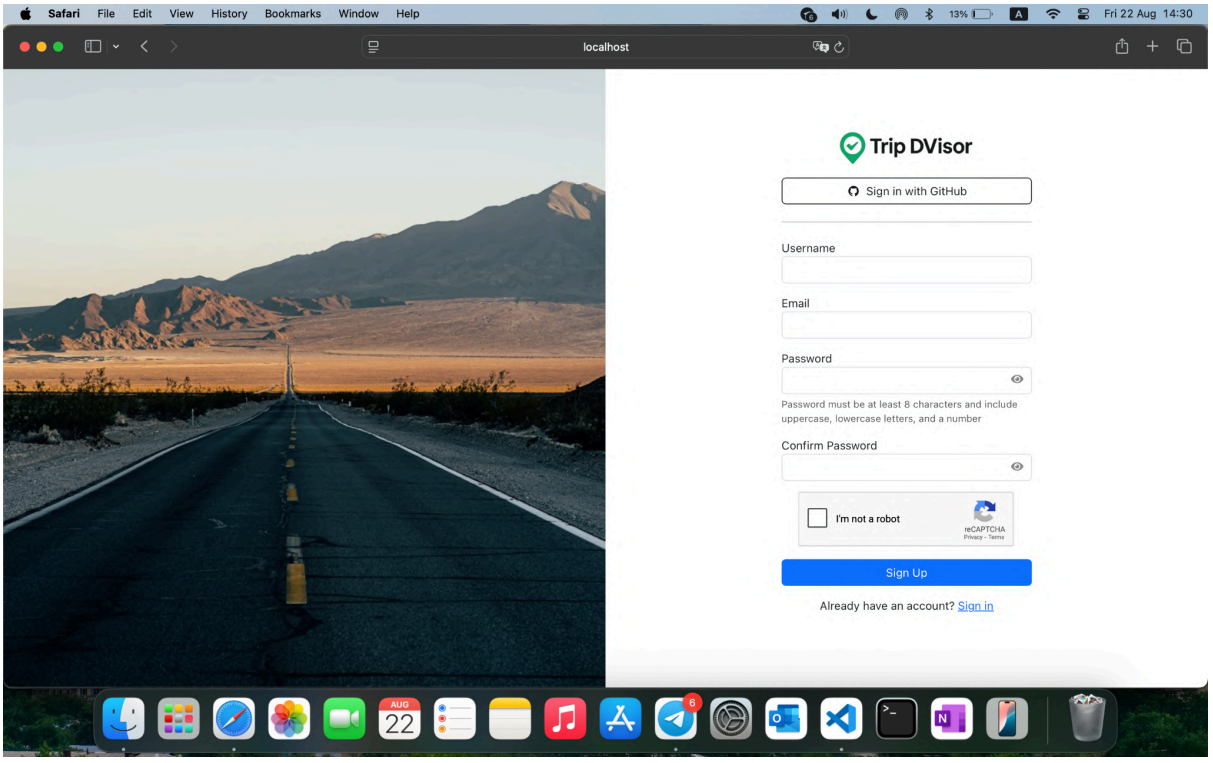
1. **User Registration and Login** – secure sign-up and sign-in with email, password, and CAPTCHA verification.
2. **GitHub OAuth Integration** – alternative login method without CAPTCHA for faster access.
3. **Two-Factor Authentication (2FA)** – optional email-based verification codes during login, with email notifications when 2FA state changes.

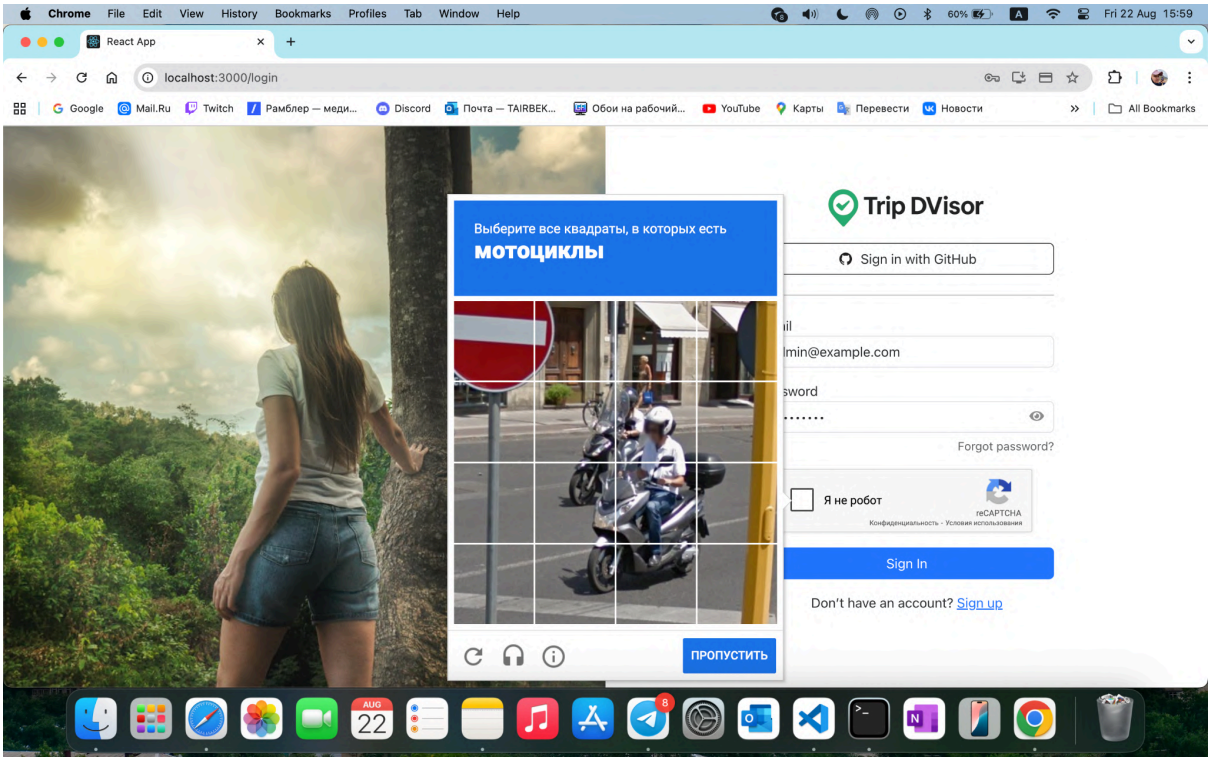
4. **Account Management** – ability to reset password, enable/disable 2FA, and manage subscription/weekly plan settings (username and email are fixed).
5. **Subscription via Stripe** – secure payment flow for premium access, with redirect to checkout, webhook confirmation, and success emails.
6. **Weekly AI Trip Suggestions** – automatically generated "Trip of the Week" for premium users, with opt-in/out toggle in account settings.
7. **Planner with Conversational AI** – interactive Q&A with AI assistant to collect trip details (destination, budget, transport, accommodation).
8. **Trip Summary Generation** – AI produces a structured trip plan, enriched with live API data (flights via Amadeus, hotels via Hotelbeds).
9. **Save Trip** – users can store trip summaries in Favorites and download them as formatted PDFs.
10. **Favorites Page** – central storage of saved trips with options to view, delete, or create polls; premium users also see the "Trip of the Week."
11. **Poll Creation** – polls can be created from saved trips, with settings for title, description, deadline, vote threshold, and anonymity.
12. **Public Poll Sharing** – shareable poll links allow group participation without requiring accounts.
13. **Voting and Commenting** – participants can vote "for" or "against" a trip and leave comments, with live statistics shown on the poll page.
14. **Automatic Poll Closure** – polls close when reaching deadline or vote threshold, freezing results.
15. **Email Notifications** – system emails for password changes, 2FA toggles, subscription updates, weekly trips, and poll results.
16. **Dark Mode** – toggleable theme for improved user experience.
17. **Admin Dashboard** – available only to admins, showing detailed info about users, trips, and voting sessions, with ability to delete data if necessary.
18. **Security Features** – password hashing, CAPTCHA on login/registration, secure tokens for session management, and webhook validation.
19. **Responsive UI** – frontend optimized for both desktop and mobile users.
20. **Cross-Platform Compatibility** – tested to ensure consistent functionality across browsers and devices.

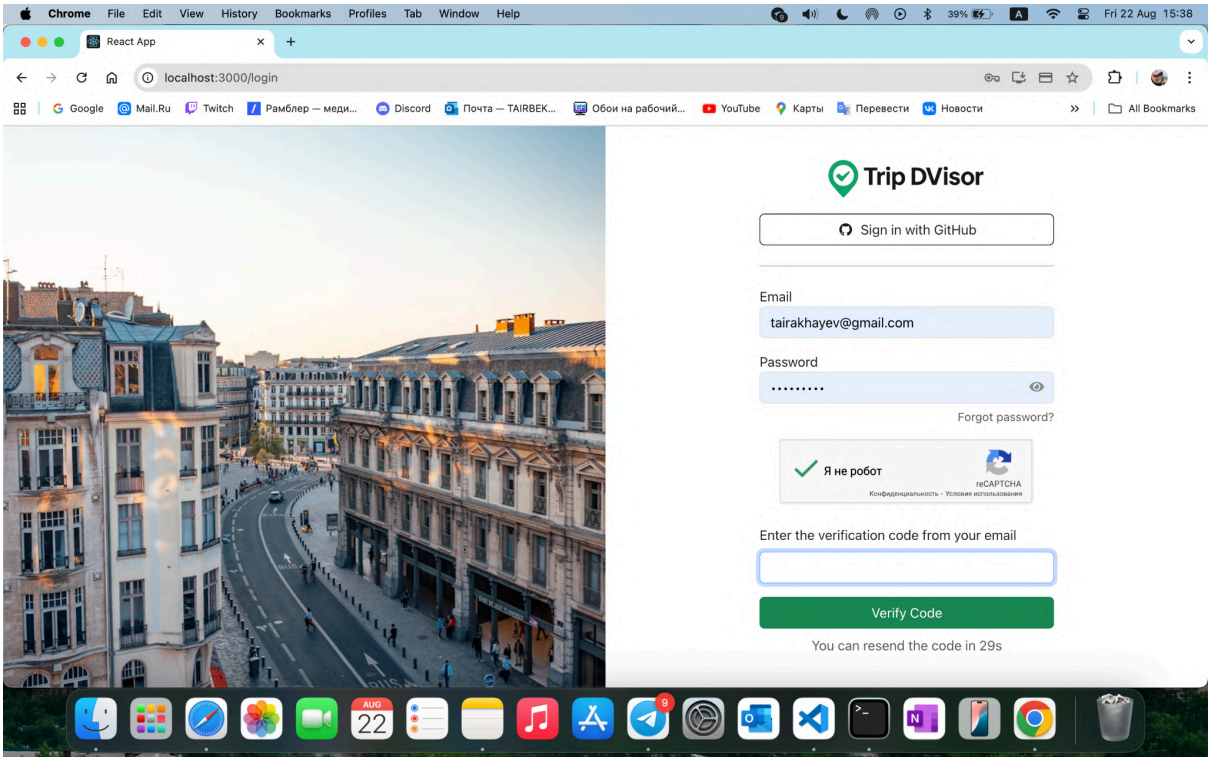
9. Website mock-up

9.1. PC browser view






Chrome File Edit View History Bookmarks Profiles Tab Window Help
Fri 22 Aug 15:59
React App
localhost:3000/login
Выберите все квадраты, в которых есть
мотоциклы
ПРОПУСТИТЬ
Trip DVisor
Sign in with GitHub
Forgot password?
Я не робот
reCAPTCHA
Sign In
Don't have an account? Sign up


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Fri 22 Aug 15:38
React App
localhost:3000/login
Trip DVisor
Sign in with GitHub
Email
tairakhayev@gmail.com
Password
Forgot password?
Я не робот
reCAPTCHA
Enter the verification code from your email
Verify Code
You can resend the code in 29s

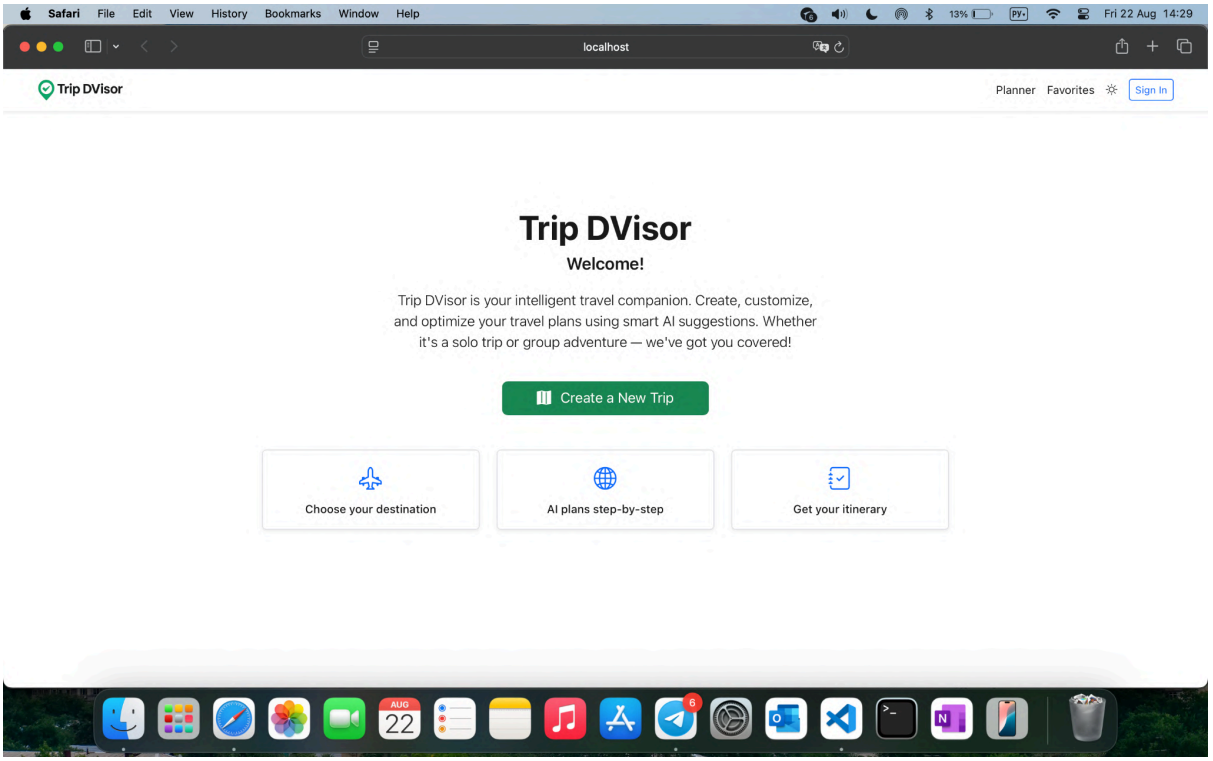
2FA Verification Code

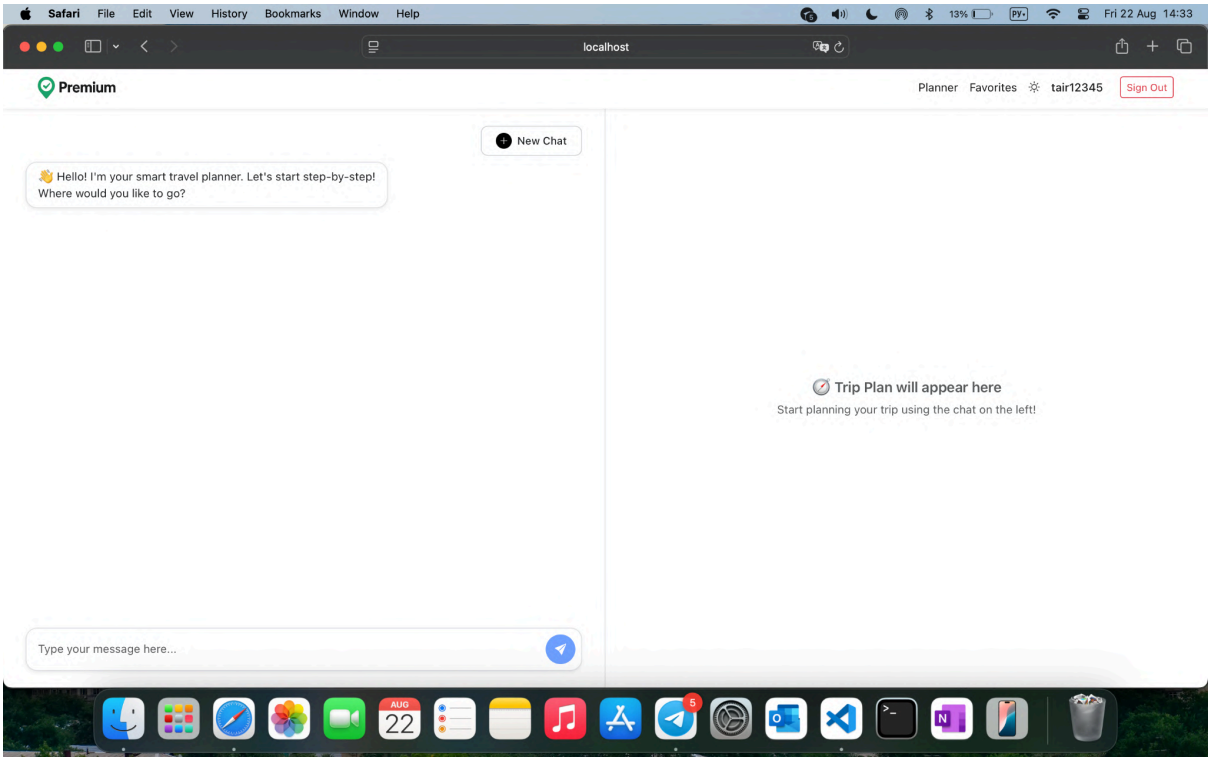
Inbox ×

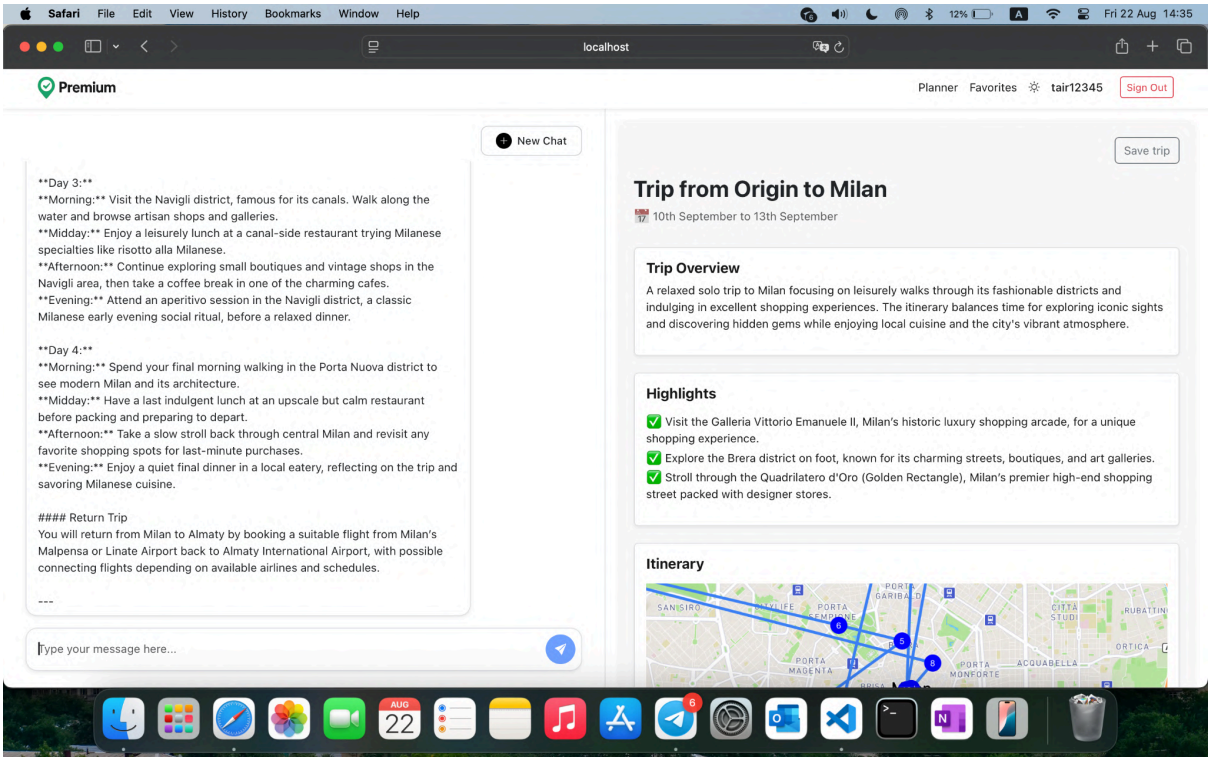
Trip DVisor <vperedmuslims@gmail.com>

to me

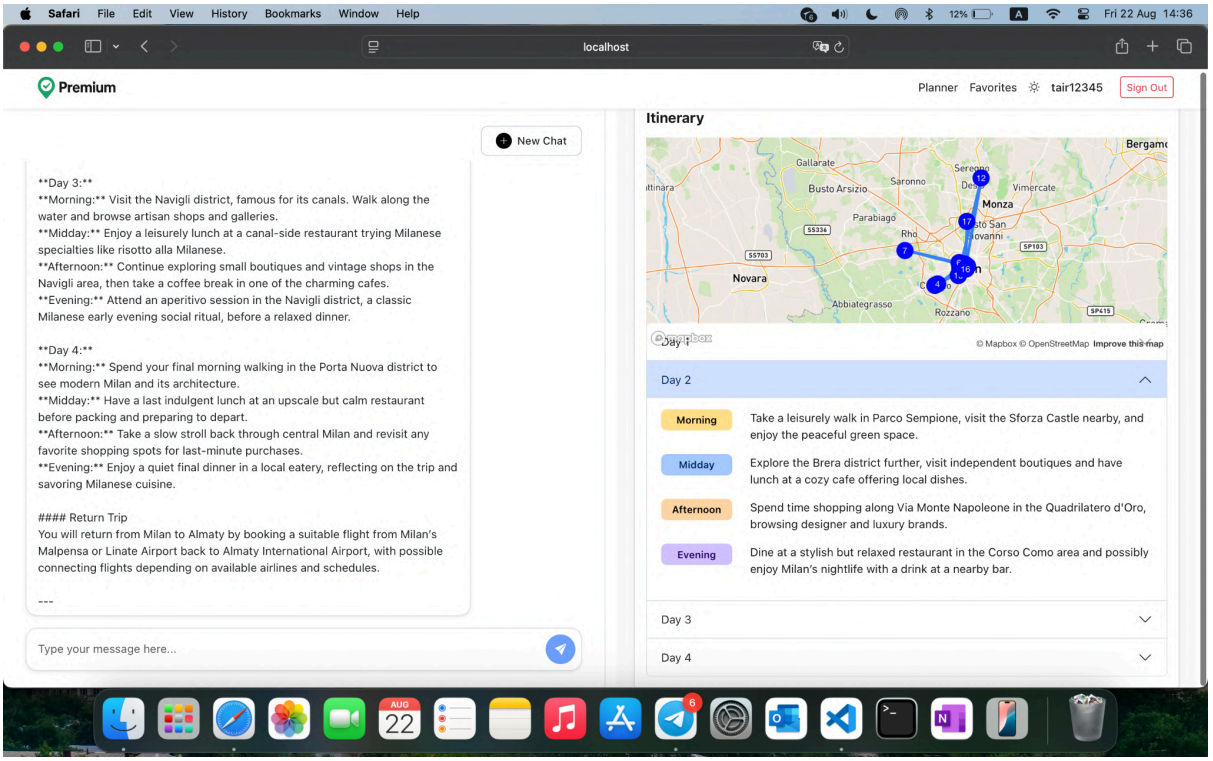
Your verification code is: 510382



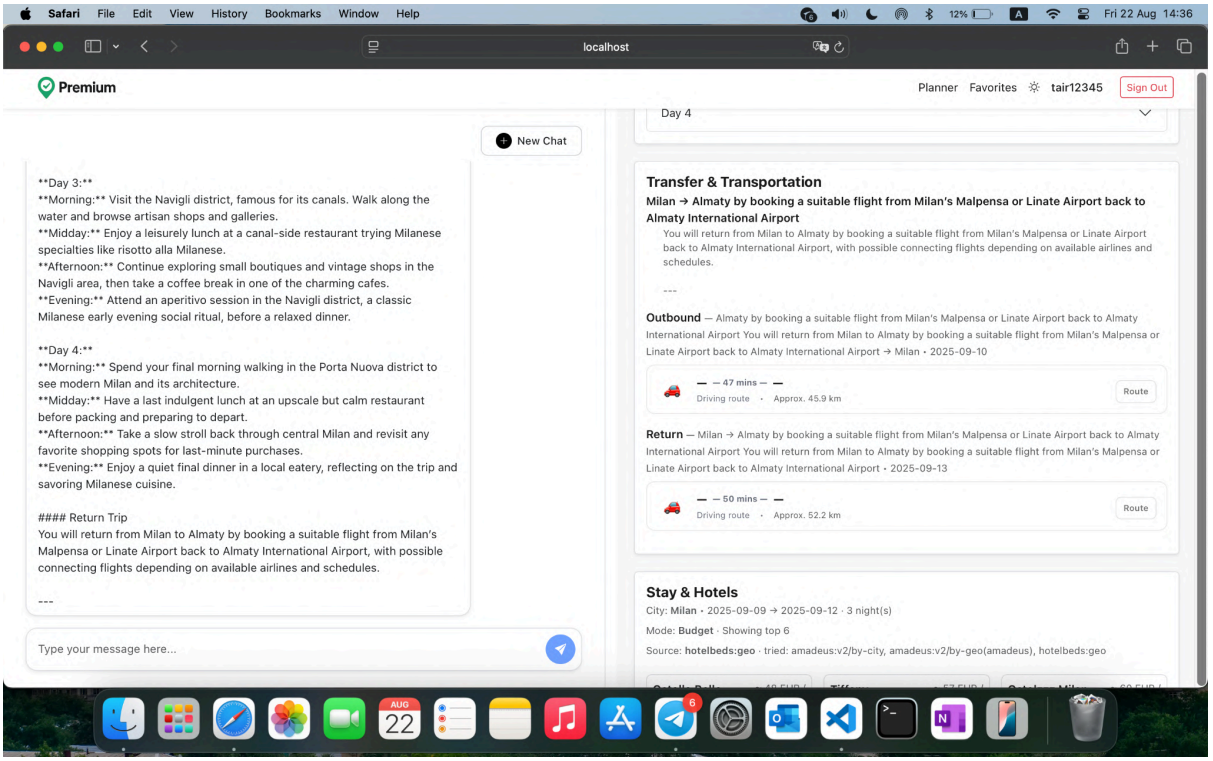

Safari File Edit View History Bookmarks Window Help
Fri 22 Aug 14:33
localhost
Premium
Planner Favorites tair12345 Sign Out
New Chat
Hello! I'm your smart travel planner. Let's start step-by-step!
Where would you like to go?
Trip Plan will appear here
Start planning your trip using the chat on the left!
Type your message here...


Safari File Edit View History Bookmarks Window Help
Fri 22 Aug 14:35
localhost
Premium
Planner Favorites tair12345 Sign Out
New Chat
Day 3:
Morning: Visit the Navigli district, famous for its canals. Walk along the water and browse artisan shops and galleries.
Midday: Enjoy a leisurely lunch at a canal-side restaurant trying Milanese specialties like risotto alla Milanese.
Afternoon: Continue exploring small boutiques and vintage shops in the Navigli area, then take a coffee break in one of the charming cafes.
Evening: Attend an aperitivo session in the Navigli district, a classic Milanese early evening social ritual, before a relaxed dinner.
Day 4:
Morning: Spend your final morning walking in the Porta Nuova district to see modern Milan and its architecture.
Midday: Have a last indulgent lunch at an upscale but calm restaurant before packing and preparing to depart.
Afternoon: Take a slow stroll back through central Milan and revisit any favorite shopping spots for last-minute purchases.
Evening: Enjoy a quiet final dinner in a local eatery, reflecting on the trip and savoring Milanese cuisine.
Return Trip
You will return from Milan to Almaty by booking a suitable flight from Milan's Malpensa or Linate Airport back to Almaty International Airport, with possible connecting flights depending on available airlines and schedules.

Type your message here...
Save trip
Trip from Origin to Milan
10th September to 13th September
Trip Overview
A relaxed solo trip to Milan focusing on leisurely walks through its fashionable districts and indulging in excellent shopping experiences. The itinerary balances time for exploring iconic sights and discovering hidden gems while enjoying local cuisine and the city's vibrant atmosphere.
Highlights
✅ Visit the Galleria Vittorio Emanuele II, Milan's historic luxury shopping arcade, for a unique shopping experience.
✅ Explore the Brera district on foot, known for its charming streets, boutiques, and art galleries.
✅ Stroll through the Quadrilatero d'Oro (Golden Rectangle), Milan's premier high-end shopping street packed with designer stores.
Itinerary
SANSIRO
PORTA GARIBALDI
PORTA MAGENTA
PORTA MONFORTE
ACQUABELLA
ORTICA

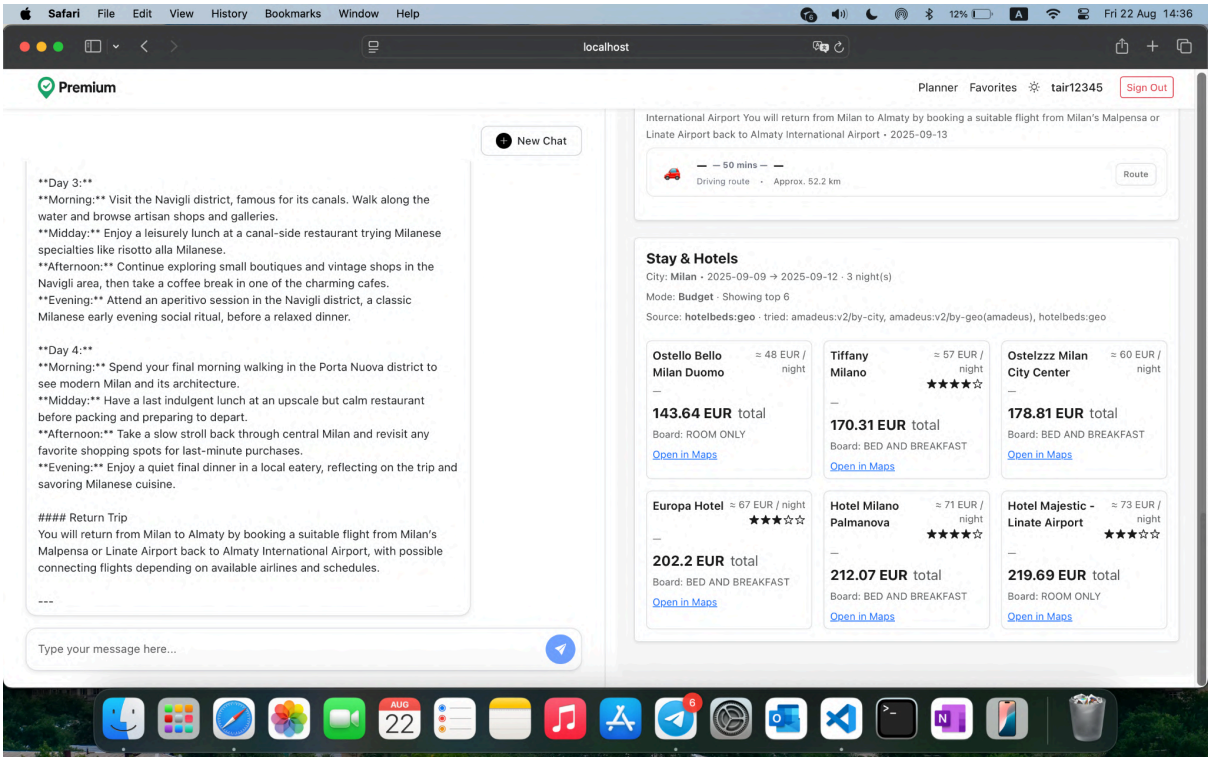

Safari File Edit View History Bookmarks Window Help
Fri 22 Aug 14:36
localhost
Premium
Planner Favorites tair12345 Sign Out
New Chat
Day 3:
Morning: Visit the Navigli district, famous for its canals. Walk along the water and browse artisan shops and galleries.
Midday: Enjoy a leisurely lunch at a canal-side restaurant trying Milanese specialties like risotto alla Milanese.
Afternoon: Continue exploring small boutiques and vintage shops in the Navigli area, then take a coffee break in one of the charming cafes.
Evening: Attend an aperitivo session in the Navigli district, a classic Milanese early evening social ritual, before a relaxed dinner.
Day 4:
Morning: Spend your final morning walking in the Porta Nuova district to see modern Milan and its architecture.
Midday: Have a last indulgent lunch at an upscale but calm restaurant before packing and preparing to depart.
Afternoon: Take a slow stroll back through central Milan and revisit any favorite shopping spots for last-minute purchases.
Evening: Enjoy a quiet final dinner in a local eatery, reflecting on the trip and savoring Milanese cuisine.
Return Trip
You will return from Milan to Almaty by booking a suitable flight from Milan's Malpensa or Linate Airport back to Almaty International Airport, with possible connecting flights depending on available airlines and schedules.

Type your message here...
Itinerary
Gallarate
Busto Arsizio
Saronno
Monza
Parabiago
Rho
Novara
Abbiategrasso
Rozzano
Bergamo
Vimercate
SS336
SS703
SP103
SP415
© Mapbox © OpenStreetMap Improve this map
Day 1
Day 2
Morning
Take a leisurely walk in Parco Sempione, visit the Sforza Castle nearby, and enjoy the peaceful green space.
Midday
Explore the Brera district further, visit independent boutiques and have lunch at a cozy cafe offering local dishes.
Afternoon
Spend time shopping along Via Monte Napoleone in the Quadrilatero d'Oro, browsing designer and luxury brands.
Evening
Dine at a stylish but relaxed restaurant in the Corso Como area and possibly enjoy Milan's nightlife with a drink at a nearby bar.
Day 3
Day 4

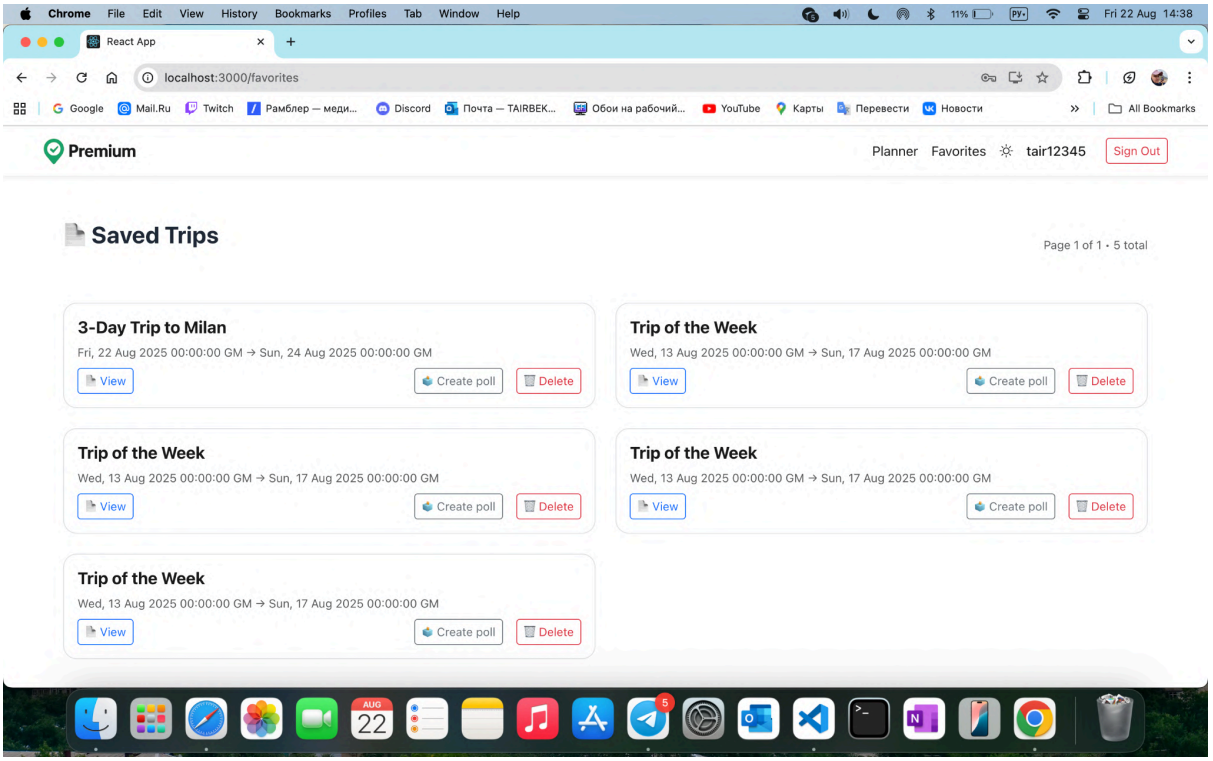

Safari File Edit View History Bookmarks Window Help
Fri 22 Aug 14:36
localhost
Premium
Planner Favorites tair12345 Sign Out
New Chat
Day 3:
Morning: Visit the Navigli district, famous for its canals. Walk along the water and browse artisan shops and galleries.
Midday: Enjoy a leisurely lunch at a canal-side restaurant trying Milanese specialties like risotto alla Milanese.
Afternoon: Continue exploring small boutiques and vintage shops in the Navigli area, then take a coffee break in one of the charming cafes.
Evening: Attend an aperitivo session in the Navigli district, a classic Milanese early evening social ritual, before a relaxed dinner.
Day 4:
Morning: Spend your final morning walking in the Porta Nuova district to see modern Milan and its architecture.
Midday: Have a last indulgent lunch at an upscale but calm restaurant before packing and preparing to depart.
Afternoon: Take a slow stroll back through central Milan and revisit any favorite shopping spots for last-minute purchases.
Evening: Enjoy a quiet final dinner in a local eatery, reflecting on the trip and savoring Milanese cuisine.
Return Trip
You will return from Milan to Almaty by booking a suitable flight from Milan's Malpensa or Linate Airport back to Almaty International Airport, with possible connecting flights depending on available airlines and schedules.

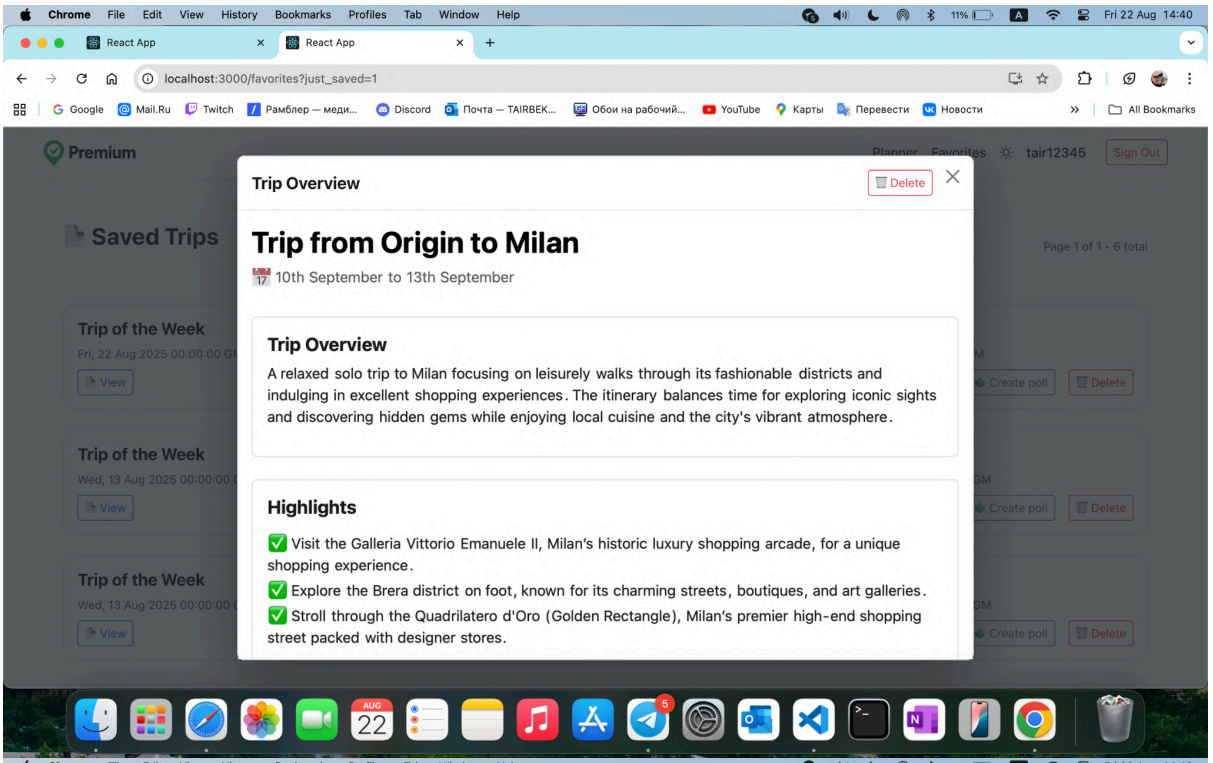
Type your message here...
Day 4
Transfer & Transportation
Milan → Almaty by booking a suitable flight from Milan's Malpensa or Linate Airport back to Almaty International Airport
You will return from Milan to Almaty by booking a suitable flight from Milan's Malpensa or Linate Airport back to Almaty International Airport, with possible connecting flights depending on available airlines and schedules.

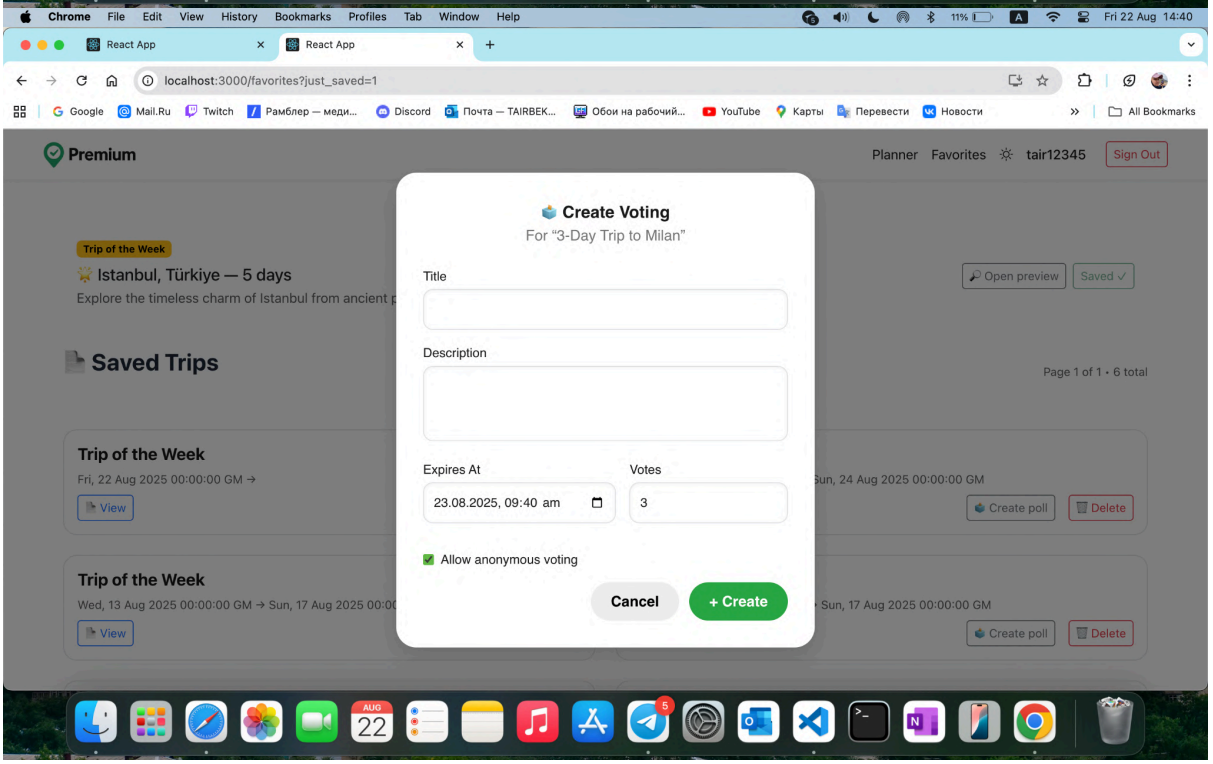
Outbound — Almaty by booking a suitable flight from Milan's Malpensa or Linate Airport back to Almaty International Airport You will return from Milan to Almaty by booking a suitable flight from Milan's Malpensa or Linate Airport back to Almaty International Airport → Milan • 2025-09-10
— 47 mins —
Driving route • Approx. 45.9 km
Route
Return — Milan → Almaty by booking a suitable flight from Milan's Malpensa or Linate Airport back to Almaty International Airport You will return from Milan to Almaty by booking a suitable flight from Milan's Malpensa or Linate Airport back to Almaty International Airport • 2025-09-13
— 50 mins —
Driving route • Approx. 52.2 km
Route
Stay & Hotels
City: Milan • 2025-09-09 → 2025-09-12 • 3 night(s)
Mode: Budget • Showing top 6
Source: hotelbeds:geo • tried: amadeus:v2/by-city, amadeus:v2/by-geo(amadeus), hotelbeds:geo


Safari File Edit View History Bookmarks Window Help
Fri 22 Aug 14:36
localhost
Premium
Planner Favorites tair12345 Sign Out
New Chat
Day 3:
Morning: Visit the Navigli district, famous for its canals. Walk along the water and browse artisan shops and galleries.
Midday: Enjoy a leisurely lunch at a canal-side restaurant trying Milanese specialties like risotto alla Milanese.
Afternoon: Continue exploring small boutiques and vintage shops in the Navigli area, then take a coffee break in one of the charming cafes.
Evening: Attend an aperitivo session in the Navigli district, a classic Milanese early evening social ritual, before a relaxed dinner.
Day 4:
Morning: Spend your final morning walking in the Porta Nuova district to see modern Milan and its architecture.
Midday: Have a last indulgent lunch at an upscale but calm restaurant before packing and preparing to depart.
Afternoon: Take a slow stroll back through central Milan and revisit any favorite shopping spots for last-minute purchases.
Evening: Enjoy a quiet final dinner in a local eatery, reflecting on the trip and savoring Milanese cuisine.
Return Trip
You will return from Milan to Almaty by booking a suitable flight from Milan's Malpensa or Linate Airport back to Almaty International Airport, with possible connecting flights depending on available airlines and schedules.

Type your message here...
International Airport You will return from Milan to Almaty by booking a suitable flight from Milan's Malpensa or Linate Airport back to Almaty International Airport • 2025-09-13
— 50 mins —
Driving route • Approx. 52.2 km
Route
Stay & Hotels
City: Milan • 2025-09-09 → 2025-09-12 · 3 night(s)
Mode: Budget · Showing top 6
Source: hotelbeds:geo · tried: amadeus:v2/by-city, amadeus:v2/by-geo(amadeus), hotelbeds:geo
Ostello Bello Milan Duomo
≈ 48 EUR / night
143.64 EUR total
Board: ROOM ONLY
Open in Maps
Tiffany Milano
≈ 57 EUR / night
★★★★☆
170.31 EUR total
Board: BED AND BREAKFAST
Open in Maps
Ostelzzz Milan City Center
≈ 60 EUR / night
178.81 EUR total
Board: BED AND BREAKFAST
Open in Maps
Europa Hotel
≈ 67 EUR / night
★★★☆☆
202.2 EUR total
Board: BED AND BREAKFAST
Open in Maps
Hotel Milano Palmanova
≈ 71 EUR / night
★★★★☆
212.07 EUR total
Board: BED AND BREAKFAST
Open in Maps
Hotel Majestic - Linate Airport
≈ 73 EUR / night
★★★☆☆
219.69 EUR total
Board: ROOM ONLY
Open in Maps

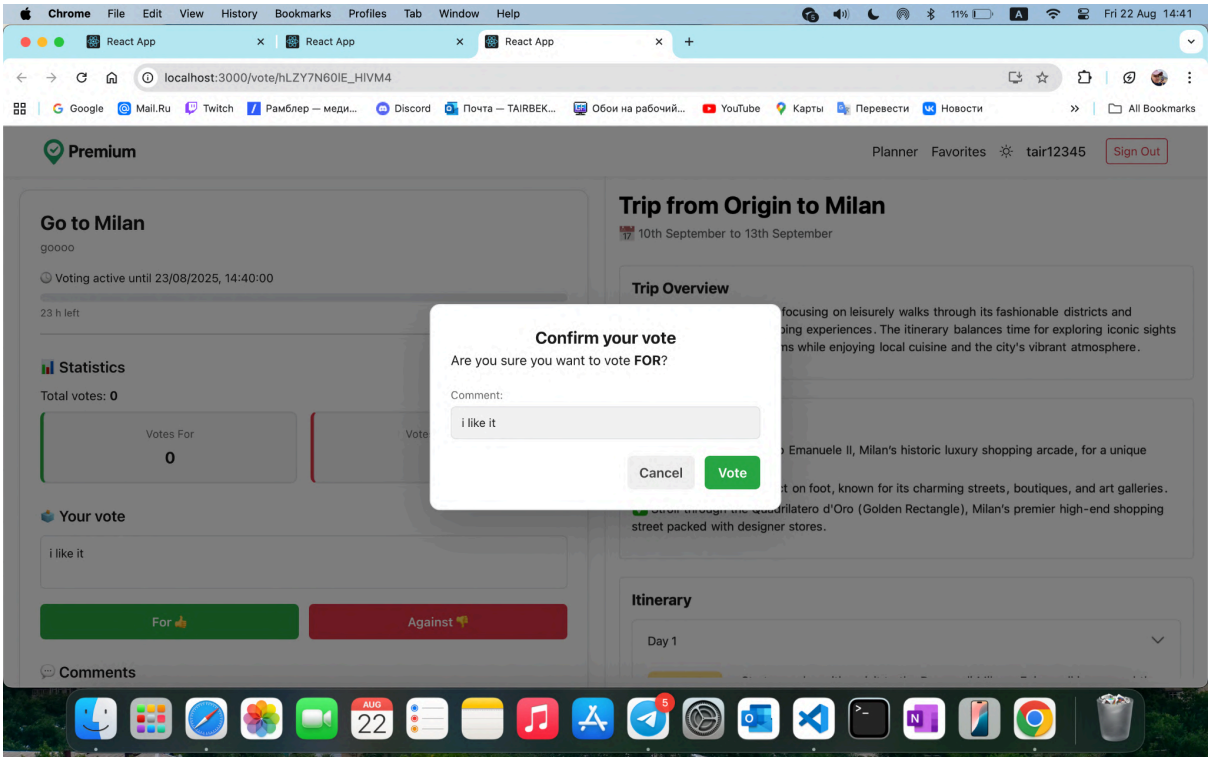

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Fri 22 Aug 14:38
React App
localhost:3000/favorites
Google Mail.Ru Twitch Рамблер — меди... Discord Почта — TAIRBEK... Обои на рабочий... YouTube Карты Перевести Новости All Bookmarks
Premium
Planner Favorites tair12345 Sign Out
Saved Trips
Page 1 of 1 • 5 total
3-Day Trip to Milan
Fri, 22 Aug 2025 00:00:00 GM → Sun, 24 Aug 2025 00:00:00 GM
View Create poll Delete
Trip of the Week
Wed, 13 Aug 2025 00:00:00 GM → Sun, 17 Aug 2025 00:00:00 GM
View Create poll Delete
Trip of the Week
Wed, 13 Aug 2025 00:00:00 GM → Sun, 17 Aug 2025 00:00:00 GM
View Create poll Delete
Trip of the Week
Wed, 13 Aug 2025 00:00:00 GM → Sun, 17 Aug 2025 00:00:00 GM
View Create poll Delete
Trip of the Week
Wed, 13 Aug 2025 00:00:00 GM → Sun, 17 Aug 2025 00:00:00 GM
View Create poll Delete

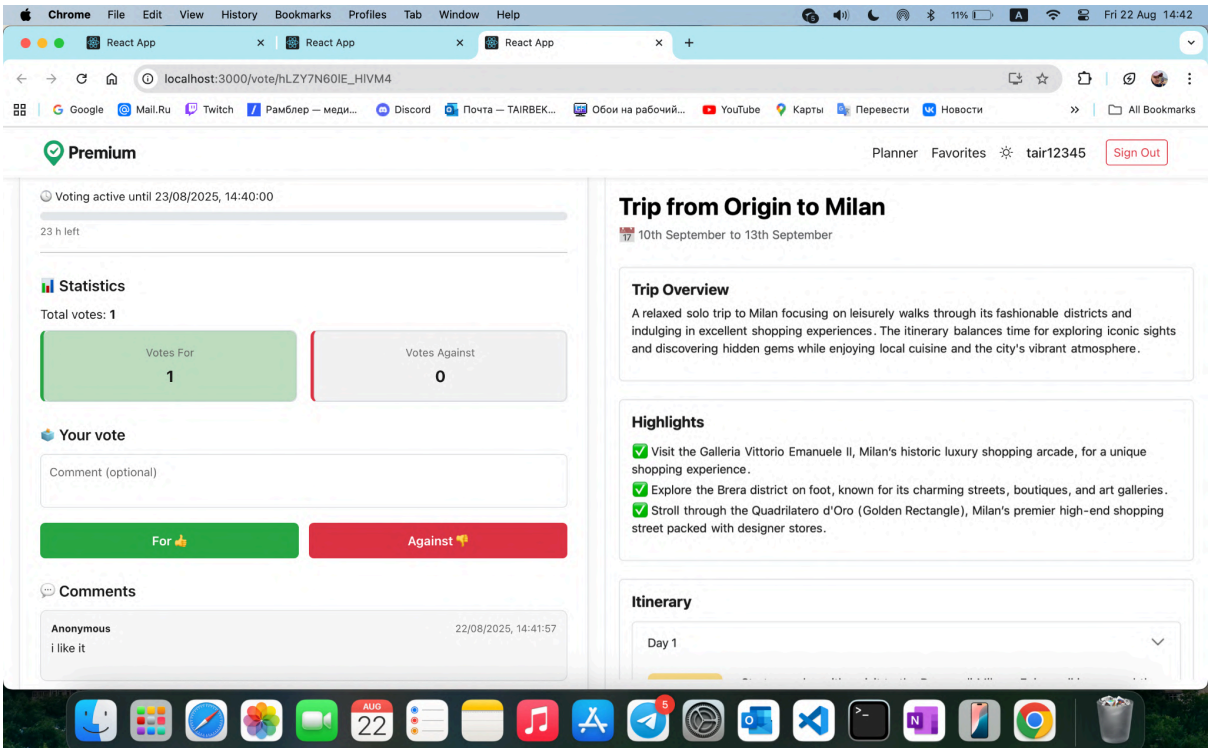

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Fri 22 Aug 14:40
React App
localhost:3000/favorites?just_saved=1
Premium
Planner Favorites tair12345 Sign Out
Saved Trips
Page 1 of 1 • 6 total
Trip Overview
Delete
Trip from Origin to Milan
10th September to 13th September
Trip Overview
A relaxed solo trip to Milan focusing on leisurely walks through its fashionable districts and indulging in excellent shopping experiences. The itinerary balances time for exploring iconic sights and discovering hidden gems while enjoying local cuisine and the city's vibrant atmosphere.
Highlights
✅ Visit the Galleria Vittorio Emanuele II, Milan's historic luxury shopping arcade, for a unique shopping experience.
✅ Explore the Brera district on foot, known for its charming streets, boutiques, and art galleries.
✅ Stroll through the Quadrilatero d'Oro (Golden Rectangle), Milan's premier high-end shopping street packed with designer stores.

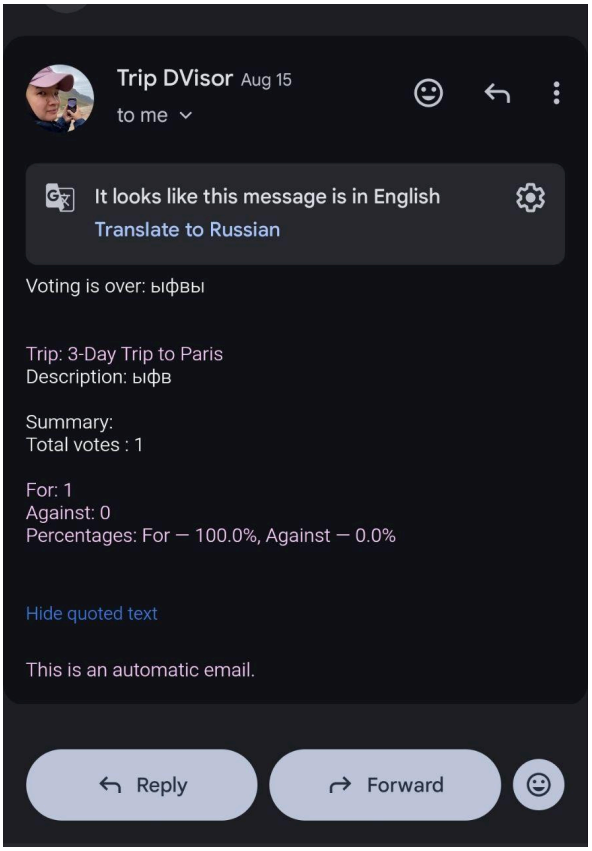

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Fri 22 Aug 14:40
React App
localhost:3000/favorites?just_saved=1
Premium
Planner Favorites tair12345 Sign Out
Create Voting
For "3-Day Trip to Milan"
Title
Description
Expires At
23.08.2025, 09:40 am
Votes
3
Allow anonymous voting
Cancel
+ Create
Saved Trips
Page 1 of 1 • 6 total

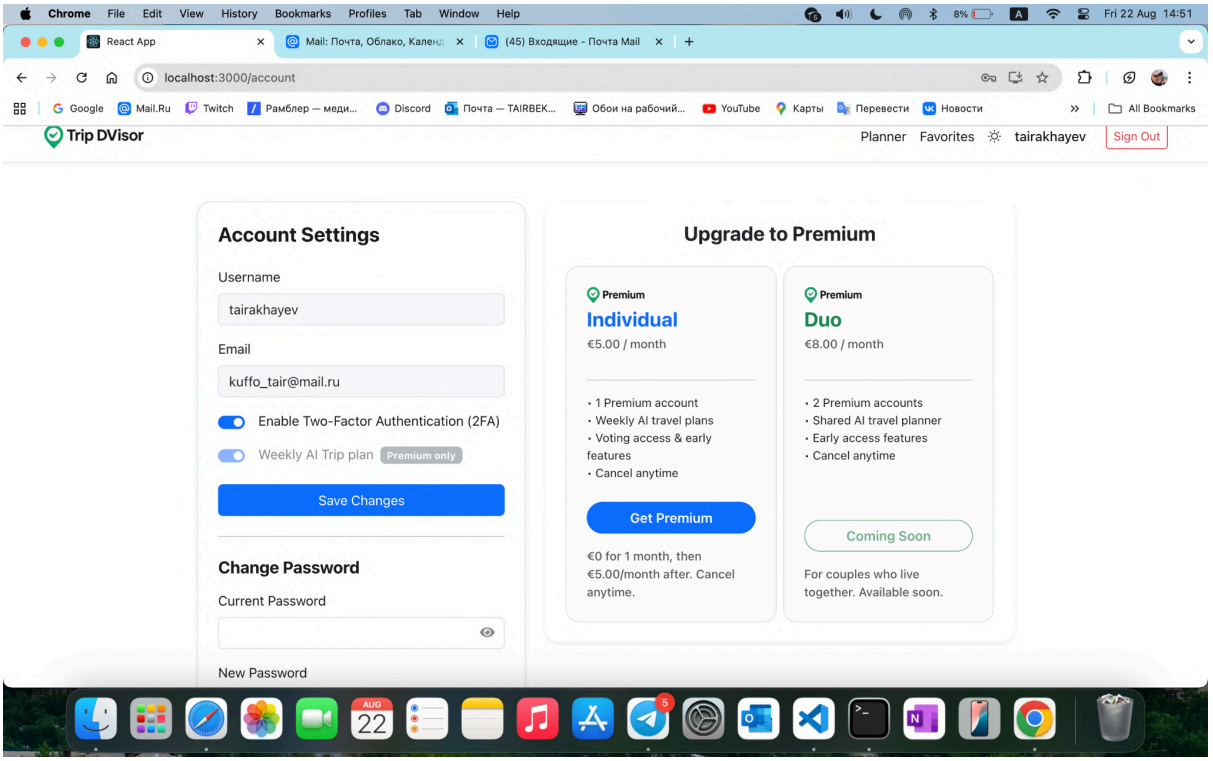
✅ Voting created! Link copied.

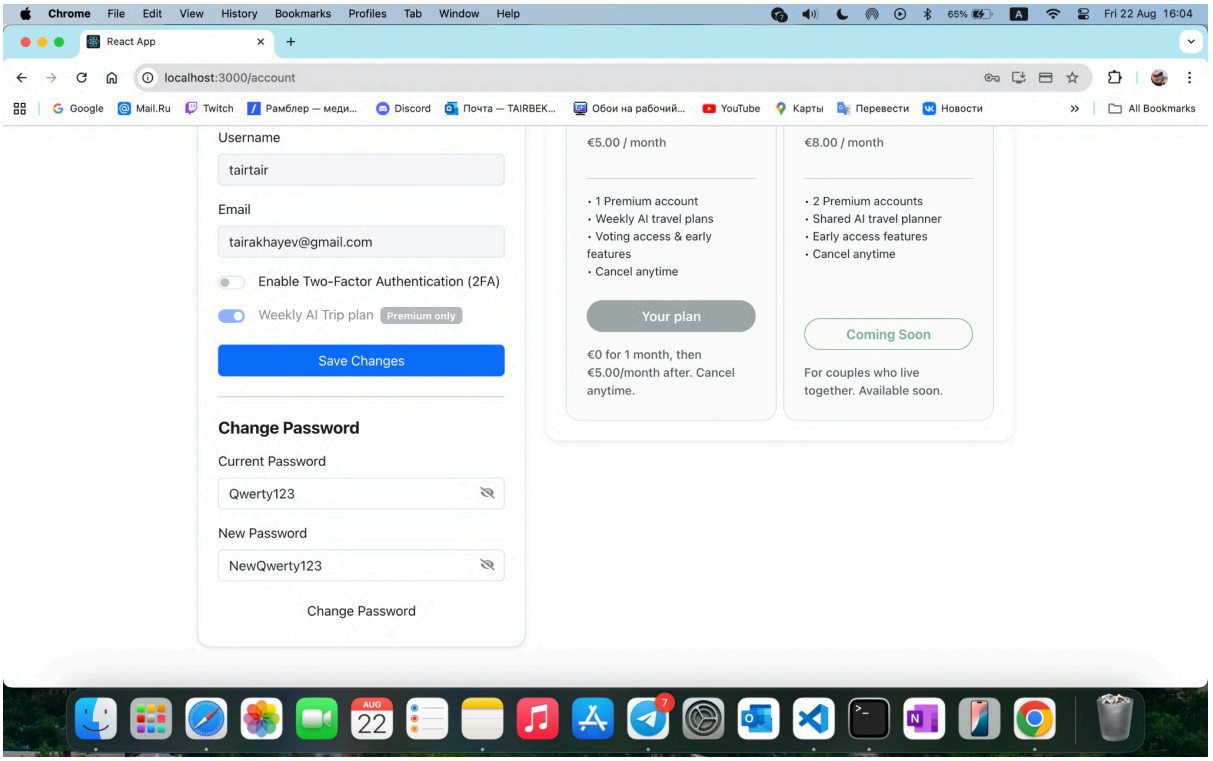
http://localhost:3000/vote/hLZY7N60lE_HlVM4

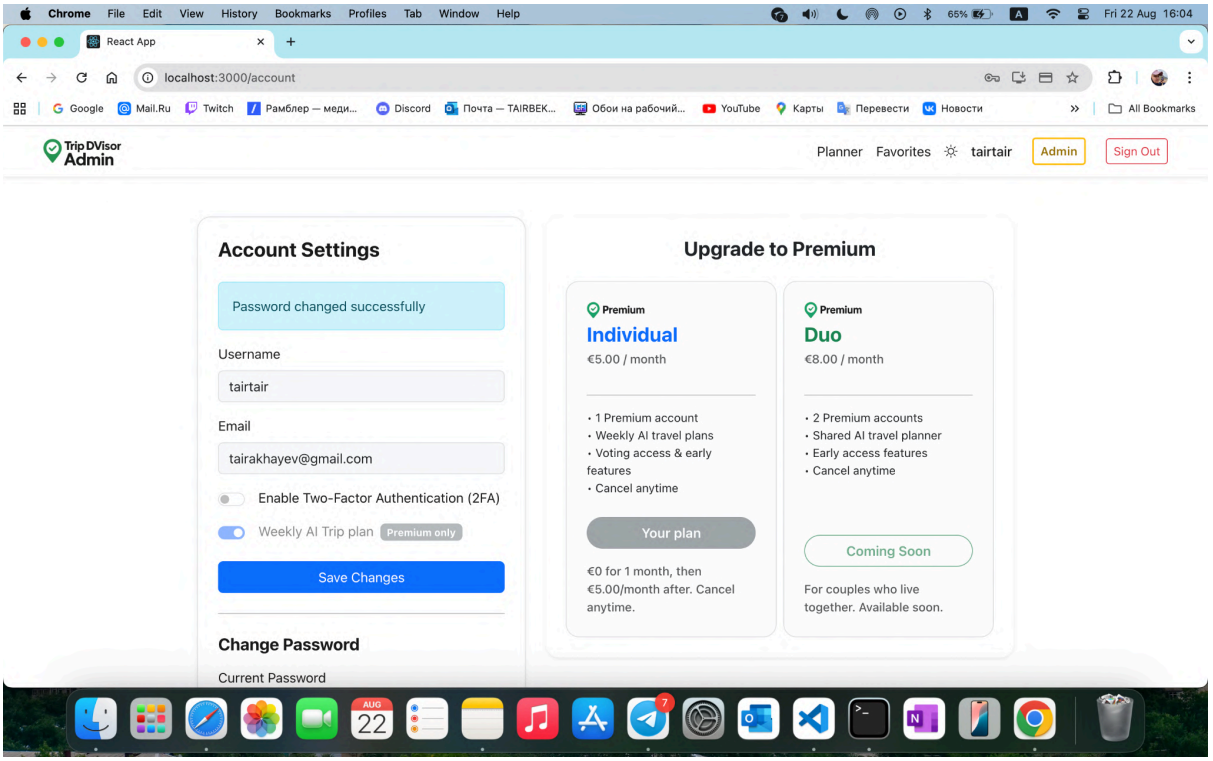





Trip DVisor Aug 15
to me
It looks like this message is in English
Translate to Russian
Voting is over: ыфвы
Trip: 3-Day Trip to Paris
Description: ыфв
Summary:
Total votes : 1
For: 1
Against: 0
Percentages: For — 100.0%, Against — 0.0%
Hide quoted text
This is an automatic email.
Reply
Forward


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Fri 22 Aug 14:51
React App
localhost:3000/account
Trip DVisor
Planner
Favorites
tairakhayev
Sign Out
Account Settings
Username
tairakhayev
Email
kuffo_tair@mail.ru
Enable Two-Factor Authentication (2FA)
Weekly AI Trip plan
Premium only
Save Changes
Change Password
Current Password
New Password
Upgrade to Premium
Premium
Individual
€5.00 / month
• 1 Premium account
• Weekly AI travel plans
• Voting access & early features
• Cancel anytime
Get Premium
€0 for 1 month, then €5.00/month after. Cancel anytime.
Premium
Duo
€8.00 / month
• 2 Premium accounts
• Shared AI travel planner
• Early access features
• Cancel anytime
Coming Soon
For couples who live together. Available soon.

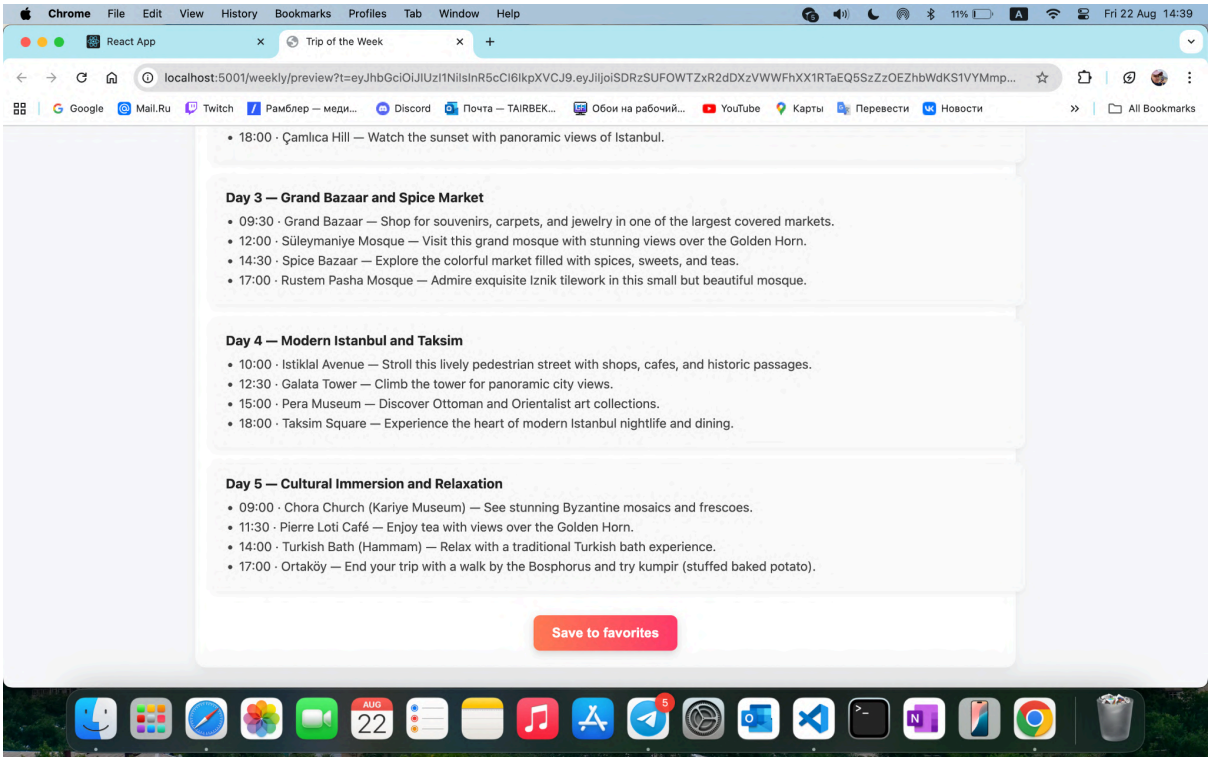

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Fri 22 Aug 16:04
React App
localhost:3000/account
Username
tairtair
Email
tairakhayev@gmail.com
Enable Two-Factor Authentication (2FA)
Weekly AI Trip plan
Premium only
Save Changes
Change Password
Current Password
Qwerty123
New Password
NewQwerty123
Change Password
€5.00 / month
• 1 Premium account
• Weekly AI travel plans
• Voting access & early features
• Cancel anytime
Your plan
€0 for 1 month, then €5.00/month after. Cancel anytime.
€8.00 / month
• 2 Premium accounts
• Shared AI travel planner
• Early access features
• Cancel anytime
Coming Soon
For couples who live together. Available soon.

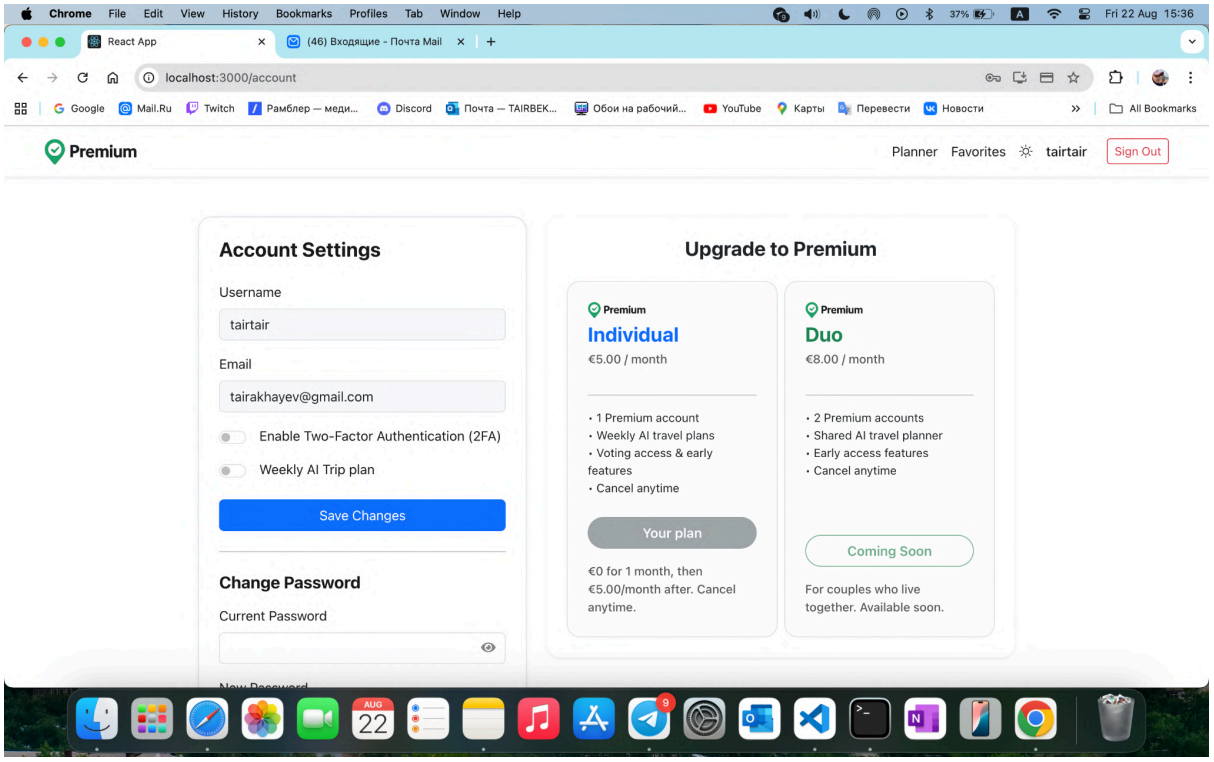


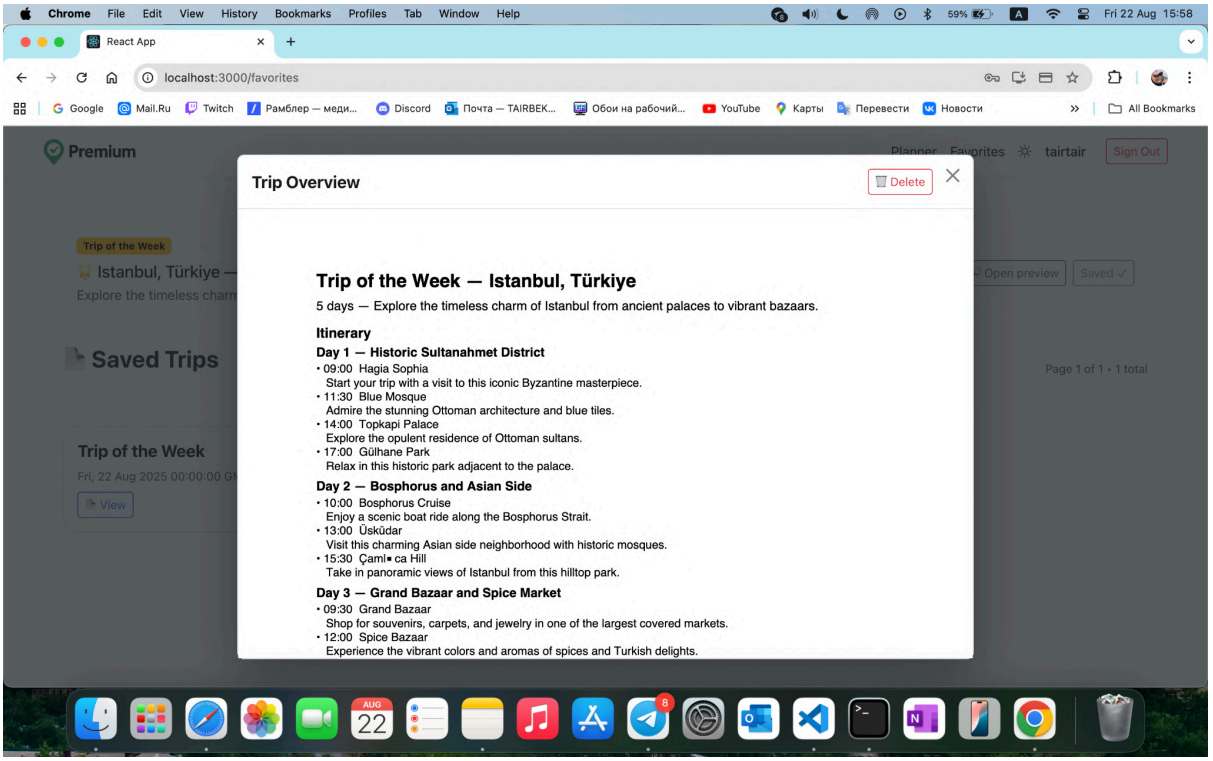
Subscription Activated

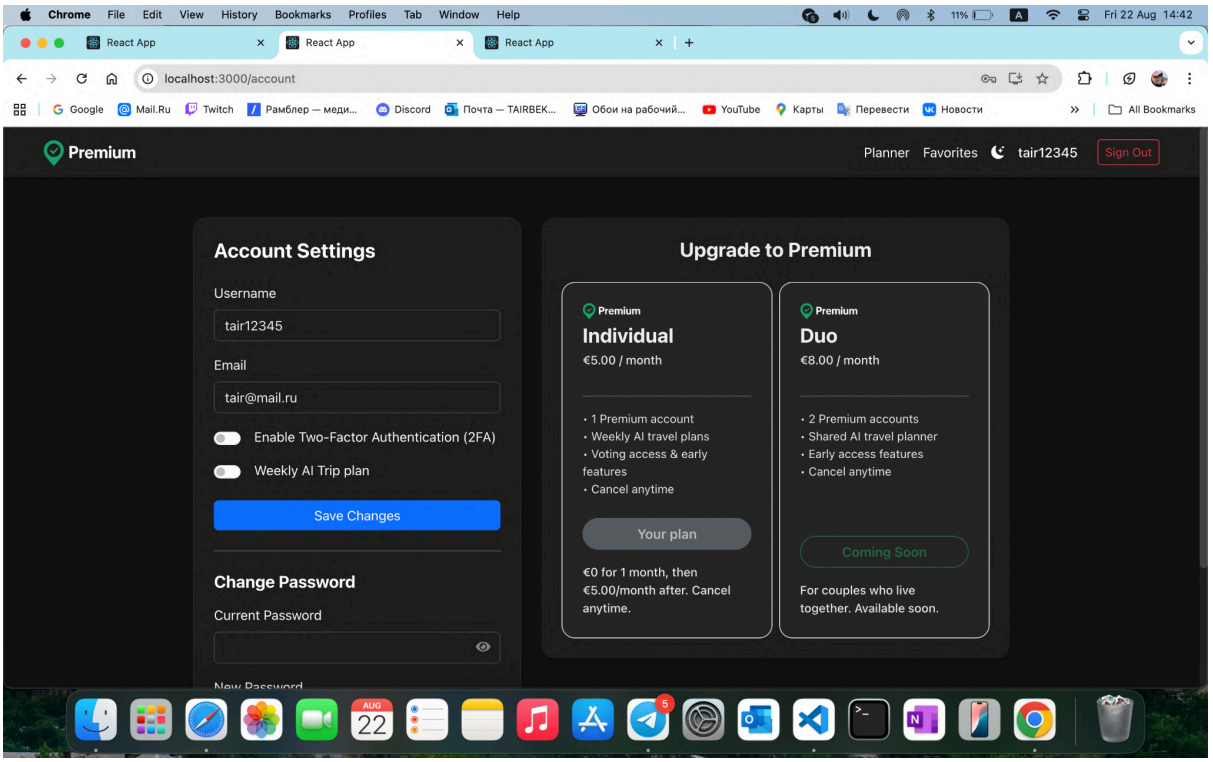
Trip DVisor <vperedmuslims@gmail.com>
to me

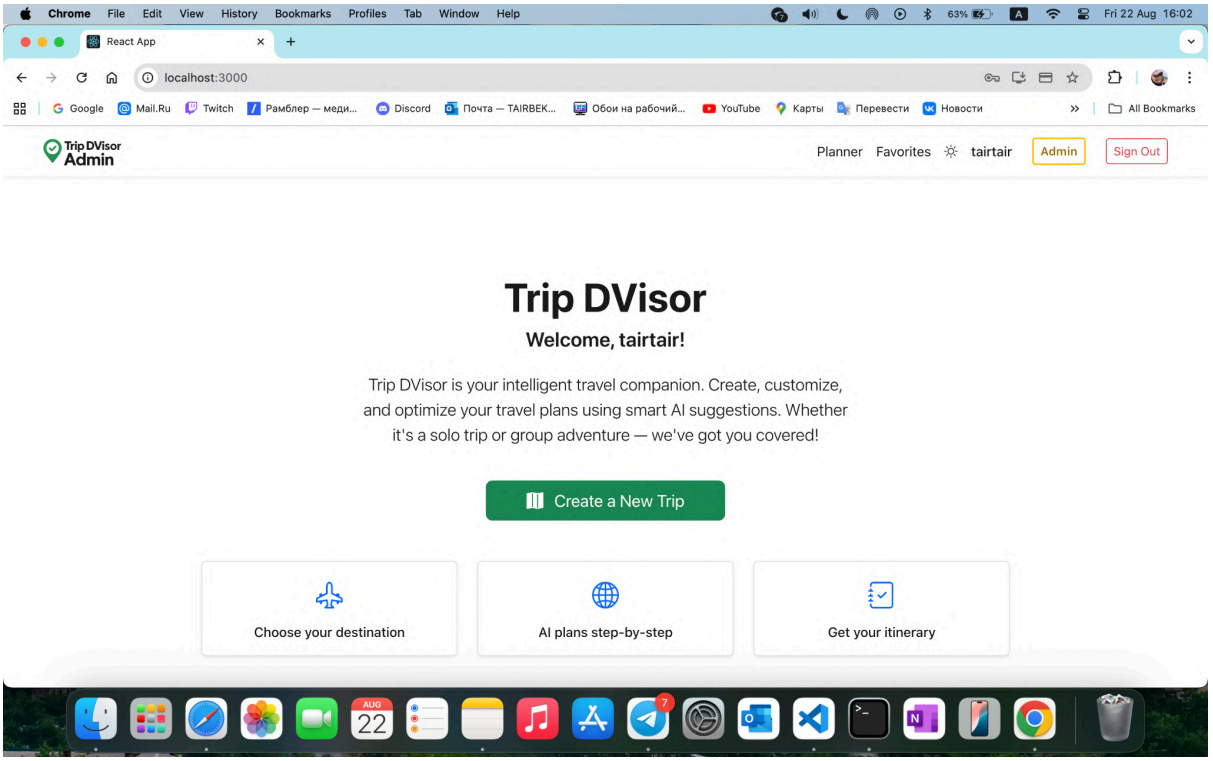
Thank you for subscribing to TripDVisor Premium! You now have access to exclusive features.

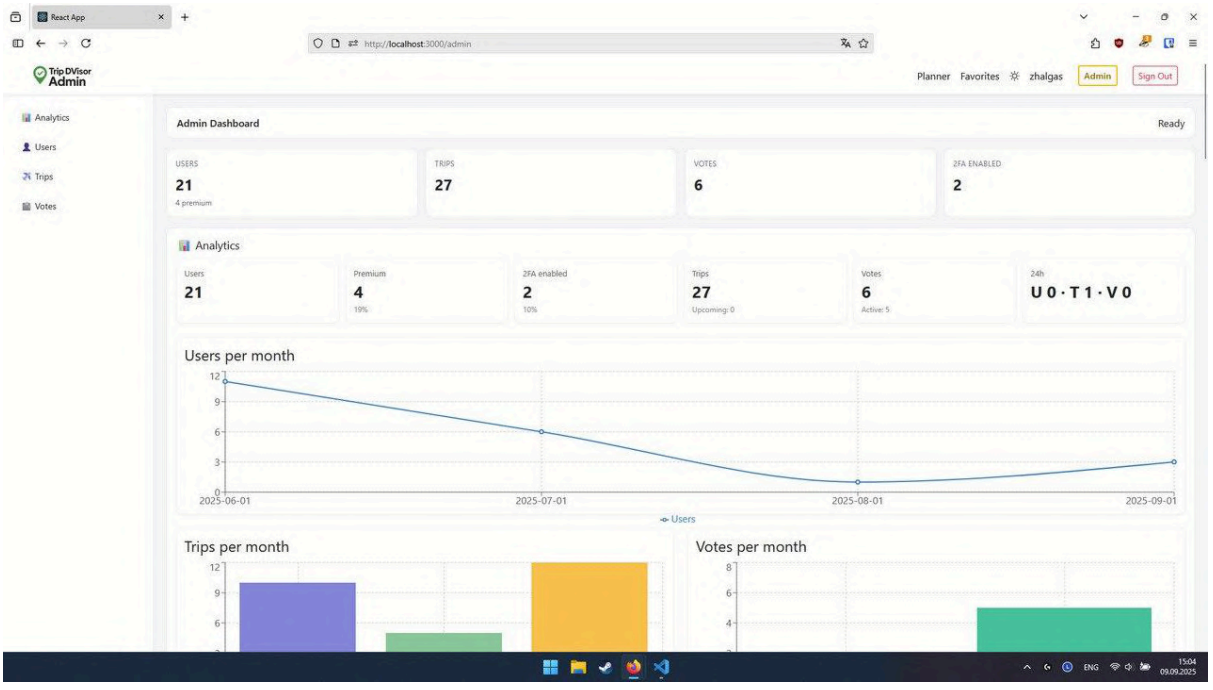

• 18:00 · Çamlıca Hill — Watch the sunset with panoramic views of Istanbul.
Day 3 — Grand Bazaar and Spice Market
• 09:30 · Grand Bazaar — Shop for souvenirs, carpets, and jewelry in one of the largest covered markets.
• 12:00 · Süleymaniye Mosque — Visit this grand mosque with stunning views over the Golden Horn.
• 14:30 · Spice Bazaar — Explore the colorful market filled with spices, sweets, and teas.
• 17:00 · Rustem Pasha Mosque — Admire exquisite Iznik tilework in this small but beautiful mosque.
Day 4 — Modern Istanbul and Taksim
• 10:00 · Istiklal Avenue — Stroll this lively pedestrian street with shops, cafes, and historic passages.
• 12:30 · Galata Tower — Climb the tower for panoramic city views.
• 15:00 · Pera Museum — Discover Ottoman and Orientalist art collections.
• 18:00 · Taksim Square — Experience the heart of modern Istanbul nightlife and dining.
Day 5 — Cultural Immersion and Relaxation
• 09:00 · Chora Church (Kariye Museum) — See stunning Byzantine mosaics and frescoes.
• 11:30 · Pierre Loti Café — Enjoy tea with views over the Golden Horn.
• 14:00 · Turkish Bath (Hammam) — Relax with a traditional Turkish bath experience.
• 17:00 · Ortaköy — End your trip with a walk by the Bosphorus and try kumpir (stuffed baked potato).
Save to favorites

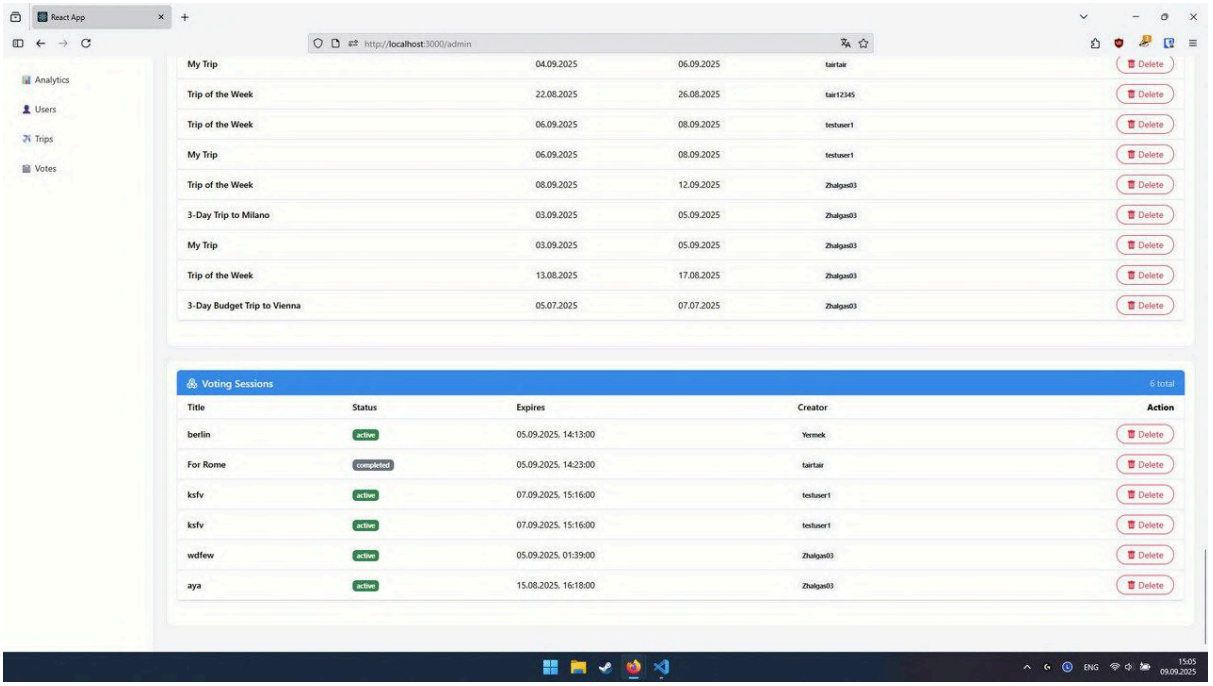

Premium
Planner
Favorites
tairtair
Sign Out
Account Settings
Username
tairtair
Email
tairakhayev@gmail.com
Enable Two-Factor Authentication (2FA)
Weekly AI Trip plan
Save Changes
Change Password
Current Password
Upgrade to Premium
Premium
Individual
€5.00 / month
• 1 Premium account
• Weekly AI travel plans
• Voting access & early features
• Cancel anytime
Your plan
€0 for 1 month, then €5.00/month after. Cancel anytime.
Premium
Duo
€8.00 / month
• 2 Premium accounts
• Shared AI travel planner
• Early access features
• Cancel anytime
Coming Soon
For couples who live together. Available soon.

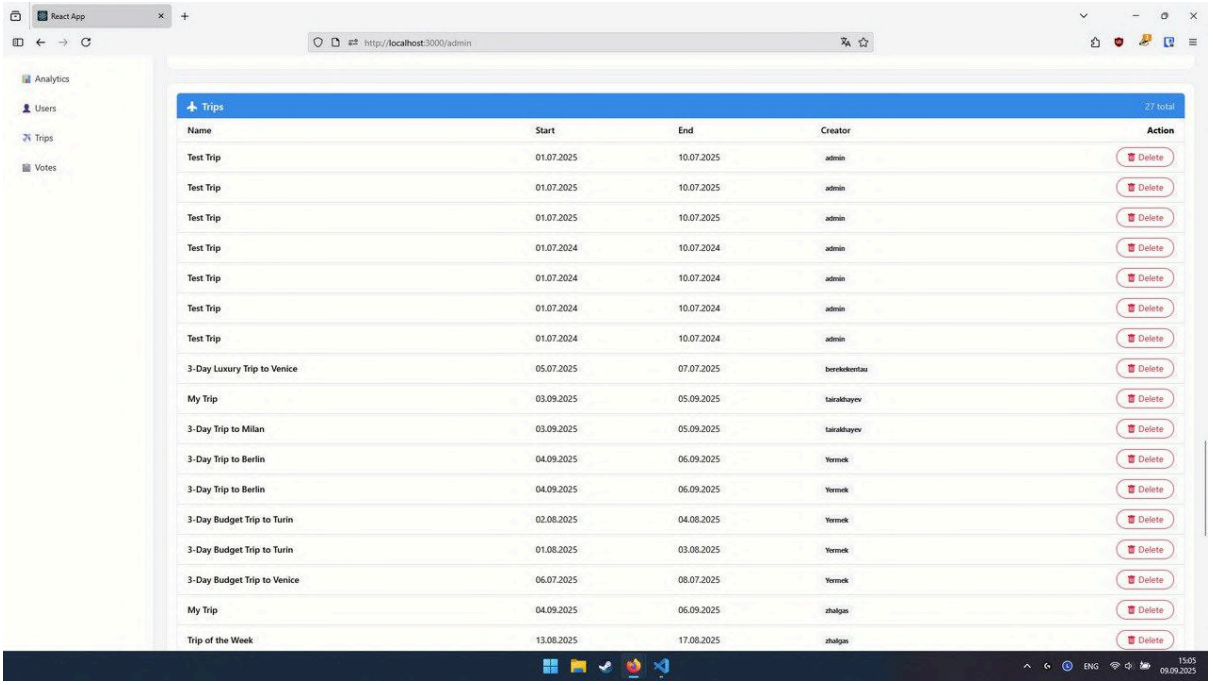

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Fri 22 Aug 15:58
React App
localhost:3000/favorites
Premium
Planner Favorites tairtair Sign Out
Trip Overview
Delete
Trip of the Week — Istanbul, Türkiye
5 days — Explore the timeless charm of Istanbul from ancient palaces to vibrant bazaars.
Itinerary
Day 1 — Historic Sultanahmet District
• 09:00 Hagia Sophia
Start your trip with a visit to this iconic Byzantine masterpiece.
• 11:30 Blue Mosque
Admire the stunning Ottoman architecture and blue tiles.
• 14:00 Topkapi Palace
Explore the opulent residence of Ottoman sultans.
• 17:00 Gülhane Park
Relax in this historic park adjacent to the palace.
Day 2 — Bosphorus and Asian Side
• 10:00 Bosphorus Cruise
Enjoy a scenic boat ride along the Bosphorus Strait.
• 13:00 Üsküdar
Visit this charming Asian side neighborhood with historic mosques.
• 15:30 Çamlıca Hill
Take in panoramic views of Istanbul from this hilltop park.
Day 3 — Grand Bazaar and Spice Market
• 09:30 Grand Bazaar
Shop for souvenirs, carpets, and jewelry in one of the largest covered markets.
• 12:00 Spice Bazaar
Experience the vibrant colors and aromas of spices and Turkish delights.
Saved Trips
Trip of the Week
Open preview
Saved
Page 1 of 1 • 1 total

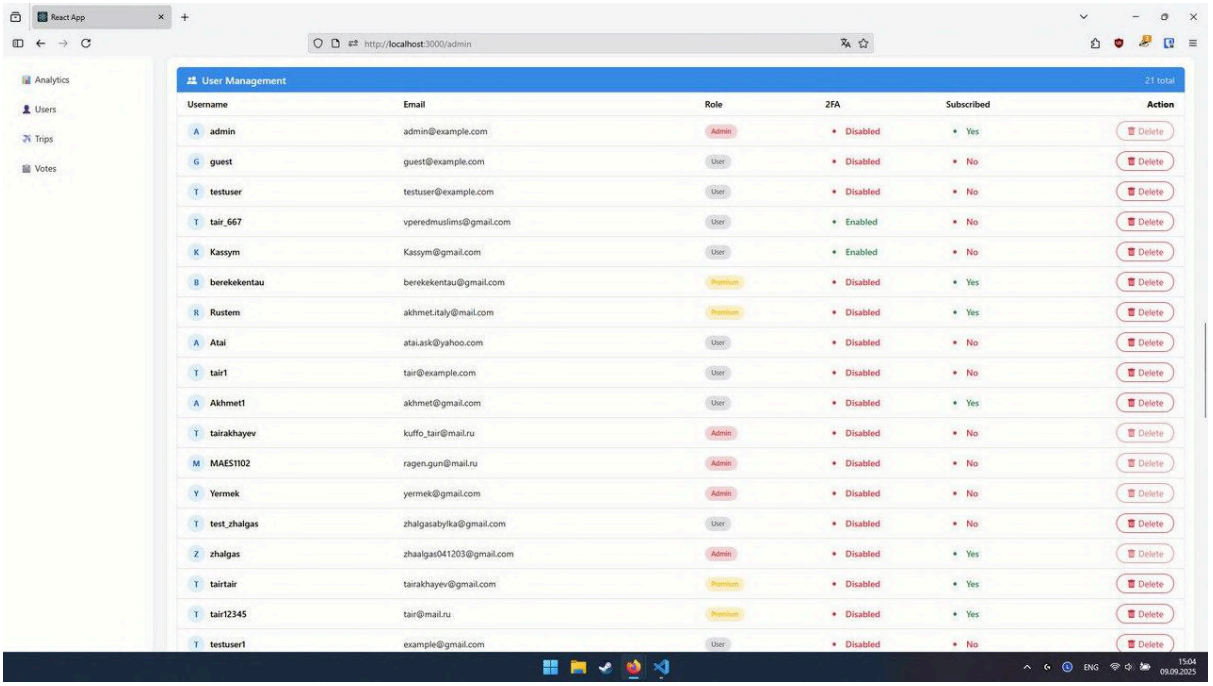

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Fri 22 Aug 14:42
React App
localhost:3000/account
Premium
Planner Favorites tair12345 Sign Out
Account Settings
Username
tair12345
Email
tair@mail.ru
Enable Two-Factor Authentication (2FA)
Weekly AI Trip plan
Save Changes
Change Password
Current Password
Upgrade to Premium
Premium
Individual
€5.00 / month
• 1 Premium account
• Weekly AI travel plans
• Voting access & early features
• Cancel anytime
Your plan
€0 for 1 month, then €5.00/month after. Cancel anytime.
Premium
Duo
€8.00 / month
• 2 Premium accounts
• Shared AI travel planner
• Early access features
• Cancel anytime
Coming Soon
For couples who live together. Available soon.

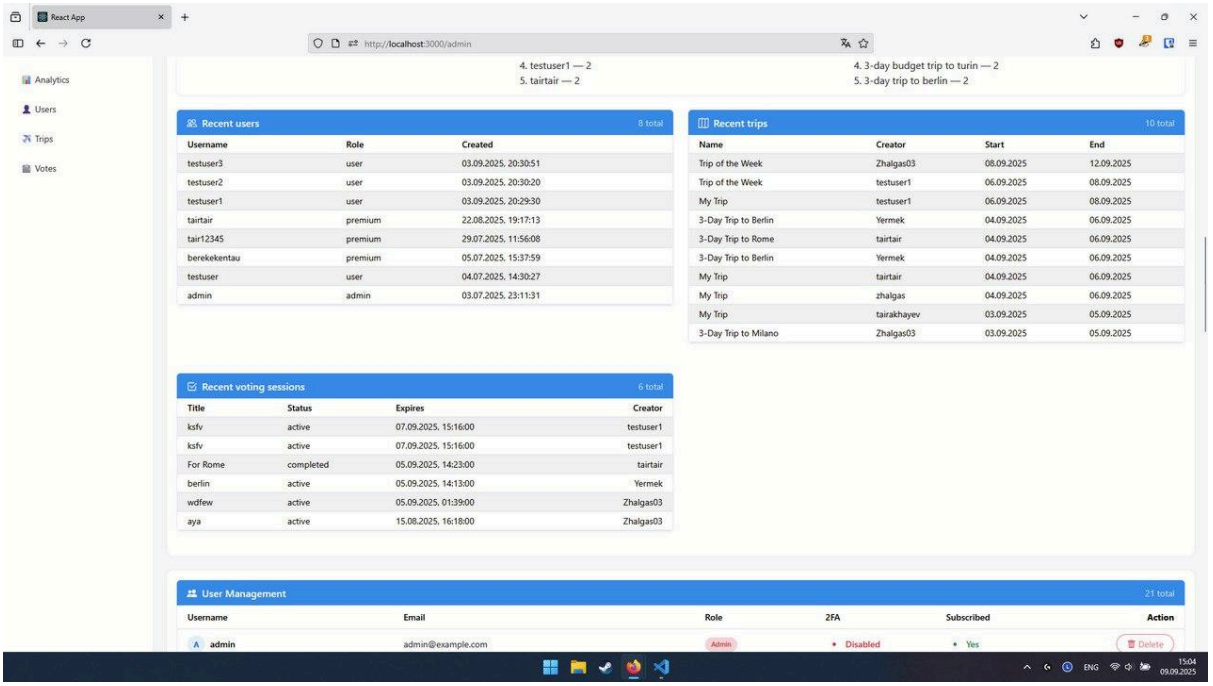

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Fri 22 Aug 16:02
React App
localhost:3000
Trip DVisor Admin
Planner Favorites tairtair Admin Sign Out
Trip DVisor
Welcome, tairtair!
Trip DVisor is your intelligent travel companion. Create, customize, and optimize your travel plans using smart AI suggestions. Whether it's a solo trip or group adventure — we've got you covered!
Create a New Trip
Choose your destination
AI plans step-by-step
Get your itinerary

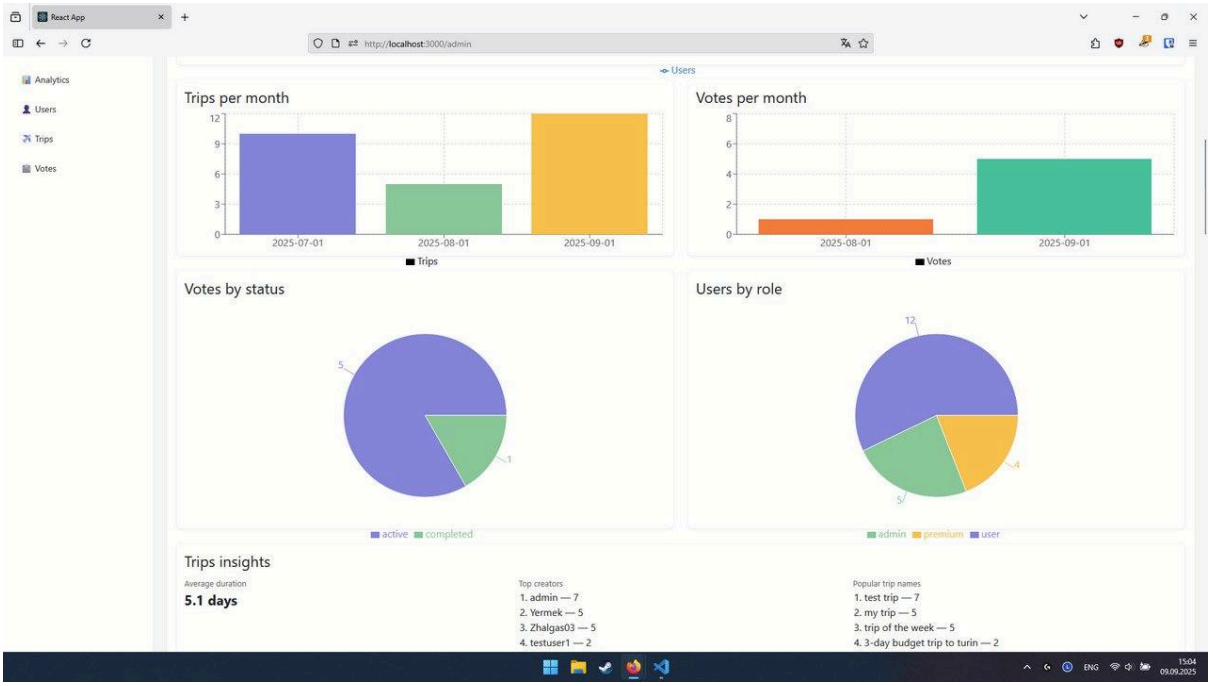

React App
http://localhost:3000/admin
Trip DVisor Admin
Planner Favorites zhalgas Admin Sign Out
Analytics
Users
Trips
Votes
Admin Dashboard
Ready
USERS 21 4 premium
TRIPS 27
VOTES 6
2FA ENABLED 2
Analytics
Users 21
Premium 4 19%
2FA enabled 2 10%
Trips 27 Upcoming: 0
Votes 6 Active: 5
24h U0·T1·V0
Users per month
2025-06-01
2025-07-01
2025-08-01
2025-09-01
Users
Trips per month
Votes per month


React App
http://localhost:3000/admin
Analytics
Users
Trips
Votes
My Trip 04.09.2025 06.09.2025 tairtair Delete
Trip of the Week 22.08.2025 26.08.2025 tair12345 Delete
Trip of the Week 06.09.2025 08.09.2025 testuser1 Delete
My Trip 06.09.2025 08.09.2025 testuser1 Delete
Trip of the Week 08.09.2025 12.09.2025 Zhalgas03 Delete
3-Day Trip to Milano 03.09.2025 05.09.2025 Zhalgas03 Delete
My Trip 03.09.2025 05.09.2025 Zhalgas03 Delete
Trip of the Week 13.08.2025 17.08.2025 Zhalgas03 Delete
3-Day Budget Trip to Vienna 05.07.2025 07.07.2025 Zhalgas03 Delete
Voting Sessions 6 total
Title Status Expires Creator Action
berlin active 05.09.2025, 14:13:00 Yermek Delete
For Rome completed 05.09.2025, 14:23:00 tairtair Delete
ksfv active 07.09.2025, 15:16:00 testuser1 Delete
ksfv active 07.09.2025, 15:16:00 testuser1 Delete
wdfew active 05.09.2025, 01:39:00 Zhalgas03 Delete
aya active 15.08.2025, 16:18:00 Zhalgas03 Delete
ENG
15:05
09.09.2025

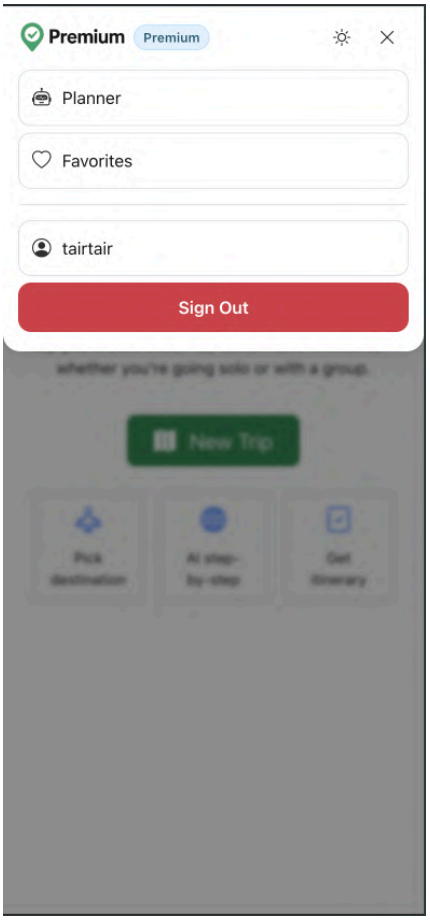

React App
http://localhost:3000/admin
Analytics
Users
Trips
Votes
Trips 27 total
Name Start End Creator Action
Test Trip 01.07.2025 10.07.2025 admin Delete
Test Trip 01.07.2025 10.07.2025 admin Delete
Test Trip 01.07.2025 10.07.2025 admin Delete
Test Trip 01.07.2024 10.07.2024 admin Delete
Test Trip 01.07.2024 10.07.2024 admin Delete
Test Trip 01.07.2024 10.07.2024 admin Delete
Test Trip 01.07.2024 10.07.2024 admin Delete
3-Day Luxury Trip to Venice 05.07.2025 07.07.2025 berekekentau Delete
My Trip 03.09.2025 05.09.2025 tairaldiayev Delete
3-Day Trip to Milan 03.09.2025 05.09.2025 tairaldiayev Delete
3-Day Trip to Berlin 04.09.2025 06.09.2025 Yermek Delete
3-Day Trip to Berlin 04.09.2025 06.09.2025 Yermek Delete
3-Day Budget Trip to Turin 02.08.2025 04.08.2025 Yermek Delete
3-Day Budget Trip to Turin 01.08.2025 03.08.2025 Yermek Delete
3-Day Budget Trip to Venice 06.07.2025 08.07.2025 Yermek Delete
My Trip 04.09.2025 06.09.2025 zhalgas Delete
Trip of the Week 13.08.2025 17.08.2025 zhalgas Delete
ENG
15:05
09.09.2025

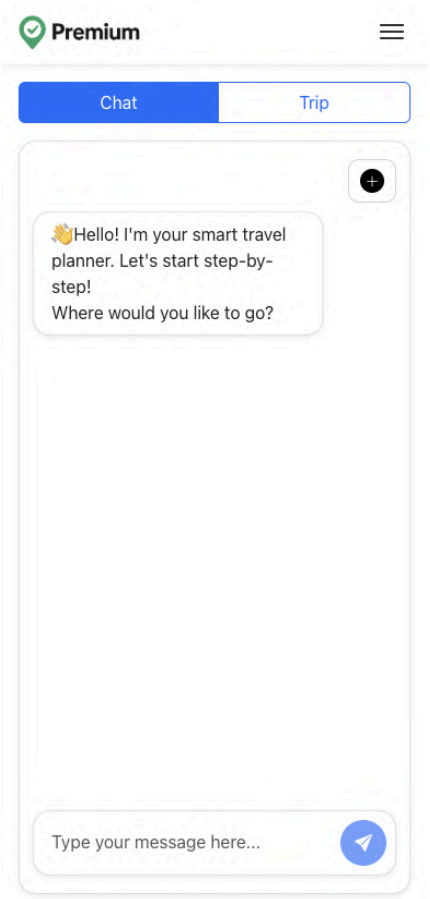

React App
http://localhost:3000/admin
Analytics
Users
Trips
Votes
User Management
21 total
Username
Email
Role
2FA
Subscribed
Action
admin
admin@example.com
Admin
Disabled
Yes
Delete
guest
guest@example.com
User
Disabled
No
testuser
testuser@example.com
tair_667
vperedmuslims@gmail.com
Enabled
Kassym
Kassym@gmail.com
berekekentau
berekekentau@gmail.com
Premium
Rustem
akhmet.italy@mail.com
Atai
atai.ask@yahoo.com
tair1
tair@example.com
Akhmet1
akhmet@gmail.com
tairakhayev
kuffo_tair@mail.ru
MAES1102
ragen.gun@mail.ru
Yermek
yermek@gmail.com
test_zhalgas
zhalgasabylka@gmail.com
zhalgas
zhaalgas041203@gmail.com
tairtair
tairakhayev@gmail.com
tair12345
tair@mail.ru
testuser1
example@gmail.com
ENG
15:04
09.09.2025

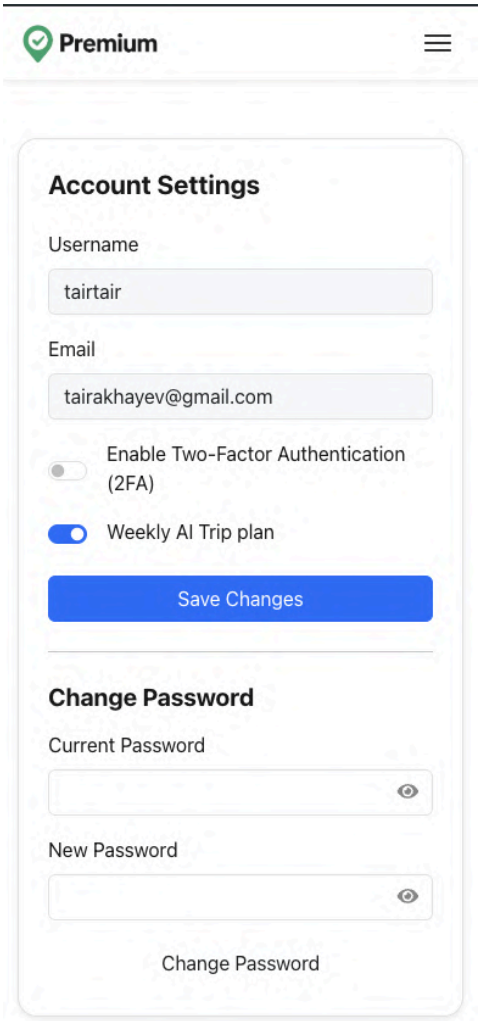

React App
http://localhost:3000/admin
Analytics
Users
Trips
Votes
4. testuser1 — 2
5. tairtair — 2
4. 3-day budget trip to turin — 2
5. 3-day trip to berlin — 2
Recent users
8 total
Username
Role
Created
testuser3
user
03.09.2025, 20:30:51
testuser2
user
03.09.2025, 20:30:20
testuser1
user
03.09.2025, 20:29:30
tairtair
premium
22.08.2025, 19:17:13
tair12345
premium
29.07.2025, 11:56:08
berekekentau
premium
05.07.2025, 15:37:59
testuser
user
04.07.2025, 14:30:27
admin
admin
03.07.2025, 23:11:31
Recent trips
10 total
Name
Creator
Start
End
Trip of the Week
Zhalgas03
08.09.2025
12.09.2025
Trip of the Week
testuser1
06.09.2025
08.09.2025
My Trip
testuser1
06.09.2025
08.09.2025
3-Day Trip to Berlin
Yermek
04.09.2025
06.09.2025
3-Day Trip to Rome
tairtair
04.09.2025
06.09.2025
3-Day Trip to Berlin
Yermek
04.09.2025
06.09.2025
My Trip
tairtair
04.09.2025
06.09.2025
My Trip
zhalgas
04.09.2025
06.09.2025
My Trip
tairakhayev
03.09.2025
05.09.2025
3-Day Trip to Milano
Zhalgas03
03.09.2025
05.09.2025
Recent voting sessions
6 total
Title
Status
Expires
Creator
ksfv
active
07.09.2025, 15:16:00
testuser1
ksfv
active
07.09.2025, 15:16:00
testuser1
For Rome
completed
05.09.2025, 14:23:00
tairtair
berlin
active
05.09.2025, 14:13:00
Yermek
wdfew
active
05.09.2025, 01:39:00
Zhalgas03
aya
active
15.08.2025, 16:18:00
Zhalgas03
User Management
21 total
Username
Email
Role
2FA
Subscribed
Action
admin
admin@example.com
Admin
Disabled
Yes
Delete
ENG
15:04
09.09.2025



9.2 Mobile view




Premium
Chat
Trip
👋Hello! I'm your smart travel planner. Let's start step-by-step!
Where would you like to go?
Type your message here...


Premium
Account Settings
Username
tairtair
Email
tairakhayev@gmail.com
Enable Two-Factor Authentication (2FA)
Weekly AI Trip plan
Save Changes
Change Password
Current Password
New Password
Change Password