

Thesis submitted to the
UNIVERSITY OF MOHAMED BOUDIAF – MSILA



FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

In partial fulfillment of the requirements for the degree of
Master in Computer science

By
Attig, Youcef Massaoud
Refice, Ilies

Title of the thesis

TripTale: A Personalized Travel Exploration and Sharing Platform

Under the supervision of

Adel Moussaoui

Composition of the jury

Ali DABBA

University of Msila

President

Adel Moussaoui

University of Msila

Reporter

ABDELGHANI Boudaa

University of Msila

Examiner

June, 2026

Abstract

TripTale is a mobile application designed to enhance travel experiences through trip discovery, planning, and social interaction. The application allows users to explore travel destinations, create and share trip posts, save favorite locations, and interact with a travel community. The system was developed using React Native for the mobile frontend and a modern backend architecture with database integration and cloud services. Additional features such as personalized recommendations, trip analytics, premium subscriptions, and media sharing were integrated to improve user engagement and usability. The project aimed to provide travelers with a modern and intuitive platform that combines travel planning with social networking functionalities. The final application demonstrates efficient performance, responsive user interface design, and scalable architecture suitable for future improvements and deployment.

Keywords: Mobile Application, React Native, Travel Platform, Social Networking, Trip Planning.

الملخص

TripTale هو تطبيق هاتف ذكي يهدف إلى تحسين تجربة السفر من خلال اكتشاف الرحلات، التخطيط لها، والتفاعل الاجتماعي بين المستخدمين. يتيح التطبيق للمستخدمين استكشاف الوجهات السياحية، إنشاء ومشاركة منشورات السفر، حفظ الأماكن المفضلة، والتفاعل مع مجتمع خاص بالسفر. تم تطوير النظام باستخدام React Native للواجهة الأمامية مع بنية خلفية حديثة وقاعدة بيانات وخدمات سحابية. كما تم دمج ميزات إضافية مثل التوصيات المخصصة، إحصائيات الرحلات، الاشتراكات المميزة، ومشاركة الوسائط لتحسين تجربة المستخدم. يهدف المشروع إلى توفير منصة حديثة وسهلة الاستخدام تجمع بين تخطيط الرحلات وميزات الشبكات الاجتماعية. أظهرت النتائج النهائية تطبيقاً يتميز بالأداء الجيد، وواجهة مستخدم متجاوبة، وبنية قابلة للتطوير مستقبلاً.

الكلمات المفتاحية: تطبيقات الهاتف، React Native، السفر، الشبكات الاجتماعية، تخطيط الرحلات

Dedications

"I dedicate this work to my beloved parents and my brother, whose endless love, sacrifices, support, and encouragement have been the foundation of everything I have achieved. Your patience and belief in me gave me the strength to continue throughout this journey.

To the memory of my deceased brother, your presence remains forever in my heart. This achievement is dedicated to you with love and remembrance.

I would also like to dedicate this work to my close friends, who stood beside me during difficult moments, encouraged me, and made this journey more meaningful.

Finally, I dedicate this thesis to everyone who helped, supported, and guided me throughout the development of this project and during my academic journey.

Thank you all for being part of this achievement."

Attig Youcef Massoud

تشكرات

" بدايةً، نحمد الله ونشكره على ما وفقنا إليه من إنجاز هذا البحث .

ونود أن نتقدم بجزيل الشكر والتقدير إلى كل من ساهم في إنجاز هذا البحث وقدم لنا الدعم والمساندة، ونخص بالذكر الأستاذ المشرف مساوي عادل على إشرافه المباشر وتوجيهاته القيمة التي ساعدتنا على إنجاز البحث على أكمل وجه .

نقدر ونثمن مساهمات الجميع ونأمل أن يكون هذا البحث مفيداً وذا قيمة علمية "

الياس ارفيس

Acknowledgments

We are deeply indebted to our thesis supervisor Adel Moussaoui whose unlimited steadfast support and inspirations have made this project a great success. In a very special way, we thank him for every support he has rendered unto us to see that we succeed in this challenging study.

Special thanks go to our friends and families who have contained the hectic moments and stress we have been through during the course of the research project.

We thank the school for giving us the grand opportunity to work as a team which has indeed promoted our team work spirit and communication skills. We also thank the individual group members for the good team spirit and solidarity.

Contents

تَشْكُرات	4
GENERAL INTRODUCTION	12
This thesis is organized as follows:	13
1. Chapter 1 Study and Analysis of Existing Solutions	14
1.1 Introduction to Travel Applications	14
1.2 Comparative Study of Existing Applications	15
1.2.1 TripAdvisor	15
1.2.2 Polarsteps	16
1.2.3 Airbnb	16
1.2.4 Instagram	16
1.2.5 Synthesis	16
1.3 Problem Statement	17
1.3.1 Difficulty of organizing trips	17
1.4 Proposed Solution: TripTale	18
1.5 Functional Requirements	20
1.6 Non-Functional Requirements	21
1.7 Conclusion	22
2. Chapter 2 — System Analysis and Design (TripTale)	23
2.1 Development Methodology	24
2.1.1 Overview: Why Agile for TripTale	24
2.1.2 Step by Step Development Approach	24
2.2 Use Case Diagram	25
2.2.1 Actors	25
2.2.4 Use Case Diagram	26
2.3 Application Flowchart	27

2.3.1 Authentication Flow (App Start → Valid Session → App).....	28
2.3.2 Discover Flow (Categories → Trips → Trip Details).....	29
2.3.4 Profile Flow (My Trips → Create → Preview → Publish).....	30
2.3.5 Saved Trips Flow (Planning Tab).....	31
2.3.6 Logout Flow	32
2.4 System Architecture.....	32
2.4.1 High-Level Architecture	32
2.4.2 Frontend Architecture	32
2.4.3 Backend Architecture.....	33
2.4.4 API Communication	33
2.4.5 Database Choice: SQLite → PostgreSQL	33
2.5 Database Design.....	34
2.5.1 ER Design Goals.....	34
2.5.2 Relationships Summary (Cardinality)	35
2.5.3 Normalization and Integrity.....	35
2.5.4 Indexing (Performance Considerations).....	35
2.6 UI/UX Design	36
2.6.1 Navigation Structure and Key Screens	36
2.6.2 Mobile-First Design Principles.....	38
2.6.3 Visual Design and Color/Brand Choices.....	38
2.6.4 UX Decisions Explained.....	39
2.7 Conclusion.....	39
3. Chapter 3 Technologies and Development Tools	40
3.1 Frontend Technologies.....	41
3.1.1 React Native.....	41
3.1.2 Expo and Expo Router.....	42
3.1.3 JSX and the React component model	42

3.1.4	Navigation and supporting libraries	42
3.2	Backend Technologies	43
3.2.1	Node.js	43
3.2.2	Express	43
3.2.3	REST API architecture	44
3.2.4	Authentication and security middleware	44
3.2.5	Validation, uploads, logging, and documentation	45
3.3	Database Technologies	45
3.3.1	PostgreSQL	46
3.3.2	Prisma.....	46
3.3.3	Seeding and demo data	46
3.4	Development Tools	46
3.4.1	Visual Studio Code	47
3.4.2	Git and GitHub.....	47
3.4.3	Docker and Docker Compose	47
3.4.4	npm, ESLint, and the TypeScript compiler	48
3.4.5	API exploration and mobile preview	48
3.5	Justification of Technology Choices	49
3.5.1	Cross-platform development.....	49
3.5.2	Rapid iteration.....	49
3.5.3	Scalability and robustness.....	49
3.5.4	Community and ecosystem	50
3.5.5	Open-source licensing and cost	50
3.6	Conclusion.....	50
4.	Chapter 4 — Implementation of TripTale	51
4.1	Authentication System	51
4.1.1	Signup (Registration)	51

4.1.2 Login	52
4.1.3 Token / Session Storage.....	52
4.1.4 AuthContext.....	52
4.2 Trip Creation Module	53
4.2.1 Trip Publishing Flow	53
4.2.2 Image Uploading.....	53
4.2.3 Preview System.....	54
4.2.4 Edit / Discard Logic	54
4.3 Premium Features	54
4.3.1 Premium Badge.....	54
4.3.2 Analytics	54
4.3.3 Unlimited Uploads	54
4.4 API and Backend Implementation.....	55
4.4.1 API Routes	55
4.4.2 Request / Response Flow	55
4.4.3 Backend Validation.....	55
4.5 Database Operations	55
4.5.1 CRUD Overview.....	55
4.5.2 Prisma Usage.....	55
4.6 State Management.....	56
4.6.1 Context API.....	56
4.6.2 Hooks	56
4.6.3 Global State Handling.....	56
4.7 Security and Error Handling	56
4.7.1 Validation.....	56
4.7.2 Token Protection.....	56
4.7.3 Error Messages.....	56

4.7.4 Protected Routes	56
4.8 Challenges Encountered During Development	57
4.8.1 Authentication Issues	57
4.8.2 Database Relationships	57
4.8.3 Navigation Bugs.....	57
4.8.4 Image Uploads	57
4.8.5 Synchronization Problems	57
4.9 Future Improvements	58
4.9.1 the expansion of the premium subscription:.....	58
4.9.2 travel agencies:.....	58
4.9.3 AI enhancement:	58
4.9.4 Interactive maps:	58
4.9.5 booking systems:.....	58
4.9.6 real-time chat:	58
4.10 Conclusion.....	59
General Conclusion.....	60
References	61

Table of Contents

Figure 1-1: existing apps.....	15
Figure 1-2: Comparative feature matrix.	17
Figure 1-3: User journey before and with TripTale.	18
Figure 4: TripTale value proposition.	19
Figure 2-1: use-case diagram.....	26
Figure 2-2: flowchart diagram	27
Figure 2-3: app start (flowchart).....	28
Figure 2-4: discover tab (flowchart)	29
Figure 2-5: profile tab (flowchart)	30
Figure 2-6: saved tab (flowchart).....	31
Figure 2-7: logout (flowchart)	32
Figure 2-8: ER diagram.....	34
Figure 2-9: discover tab UI/UX	36
Figure 2-10: saved tab UI/UX.....	37
Figure 2-11: profile tab UI/UX	38
Figure 1-1: Three-tier architecture of TripTale.	40
Figure 1-2: TripTale technology stack.....	41
Figure 1-3: Backend module organization.	43
Figure 1-4: JWT authentication flow.....	45
Figure 1-5: Docker Compose stack.	48

GENERAL INTRODUCTION

The rapid growth of mobile technologies and social networking platforms has significantly transformed the way people plan, experience, and share their travels. Modern travelers increasingly rely on digital applications to discover destinations, obtain practical information, organize itineraries, and communicate with other travelers. Although many travel applications exist, most focus on a specific aspect of the travel experience, such as destination discovery, trip planning, or social interaction, requiring users to switch between multiple platforms to satisfy their needs.

In Algeria, this limitation is even more evident. While international platforms provide access to global travel content, they often lack localized information tailored to Algerian travelers and destinations. As a result, travelers face difficulties finding relevant recommendations, sharing experiences, and organizing their trips within a single unified environment.

To address this issue, we propose **TripTale**, a mobile application dedicated to travel exploration and experience sharing. The application combines destination discovery, trip publication, travel planning, social interaction, and premium services within a single platform. By integrating these functionalities, TripTale aims to provide travelers with a more intuitive and engaging experience while promoting local tourism and community participation.

The development of TripTale follows a structured software engineering approach involving requirement analysis, system design, technology selection, implementation, and evaluation. The project also adopts modern technologies such as React Native, Express.js, Prisma ORM, and PostgreSQL to ensure portability, scalability, and maintainability.

This thesis is organized as follows:

- **Chapter 1** presents the study of existing travel solutions, identifies their limitations, and defines the requirements of the proposed system.
- **Chapter 2** focuses on the analysis and design of TripTale, including use cases, system architecture, database design, and user interface considerations.
- **Chapter 3** describes the technologies, frameworks, and development tools used throughout the project.
- **Chapter 4** details the implementation of the application and explains the realization of its main functionalities.

Finally, the thesis concludes with a summary of the achieved work, the main contributions of the project, and several perspectives for future improvements.

1. Chapter 1 Study and Analysis of Existing Solutions

Before developing any software solution, it is essential to understand the problem domain, analyze existing solutions, and identify unmet user needs. In the field of tourism and travel, numerous applications have been developed to assist travelers in discovering destinations, planning trips, and sharing experiences. However, these solutions often focus on specific functionalities and may not fully address the requirements of local travelers.

This chapter presents a study of the travel application ecosystem and examines several existing platforms. Based on this analysis, the limitations of current solutions are identified, leading to the definition of the problem statement and the proposal of TripTale as a comprehensive travel-sharing platform. The chapter concludes by presenting the functional and non-functional requirements that guide the development of the system.

1.1 Introduction to Travel Applications

A travel application is a software product, generally installed on a mobile device, that supports one or several stages of the traveler's journey: inspiration, planning, booking, on-site navigation, and post-trip sharing. The widespread adoption of smartphones has led to a fragmented but rich ecosystem of such applications. Three large families can be distinguished in this ecosystem.

Trip planning applications focus on the organization of an upcoming journey. They allow the user to build itineraries, store flight and hotel reservations, manage budgets, and synchronize these elements across devices. TripIt, Wanderlog, and Sygic Travel are well-known representatives of this category. Their value lies in centralizing information that would otherwise be scattered across emails, calendar entries, and paper documents.

Social travel applications, on the other hand, place the emphasis on sharing and discovering experiences rather than on logistical planning. Polarsteps, Wonderment, and travel-oriented uses of Instagram allow travelers to publish stories that combine photographs, narrative text, maps, and itineraries. The accumulated content becomes a community resource that future travelers consult to obtain inspiration and recommendations.

Finally, destination-sharing and review platforms specialize in helping users decide where to go and what to do once they arrive. TripAdvisor, Google Maps, and Yelp aggregate user reviews of

restaurants, hotels, and attractions; Airbnb adds a transactional layer with peer-to-peer accommodation booking. These applications excel at providing factual information about a destination but are less suited to telling a personal travel story.

Each of these families has its own strengths, but no existing application covers, in a single coherent interface, the full life-cycle that an Algerian traveler actually follows: discovering destinations within Algeria, organizing a trip, sharing a souvenir with photographs and guides, saving inspiring trips of other users for later, and interacting socially through ratings, comments, and follows. This observation motivates the comparative study presented in the next section.

1.2 Comparative Study of Existing Applications

In order to position our work, we have analyzed four widely used applications that, at first sight, could partially cover the needs of an Algerian traveler: TripAdvisor, Polarsteps, Airbnb, and Instagram. For each application we identify its main features, its strengths, and its limitations with respect to our target use case.



Figure 1-1: existing apps

1.2.1 TripAdvisor

TripAdvisor is a destination-review platform that gathers user-generated ratings on hotels, restaurants, and attractions worldwide. Its strengths are the sheer volume of reviews and the maturity of its filtering and ranking features. However, the platform is not designed to let users compose narrative travel stories; published content is essentially a review attached to a venue. Coverage of Algerian destinations is also uneven: large cities such as Algiers and Constantine are well represented, while remote sites of high heritage value (Tassili, Hoggar, Ghardaia valleys) are described mainly by foreign visitors and lack the perspective of local travelers.

1.2.2 Polarsteps

Polarsteps automatically tracks a user's location during a trip and produces a timeline that the user can enrich with photographs and short stories. The application is elegant and well suited to long international journeys, but it is centered on the individual traveler's diary: the social discovery dimension is limited, and there is no concept of structured guides such as where to eat or must-visit places associated with a destination. The Algerian audience of Polarsteps is also small, which limits the relevance of its social feed for our target users.

1.2.3 Airbnb

Airbnb is essentially a marketplace for short-term accommodation, extended in recent years with an Experiences module that lists local activities. Its strength lies in the transactional layer (secure payment, reviews, host verification). The application is not, however, a travel-sharing platform: users cannot publish a free-form trip narrative, and the local offer in Algeria remains modest because the legal framework of short-term rental is still emerging.

1.2.4 Instagram

Instagram is, in practice, the dominant channel through which Algerian travelers share souvenirs of their trips. Its visual editor is excellent and its audience is enormous. However, Instagram is a general-purpose social network: travel content is mixed with unrelated content, destination information is not structured (no category, no budget, no explicit guide), and users cannot easily save trips for later planning. Discoverability relies on hashtags, which are noisy and inconsistent.

1.2.5 Synthesis

Table 1.1 summarizes the comparison along five criteria: trip planning, social sharing, structured destination guides, save-for-later feature, and focus on Algerian destinations. A symbol Yes indicates that the feature is fully supported, Partial that the feature exists but with important limitations, and No that the feature is absent.

Application	Trip planning	Social sharing	Guides	Save for later	Algerian focus
TripAdvisor	No	Partial	Partial	Partial	Partial
Polarsteps	Partial	Yes	No	Yes	No
Airbnb	No	No	Partial	Yes	No
Instagram	No	Yes	No	Partial	No
TripTale	Partial	Yes	Yes	Yes	Yes

Table 1-2: Comparative feature matrix.

The matrix shows that no single existing application combines structured trip publication, social discovery, save-for-later capability, and a focus on Algerian destinations. Each of the four reference applications leaves at least three of these dimensions either unsupported or only partially supported. This gap is precisely what TripTale addresses.

1.3 Problem Statement

The comparative study highlights three concrete problems that an Algerian traveler faces when planning, living, and sharing a trip.

1.3.1 Difficulty of organizing trips

A traveler who plans a journey within Algeria has to combine information from multiple, heterogeneous sources: a search engine for inspiration, social networks for souvenirs of previous travelers, dedicated booking applications for accommodation, and personal note-taking applications to keep track of the itinerary. The result is a fragmented planning experience, in which information is constantly lost or duplicated, and in which it is difficult to obtain a global view of the trip.

1.3.2 Lack of dedicated social trip sharing

Existing social applications, primarily Instagram, are not designed for structured trip sharing. A travel post is essentially a series of photographs and a free-form caption: the budget, the duration, the category of the trip, and the practical guides associated with the destination are not represented as first-class entities. As a consequence, the content produced by Algerian travelers is rich but weakly reusable: a future traveler cannot easily filter posts by category or budget, or save a trip with personal notes for a future journey.

1.3.3 Scattered and non-local travel inspiration

Global platforms underrepresent Algerian destinations, and when they do expose them, the perspective is most often that of an international visitor. Algerian travelers, who are best placed to identify the cultural, gastronomic, and natural specificities of each region, do not have a dedicated platform where they can publish, discover, and discuss trips inside Algeria. This is particularly true for emerging interior destinations such as Taghit, Tikjda, Chrea, Habibas, or the heritage sites of the Sahara, for which inspiring local content is currently scattered.

Table 1.2 makes the contrast explicit: the typical Algerian traveler currently moves from one third-party application to another at every step of the journey, whereas the TripTale model channels the same five-step life-cycle into a single coherent application.

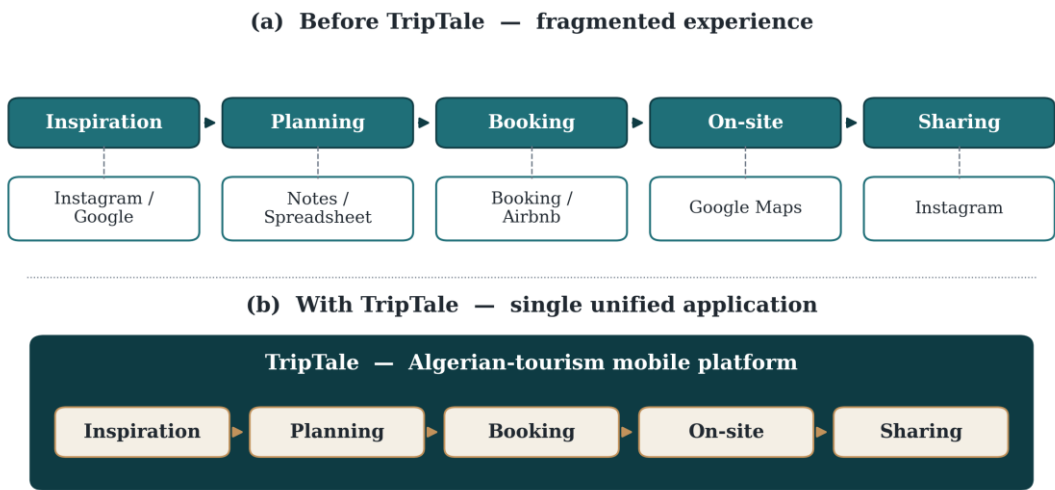


Figure 1-3: User journey before and with TripTale.

1.4 Proposed Solution: TripTale

TripTale is a mobile social application dedicated to Algerian tourism. It combined in a single product, namely discovering destinations, creating travel posts, saving trips for later, and interacting socially with other travelers. The application targets in priority the Algerian audience and proposes a curated catalogue of twenty-seven Algerian destinations across categories such as camping, hiking, beach, city tour, road trip, desert, mountain, cultural, food and wine, festival, and winter sports. Figure 1.3 summarizes the value proposition as a central node connected to the six functional pillars described in the subsections below.

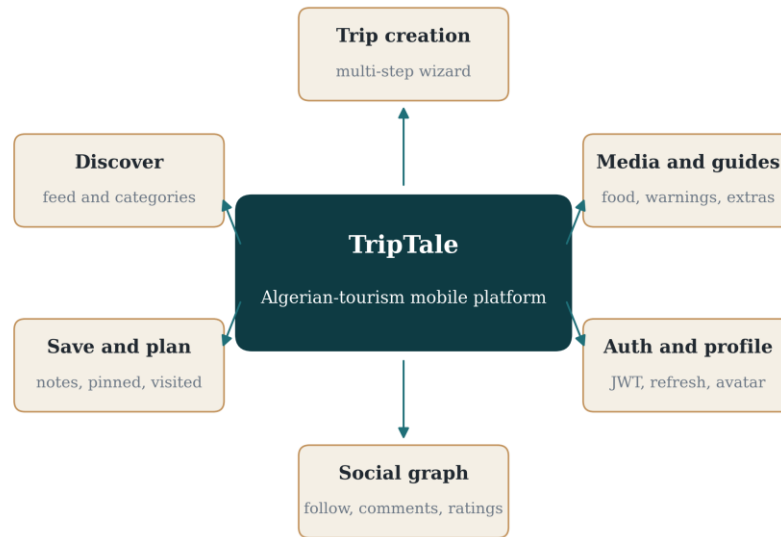


Figure 1-4: TripTale value proposition.

1.4.1 Discover trips

The discover module exposes a feed of public trips, organized by category. A user can browse the catalogue from the home screen, drill down into a category to filter trips, and open the detail page of a trip to consult its full description, media gallery, ratings, comments, and associated guides.

1.4.2 Create travel posts

The trip creation module is structured as a multi-step wizard that walks the user through the construction of a complete trip: title, description, region and country, cover photograph, estimated budget, category, supplementary media, and structured guides covering must-visit places, food, warnings, and extra advice. At each step the user can preview the current state of the trip before publishing it.

1.4.3 Save trips and personal notes

Each trip discovered in the feed can be saved into a personal collection. The saved-trips module supports priority, pinned status, visited flag, visit date, and a personal comment, which together allow the user to use TripTale as a planning notebook in addition to a social feed.

1.4.4 Social interaction

Each user has a public profile with biography, profile picture, and the list of trips they have published. A user can follow another user to be exposed more frequently to their content, rate any trip on a five-point scale, and post comments which are displayed together with the author's username, full name, profile picture, and timestamp.

1.4.5 Planned premium features

A premium tier is foreseen as future work and is not implemented in the current release. The intended scope of this tier includes an enhanced profile badge, advanced analytics on the published trips, unlimited media uploads, and partnerships with travel agencies. These features are discussed in more detail in the perspectives section of the general conclusion.

1.5 Functional Requirements

The functional requirements describe what the system is expected to do, independently of the technologies used to implement it. They are expressed below as a list of capabilities exposed to the end user.

- **FR1 Authentication.** The system must allow a visitor to sign up with a unique username, a unique e-mail address, and a password; to log in with their e-mail and password; to obtain an access token and a refresh token; and to log out.
- **FR2 Profile management.** An authenticated user must be able to consult and edit their profile, including full name, biography, and profile picture, and to consult the public profile of any other user.
- **FR3 Category catalogue.** The system must expose a catalogue of tourism categories that can be listed, selected, and used as a filter for trip discovery.
- **FR4 Trip creation.** An authenticated user must be able to create a trip with a title, a textual description, a location split into region and country, a cover photograph, an estimated budget, a category, additional media items, and structured guides covering must-visit places, food, warnings, and extra advice.

- **FR5 Trip discovery.** Any user, authenticated or not, must be able to browse public trips, filter them by category, and consult the detail page of a trip.
- **FR6 Comments and ratings.** An authenticated user must be able to post a comment on a trip and to assign a rating between one and five. Each comment must be displayed with the author's avatar, full name, username, and creation date.
- **FR7 Save trips.** An authenticated user must be able to save a trip into a personal collection, attach a note to a saved trip, mark it as pinned or visited, record a visit date, and remove it from the collection.
- **FR8 Social graph.** An authenticated user must be able to follow and unfollow another user, and to consult the lists of followers and followings of any user.
- **FR9 Media upload.** An authenticated user must be able to upload images for their trips, with a configurable maximum file size, and the server must serve these images statically.
- **FR10 API documentation.** The backend must expose an interactive Swagger/OpenAPI documentation of all the endpoints described above, so that the mobile client and any third-party consumer can discover and test the API.

1.6 Non-Functional Requirements

The non-functional requirements describe properties of the system rather than additional features. They are structured along the quality characteristics defined by the ISO/IEC 25010 software-product quality model [1] and they constrain the architectural and technological decisions discussed in chapters 2 and 3.

- **NFR1 Security.** User passwords must never be stored in clear text; they are persisted as bcrypt hashes. Authenticated endpoints must be protected by JSON Web Tokens and a refresh-token rotation mechanism. Standard HTTP-level hardening (Helmet, CORS whitelist, body size limit) must be in place.
- **NFR2 Performance.** The mobile client must remain responsive on a mid-range smartphone over a 4G connection. The backend must use HTTP compression and indexed database queries so that the most frequent endpoints (trip list, trip detail, category list) answer in less than 300 milliseconds on the development environment.

- **NFR3 Usability.** The application must follow the mobile-first principle: every interaction must be reachable with one hand on a vertical screen. The visual identity must be coherent (dark teal and beige palette, minimalist black icons). The design is informed by Nielsen's ten usability heuristics [2].
- **NFR4 Scalability.** The chosen persistence layer must support multi-device concurrent access. PostgreSQL is preferred over file-based alternatives because all users of the same instance read and write the same shared database in real time.
- **NFR5 Portability.** The mobile client must run on both Android and iOS without code duplication. The backend must be reproducible on any host that supports Docker Engine, in order to make the deployment as portable as possible.
- **NFR6 Reliability and data integrity.** Foreign-key constraints and unique constraints must be enforced by the database; database migrations must be versioned and repayable on a fresh instance via a deterministic seed script.
- **NFR7 Maintainability.** The backend must follow a modular organization, with one folder per domain module. Static typing must be enforced via TypeScript on the backend.
- **NFR8 Openness.** The HTTP API must be self-described through an OpenAPI specification, so that the application can later be consumed by additional clients (web, third-party services).

1.7 Conclusion

In this chapter, we have presented the family of travel applications and shown, through a structured comparison of four widely used products, that no existing solution fully covers the needs of an Algerian traveler who wants to discover, share, and save travel experiences inside Algeria. From this gap, we have derived a clear problem statement and we have introduced TripTale, our proposed solution, together with its functional and non-functional requirements. The next chapter formalizes these requirements through use-case diagrams, the application flow, the system architecture, and the database design, before chapter 3 details the technologies and tools that have been chosen to implement the system.

2. Chapter 2 — System Analysis and Design (TripTale)

After identifying the project requirements and defining the proposed solution, the next step consists of transforming these requirements into a structured system design. System analysis and design play a crucial role in ensuring that the final application is coherent, scalable, and aligned with user expectations.

This chapter presents the analysis and design of TripTale. It introduces the system architecture, use case diagrams, application workflows, database design, and user interface considerations. These design elements provide a clear blueprint for the implementation phase and establish the foundation upon which the application is built.

System Name: TripTale

Type: Mobile Tourism & Social Discovery Application

Frontend: React Native + Expo (file-based routing)

Backend: Node.js + Express + TypeScript

Database: SQLite (development) → PostgreSQL (production / shared)

ORM: Prisma

Authentication: JWT Access Token + Refresh Token (rotation/revocation)

Primary Features: Authentication, browse trips by categories, create/publish trips, save trips with planning notes, comments, ratings, media upload, user profile, follow system (backend), trip guides (backend)

2.1 Development Methodology

2.1.1 Overview: Why Agile for TripTale

TripTale is a feature-rich mobile application that combines (1) content discovery (destinations, trips, categories), (2) social interaction (ratings, comments, author identity), and (3) personal planning (saved trips with notes, priority/pin concepts). These features evolve rapidly during development because user experience and data relationships become clearer as screens and APIs are tested on real devices.

To handle iterative feature building and continuous feedback, **Agile methodology** is a natural fit. Agile supports:

- incremental delivery of working features (screens + endpoints)
- frequent integration between mobile app and backend API
- rapid adaptation when requirements change (e.g., switching DB provider, adding Premium later)

In TripTale, Agile allows the team to release a usable minimum version (authentication + browsing + trip details) early and then expand into advanced features (media, saved planning, followers, premium, analytics).

2.1.2 Step by Step Development Approach

TripTale follows a Step by Step approach where each Step improves:

- **UI flows** (discover → details → actions such as rate/save/comment)
- **API capabilities** (filters, pagination, upload endpoints, role-based expansions)
- **data model** (adding relations, indices, and constraints to preserve integrity)

A practical loop for TripTale looks like:

1. Define/adjust a requirement (e.g., “comments show author avatar + date”).
2. Confirm backend has required data (join comment with user profile).
3. Implement frontend UI rendering.
4. Test on device(s), verify real-time shared DB behavior.
5. Refine (fallback avatar logic, date formatting, error handling).

2.2 Use Case Diagram

2.2.1 Actors

TripTale supports two primary actors at the system level.

1. User (Traveler)

A standard registered user who can browse trips, interact socially, and manage saved plans.

2. Premium User (Premium Traveler)

A paying user with additional benefits:

- Premium badge displayed on profile and alongside user-generated content
- Ad-free experience
- Post analytics (views, saves, ratings interactions, comment counts, reach)
- Unlimited media uploads per trip
- More exposure (ranking boost of published trips in discovery)

2.2.4 Use Case Diagram

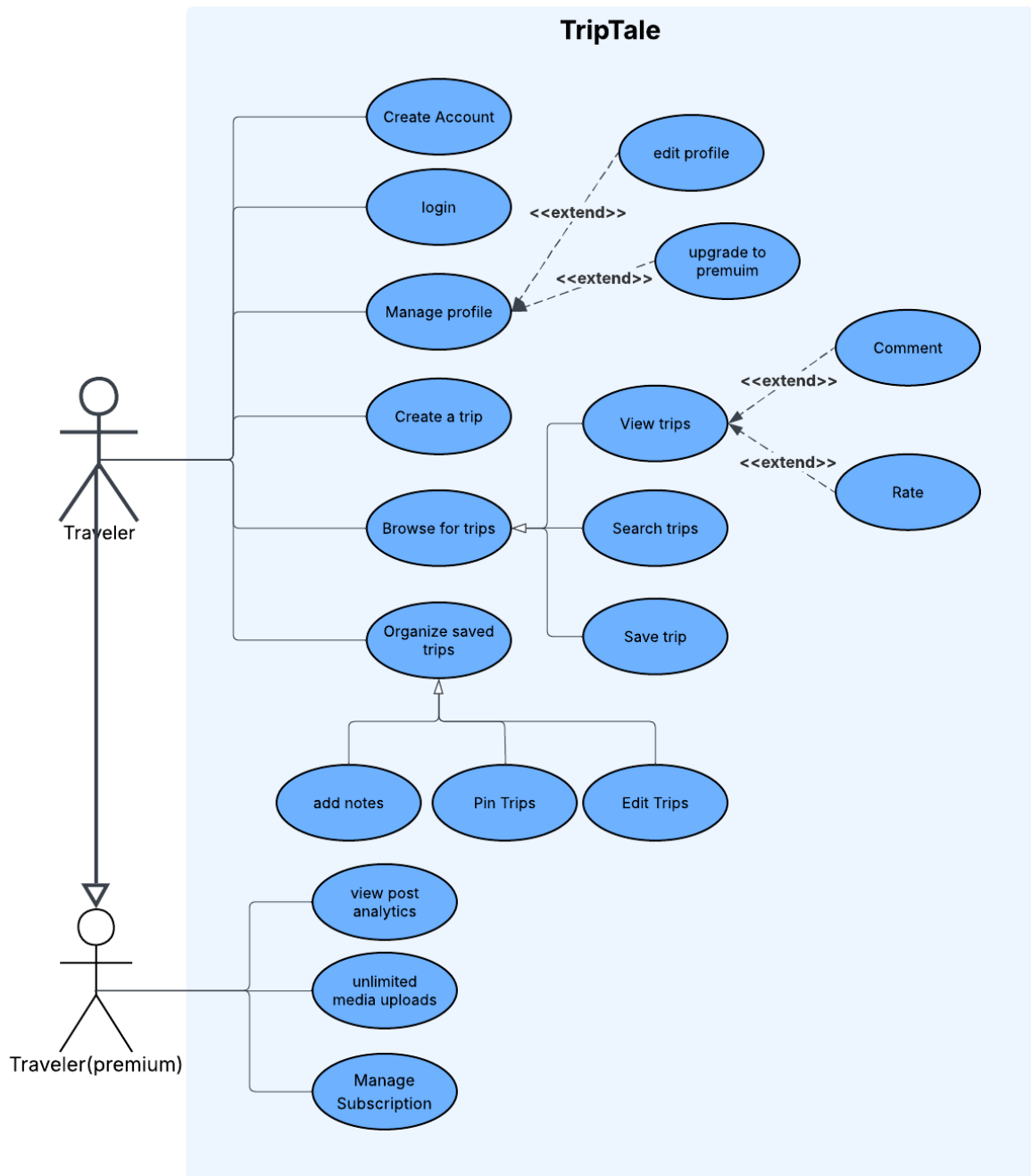


Figure 2-1: use-case diagram

2.3 Application Flowchart

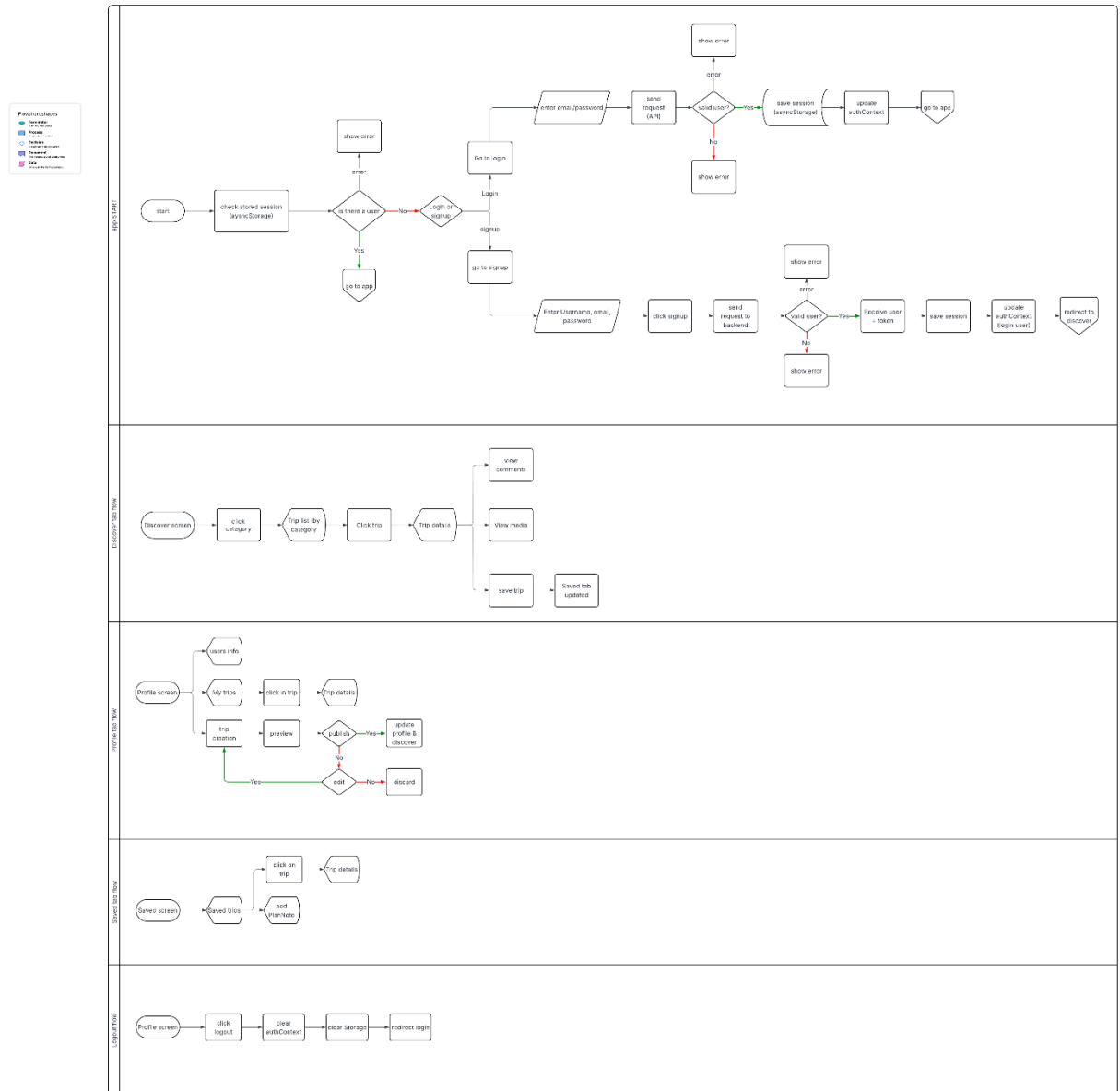


Figure 2-2: flowchart diagram

This section explains the functional flow of TripTale from app launch to day-to-day use. The flowchart provides the high-level logic; and expands it into implementation-level steps and design rationale.

2.3.1 Authentication Flow (App Start → Valid Session → App)

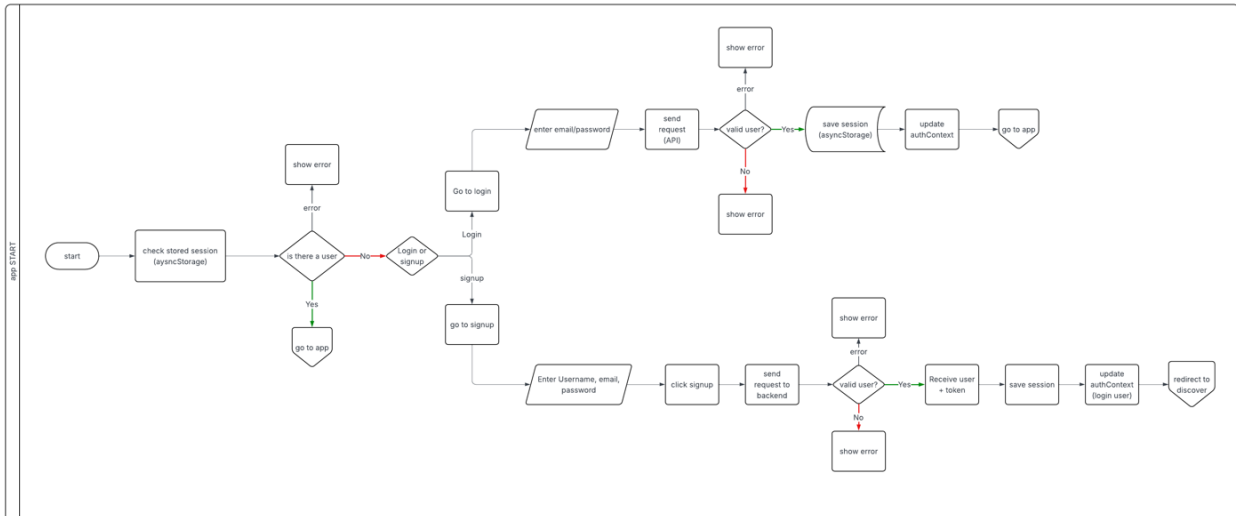


Figure 2-3: app start (flowchart)

Step 1 — App Launch and Session Check

1. App executes an initialization routine.
2. It checks local storage for an existing session (access token + refresh token).
3. Decision: Is there a stored user session?
 - If Yes: proceed to the app (may still validate token).
 - If No: route to login/signup.

Step 2 — Login Flow

1. User enters email/password.
2. App sends POST /api/auth/login.
3. Backend validates credentials and returns { user, accessToken, refreshToken } on success.
4. App stores tokens in AsyncStorage and updates AuthContext.
5. User is redirected to the main app (tabs).

Error handling: show clear messages; handle network errors; avoid infinite loading states.

Step 3 — Signup Flow

1. User enters username, email, password.
2. App sends `POST /api/auth/register`.
3. Backend creates the account and returns tokens.
4. App stores tokens and routes user into the app (discover).

Step 4 — Token Refresh (Security + UX)

1. When access token expires, app triggers `POST /api/auth/refresh`.
2. Backend validates refresh token and performs rotation (revokes old token, issues new pair).
3. App updates stored tokens and retries failed request.

Why rotation matters: used tokens become invalid, reducing theft impact.

2.3.2 Discover Flow (Categories → Trips → Trip Details)

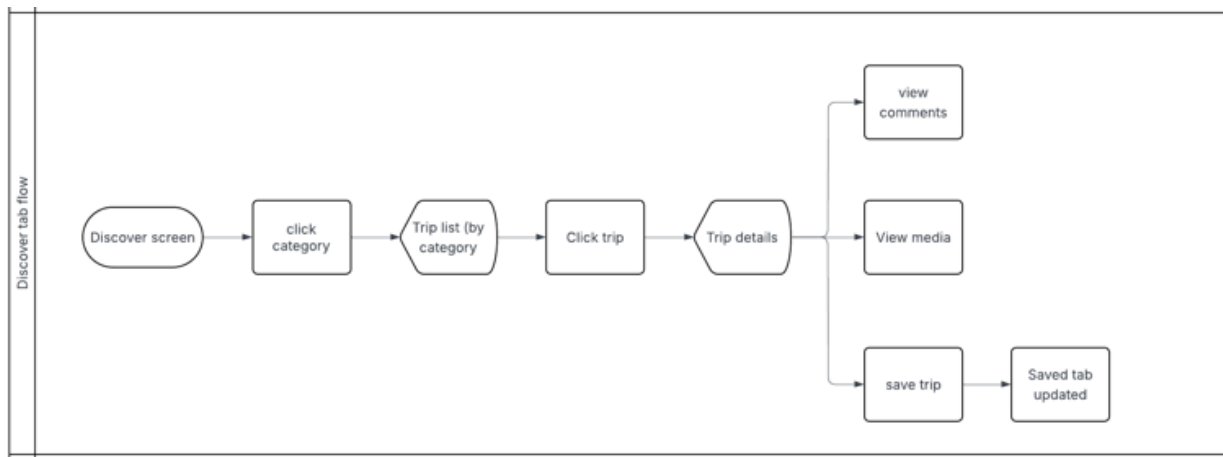


Figure 2-4: discover tab (flowchart)

Step 1 — Discover Screen Opens

- Category cards (minimalist icons)
- Trips list or category-first navigation

Step 2 — Select Category

1. User taps a category.

2. App requests trips filtered by category
3. Trips list is displayed.

Step 3 — Browse Trips List

- Scroll trip cards (title, location, cover photo, rating summary).
- Backend supports sorting/filtering (recent/popular, budget range, min rating, q search).

Step 4 — Open Trip Details

1. User taps a trip.
2. App loads detail data and associated lists (media/comments/ratings summary).
3. User can view media, comments, rate, and save the trip.

2.3.4 Profile Flow (My Trips → Create → Preview → Publish)

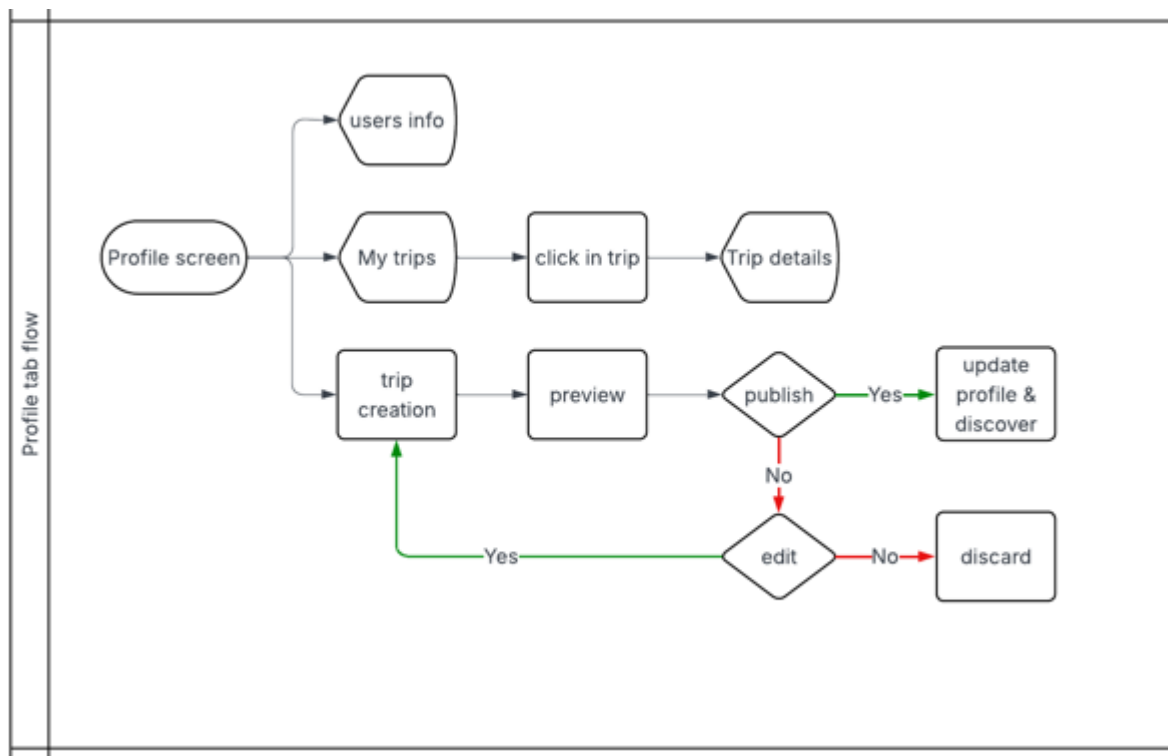


Figure 2-5: profile tab (flowchart)

Step 1 — Profile Screen

- User info (avatar, bio)
- My trips list
- Create trip entry point
- Logout action

Step 2 — Trip Creation

- title, description, location/region/country
- category, budget, cover photo

Step 3 — Preview

Decision: Publish?

- If Yes: POST /api/trips; on success update profile + discover.
- If No: edit again or discard.

2.3.5 Saved Trips Flow (Planning Tab)

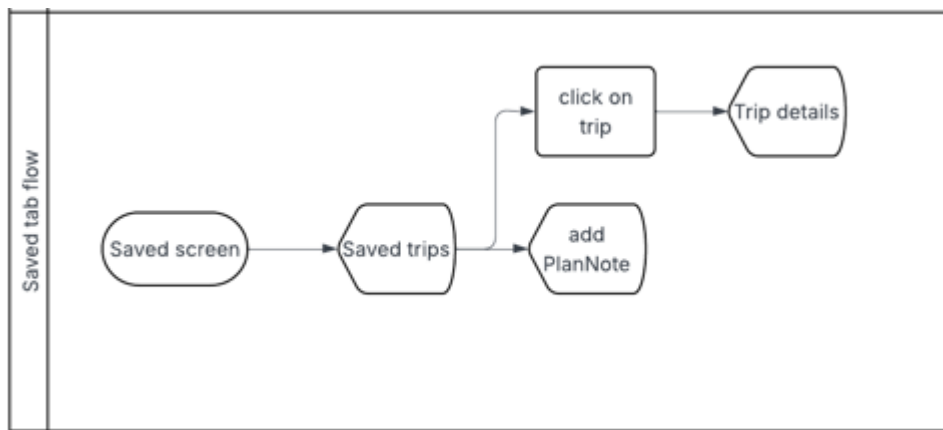


Figure 2-6: saved tab (flowchart)

1. Open Saved screen.
2. Load saved trips: GET /api/saved-trips.
3. Update notes/priority/pin: PATCH /api/saved-trips/:id.
4. Remove saved: DELETE /api/saved-trips/:id.

2.3.6 Logout Flow

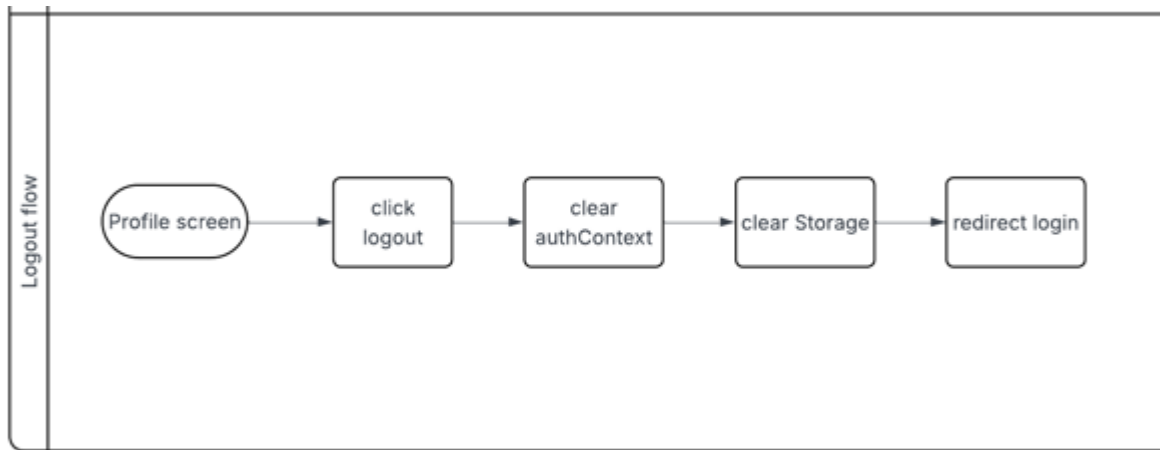


Figure 2-7: logout (flowchart)

1. User clicks logout.
2. Clear auth context and AsyncStorage session.
3. Optionally call POST /api/auth/logout to revoke refresh token.
4. Redirect to login screen.

2.4 System Architecture

2.4.1 High-Level Architecture

- **Client (Mobile App):** React Native + Expo
- **Server (REST API):** Express + TypeScript
- **Database:** SQLite (dev) → PostgreSQL (prod/shared)
- **ORM:** Prisma

2.4.2 Frontend Architecture

- File-based routing organizes screens by paths.
- Tab navigation separates: discover, saved, profile.
- AuthContext holds login state and current user.

2.4.3 Backend Architecture

The backend is modular: auth, users, categories, trips, comments, ratings, saved-trips, media; plus social graph and trip guides.

- Auth middleware: required auth and optional auth
- Validation middleware: request validation
- Error handler: centralized response formatting
- Upload middleware: multipart media uploads with limits

2.4.4 API Communication

- REST endpoints with JSON requests/responses.
- Authorization: Bearer <accessToken> for authenticated routes.
- multipart/form-data for media uploads.

2.4.5 Database Choice: SQLite → PostgreSQL

SQLite (Development) enables quick local iteration with zero configuration.

PostgreSQL (Production / Shared) supports multi-device shared data and is suitable for deployment.

Prisma reduces migration friction by keeping models stable while changing provider and connection string.

2.5 Database Design

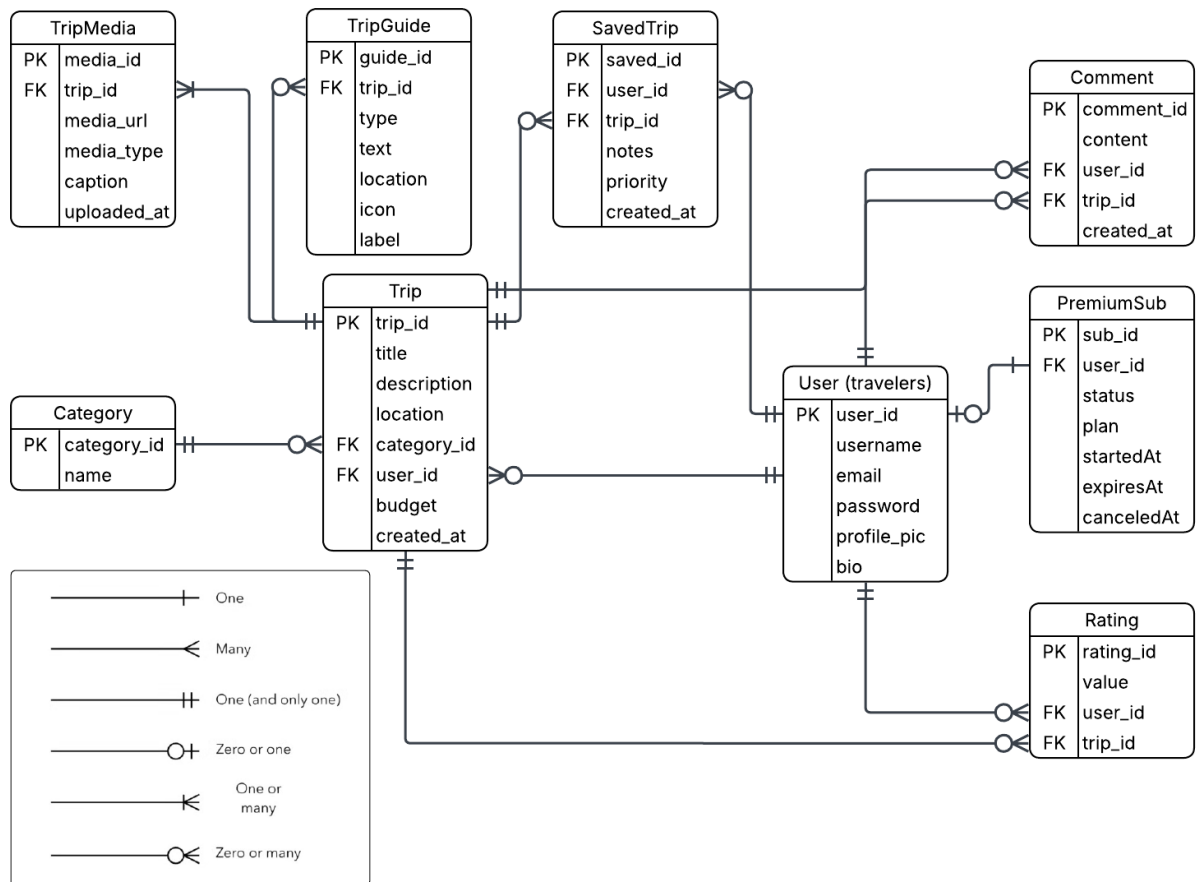


Figure 2-8: ER diagram

2.5.1 ER Design Goals

- Ownership of trips by users
- Categorized discover browsing
- Social interactions (comments, ratings)
- Personal planning (saved trips + notes/priority)
- Media attachments for rich posts
- Premium expansion (PremiumSubscription table)

2.5.2 Relationships Summary (Cardinality)

- User (1) → (Many) Trip
- Category (1) → (Many) Trip
- Trip (1) → (Many) TripMedia
- Trip (1) → (Many) Comment
- Trip (1) → (Many) Rating
- User (1) → (Many) Comment
- User (1) → (Many) Rating
- User (Many) ↔ (Many) Trip via SavedTrip
- User (0..1) → PremiumSubscription

2.5.3 Normalization and Integrity

- Separation: media, comments, ratings, saved planning are separate tables.
- Uniqueness constraints enforce correct behavior (one rating per trip per user, one saved record per trip per user).
- Cascade deletes prevent orphan rows when a user or trip is deleted.

2.5.4 Indexing (Performance Considerations)

- Trip.categoryId for category browsing
- Trip.userId for “My trips”
- Comment.tripId for listing comments
- Rating.tripId for aggregates
- SavedTrip.userId for saved planning list
- Follow.followerId / followingId for profile social stats

2.6 UI/UX Design

2.6.1 Navigation Structure and Key Screens

TripTale uses a mobile-first navigation structure where main flows are separated into tabs.

A) **Discover Tab:** categories → trips list → trip details

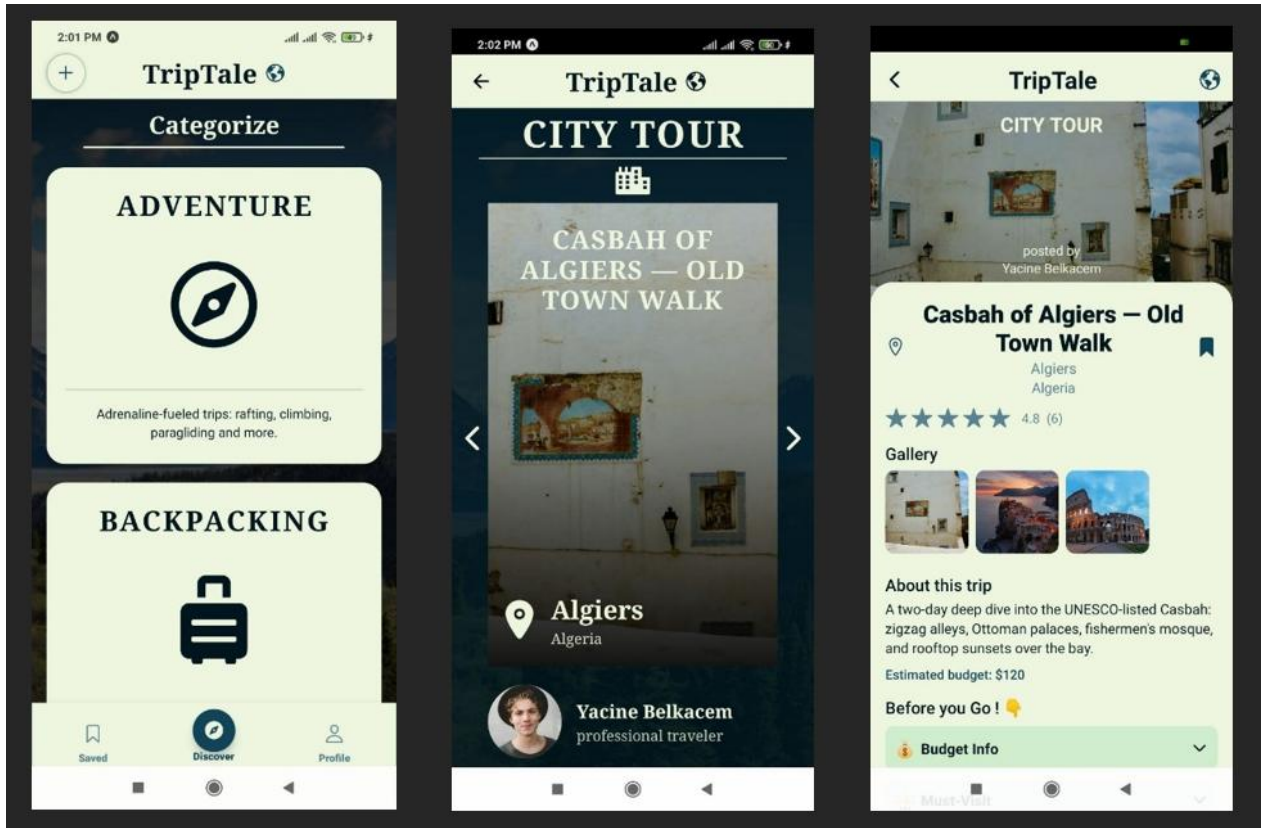


Figure 2-9: discover tab UI/UX

B) Saved Tab: saved trips + notes/priority/pin organization

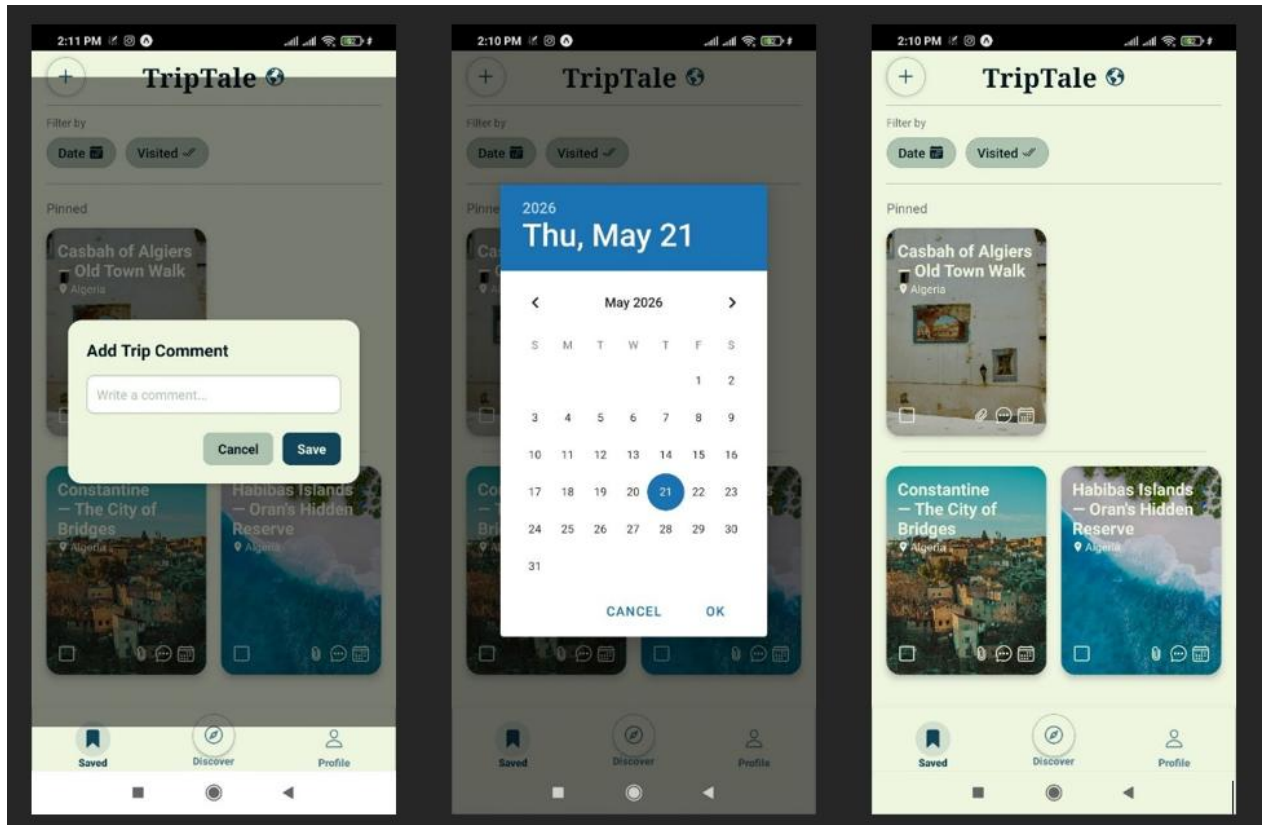


Figure 2-10: saved tab UI/UX

C) **Profile Tab:** user profile, my trips, create/publish trips, logout

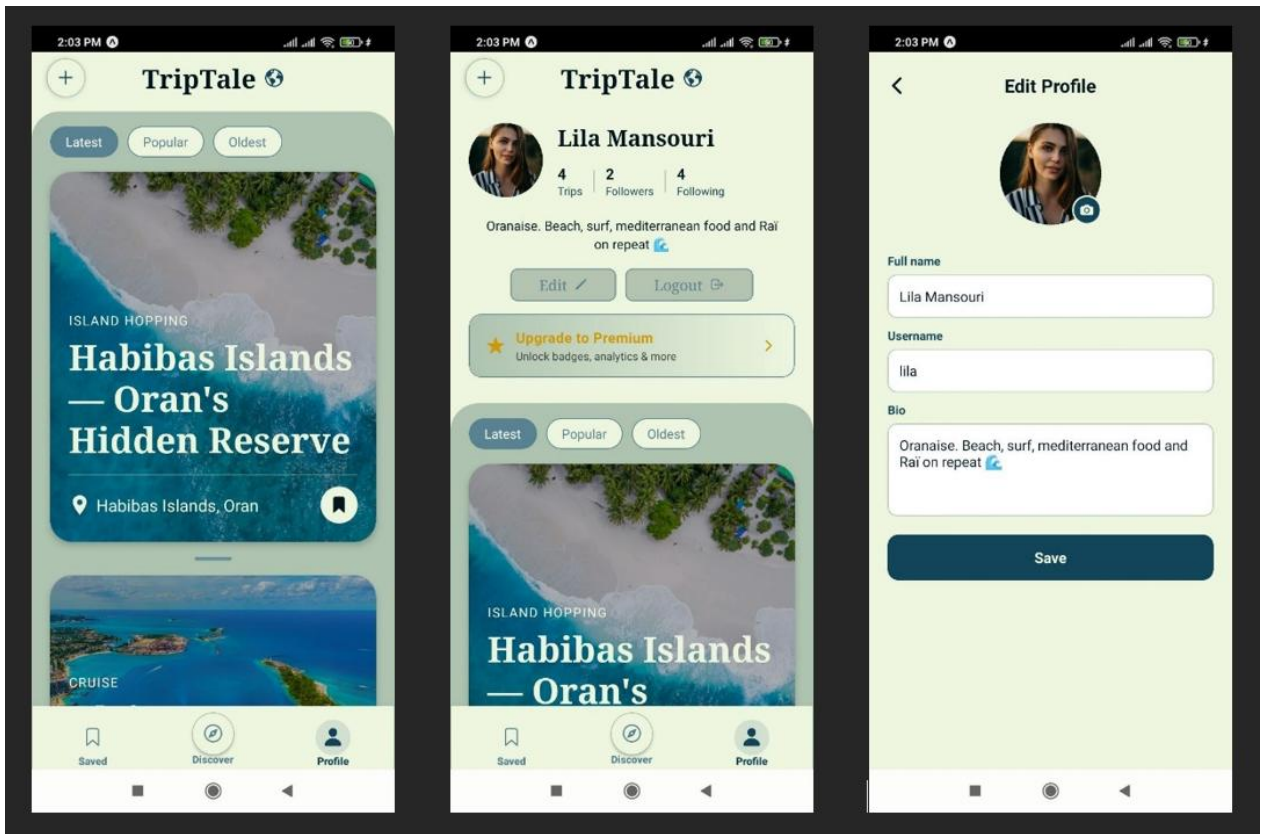


Figure 2-11: profile tab UI/UX

Authentication screens exist before the tabs: login and signup.

2.6.2 Mobile-First Design Principles

- Large touch targets and card-based layouts
- Vertical scrolling lists for discover browsing
- Clear icons for categories
- Fast path to actions: save, comment, rate

Mobile constraints addressed: variable network quality, limited screen space, performance.

2.6.3 Visual Design and Color/Brand Choices

- Minimalist iconography (black icons) for categories
- Consistent card layouts and readable typography

- Content-first design (trips and media are the focus)

2.6.4 UX Decisions Explained

A) Session Persistence

refresh tokens reduces friction and maintains continuity across launches.

B) Discover → Details → Action Pattern

Actions are placed close to information, reducing navigation complexity.

C) Planning via Saved Trips

Saved trips transform browsing into real planning with notes and priorities.

D) Premium UX

Premium adds value (badge, analytics, ad-free) while exposure boosting remains controlled and fair.

2.7 Conclusion

TripTale is a full-stack tourism and social discovery system built with a modern architecture: a mobile-first React Native + Expo client, a modular Express + TypeScript backend, and Prisma-managed relational databases (SQLite in development and PostgreSQL in production/shared).

The system supports secure authentication, scalable content features (trips, media, comments, ratings), planning tools (saved trips with notes), social growth (followers/following), and planned Premium extensions including badge, ad-free UI, analytics, unlimited uploads, and more exposure.

This chapter provided Agile methodology justification, completed use cases, detailed flow explanation, system architecture, database design (including planned PremiumSubscription), and UI/UX design decisions.

3. Chapter 3 Technologies and Development Tools

After having defined, in chapter 1, the functional and non-functional requirements of TripTale, and after having designed the architecture, the use-cases, and the data model in chapter 2, this chapter focuses on the concrete technologies that have been selected to implement the application. The chapter is organised in five thematic sections covering the frontend stack, the backend stack, the database stack, the development tools, and finally the justification of the global technological choices with respect to the non-functional requirements. Figure 3.1 projects the chapter onto the three-tier client-application-data architecture, and Figure 3.2 groups the official logos of every technology covered in this chapter into four functional tiers.

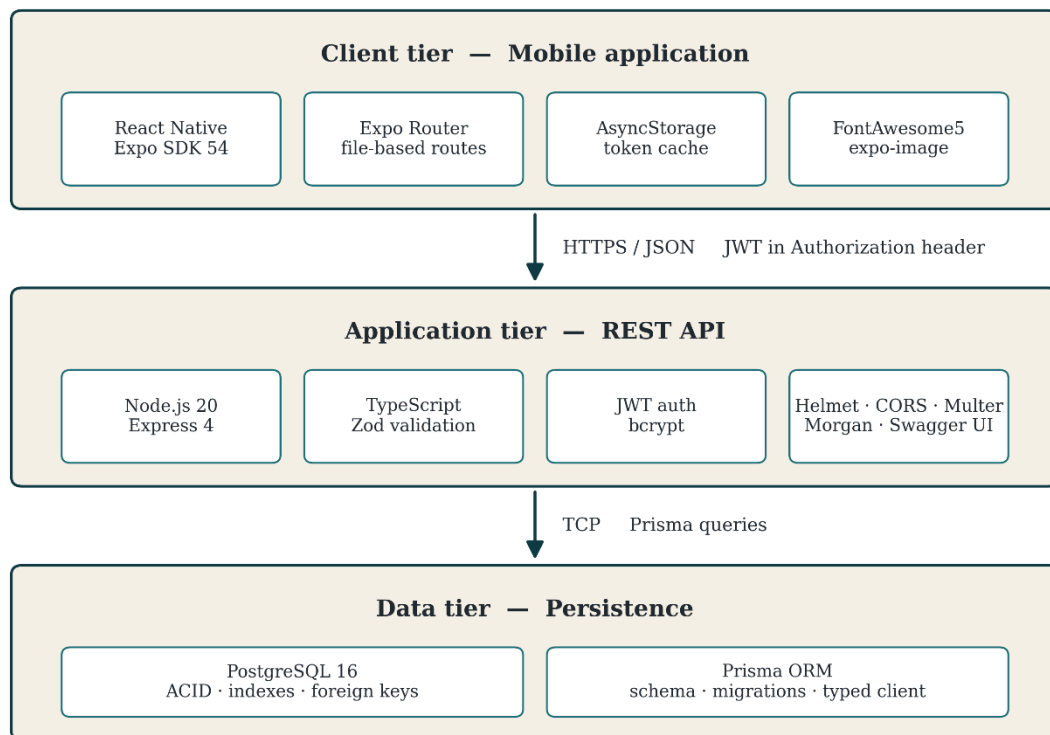


Figure 3-1: Three-tier architecture of TripTale.

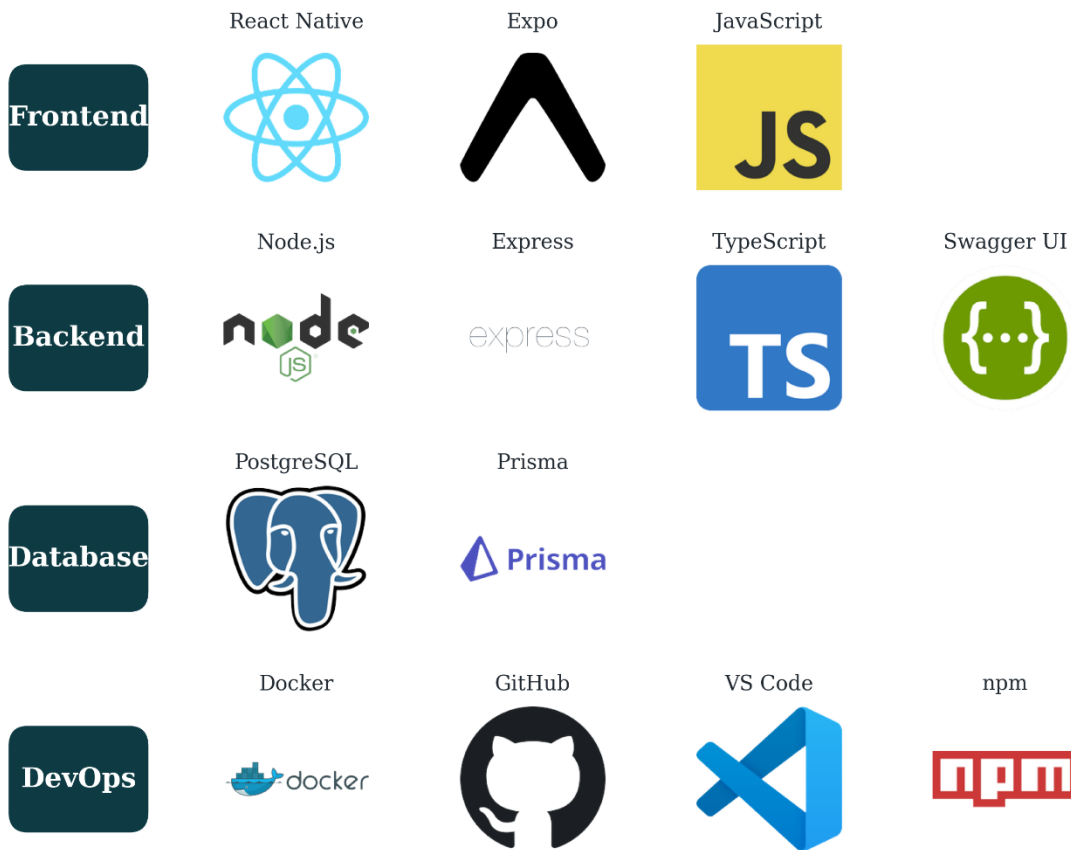


Figure 3-2: TripTale technology stack.

3.1 Frontend Technologies

The frontend of TripTale is a mobile application built on the React Native ecosystem. The choice of a single cross-platform code base is directly motivated by the portability requirement (NFR5) and by the limited development resources of a graduation project. The frontend is deliberately organised around a small set of well-known technologies rather than an exhaustive list of optional libraries, in order to make the implementation predictable and reviewable.

3.1.1 React Native

React Native is an open-source framework, initially released by Meta in 2015, that allows developers to build native mobile applications using JavaScript and the React component model [4]. Unlike hybrid frameworks that render through an embedded web view, React Native translates declarative components into the platform's native user-interface primitives, so that the final application behaves and performs like a fully native product. In TripTale, every screen, navigation

element, and form widget is expressed as a React Native component, which makes the visual identity of the application uniform between Android and iOS.

3.1.2 Expo and Expo Router

Expo is a managed framework built on top of React Native that takes care of the low-level configuration of the native projects and exposes a unified JavaScript API for the most frequent native capabilities: image picker, fonts, splash screen, haptics, web browser, status bar, and many more [5]. The TripTale frontend is built on Expo SDK 54 and relies on the Expo managed workflow, which removes the need for a local Android or Xcode toolchain during development and considerably accelerates iteration. Routing is provided by Expo Router, a file-system based router inspired by Next.js: each file inside the app directory automatically becomes a screen, which makes the navigation tree immediately readable from the source tree.

3.1.3 JSX and the React component model

JSX is a syntactic extension of JavaScript that allows user-interface trees to be written with a markup-oriented syntax inside JavaScript modules. Combined with React's unidirectional data flow and component model [3], it produces a code base in which the visual structure of each screen is co-located with its behaviour. In TripTale, JSX is used exclusively in .jsx files for the user-facing layer, while shared logic (API client, authentication context) lives in plain JavaScript modules inside the src directory.

3.1.4 Navigation and supporting libraries

The bottom-tab navigation of the home screen is implemented with `@react-navigation/bottom-tabs`, while modal and stack navigation is provided by Expo Router on top of `@react-navigation/native`. Token persistence between application restarts relies on `@react-native-async-storage/async-storage`. Iconography uses the `@expo/vector-icons` package, which exposes `FontAwesome5` and is responsible for the minimalist black icons displayed on the category cards. Media handling is delegated to `expo-image`, `expo-image-picker`, and the standard React Native image component, depending on whether the source is local, remote, or selected interactively by the user.

3.2 Backend Technologies

The backend of TripTale is a REST API [4] written in TypeScript, executed by the Node.js runtime, and built on the Express web framework [5]. This section presents each layer of the backend stack and the cross-cutting middleware that protect and document the API. Internally, the source tree is split between a generic `src/` folder and ten dedicated domain modules, as shown in Figure 3.3.

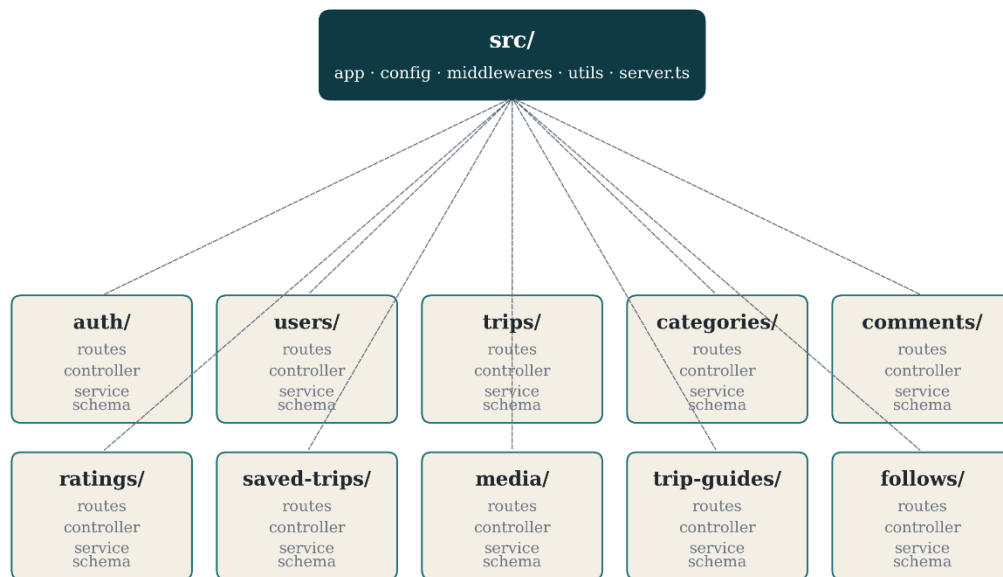


Figure 3-3: Backend module organization.

3.2.1 Node.js

Node.js is a JavaScript runtime built on Chromium's V8 engine [6]. It is designed around a single-threaded event loop and non-blocking input/output, which makes it particularly well adapted to web back ends in which most of the work consists in waiting for input/output operations (database queries, file uploads, HTTP requests). TripTale targets the Node.js 20 LTS line, which offers long-term support and a complete compatibility with modern ECMAScript features.

3.2.2 Express

Express is a minimal and unopinionated web framework for Node.js [7]. It provides routing, middleware composition, and a small surface of request and response helpers, while leaving the

choice of every other concern (validation, ORM, authentication) to the application. This minimalism makes Express a natural fit for graduation projects, because the developer keeps full control over the architecture and the dependencies are easy to audit.

3.2.3 REST API architecture

The backend of TripTale exposes a Representational State Transfer (REST) interface, in the sense originally formalised by Fielding [8]. Every resource of the domain (user, trip, category, comment, rating, saved trip, follow, media, trip guide) is mapped to a set of endpoints organised under a stable URL hierarchy. The verbs of HTTP carry the semantics of the operation: GET to read, POST to create, PUT or PATCH to update, and DELETE to remove. Responses are serialised as JSON, and errors are returned with appropriate status codes accompanied by a structured error body. Each domain is encapsulated in its own module containing the route file, the controller, the service, and the Zod schema that validates the request payloads.

3.2.4 Authentication and security middleware

Authentication relies on JSON Web Tokens [9], issued by the `jsonwebtoken` library. A short-lived access token is delivered together with a long-lived refresh token, the latter being persisted in the database and rotated on every refresh. Passwords are hashed with `bcryptjs`. Cross-cutting protection is provided by `Helmet`, which sets a set of conservative HTTP security headers; `CORS`, which restricts the origins allowed to call the API; `compression`, which compresses HTTP responses with `gzip`; and a body-size limit, which bounds the maximum size of incoming JSON payloads to five megabytes.

Figure 3.4 traces the corresponding sequence of HTTP exchanges between the mobile client, the Express API, and PostgreSQL during a typical login, an authenticated request, and a refresh.

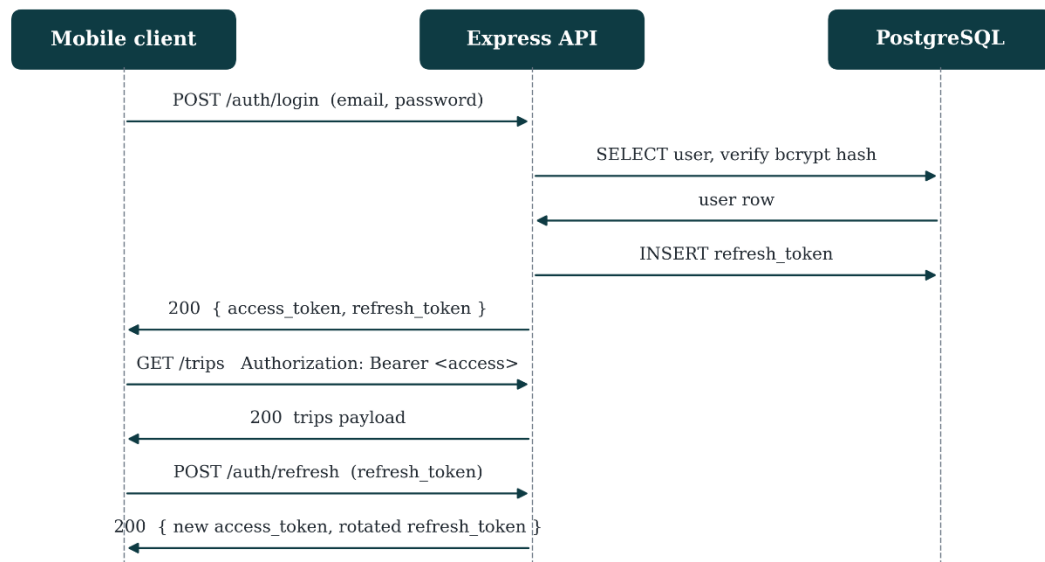


Figure 3-4: JWT authentication flow.

3.2.5 Validation, uploads, logging, and documentation

Every incoming payload is validated with Zod, a schema-first validation library that produces precise error messages and type-safe request handlers. File uploads are handled by Multer and persisted on disk under the configurable uploads directory, which Express then serves statically. Request logging in development uses Morgan with the compact dev formatter. Finally, the entire surface of the API is documented inline using swagger-jsdoc annotations following the OpenAPI specification and exposed as an interactive Swagger UI at /api/docs, which satisfies the openness requirement (NFR8) and makes the API immediately testable from a browser without writing any client code.

3.3 Database Technologies

The data tier of TripTale combines a relational database engine, PostgreSQL, with a type-safe object-relational mapper, Prisma. This combination provides a strong foundation for the integrity, scalability, and maintainability requirements identified in chapter 1.

3.3.1 PostgreSQL

PostgreSQL [10] is a mature open-source relational database engine that supports ACID transactions, foreign keys, indexes, JSON columns, full-text search, and a rich extension ecosystem. TripTale targets PostgreSQL 16, which is the most recent stable major release at the time of writing. The choice of PostgreSQL is directly motivated by the scalability requirement (NFR4): multiple mobile clients connected to the same instance read and write a shared dataset concurrently, and the database engine guarantees that comments, ratings, and saved trips remain consistent across devices. A lightweight SQLite fallback remains documented in the schema for offline experimentation, but the production deployment exclusively targets PostgreSQL.

3.3.2 Prisma

Prisma is an open-source toolkit that combines an object-relational mapper, a declarative schema language, and a database migration tool [11]. The schema file `prisma/schema.prisma` describes every entity of TripTale — User, RefreshToken, Follow, Category, Trip, TripGuide, Comment, Rating, SavedTrip, TripMedia — together with their fields, constraints, and relations. From this declarative description, the Prisma CLI generates a fully typed client that the TypeScript code uses to query and mutate the database. Migrations are produced by `prisma migrate dev` during development and replayed in production with `prisma migrate deploy`, which guarantees that any developer or any deployment environment converges to the same schema state.

3.3.3 Seeding and demo data

A dedicated seed script populates the database with the seventeen tourism categories, the twenty-seven Algerian destinations, the eight demonstration users, and an initial set of comments and ratings. The seed is replayed automatically when the Docker Compose stack is started for the first time, so that a freshly cloned repository can be brought to a fully usable state with a single command. This directly supports the reproducibility goal of NFR5.

3.4 Development Tools

The day-to-day development of TripTale combines a code editor, a version-control system, a containerisation runtime, and a small set of complementary tools that together form a reproducible and modern toolchain.

3.4.1 Visual Studio Code

Visual Studio Code is the editor used to write and refactor the TypeScript backend and the JavaScript frontend. Its integration with the TypeScript language server provides type checking, automatic imports, and quick fixes directly in the editor, which considerably shortens the feedback loop. Useful auxiliary extensions for the project include the Prisma plug-in, which formats and validates the schema file, and the ESLint plug-in, which enforces coding conventions while files are edited.

3.4.2 Git and GitHub

Source-code revisions are managed with Git and pushed to a GitHub repository. Each significant evolution is recorded as a commit with a descriptive message, which produces a linear history that can be inspected to understand the construction of the application. GitHub also serves as the canonical source for the supervisor and the jury to consult the implementation.

3.4.3 Docker and Docker Compose

The backend and the database are packaged as a Docker Compose stack [12]. A single declarative file, `docker-compose.yml`, describes two services: a PostgreSQL 16 container that persists its data on a named volume, and a Node.js container that installs the dependencies, deploys the Prisma migrations, executes the seed script, and starts the Express server. The end user only has to run `docker compose up -d` to obtain a fully functional backend, which is the most concrete expression of the reproducibility and portability requirements. Figure 3.5 details the structure of this two-service stack and its persistent volumes.

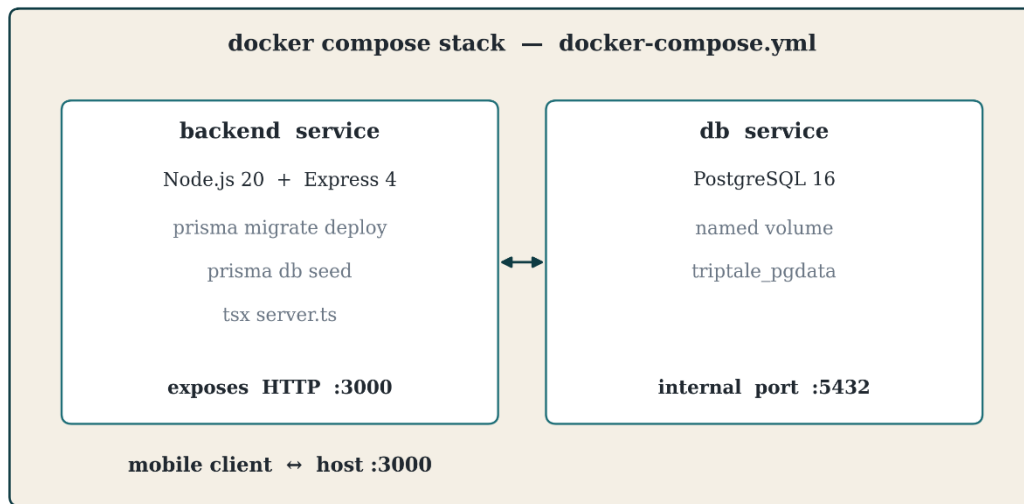


Figure 3-5: Docker Compose stack.

3.4.4 npm, ESLint, and the TypeScript compiler

Dependencies are declared and installed with npm. The backend uses the TypeScript compiler in two complementary modes: `tsc --noEmit` for type checking and `tsx` for direct execution of `.ts` files during development, which removes the need for a separate compilation step. Coding conventions are enforced by ESLint, with a rule set extended from the recommended TypeScript and Expo configurations.

3.4.5 API exploration and mobile preview

The Swagger UI exposed by the backend at `/api/docs` is used as the primary tool to explore and test the REST API during development. On the mobile side, the Expo Go application is used to run the frontend on a physical device or on an emulator simply by scanning a QR code produced by the Expo development server. The combination of these two tools makes the iteration cycle extremely short.

3.5 Justification of Technology Choices

The technologies described in the previous sections were not selected independently from one another: they form a coherent stack that answers, jointly, the non-functional requirements stated in chapter 1, and they apply well-established enterprise-application patterns to a mobile-first context. This section makes the alignment explicit along five axes: cross-platform development, rapid iteration, scalability, robustness, and community support.

3.5.1 Cross-platform development

React Native combined with Expo provides a single JavaScript code base that compiles to native Android and iOS binaries, which directly satisfies NFR5. Compared to two parallel native code bases (one in Kotlin for Android, one in Swift for iOS), this approach divides the implementation effort approximately by two and removes the risk of divergent behaviour between platforms.

3.5.2 Rapid iteration

Several elements of the stack are explicitly designed to shorten the feedback loop. The Expo development server hot-reloads the mobile client on every file save; tsx executes TypeScript files without an intermediate compilation step; Prisma generates a typed client on schema change in a few seconds; and Swagger UI exposes the backend as a clickable interface that does not require a third-party tool such as Postman. The cumulative effect is a development experience in which the time between writing a line of code and observing its effect on the running application is consistently below five seconds.

3.5.3 Scalability and robustness

PostgreSQL is a battle-tested engine that scales from a single developer's machine to multi-tenant production clusters without changing the application code. Prisma adds a layer of static typing and explicit migrations that prevents an entire class of runtime errors that would otherwise occur on a schema change. JWT-based authentication is stateless, which means that a future horizontal scaling of the backend (multiple Express instances behind a load balancer) does not require any session-affinity mechanism. Containerisation with Docker Compose finally ensures that the deployment environment is identical to the development environment, which eliminates the most frequent source of production incidents.

3.5.4 Community and ecosystem

Each technology of the stack is associated with an active open-source community: React Native and Expo are maintained by Meta and Expo Inc. respectively and supported by a large ecosystem of third-party libraries; Node.js and Express are among the most widely used web technologies in the industry; PostgreSQL and Prisma both publish public roadmaps and answer questions on dedicated forums; Docker has become the de facto standard for application packaging. This maturity provides reliable documentation, a steady stream of security fixes, and a deep pool of recruitable expertise.

3.5.5 Open-source licensing and cost

Every component of the stack is distributed under a permissive open-source licence (MIT for React Native, Express, Prisma, jsonwebtoken, bcryptjs, Zod, Helmet, Multer, and Morgan; the PostgreSQL Licence for the database engine; the Apache 2.0 licence for Docker Engine). There is, as a consequence, no licensing fee associated with the deployment of TripTale, which is a significant advantage for an academic project and for a future commercial roll-out in a country where access to commercial cloud platforms is still constrained.

3.6 Conclusion

This chapter has described the complete technological stack of TripTale: a React Native and Expo frontend, a Node.js and Express backend written in TypeScript, a PostgreSQL database accessed through the Prisma ORM, and a development workflow built on Visual Studio Code, Git, GitHub, Docker Compose, ESLint, and Swagger UI. We have also argued that this stack is not merely a collection of fashionable tools but a coherent answer to the non-functional requirements stated in chapter 1, namely portability, rapid iteration, scalability, robustness, ecosystem support, and cost-effectiveness. The next chapter exploits this technological foundation to describe, module by module, how the requirements of chapter 1 and the design of chapter 2 are translated into the running application.

4. Chapter 4 — Implementation of TripTale

Once the analysis, design, and technological choices have been established, the final stage consists of implementing the proposed solution. The implementation phase transforms the theoretical design into a fully functional application by integrating all modules and services defined in the previous chapters.

This chapter presents the implementation of TripTale and explains the realization of its main features. It covers the authentication system, trip management functionalities, media handling, premium services, backend APIs, and database integration. Screenshots and implementation details are provided to illustrate the behavior and functionality of the application.

4.1 Authentication System

TripTale's authentication system is responsible for securely identifying users and maintaining their session across app launches. The implementation is split between the mobile frontend (React Native/Expo) and the backend (Express + JWT + Prisma).

4.1.1 Signup (Registration)

Frontend workflow:

5. User opens the Signup screen and enters *username*, *email*, and *password*.
6. The app sends a request to the backend registration endpoint.
7. If successful, the backend returns user data and authentication tokens.
8. The app stores the tokens and transitions the user into the main application (tabs).

```
POST /api/auth/register
Content-Type: application/json
```

```
{
  "username": "exampleUser",
  "email": "user@example.com",
  "password": "password123"
}
```

4.1.2 Login

Frontend workflow:

9. User enters email/password on the Login screen.
10. The app calls POST /api/auth/login.
11. On success, backend returns { user, accessToken, refreshToken }.
12. The app saves tokens locally and updates the global auth state (AuthContext).

```
POST /api/auth/login
Content-Type: application/json
```

```
{
  "email": "user@example.com",
  "password": "password123"
}
```

Screenshot placeholder: Insert a screenshot of the Login screen (inputs + login action). Add a short caption below the image.

4.1.3 Token / Session Storage

TripTale uses local storage (commonly AsyncStorage) to persist authentication state. This enables the application to keep the user signed in after the app is closed and re-opened.

- **Access Token (JWT):** short-lived token used to access protected endpoints.
- **Refresh Token:** longer-lived token used to obtain a new access token when the old one expires.

4.1.4 AuthContext

The frontend uses a global state layer (AuthContext) that stores:

- current user profile (id, username, avatar, etc.)
- authentication status (logged in / logged out)
- methods such as login(), logout(), and restoreSession()

4.2 Trip Creation Module

4.2.1 Trip Publishing Flow

Trip creation is implemented as a guided flow: user enters details, previews the trip, and then publishes it. Publishing triggers a backend request that stores a new Trip record and links it to the authenticated user.

13. Open “Create Trip” in the Profile section.
14. Fill trip form (title, description, location, category, budget, cover photo).
15. Preview the trip content.
16. Publish → backend stores the Trip; frontend refreshes Profile and Discover lists.

4.2.2 Image Uploading

TripTale supports uploading images/videos to a trip using a media endpoint that accepts multipart/form-data. The database stores only the URL/path of the media file, not the file binary itself.

```
POST /api/trips/{tripId}/media
Content-Type: multipart/form-data
Authorization: Bearer <accessToken>
```

```
files=@photo.jpg
caption="View from the top"
```

4.2.3 Preview System

The preview step improves quality and helps users validate content before making it visible to others. The UI typically reuses the same layout as the trip details screen.

4.2.4 Edit / Discard Logic

If the user decides not to publish:

- **Edit:** returns to the form with values preserved.
- **Discard:** clears draft state and returns to Profile.

4.3 Premium Features

Premium features are designed to be added incrementally without breaking the core experience. The premium model improves user experience (ad-free), increases creator value (analytics), and supports monetization.

4.3.1 Premium Badge

- Premium status displays a badge on the user profile and next to the username on posts/comments.
- Badge visibility is controlled by the subscription status.

4.3.2 Analytics

Premium users can see analytics for their posts, such as:

- views/impressions per trip
- saves count and change over time
- comments and ratings trends

4.3.3 Unlimited Uploads

Premium removes media limits. The backend can enforce:

- standard users: maximum N files per trip (or per day)
- premium users: unlimited or a significantly higher limit

4.4 API and Backend Implementation

4.4.1 API Routes

TripTale exposes REST endpoints grouped by module (auth, trips, categories, etc.). These routes are documented using Swagger for testing and debugging.

4.4.2 Request / Response Flow

17. Frontend triggers a request (API client).
18. Backend validates request (body/query/params).
19. Middleware checks authentication (if required).
20. Controller/service executes business logic.
21. Prisma executes database operations.
22. Backend returns JSON response to the client.

4.4.3 Backend Validation

Backend validation prevents invalid data from entering the system:

- schema validation (required fields, types, formats)
- authorization rules (owner-only editing and deletion)
- constraints enforced by DB (unique keys)

4.5 Database Operations

4.5.1 CRUD Overview

- **Create:** register user, create trip, add comment/rating, save trip.
- **Read:** list categories, browse trips with filters, view trip details.
- **Update:** edit trip, update profile, update saved notes/priority.
- **Delete:** delete comment, remove saved trip, delete media.

4.5.2 Prisma Usage

Prisma provides:

- type-safe queries in TypeScript
- migrations for consistent schema across environments

- easy switching from SQLite (dev) to PostgreSQL (prod) by updating datasource config

4.6 State Management

4.6.1 Context API

TripTale uses Context API for global state such as authentication and user session data. This reduces prop drilling and simplifies protected routing.

4.6.2 Hooks

- hooks manage loading/error states consistently
- custom hooks can wrap API requests and state updates

4.6.3 Global State Handling

- AuthContext stores tokens/user and exposes login/logout actions
- screen-level state handles local UI (forms, selected category, etc.)

4.7 Security and Error Handling

4.7.1 Validation

- frontend validates basic inputs for UX
- backend validates strictly for correctness and safety

4.7.2 Token Protection

- access tokens required for protected endpoints
- refresh tokens rotated and revoked on logout
- backend verifies JWT signature and expiration

4.7.3 Error Messages

- backend returns consistent error JSON
- frontend shows user-friendly errors (invalid login, network issues)

4.7.4 Protected Routes

- trip creation requires authentication
- edit/delete require ownership checks
- saved trips list is user-specific and protected

4.8 Challenges Encountered During Development

Very important section: Replace the bullets below with the real issues you encountered and how you solved them.

4.8.1 Authentication Issues

- token expiration during testing
- handling refresh token rotation correctly
- session persistence across app restarts

Solution approach: centralize token handling in API client, implement refresh flow, and ensure logout clears tokens.

4.8.2 Database Relationships

- ensuring correct foreign keys between user, trip, comment, rating
- preventing duplicates (rating uniqueness, saved trip uniqueness)

Solution approach: enforce unique constraints and use Prisma relations with cascade deletes.

4.8.3 Navigation Bugs

- incorrect redirects after login/signup
- back navigation issues after publish/discard

Solution approach: standardize navigation entry points and use consistent route guards.

4.8.4 Image Uploads

- multipart request formatting
- file size limits and unsupported formats
- displaying uploaded media reliably in UI

Solution approach: backend upload middleware + stable client upload function + fallback UI.

4.8.5 Synchronization Problems

- stale data if caching not handled
- ensuring saved/comments/ratings reflect shared PostgreSQL state

Solution approach: refresh lists after actions and avoid overly aggressive caching.

4.9 Future Improvements

Although TripTale already offers essential features for travel discovery, planning, and social interaction, several future improvements can further enhance the platform and provide users with a more complete travel experience.

4.9.1 the expansion of the premium subscription:

One important improvement is the expansion of the premium subscription system. Future premium features may include advanced trip analytics, unlimited media uploads, personalized profile customization, AI-based recommendations, and an ad-free experience. In addition, affiliate partnerships with hotels, airlines, and tourism services could be integrated to provide users with special offers while generating revenue for the platform.

4.9.2 travel agencies:

TripTale can also collaborate with travel agencies by introducing verified agency profiles. These profiles would allow agencies to publish travel packages, promotional offers, and organized trips directly inside the application. Users would then be able to interact with agencies, explore travel deals, and access professional tourism services more easily.

4.9.3 AI enhancement:

Artificial Intelligence is another promising enhancement for the application. AI could analyze user preferences and travel history to generate personalized destination recommendations, intelligent itineraries, and activity suggestions based on budget, interests, and season. This would improve user experience and make travel planning faster and more efficient.

4.9.4 Interactive maps:

Future versions of TripTale may also include maps and geolocation services. Interactive maps would help users explore destinations, locate nearby attractions, and visualize travel routes. Push notifications could also be added to inform users about comments, likes, new followers, travel reminders, and special offers in real time.

4.9.5 booking systems:

Another major enhancement is the integration of booking systems, allowing users to reserve hotels, flights, transportation, and travel activities directly through the application. An offline mode could also be implemented so travelers can access saved trips, guides, and maps without internet access.

4.9.6 real-time chat:

Finally, a real-time chat system would improve communication between travelers by enabling private conversations, group discussions, and travel collaboration. These future improvements would transform TripTale into a more intelligent, interactive, and complete travel platform capable of meeting the needs of modern travelers.

4.10 Conclusion

This chapter explained how TripTale is implemented across the frontend, backend, and database layers. It covered authentication, trip creation, premium strategy, API flow, database CRUD via Prisma, state management, security, and development challenges. The implementation choices support a scalable roadmap for future features such as AI recommendations, maps, and real-time communication.

General Conclusion

This thesis presented the design and development of TripTale, a mobile application dedicated to travel discovery, planning, and experience sharing. The project was motivated by the need for a unified platform that allows users to explore destinations, share travel experiences, and interact with a community of travelers within a single environment.

The work began with the analysis of existing travel applications and the identification of their limitations. Based on this study, the functional and non-functional requirements of the proposed solution were defined. A complete system design was then developed, including the system architecture, database design, and user interface structure. Finally, the application was implemented using modern technologies such as React Native, Express.js, Prisma ORM, and PostgreSQL.

The resulting system successfully integrates essential travel-related functionalities, including trip discovery, content creation, social interaction, trip saving, and premium features. The project demonstrates how modern software engineering practices can be applied to create a scalable and user-friendly mobile application.

As future work, several enhancements can be considered, such as AI-powered travel recommendations, map integration, real-time messaging, booking services, push notifications, and offline support. These improvements would further enrich the user experience and increase the practical value of the platform.

In conclusion, TripTale provides a solid foundation for a modern travel-sharing platform and represents a promising contribution to the digital tourism ecosystem.

References

- [1] iso25000. [Online]. Available: <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010>.
- [2] "nngroup," [Online]. Available: <https://www.nngroup.com/articles/ten-usability-heuristics/>.
- [3] "medium," [Online]. Available: <https://medium.com/@lizdenhup/understanding-unidirectional-data-flow-in-react-3e3524c09d8e>.
- [4] "ibm," [Online]. Available: <https://www.ibm.com/think/topics/rest-apis>.
- [5] "developer.mozilla," [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Express_Nodejs.
- [6] "v8.dev," [Online]. Available: <https://v8.dev/>.
- [7] "nodejs," [Online]. Available: <https://nodejs.org/learn/getting-started/introduction-to-nodejs>.
- [8] "ics.uci.edu," [Online]. Available: <https://ics.uci.edu/~fielding/pubs/dissertation/top.htm>.
- [9] "ibm," [Online]. Available: <https://www.ibm.com/think/topics/json-web-tokens>.
- [10] "postgresql," [Online]. Available: <https://www.postgresql.org/about/>.
- [11] "prisma," [Online]. Available: <https://www.prisma.io/docs>.
- [12] "docker," [Online]. Available: <https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-docker-compose/>.

