EÖTVÖS LORÁND UNIVERSITY
FACULTY OF INFORMATICS
DEPT. OF MEDIA AND EDUCATIONAL TECHNOLOGY

navM8

Web Application for Enhancing Travel Experiences through Local Guide Engagement

Supervisor:
Dr. Horváth Győző
Assistant Professor, PhD

Author:
Javlonbek Kosimov
Computer Science BSc

Budapest, 2024

EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

Thesis Registration Form

Student's Data:

Student's Name: Kosimov Javlonbek

Student's Neptun code: EFYBOX

Course Data:

Student's Major: Computer Science BSc

I have an internal supervisor

Internal Supervisor's Name: *Horváth Győző Dr.*

Supervisor's Home Institution: ***Department of Media and Educational Technology***

Address of Supervisor's Home Institution: ***1117, Budapest, Pázmány Péter sétány 1/C.***

Supervisor's Position and Degree: *Assistant Professor, Phd*

Thesis Title: Development of a Full-Stack Web Application for Enhancing Travel Experiences through Local Guide Engagement

Topic of the Thesis:

(Upon consulting with your supervisor, give a 150-300-word-long synopsis os your planned thesis.)

The thesis proposes the development of a full-stack web application designed to revolutionize the travel experience by connecting travelers with local guides. This application, utilizing the MERN stack (MongoDB, Express.js, React, Node.js), aims to facilitate a more immersive and personalized exploration of new destinations.

The core functionality of the application includes two separate user registration modules for travelers and guides, ensuring tailored experiences for each user type. Travelers can create profiles, detailing personal preferences and interests, while guides can showcase their skills, language proficiency, and experiences, including the option to upload relevant credentials.

A key feature of the application is the traveler interface, which allows users to search for guides based on specific criteria such as location, language, and interests. This feature is complemented by a comprehensive guide profile section, including ratings and reviews, to assist travelers in making informed decisions.

The application also integrates a booking and scheduling system, complete with a calendar feature to display guide availability. This system streamlines the process of scheduling tours, enhancing the overall user experience.

Furthermore, the application includes an in-app messaging system, enabling direct communication between travelers and guides, thus facilitating better coordination and personalized tour planning. Finally, a ratings and reviews system is incorporated, allowing travelers to provide feedback post-tour, which serves as a valuable tool for continuous improvement and guide credibility.

This thesis will cover the development process, challenges, solutions, and the impact of this application in the context of the modern travel industry.

Budapest, 2023. 11. 29.

Table of Contents

Chapter 1

1. Introduction

Traveling is about exploring new places and meeting new people. However, getting to know a city through the eyes of someone who lives there can be expensive. That's where **navM8** (short for **navigator mate**) comes in. This handy web app connects travelers with locals who give free personal tours, making it easy to dive into authentic experiences without worrying about the cost.

The idea for navM8 sparked from a love of travel and noticing how expensive it can be to get real local insight in new cities. A lot of travelers miss out on cool local spots because tour prices add up fast. navM8 changes that, offering a way to connect with local guides for free.

This thesis chose to focus on navM8 because it blends a passion for travel with a smart solution to make traveling richer for everyone. The goal is to shift how travelers interact with places they visit, putting the spotlight on exploration without breaking the bank.

In this thesis, the development of navM8, its impact on how people travel, and how it benefits both travelers and locals will be explored. It highlights the importance of easy, affordable access to authentic travel experiences, making adventures more meaningful for everyone

Chapter 2

2. User Documentation

2.1 Project Description

navM8 short for **“navigator mate”** is a full-stack web application designed to connect travelers with local guides for authentic, personalized travel experiences. The application aims to make exploring new places more accessible and enjoyable by leveraging local insights without the financial burden of traditional tour services.

Key Features:

- **Search for Tours by City:** Allows travelers to search for available tours based on specific cities.
- **Booking Tours:** Provides a secure method for booking tours with local guides.
- **Creating Tours:** Enables users to create and offer tours, showcasing unique experiences in their city.
- **Profile Creation:** Allows users to create personalized profiles to either offer or seek tour services.
- **Real-Time Messaging:** Facilitates direct communication between travelers and guides for better planning and interaction.
- **Rating and Review System:** After completing a tour, travelers can rate and review the tour, aiding future travelers in making informed decision.

2.2 System Requirements

Requirements:

- **Operating Systems:** Compatible with all major operating systems including Windows, macOS, and Linux, ensuring accessibility on desktop devices. Additionally, the web application is responsive and optimized for mobile devices such as tablets and smartphones.
- **Supported Browsers**: Supports all modern web browsers, including Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge.
- **Screen Resolution:** The app is designed to be fully responsive, adapting to various screen resolutions. Optimal viewing is achieved on devices with a resolution of 1280x720 pixels or higher, but it is functional on lower resolutions as well.

2.3 App Usage and User Interface

This section provides a detailed overview of the navM8 web application's user interface and usage. The aim is to familiarize users with the main areas of the application, including the home page, user profiles, booking interfaces, and communication tools. This will enable them to effectively utilize navM8 to enhance their travel experiences or offer their services as a local guide. Each subsection describes a specific part of the application, supported by screenshots that illustrate the workflows and interactions expected while using the app.

2.3.1 Get Started

navM8 is currently available for local testing and has not been deployed to a public server. To access the application:

1. Ensure you have the project setup on your local machine and the server is running.
2. Open your preferred web browser.
3. Navigate to the URL: http://localhost:3001/

This local URL directs to the development environment on a local machine where the navM8 server operates on port 3001.

Note: This URL is only accessible on your device where the server is running unless configured for network access within a local network.

Upon accessing navM8 at http://localhost:3001/, users are greeted by the main page, which offers a user-friendly interface to explore the available tours and features of the app.

Initial View:

- **Home Screen:** Displays a dynamic map image and a "Choose Destination" search bar at the center. Below, there is a section titled "The Most Popular Tours," showcasing available tours in various cities. *(see figure 1)*
- **Tour Cards:** Each tour is presented in a card format, providing essential information such as the city, tour description, availability, and a visual snippet of the location.

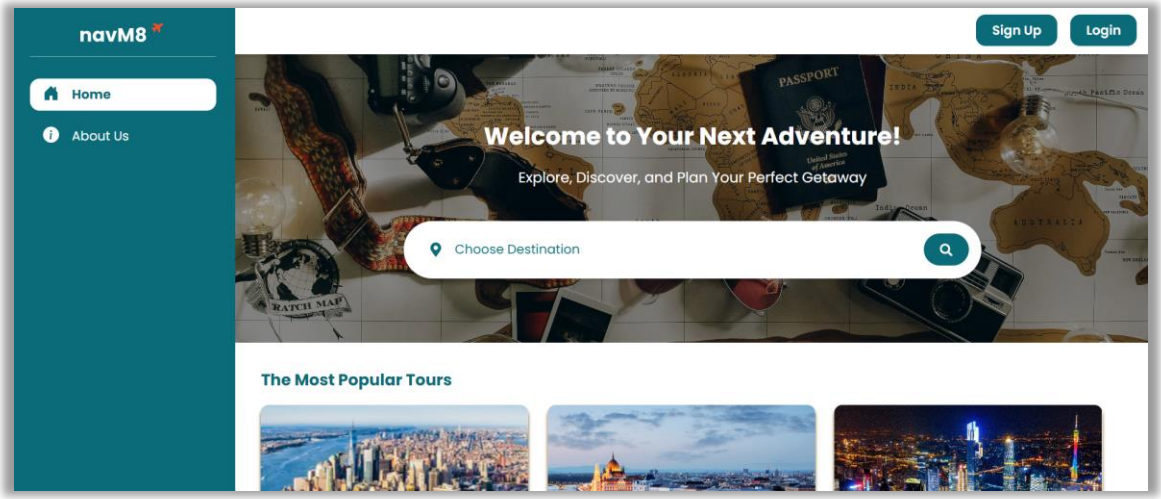


Figure 1: Home Page for guests

Interaction Before Login:

- Users can freely browse all listed tours and use the search functionality to find tours by city.

- Clicking on any tour card prompts a "Sign Up or Log In" message, as detailed interactions such as booking or viewing detailed tour information require user authentication. *(see figure 2)*

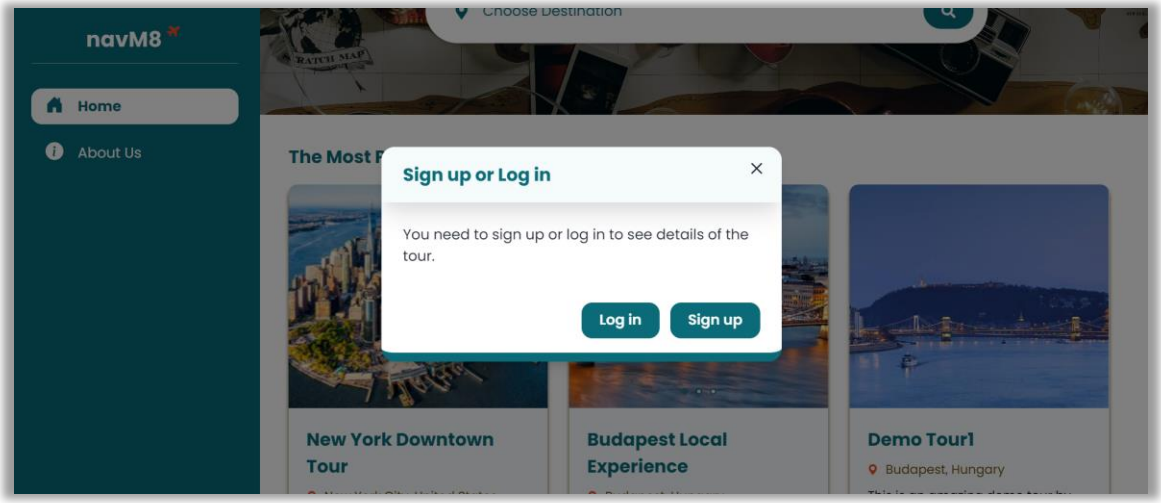


Figure 2: Modal window Sign Up/Log In

About Us Page:

- Accessible from the navigation menu, this page provides insights about navM8, including the mission and statistics such as the number of active users and tours available. *(see figure 3)*

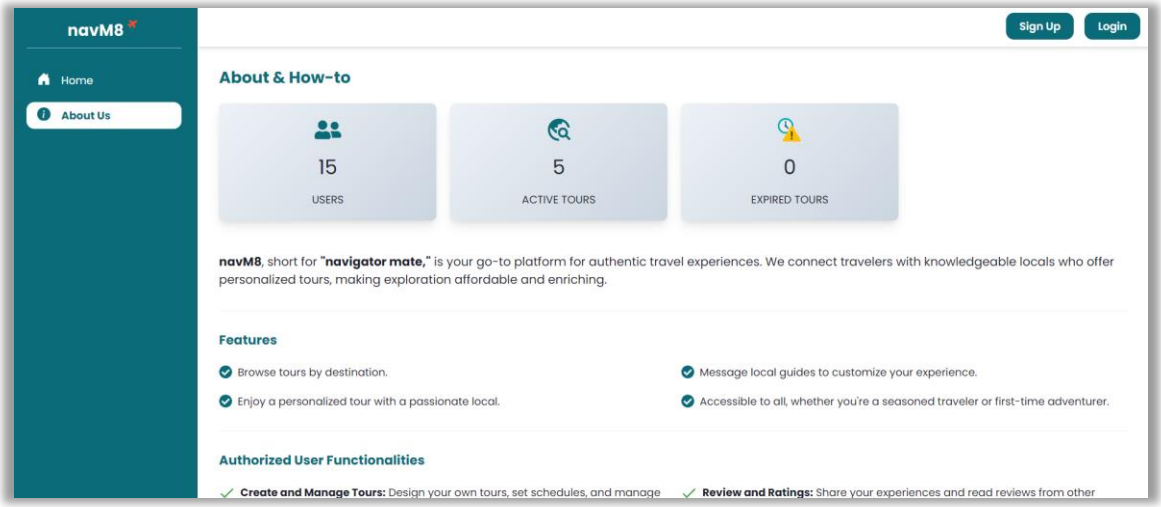


Figure 3: About Us Page for guests

2.3.2 Registration and Login

To begin exploring and booking tours with navM8, users first need to create an account. This section guides through the registration process. *(see figure 4)*

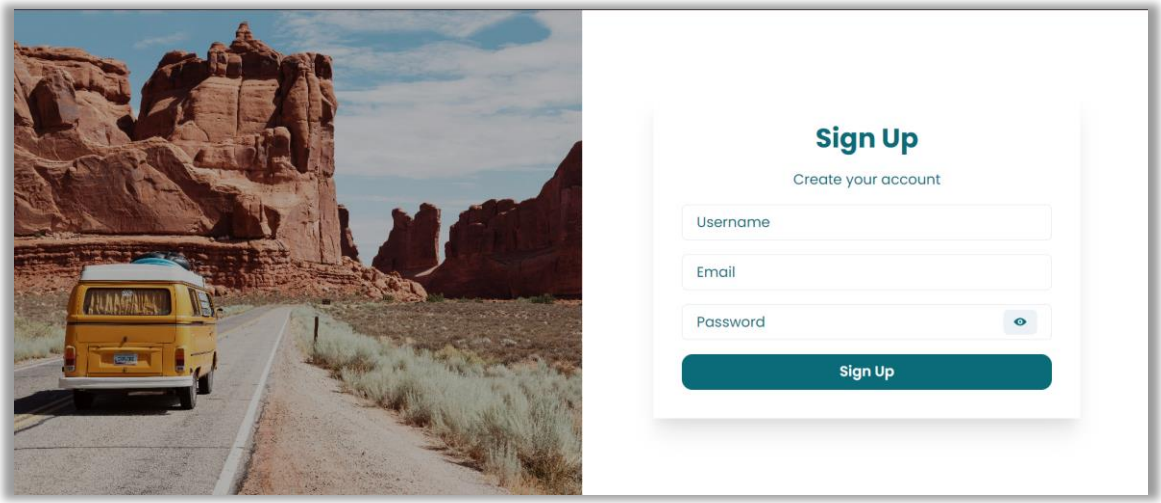


Figure 4: Sign Up Page

Steps to Register:

1. **Navigate to the Sign-Up Page:** Access the sign-up page directly by clicking on the 'Sign Up' button located at the top right corner of the homepage.
2. **Fill Out the Registration Form:** Users will need to provide the following information:
    - **Username:** Choose a unique username with at least 3 letters.
    - **Email:** Enter a valid and unique email address.
    - **Password:** Create a strong password to secure the account, meeting the following requirements:
        - Minimum length of 8 characters.
        - At least 1 lowercase letter.
        - At least 1 uppercase letter.
        - At least 1 number.
        - At least 1 symbol.
3. **Submit Your Details:** After filling out the form, click the 'Sign Up' button to complete the registration. *(see figure 4)*

Login

Once registered, users can log in to access and manage their bookings and personal account details. *(see figure 5)*

Steps to Log In:

1. **Navigate to the Login Page:**
    - Click on the 'Login' button located at the top right corner of the homepage.
2. **Enter Your Credentials:**
    - **Email:** Use the email address you registered with.
    - **Password:** Enter your password
3. **Access Your Account:** After entering your credentials, click the 'Login' button to access your profile and start using the application.

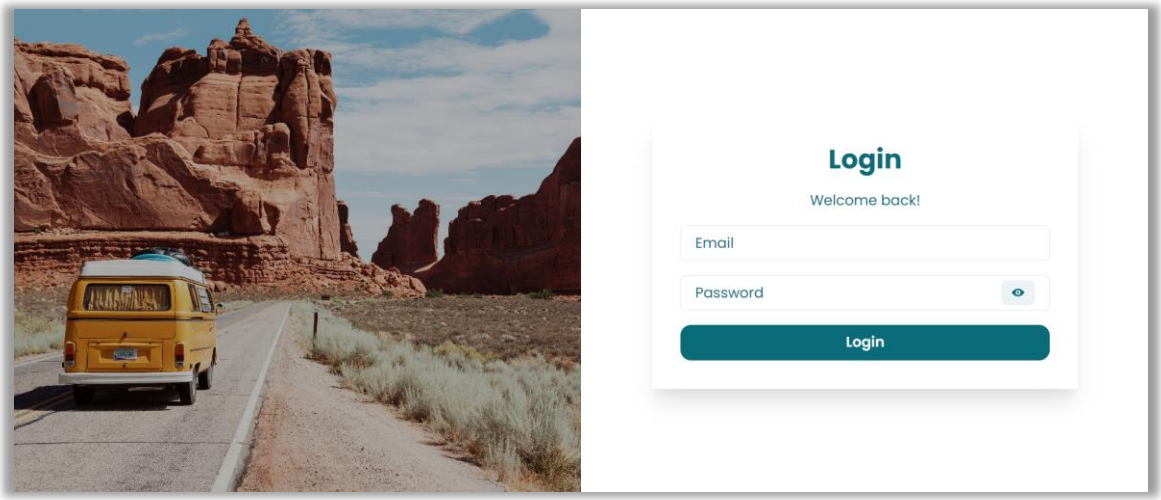


Figure 5: Log In Page

Successful Login/Signup Notification: Upon logging in or signing up, a notification appears at the bottom of the page confirming successful authentication. *(see figure 6 and figure 7)*

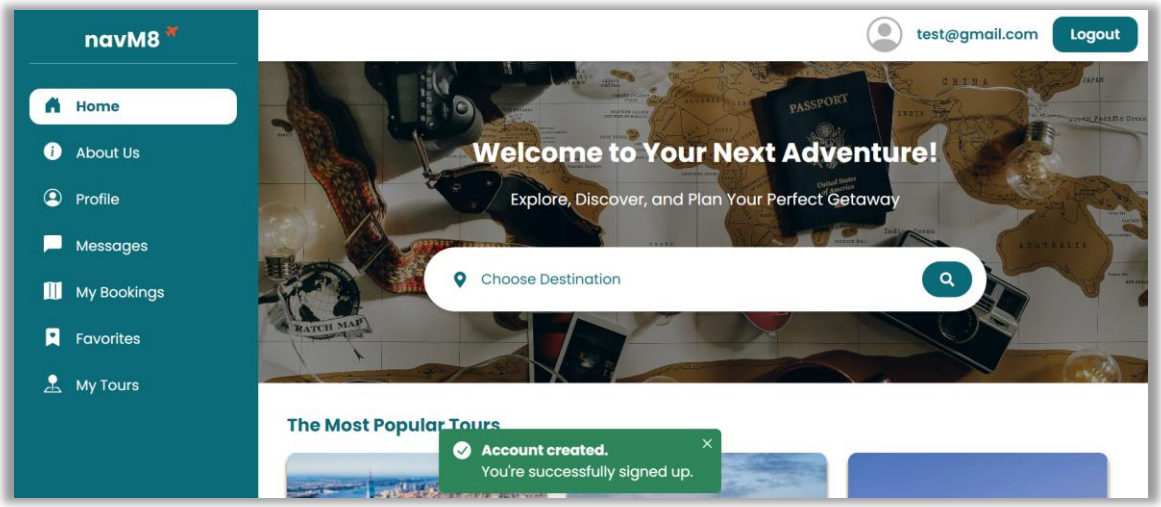


Figure 6: Successful Sign Up

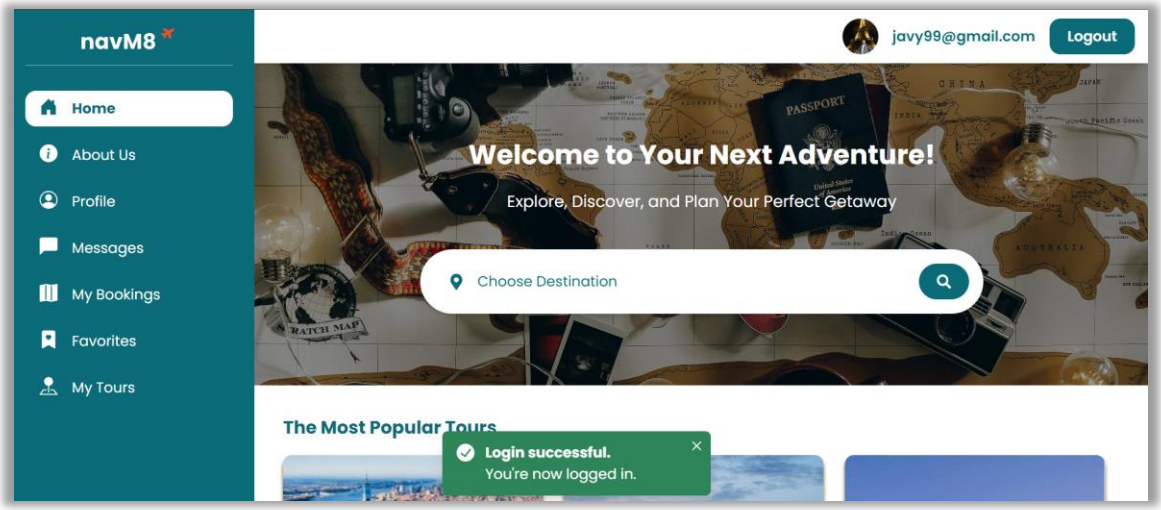


Figure 7: Successful Log In

Sidebar Navigation Post-Login:

Once logged into navM8, the sidebar becomes an essential tool for navigating through the application, offering quick access to all major sections of the platform:

- **Home:** The starting point of the navM8 journey, where users can browse and search for tours.
- **About Us:** Learn more about navM8's mission and insights into the platform.
- **Profile:** Manage personal information
- **Messages:** Communicate directly with other users or tour guides.
- **My Bookings:** View and manage upcoming and past tour bookings.

- **Favorites:** Access tours marked as favorites for future reference.
- **My Tours:** For tour guides, manage the tours offered, including bookings and tour details.

Functionalities Unlocked after Authorization:

1. **Profile Customization:**
    - **Description:** Allows users to update their profiles with personal information.
    - **Utility:** Personalizing profiles can improve interactions with guides and other travelers, making the community aspect of navM8 more engaging.
2. **Messaging:**
    - **Description:** Enables direct communication between travelers and guides or other community members.
    - **Utility:** Facilitates easier planning and coordination for tours and enhances user engagement with real-time interactions.
3. **Access to Bookings**:
    - **Description**: Users can view all their current and past bookings, manage tour details.
    - **Utility**: Keeps all travel plans organized and accessible, allowing users to manage their itineraries efficiently.
4. **Favorite Tours:**
    - **Description:** Users can save tours they find appealing to their Favorites section for quick access in the future.
    - **Utility:** Makes it convenient to return to tours that interest users, simplifying future bookings.
5. **Tour Creation**:
    - **Description:** Users can create and offer their own tours, providing unique local experiences to travelers. Also, the author of the tour can edit or delete tour as well as confirm booking of the traveler.
    - **Utility:** Empowers local experts and enthusiasts to share their knowledge and unique perspectives, enhancing the diversity of available tours.

2.3.3 Home Page

Upon successful signup/login, users are directed to the Home Page, which serves as the central hub for exploring available tours and accessing various functionalities. *(see figure 8)*

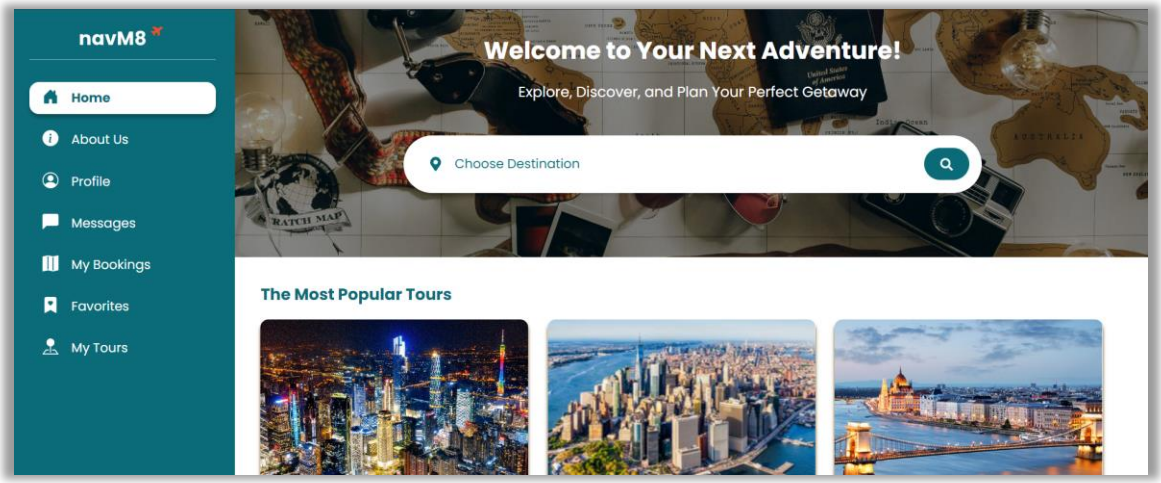


Figure 8: Home Page for authorized users

Home Page Features:

1. **Search Functionality:** The search bar allows users to search for tours by entering city names.
2. **Tour Cards**: Below the search bar, 'The Most Popular Tours' section displays cards for each tour, providing key details of the tour. Users can click on these cards to get more detailed information about each tour.

2.3.4 Tour Details Page

The Tour Details page provides comprehensive information about each tour, allowing users to make informed decisions about their bookings. *(see figure 9)*

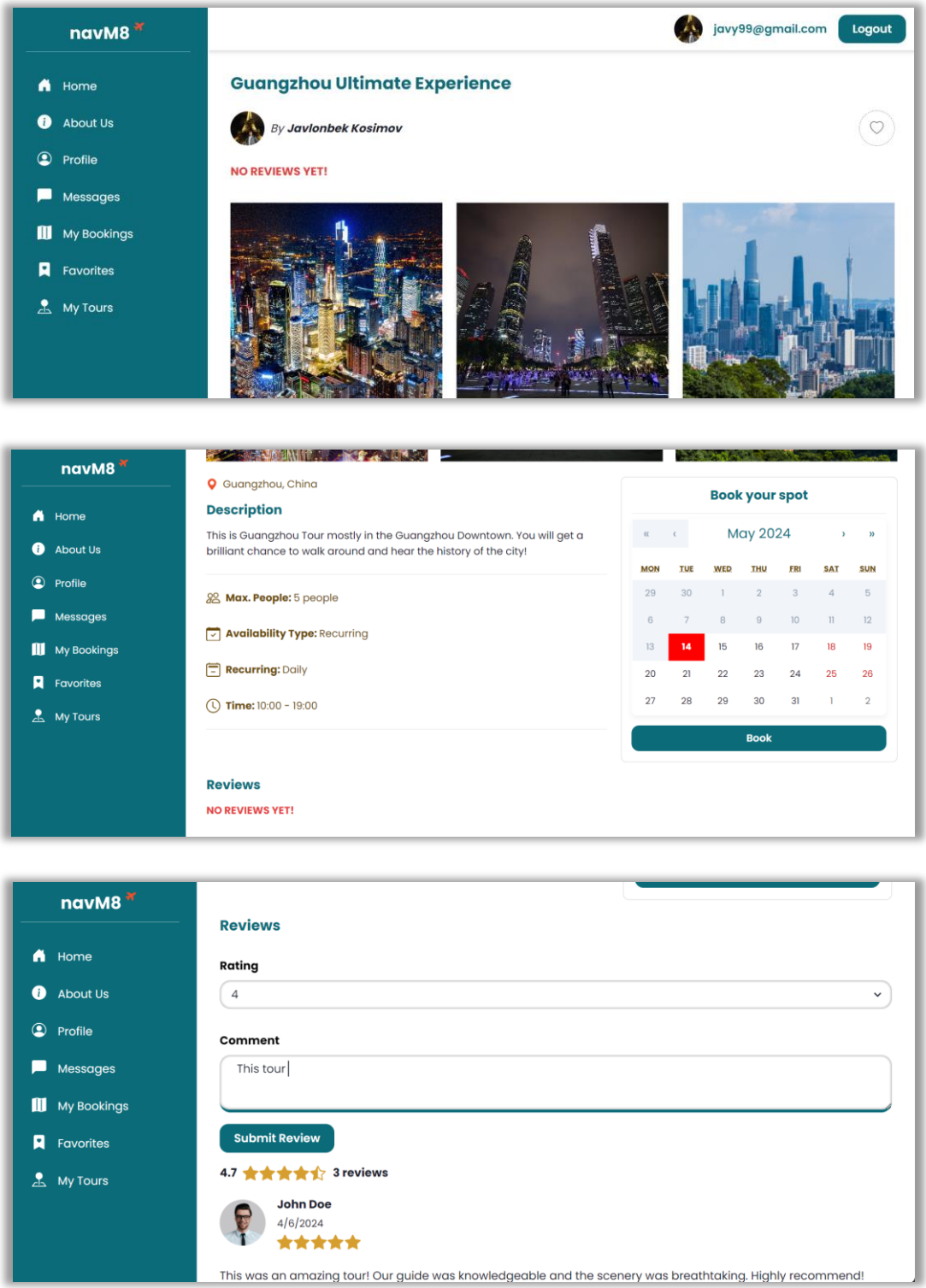


Figure 9: Tour Details Page

Page Components:

- **Detailed Description:** Each tour page includes a thorough description of the tour, highlighting key activities, points of interest, and any unique features of the tour.
- **Photos:** High-quality images showcase various aspects of the tour, giving users a visual taste of what to expect during the tour.
- **Booking Interface:** An interactive calendar allows users to select and book available dates. The 'Book' button is prominently displayed for easy access.
- **Customer Reviews:** User reviews provide insights and feedback from previous participants, enhancing transparency and trust in the tour experience.
- **Add to Favorites Button:** A clickable heart icon is located at the top right side of the page, allowing users to add the tour to their favorites.

Booking Process on the Tour Details Page:

1. **Selecting a Date:** Users select a date from the calendar based on the tour's availability (e.g., recurring daily, weekends, during weekdays or specific one-time dates).
2. **Initiating Booking:** After choosing a date, users click the 'Book' button. The bookingdate and status are displayed below the calendar *(see figure 10)*

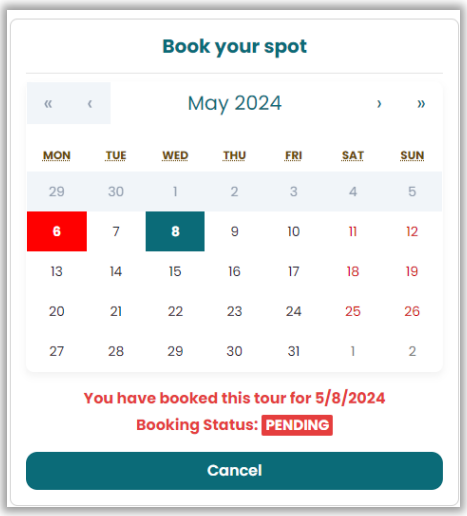


Figure 10: Booking Pending Tour

3. **Confirmation:** Once the booking is made, the tour author receives a notification

in the 'My Tours' section and should approve it. Upon approval, the traveler sees the status change below the calendar, and in 'My Bookings', the traveler can check the status of the booking *(see figure 11 and figure 12)*

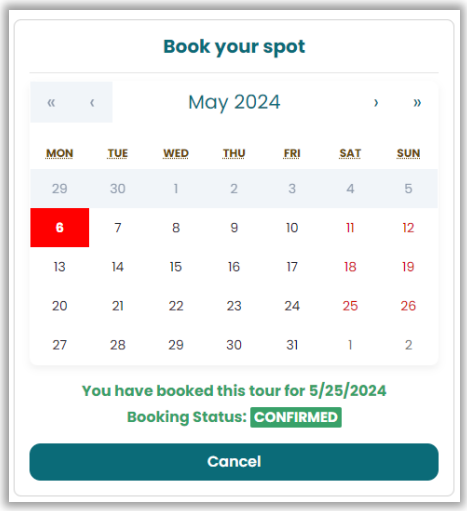


Figure 11: Booking Confirmed Tour

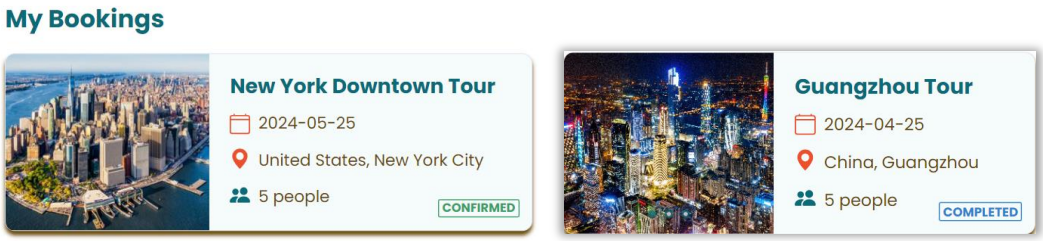


Figure 12: Tour Card in My Bookings Page

Navigating to Tour Details Page:

- **From Home Page:** Users can access the Tour Details page by clicking on a tour card from the Home Page.
- **From Favorites Page:** Tours added to 'Favorites' can be accessed, allowing users to revisit and book tours they have previously shown interest in.
- **From My Bookings Page:** Details of both upcoming and past tours can be reviewed, helping users manage their bookings and revisit past experiences.

2.3.5 About Us Page

The About Us page on navM8 provides users with a comprehensive overview of the platform's purpose, along with key statistics of the application. *(see figure 13)*

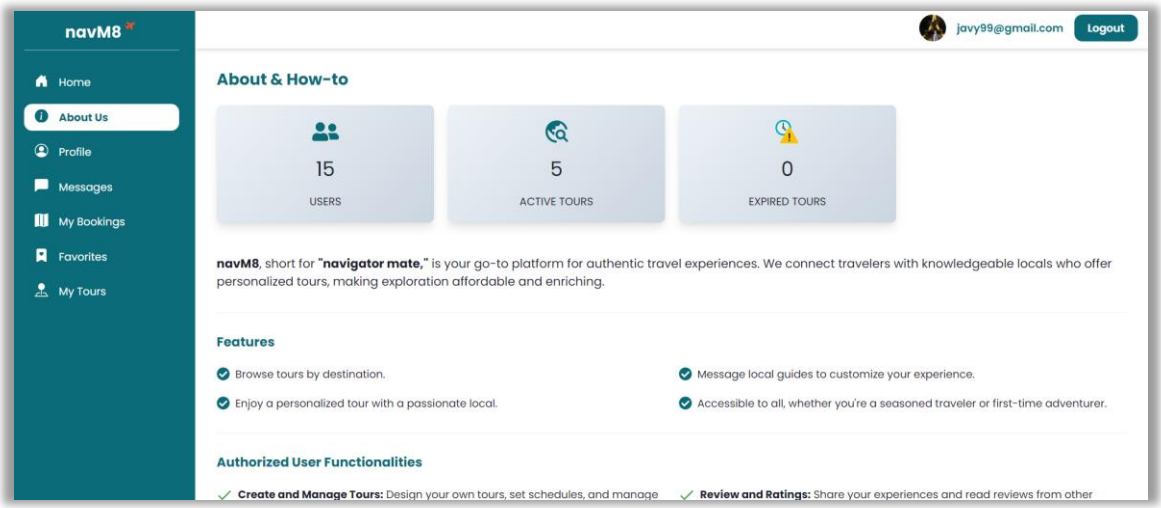


Figure 13: About Us Page for authorized users

Page Overview:

- **Mission Statement:** navM8 connects travelers with local guides for affordable, enriching travel experiences.
- **Statistics:** Displays real-time counts of users, active tours, and expired tours.
- **Features:** Highlights include browsing tours, messaging guides, personalized tours, and accessibility for all travelers.
- **Authorized User Functionalities:** Registered users can create/manage tours, access reviews/ratings, get exclusive deals, use advanced messaging, receive personalized recommendations, and enjoy priority support.

2.3.6 Profile Page

The Profile Page on navM8 serves as a dedicated environment for users to manage and update their personal information, ensuring a tailored and secure user experience. *(see figure 14)*

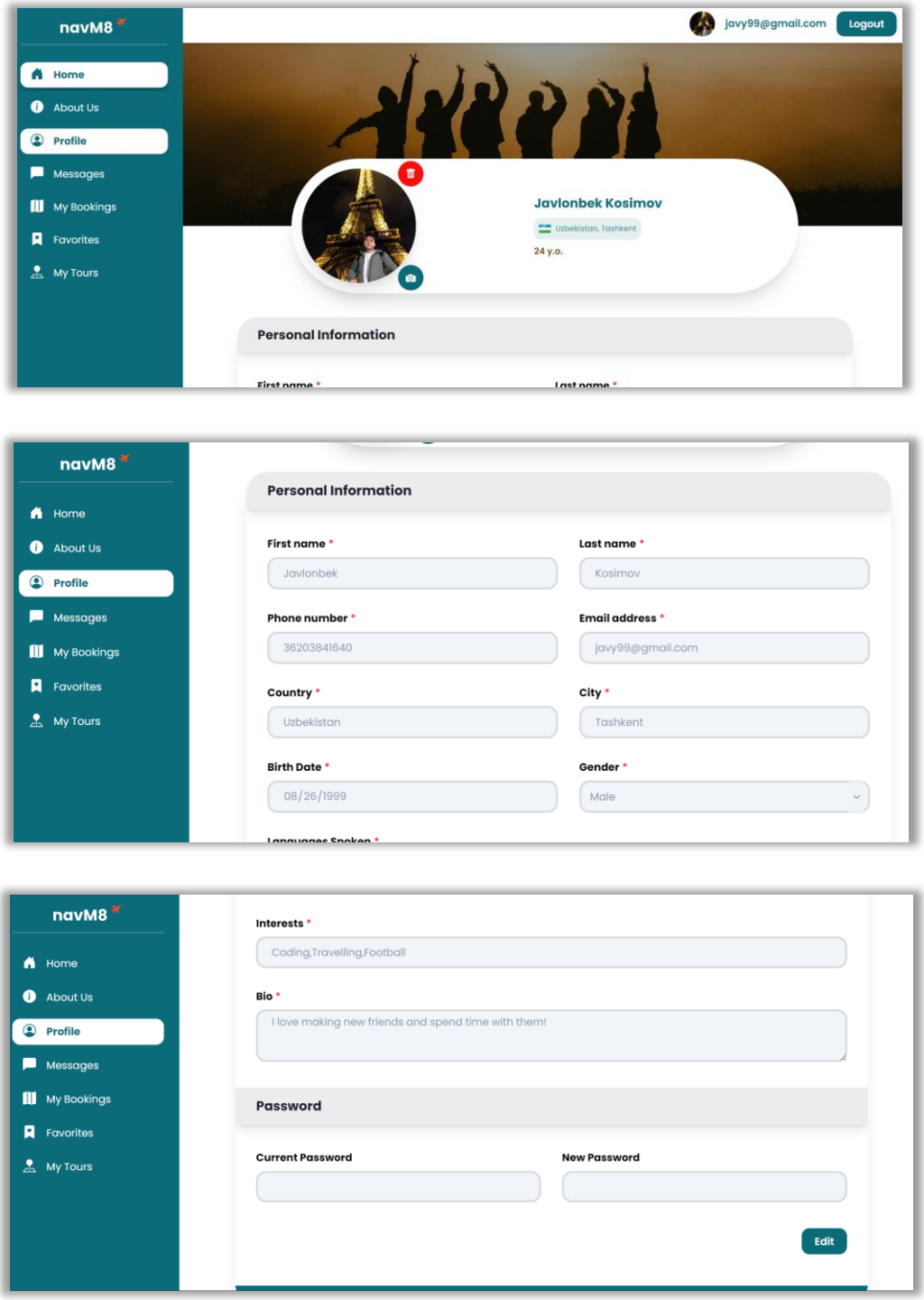


Figure 14: Profile Page

Page Components:

- **Personal Information:** This section enables users to input and modify their personal details, such as name, contact information, location, and interests. It is designed for ease of use, allowing users to keep their profile current with their latest information.
- **Password Management:** The profile page also provides an option to change the current password, enhancing security and user control over account access.

2.3.7 Messages Page

The Messages page on navM8 is designed to facilitate seamless communication between users, enhancing the social and coordination aspects of the platform through real-time messaging. This feature allows users to interact directly and efficiently, supporting a more connected community experience. *(see figure 15)*

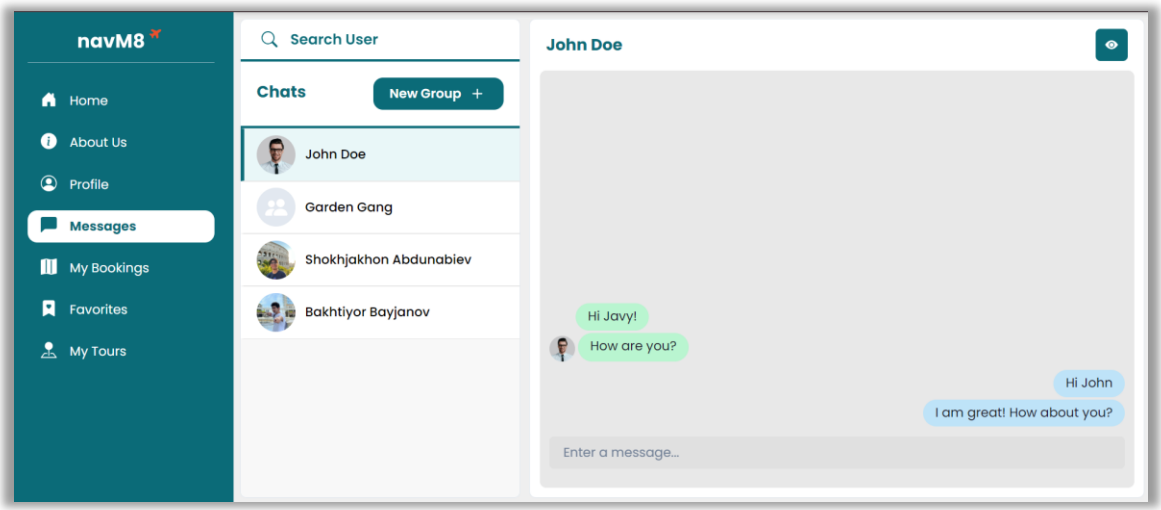


Figure 15: Messages Page

Features of the Messages Page:

- **User Search:** Located at the top of the Messages page, a search bar allows users to look up other registered members by name or email, facilitating easy and quick connections within the community. *(see figure 16)*

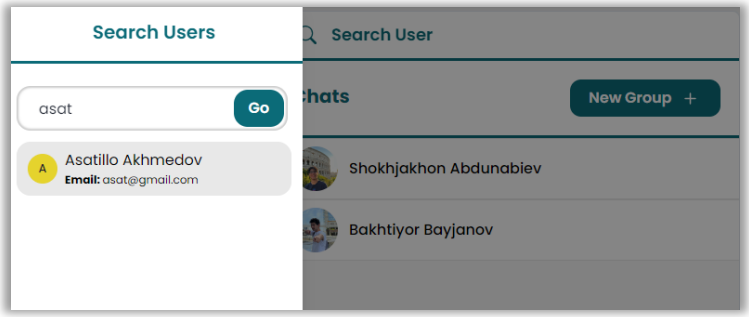


Figure 16: Search Users in Messages Page

- **One-on-One Messaging:** This feature enables users to engage in private conversations with other members.
- **Creating Group Chats:** To better coordinate among groups traveling together or planning a tour, users can initiate group chats. This is done by clicking the 'New Group +' button. A pop-up window allows them to set a chat name and add multiple users, facilitating effective group interactions. *(see figure 17)*

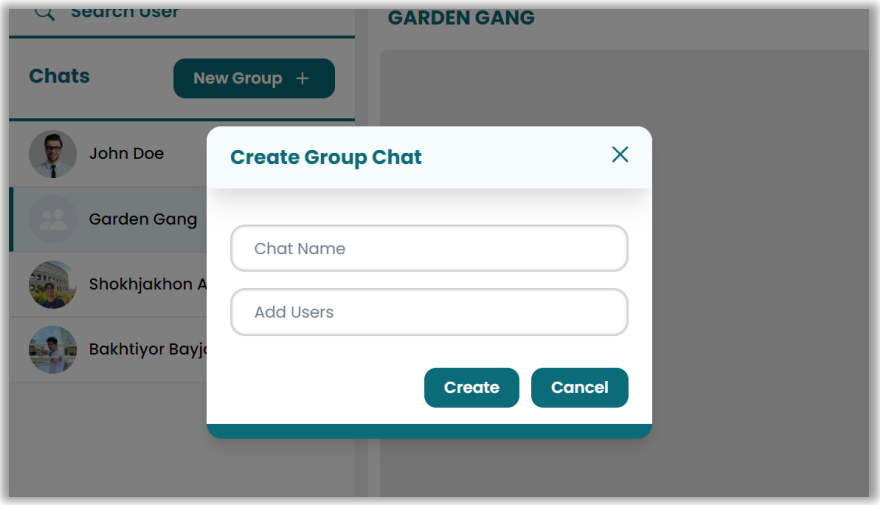


Figure 17: Create Group Chat

- **Chat Management Features:** In both group chats and one-on-one messaging, clicking the eye icon on the top right corner of the chat window allows users to view detailed user information in one-on-one chats or manage group details. In group settings, this feature enables the addition or removal of members and the ability to change the group's name. *(see figure 18)*

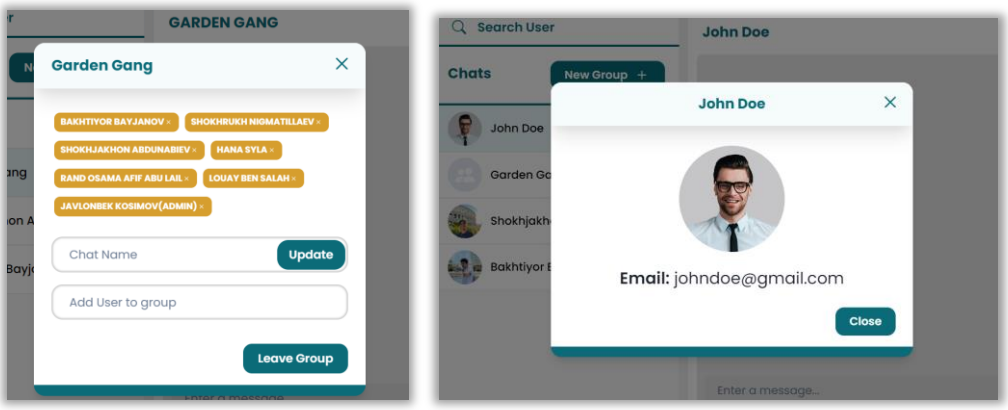


Figure 18: Chat Management

2.3.8 My Bookings Page

The "My Bookings" page on navM8 provides users with a detailed overview of both their current and past tour bookings, enhancing their ability to manage and review their travel plans effectively. *(see figure 19)*

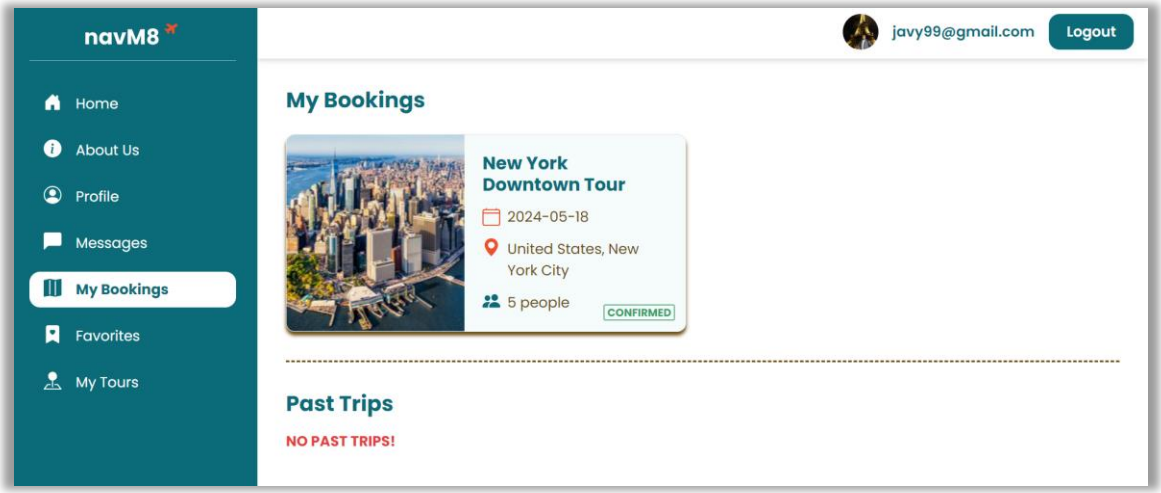


Figure 19: My Bookings Page

Features of the My Bookings Page:

- **Current Bookings:** This section displays cards for each upcoming tour the user has booked. Each card contains essential details about the tour and a brief description. Notably, the status of each booking is displayed at the bottom right

of the card, providing clear, at-a-glance information on the status of each tour, such as confirmed, pending, or needs action.

- **Past Trips:** Located below the current bookings, the "Past Trips" area lists all the tours that the user has completed. This archival feature allows users to revisit their past experiences and access details from previous tours.
- **Tour Details Access:** Users can access a detailed view of any tour by clicking on its corresponding card. This direct link to the "Tour Details Page" (as described in section 2.3.4) allows users to review specific details of the tours, further enriching their understanding and memories of their travel experiences.

2.3.9 Favorites Page

The "Favorites" section on navM8 offers a personalized space for users to save tours they are interested in, making it easier to revisit and manage preferred tours for future reference. *(see figure 20)*

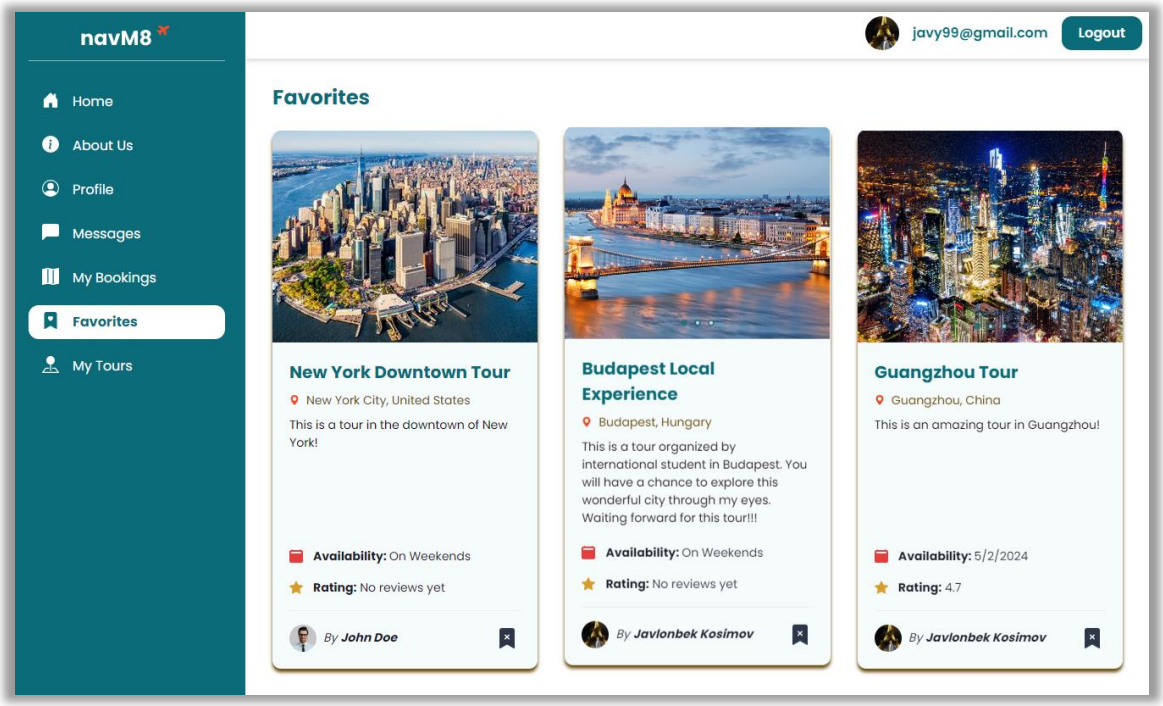


Figure 20: My Favorites Page

Overview of Features:

- **Adding to Favorites:** Users can add tours to their favorites from various sections of the app, such as the tour details page and directly from tour cards displayed on the home page. This feature provides an intuitive way for users to mark tours they find appealing by simply clicking a heart icon or a similar button on the tour card. *(see figure 21)*

Figure 21: Adding Tour to Favorites

- **Viewing Favorites:** The favorites page displays all the tours that a user has marked as favorites. Each tour card provides key details such as the tour location, availability, rating, and a brief description.
- **Managing Favorites:** Users have the flexibility to remove tours from their favorites at any time. Each tour card has a remove or heart button, which when clicked, will immediately remove the tour from the favorites list.

Functional Benefits:

- **Convenience:** This feature allows users to easily revisit and manage tours they are interested in without needing to go through the entire search process again.
- **Quick Access:** Each favorite tour card is clickable and opens the tour details page, where users can find more detailed information or proceed to book the tour.

2.3.10 My Tours Page

The "My Tours" page on navM8 is specifically designed for users who wish to offer their own tours. This section of the application allows these users, acting as local guides, to manage and showcase their tours to potential travelers. *(see figure 22)*

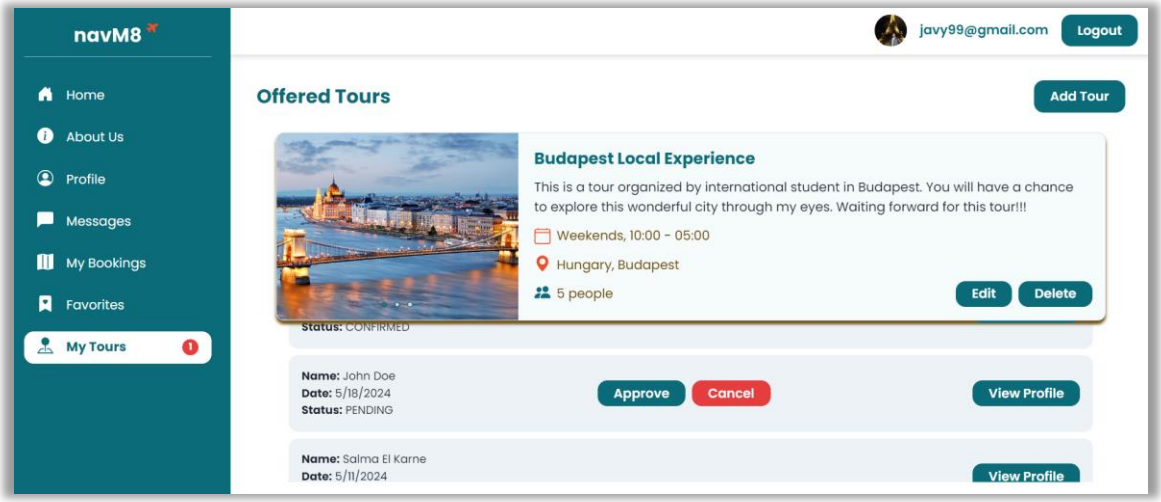


Figure 22: My Tours Page

Page Features:

1. **Offered Tours Overview:**
    - The page lists all the tours that the user has created, each presented in a card format. These cards include essential details about the tour, making it easy for potential travelers to get an overview.
2. **Create a Tour:**
    - Users can add new tours by clicking the 'Add Tour' button. This action opens a form where users can input tour details. Additionally, users can upload photos to make the tour more appealing and engaging for potential participants. *(see figure 23)*

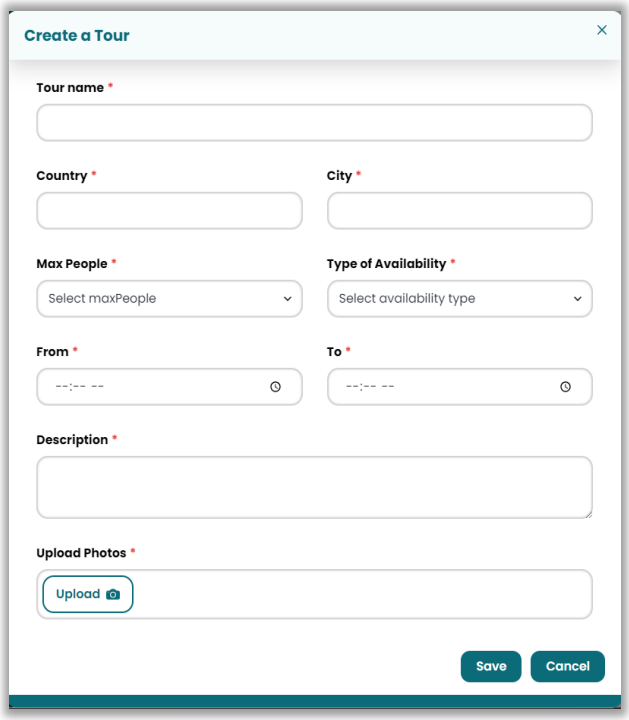


Figure 23: Create Tour Form

3. Editing and Deleting Tours:

- Each tour card has 'Edit' and 'Delete' options, allowing tour author to modify details of their tours or remove them entirely if they are no longer available or relevant. *(see figure 24)*

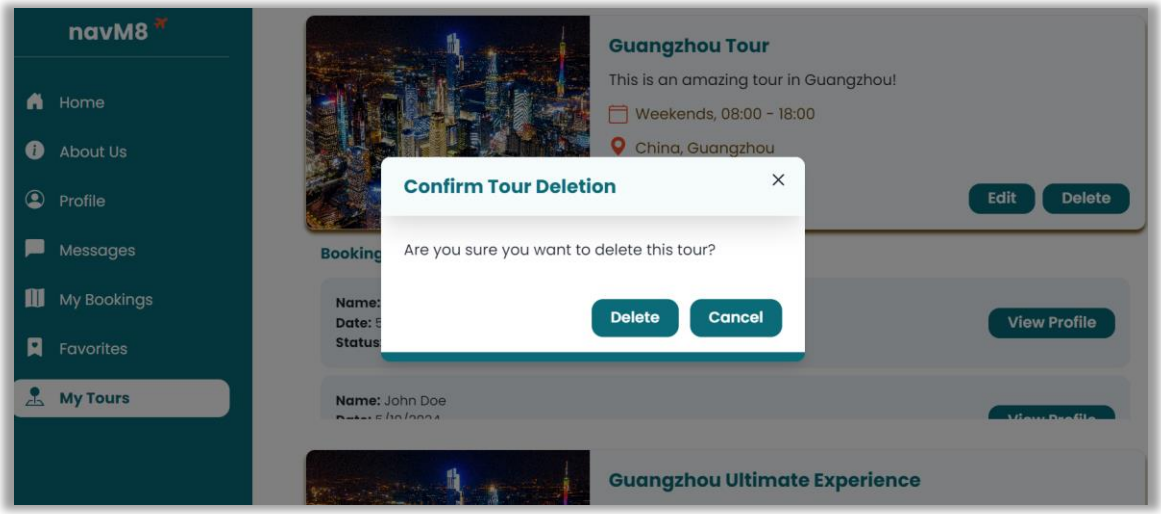


Figure 24: Delete Tour Confirmation Modal Window

4. **Booking Management:**
    - Below each tour card, there is a section for managing bookings. It displays the names of individuals who have booked the tour, providing the guide with a clear view of who will be attending. *(see figure 25)*

Figure 25: Bookings Management

5. **Approval/Cancellation of Bookings:**
    - For bookings marked as 'Pending', there is an option to approve or decline the bookings. This feature helps in managing who attends the tour, ensuring a controlled and organized tour experience. *(see figure 26)*

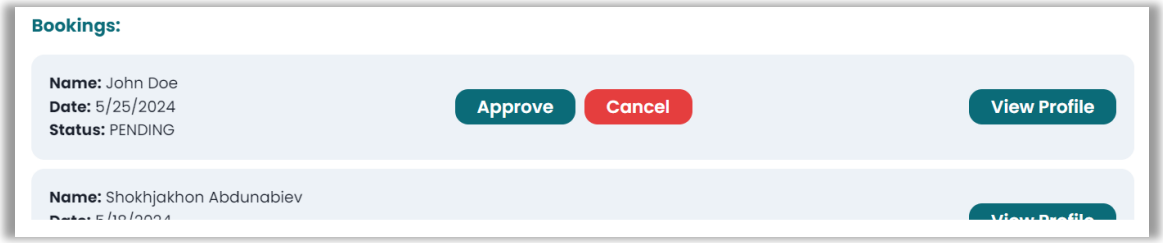


Figure 26: Booking Approval/Cancellation

2.3.11 User Details Page

The User Details Page on navM8 provides comprehensive information about a specific user, enhancing the platform's social connectivity and helping users learn more about their tour guides or fellow travelers. This page can be accessed through various points within the application, such as the Tour Details Page and the Messages Page. *(see figure 27)*

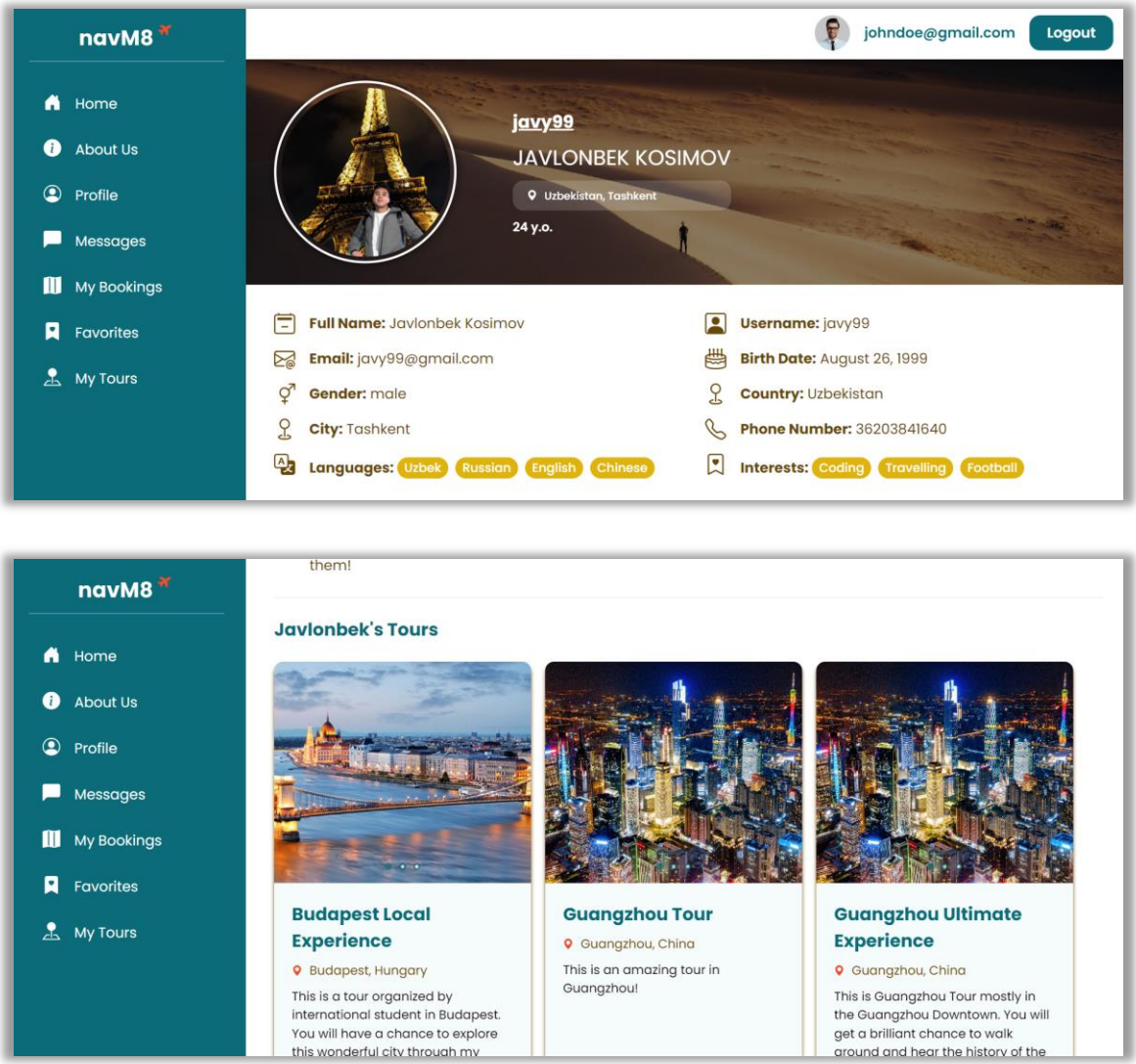


Figure 27: User Details Page

Navigating to User Details Page:

- **From Tour Details Page:** On the Tour Details Page, the author's name is clickable. Hovering over the name displays a tooltip with the text "View [Author's Name] Profile." Clicking on the name redirects the user to the User Details Page. *(see figure 28)*

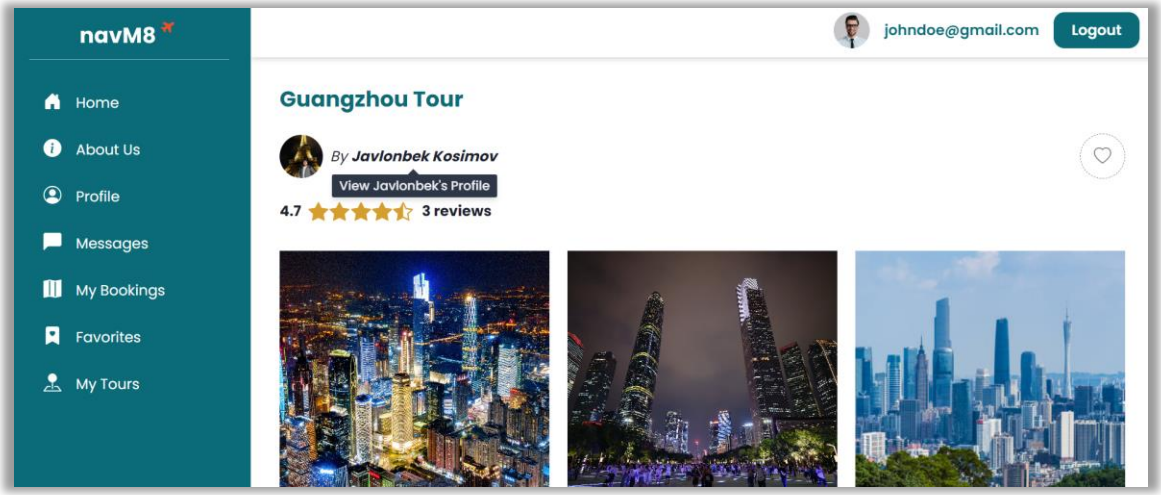


Figure 28: Navigate from Tour Details Page

2. Messages Page:

- In the Messages Page, while chatting with other users, their names are clickable within the chat interface. Hovering over the name displays a tooltip with the text "View [User's Name] Profile." Clicking on the name opens the User Details Page. *(see figure 29)*

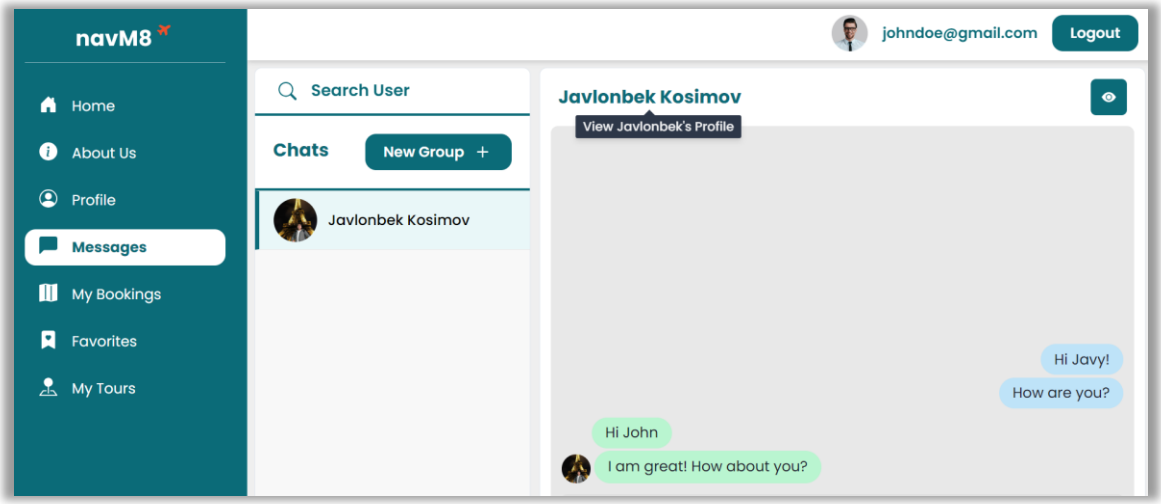


Figure 29: Navigate from Chat

Page Components

1. Personal Information:

- Displays the user's full name, profile picture, email, gender, city, country, and birth date.
- Languages spoken by the user are also listed, providing insights into their communication capabilities.
- A section dedicated to the user's interests, which can include hobbies and preferred activities like coding, traveling, or football.
- Shows the user's phone number for direct contact if needed.

2. Tour Information:

- Lists all the tours offered by the user. Each tour is presented in a card format with essential details like tour name, location, and description.
- Allows users to directly access these tours by clicking on the cards.

2.4 Error Handling and Validation

This section covers the common errors users may encounter while interacting with navM8, along with explanations and screenshots demonstrating the system's responses to these errors. The goal is to help users quickly identify and resolve issues to enhance their experience using the application.

2.4.1 Registration Errors

Error Type: Short Username:

- **Cause:** Username entered is fewer than 3 characters.
- **System Response**: The system displays an error message: "Username must be at least 3 characters long." *(see figure 30)*

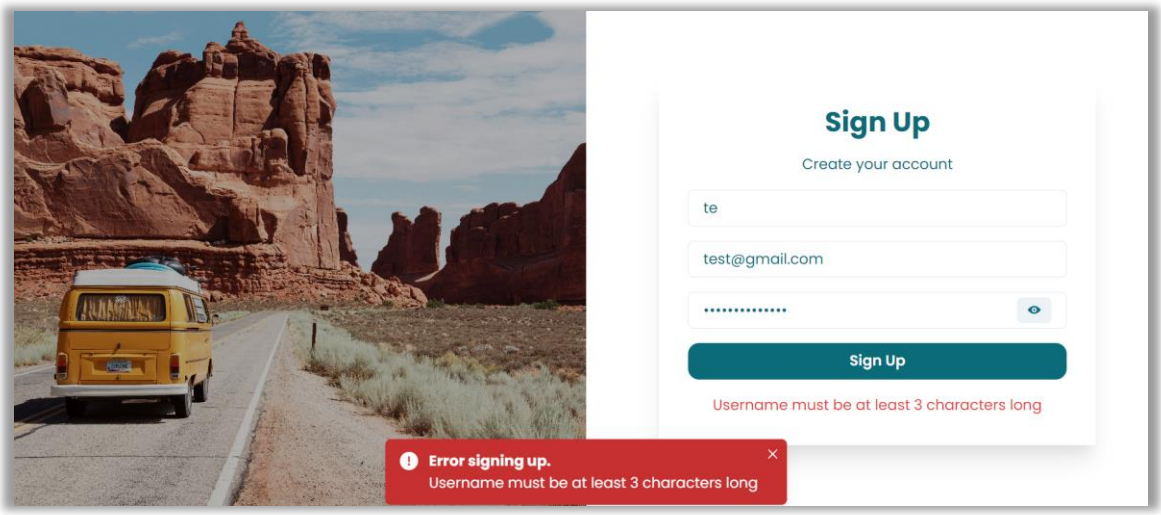


Figure 30: Short Username Error

Error Type: **Invalid Email Format**:

- **Cause:** Email entered does not match standard email formatting.
- **System Response:** The system displays an error message: "Email is not valid." *(see figure 31)*

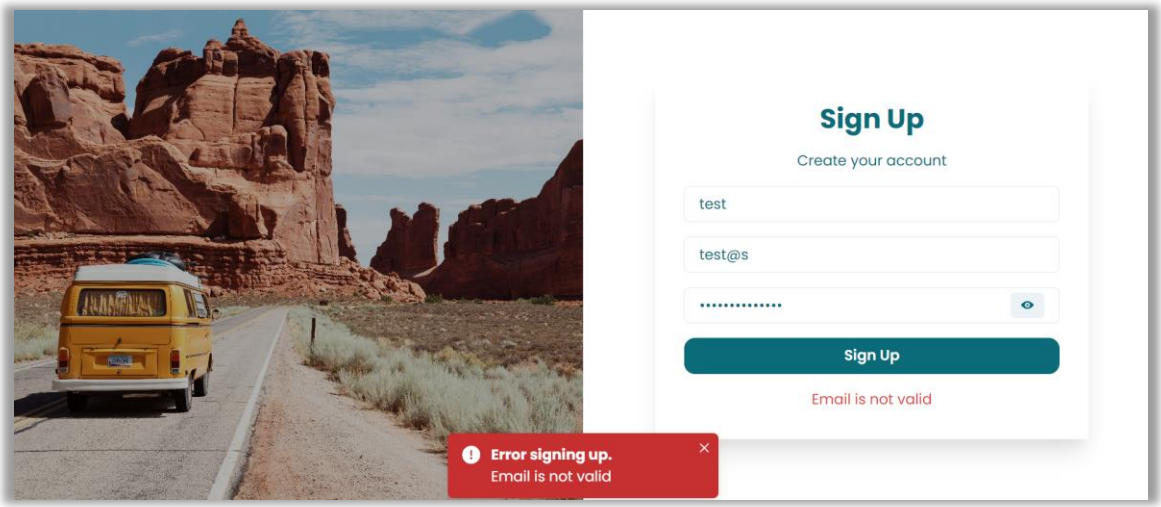


Figure 31: Invalid Email

Error Type: **Not Strong Password:**

- **Cause:** Password entered is not strong enough.
- **System Response:** The system displays an error message: "Password is not strong enough." *(see figure 32)*

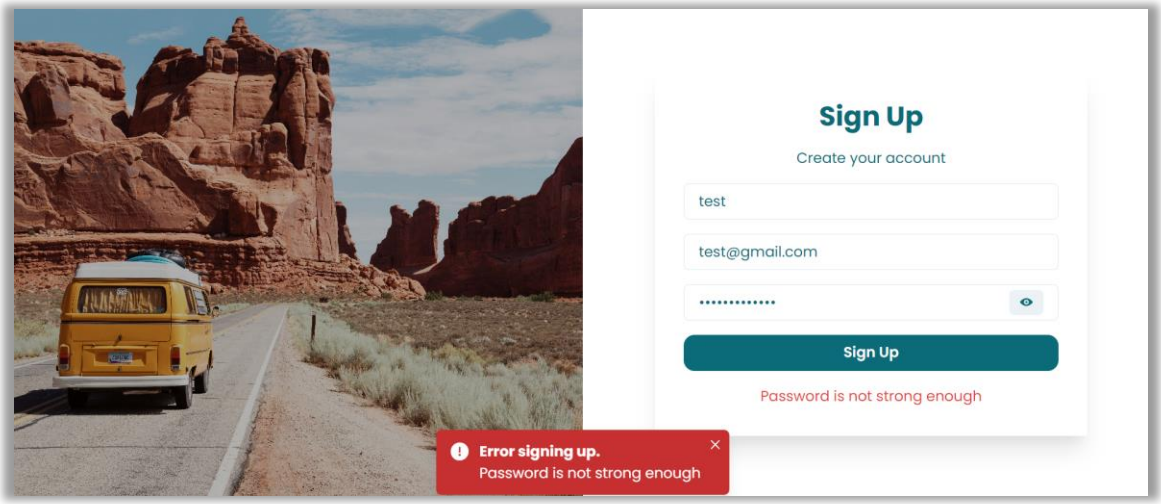


Figure 32: Not Strong Password

2.4.2 Login Errors

Error Type: **Incorrect Email:**

- **Cause:** Email entered does not exist in database.
- **System Response:** The system displays an error message: "Incorrect Email." *(see figure 33)*

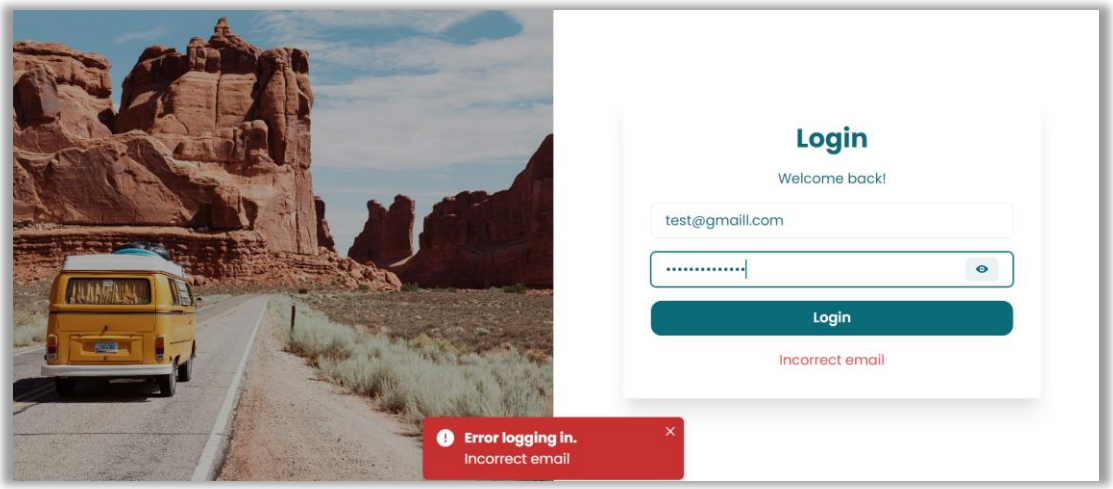


Figure 33: Incorrect Email

Error Type: **Incorrect Password:**

- **Cause:** Password entered is incorrect
- **System Response:** The system displays an error message: "Incorrect password." *(see figure 34)*

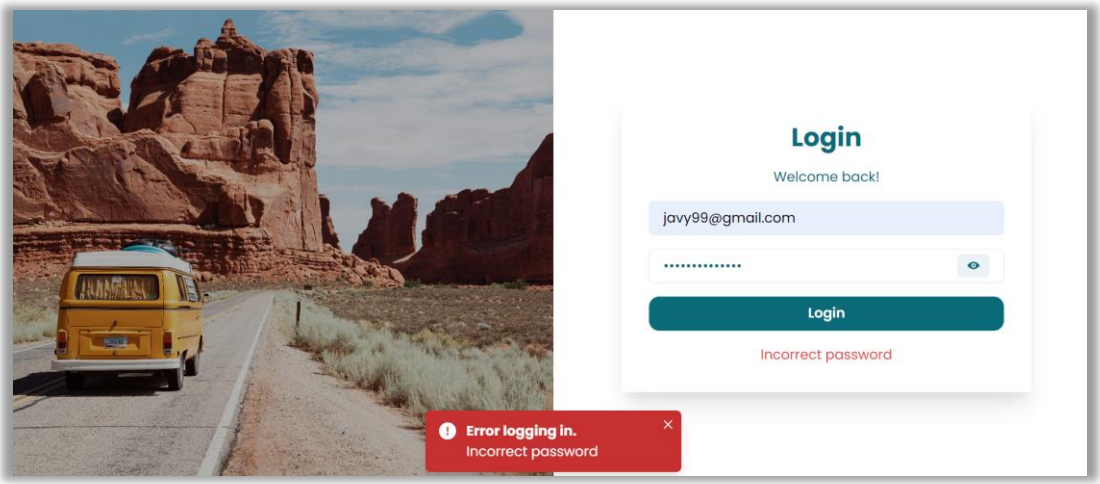


Figure 34: Incorrect Password

2.4.3 Tour Creation Errors

Error Type: Mandatory Fields Left Blank:

- **Cause:** One or more required fields in the tour creation form are left empty.
- **System Response:** The system displays an error message: "Tour validation failed: “Message.” *(see figure 35)*

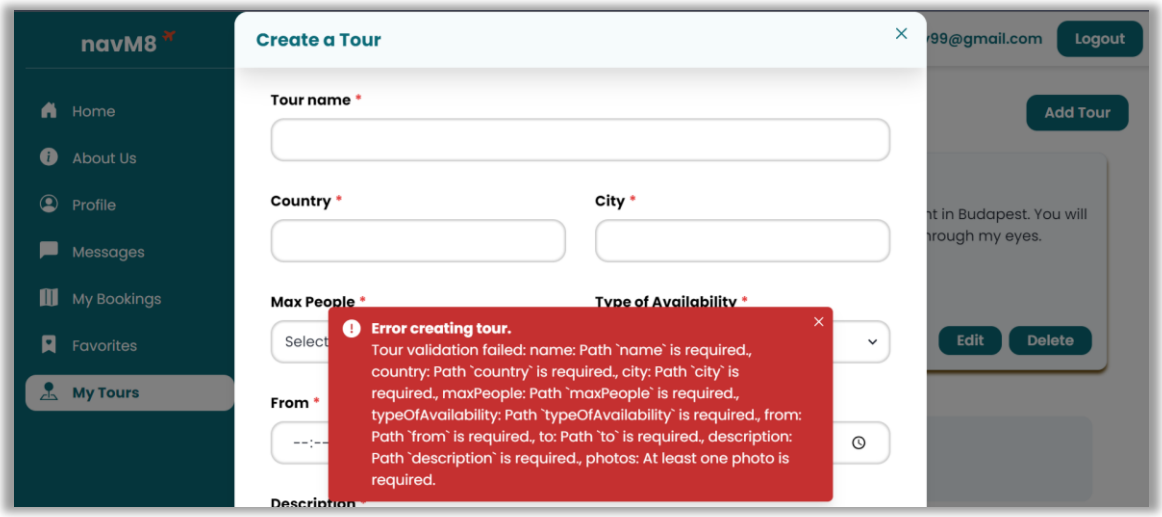


Figure 35: Empty Form

2.4.4 Tour Booking Errors

Error Type: Author Booking Own Tour:

- **Cause:** The user attempting to book the tour is the same user who created the tour.
- **System Response:** The system displays an error message: "Users cannot book their own tours." *(see figure 36)*

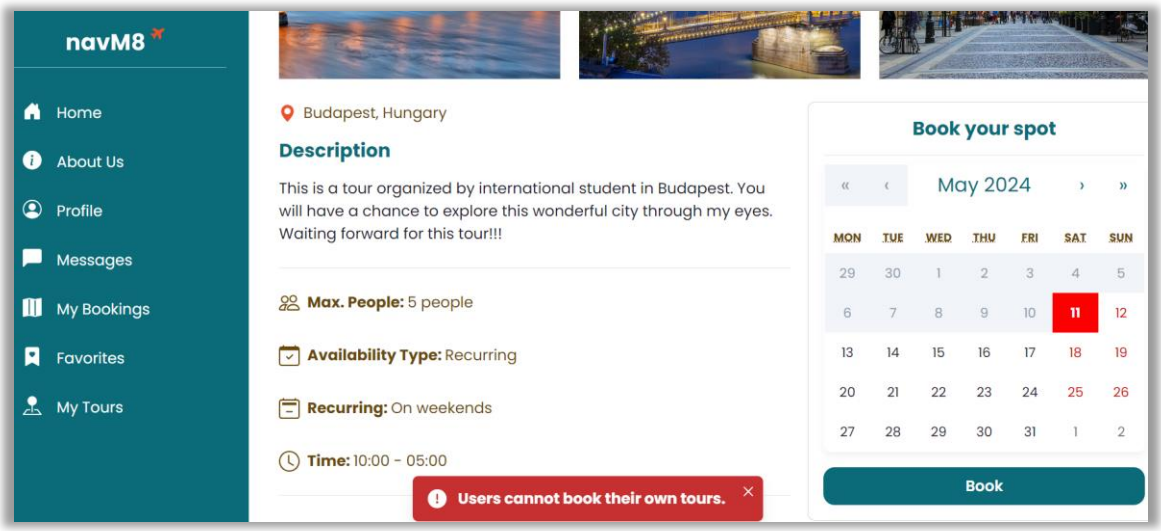


Figure 36: Author Own Tour Book

Error Type: Invalid Tour Date:

- **Cause:** The date selected for booking the tour contradicts the availability type specified for the tour (e.g., attempting to book a weekday-only tour on a weekend).
- **System Response:** The system displays an error message: "Invalid booking date for the selected tour's availability." *(see figure 37)*

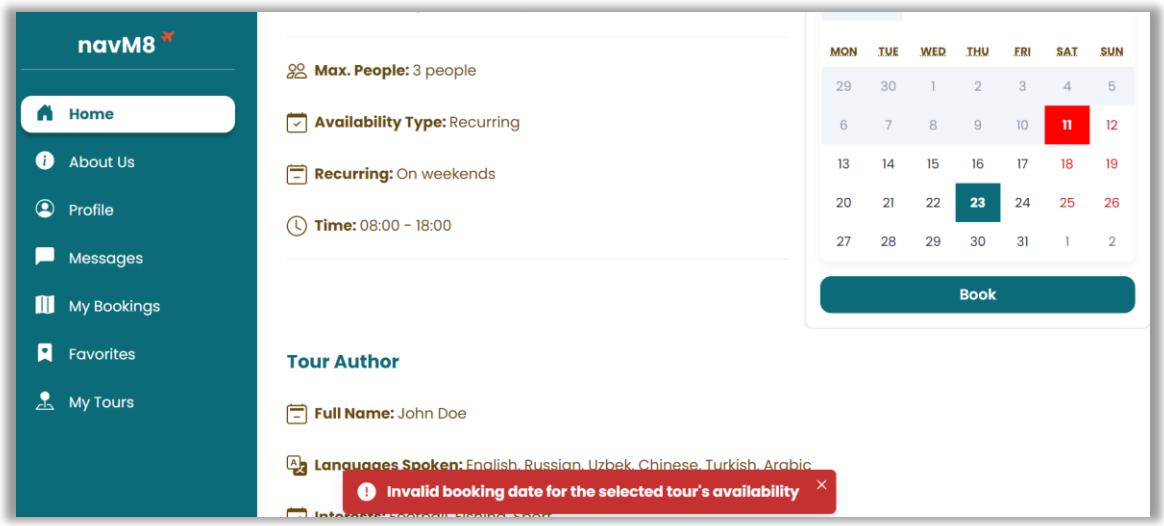


Figure 37: Invalid Date

Error Type: Incomplete User Profile:

- **Cause:** The user attempting to book the tour has not completed their profile.
- **System Response:** The system displays a warning message: "Please complete your profile to book this tour. Redirecting to your profile page..." and then redirects the user to their profile page after 3 seconds. *(see figure 38)*

Figure 38: Incomplete User Profile

Chapter 3

3. Developer Documentation

This chapter will offer insight into the project's planning, design, development, and implementation structure, as well as its extensibility. It will provide developers with a comprehensive understanding of the technology stack and functionality, covering the initial planning and design phases, through to development. This resource enables contributions, enhancements, and future development leveraging the project's components.

3.1 Requirements

Figma Design: Detailed mockups and design specifications created in Figma are used to guide the UI/UX design of the application. *(see figure 39 and figure 40)*

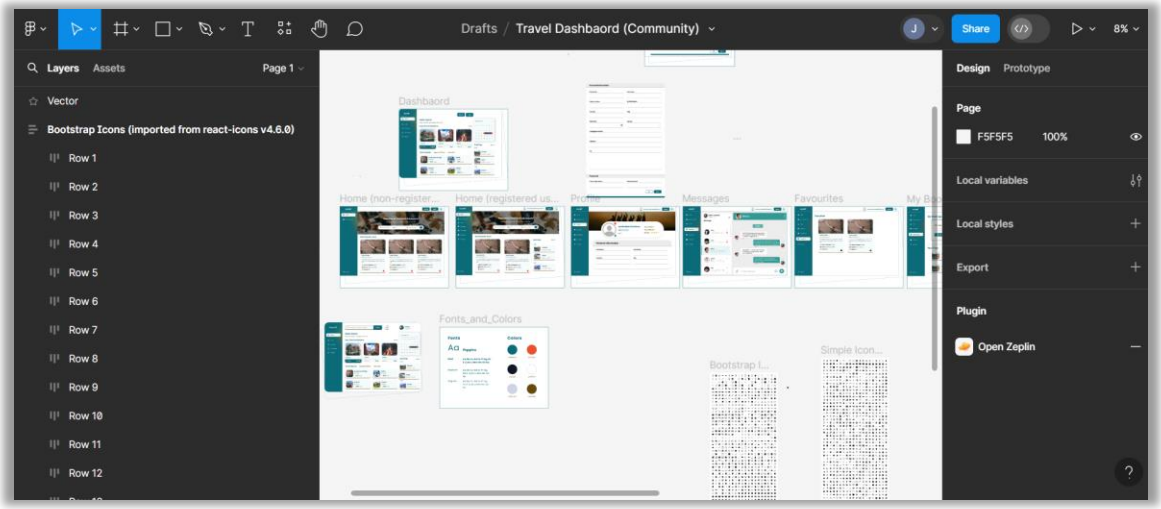


Figure 39: Figma Overvie

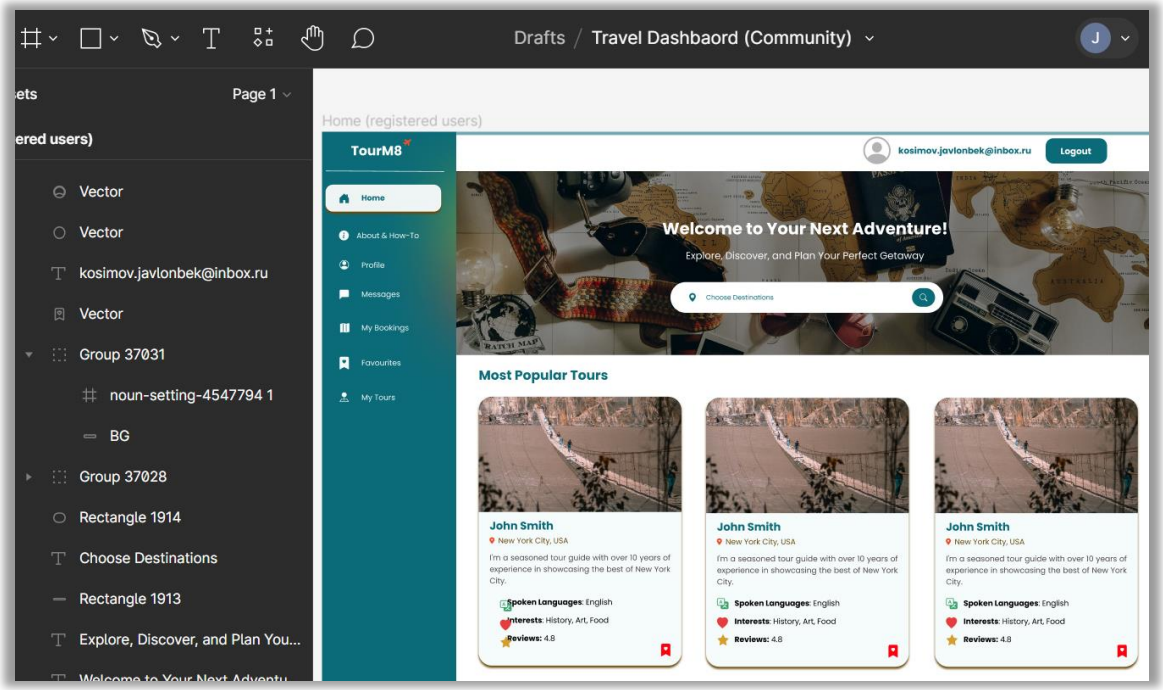


Figure 40: Figma Home Page Design

Functional Requirements:

1. User Registration:

- **User Story:** As a user, I want to register for an account so that I can access the platform's features.
- **Details:** Include fields like username, email, and password during registration.

2. User Login:

- **User Story:** As a user, I want to log into my account so that I can manage my profile and bookings.
- **Details:** Authentication should be implemented using JWT or similar methods for secure sessions.

3. Tour Booking:

- **User Story:** As a user, I want to book a tour so that I can confirm my participation in the event.
- **Details:** Provide a booking confirmation and status update (e.g., pending, confirmed).

4. Review Submission:

- **User Story:** As a user, I want to submit a review for a tour so that I can share my experience with others.
- **Details:** Reviews should include a rating and a comment section.

5. Chat Feature:

- **User Story:** As a user, I want to chat with other users and tour guides in real-time.
- **Details:** Implement real-time messaging using socket.io.

6. Profile Management:

- **User Story:** As a user, I want to view and update my personal information, including my profile picture, bio, contact information, and other details.
- **Details:** Users should be able to update their profile information from their profile page.

7. Tour Management:

- **User Story:** As a user, I want to manage the tours I have created, including adding, editing, and deleting tours.
- **Details:** Provide functionalities for creating new tours, modifying existing tours, and removing tours. Users should also be able to view bookings for their tours and manage them.

8. Favorites:

- **User Story:** As a user, I want to add tours to my list of favorites so that I can easily find them later.
- **Details:** Users should be able to mark tours as favorites and view their list of favorite tours.

9. Search Functionality:

- **User Story:** As a user, I want to search for tours by city to find relevant tours easily.
- **Details:** Include a search bar on the homepage to search for tours based on city or other criteria.

Non-functional Requirements:

1. Usability:

- The application should be easy to navigate with a clean and responsive UI.
- Ensure accessibility standards are met.

2. Performance:

- The application should load quickly and handle multiple requests efficiently.
- Optimize the application using tools like Vite for fast builds and React's virtual DOM for efficient UI updates.

3. Security:

- User data should be securely handled, with proper authentication and authorization mechanisms in place.
- Implement HTTPS, secure cookies, and appropriate data validation.

4. Scalability:

- The application should be able to scale to accommodate a growing number of users and data.
- Use scalable technologies like MongoDB for the database and Cloudinary for media management.

Use-Case Diagram

Unauthorized User:

This diagram depicts the interactions available to an unauthorized user within the navM8 application. An unauthorized user has limited access to the system and can perform the following actions. *(see figure 41)*

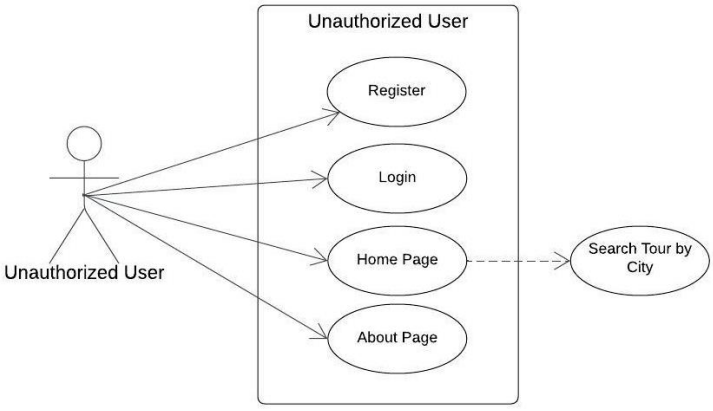


Figure 41: Use-Case Diagram of Unauthorized User

Pages:

1. **Register:** Create a new account on the platform.
2. **Login:** Authenticate using existing credentials to gain access to more features.
3. **Home Page:** View the homepage, including tour cards and search functionality. Unauthorized users can:
    - **Search Tour by City:** Use the search bar to find tours by city.
    - View basic tour information. Clicking on tour cards prompts a login/register modal.
4. **About Page:** Access information about the "navM8" application, its purpose, and its features.

Authorized User:

This diagram illustrates the interactions available to an authorized user within the navM8 application. Once logged in, a user can access various pages and perform specific actions related to managing their profile, tours, bookings, and interactions with other users. *(see figure 42)*

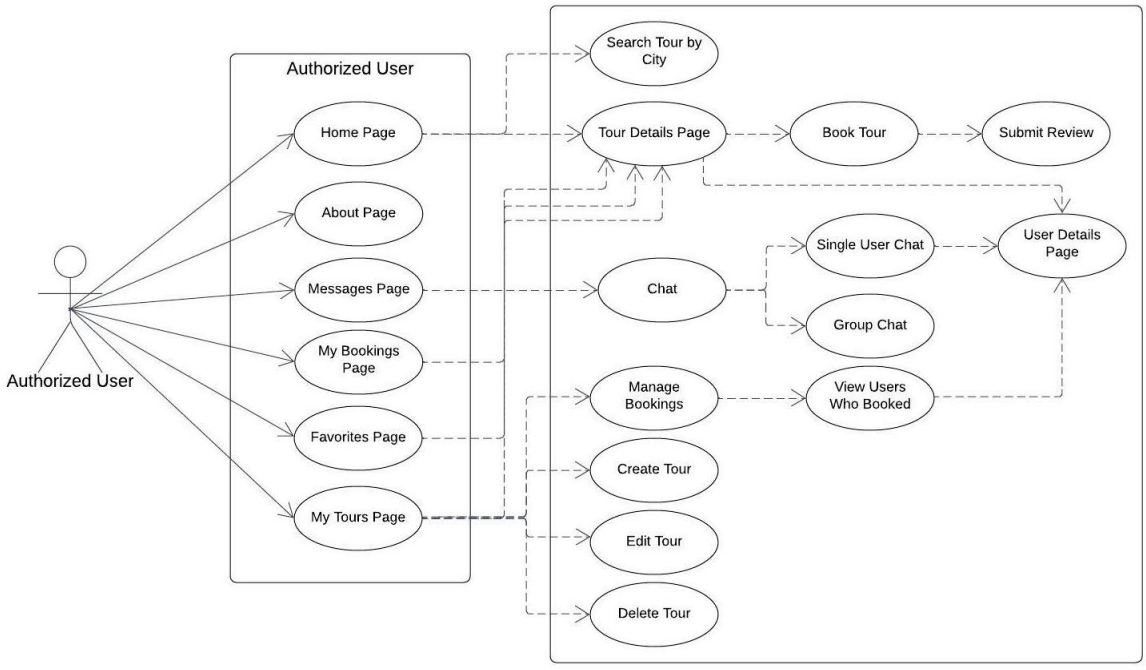


Figure 42: Use-Case Diagram of Authorized User

Pages:

1. **Home Page:** View personalized content, recommended tours, and tour cards.
    - **Search Tour by City:** Use the search bar to find tours by city.
    - View detailed tour information by clicking on tour cards.
2. **About Page:** Access detailed information about the platform.
3. **Profile Page:** View and update personal information, including profile picture, bio, contact information, and other details.
4. **Messages Page:** Engage in conversations through single user or group chats.
5. **My Bookings Page:** Manage and view the status of tour bookings.
6. **Favorites Page:** Access a list of favorite tours.
7. **My Tours Page:** Manage tours created by the user, including:
    - **Create Tour**: Add new tours to the app.
    - **Edit Tour**: Modify details of existing tours.
    - **Delete Tour:** Remove tours from the app.
    - **Manage Bookings:** View and manage users who have booked the user's tour.
    - **View Users Who Booked:** See details of users who booked their tours.
8. **User Details Page:** View detailed information about other users, such as their profile and tours they have created or booked.
9. **Tour Details Page:** Access detailed information about tours.
    - **Book Tour:** Reserve a spot in a selected tour.
    - **Submit Review:** Provide feedback and ratings for a tour.

Summary

These diagrams and descriptions provide a comprehensive overview of the user interactions within the navM8 application, distinguishing between the capabilities of unauthorized and authorized users. The detailed breakdown helps to understand the application's functionality and guides further development and enhancements.

3.2 Technologies

This section outlines the technology stack used to develop navM8, providing insights into the tools and frameworks that support the functionality of web application.

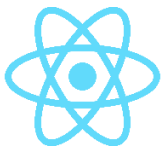
MongoDB

MongoDB is the primary database for navM8. It's chosen for its ability to handle unstructured data and its scalability. MongoDB efficiently manages user data, tour information, and reviews, which are crucial for the dynamic content on the platform. Its schema-less design is well-suited to manage the diverse data generated by user interactions.

Express.js

Express.js is used as the backend framework, integrated with Node.js. It manages requests and responses between the frontend and the database. Express streamlines the routing process and boosts the API's speed and reliability, vital for the real-time processing needs of navM8.

React JS



React powers the frontend, creating an interactive and responsive user interface. Its component-based structure is excellent for building complex interfaces efficiently. React's effective state management ensures that the user interface updates smoothly when data changes.

Node.js

Node.js is the runtime environment for backend. It's chosen for its ability to handle asynchronous tasks, such as database communication and processing multiple user requests at once. Node.js's non-blocking, event-driven architecture supports a high-performance backend.

Additional Technologies:

- **Socket.io:** Used for real-time communication between web clients and servers, enhancing the messaging features by allowing updates without page reloads.

- **Vite:** Vite speeds up frontend development and build optimization.

- **TypeScript:** TypeScript adds strong typing to the code, helping reduce bugs and improving quality.

- **Chakra UI:** A React UI library that helps quickly design a clean and accessible interface. It supports responsive design from the start, essential for ensuring "navM8" works well on different devices.

3.3 Installation

This section outlines the necessary steps to set up the development environment for navM8 and run the application locally from the provided source code. The setup process is designed to be straightforward, enabling developers to quickly get the system operational.

Development Environment Setup

The development environment for navM8 requires specific tools and versions to ensure compatibility and optimal performance.

Prerequisites:

- **Node.js** (version 18.18.0 or later)
- **npm** (version 9.8.1 or later)

Directory Structure:

- The main project folder, **navM8**, contains two subdirectories: **frontend** and **backend**.

Environment Variables:

Configuration for both the backend and frontend is managed through **.env** files.

Frontend Installation:

1. From the command line, navigate to the frontend directory.
2. Install the required dependencies by executing: `npm install`.
3. Start the development server by running: `npm run dev` *(see figure 43)*. This will launch the frontend on http://localhost:3001/.

Backend Installation:

1. From the command line, navigate to the backend directory.
2. Install the necessary dependencies by running: `npm install`.

3. Start the backend server by executing: **`npm start`** *(see figure 44)*. This script uses concurrently to run TypeScript compilation and the server with nodemon, allowing real-time updates.

Running the Application:

- Ensure that both the frontend and backend servers are running.
- The frontend can be accessed through the URL: http://localhost:3001/

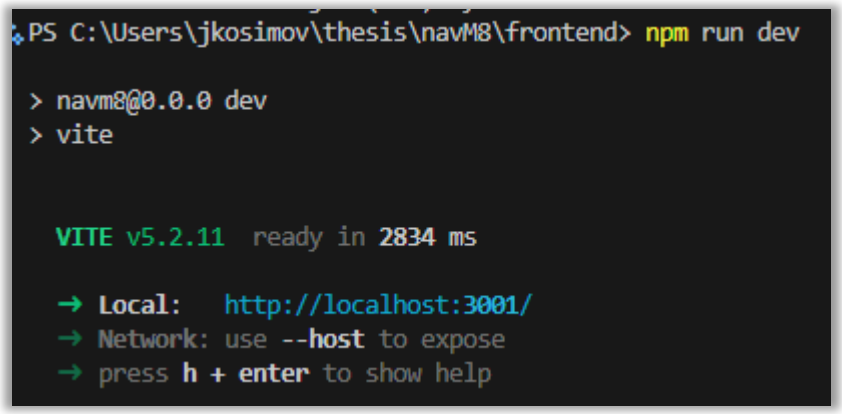
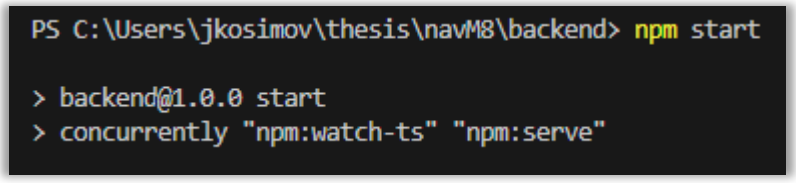


Figure 43: Running Frontend

- Backend services are running on port 3000, as indicated by the server logs when started:



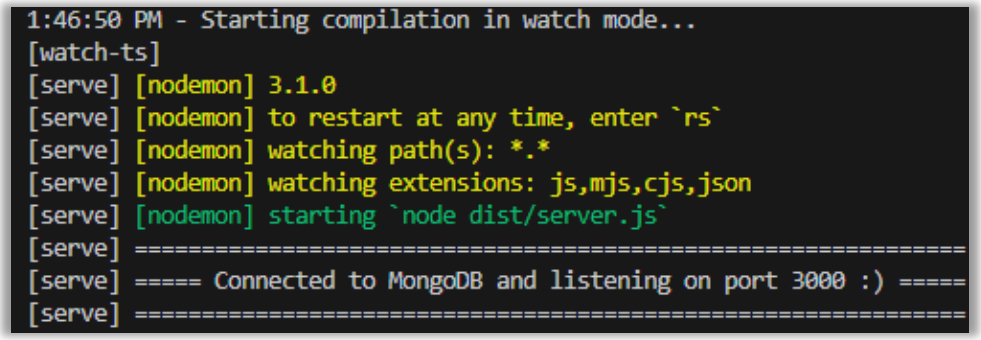


Figure 44: Running Backend

Additional Tools:

- For code linting and formatting, ESLint and Prettier are configured. To run linting, use **`npm run lint`** and for automatic fixes, use **`npm run lint:fix`**.
- TypeScript is used for type-checking. To compile TypeScript files, use **`npm run build`**.

This setup ensures a comprehensive development environment, replicating the application's operational settings and facilitating full functionality testing and development.

3.4 Architecture

Overview

The architecture section provides a comprehensive overview of application, utilizing the MERN stack. The structure is clearly divided into different layers: Frontend, Backend, and Database, with a supplementary diagram for user interaction flows. *(see figure 45)*

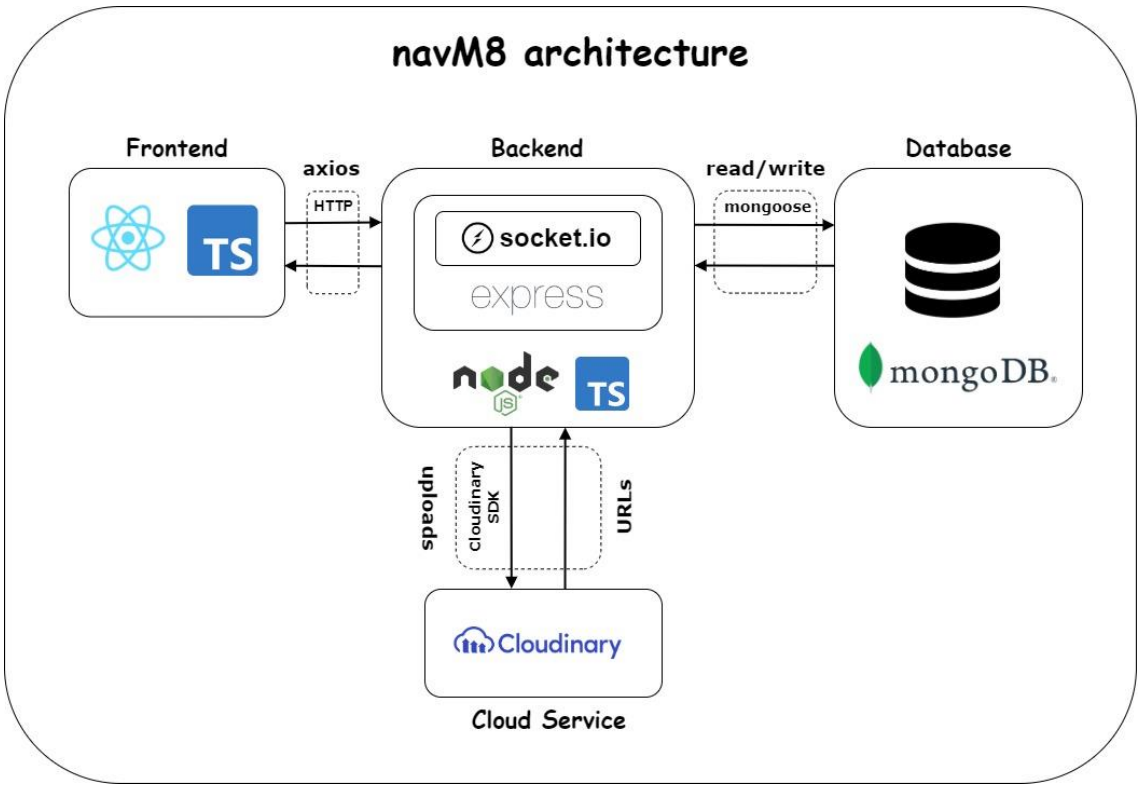


Figure 45: navM8 Architecture

Architecture Overview

1. Frontend (React JS)

- **Components:** React with TypeScript.
- **Functionality**: The frontend manages the user interface, leveraging React for building a component-based UI and TypeScript for type safety and scalability. Axios is used for making HTTP requests, enabling communication with the backend.
- **Data Flow**:
    - **Axios (HTTP):** Utilizes HTTP requests to communicate with backend services managed by Node.js and Express.js, handling data such as user profiles, bookings, and chat messages.

2. Backend (Node.js/Express.js)

- **Components:** Node.js with Express.js, Socket.io, Typescript.
- **Functionality:**
    - **API Handling:** The Express framework on Node.js with TypeScript manages API endpoints for functionalities such as user authentication, profile management, tour bookings, and reviews.
    - **Real-time Communication:** Socket.io enables real-time bi-directional communication between clients and servers, allowing for features like real-time chat without needing to refresh the web page.
    - **Media Handling:** Cloudinary is integrated for the upload and management of images, handling requests to store and retrieve media efficiently.
    - **Data Flow:**
        - **Mongoose (Read/Write):** Interactions with the MongoDB database to fetch or modify data are managed via Mongoose, an ODM (Object Data Modeling) library for MongoDB and Node.js.
        - **Cloudinary:** Direct interaction with the Cloudinary API for uploading and managing images, receiving URLs for stored media which are used within the app.

3. Database (MongoDB)

- **Component:** MongoDB
- **Functionality:** MongoDB, a NoSQL database, is used for storing various data types including user profiles, tour details, bookings, and chat histories. It is chosen for its flexibility and scalability in handling unstructured data.
- **Data Flow:**
    - **Direct Interaction with Backend:** The backend, via Mongoose, directly interacts with MongoDB to perform CRUD (Create, Read, Update, Delete) operations.

4. Cloud Service

- **Component:** Cloudinary
- **Functionality:** Acts as a cloud-based service for storing and managing images. It provides APIs to upload images directly from the frontend or backend and retrieve them for use in the application.
- **Data Flow:**
    - **Interaction with Backend and Frontend:** Both the backend and frontend can interact with Cloudinary to upload or retrieve media, facilitating dynamic content management.

3.4.1 Database Layer

Overview

The Database Layer in navM8 leverages MongoDB, a flexible NoSQL database, to store and manage a wide array of data including user profiles, tour details, bookings, reviews, and chat messages. This layer is critical for the robust handling of diverse and dynamic data structures, optimized for high-performance queries and scalability.

Quality Description:

- **MongoDB:** Offers a schema-less architecture that is ideal for handling the varied and evolving data needs of "navM8". This flexibility allows easy modifications and expansions of data structures, supporting the application’s growth and new

feature integrations.

- **Mongoose:** Acts as the Object Data Modeling (ODM) tool that provides a structured way to model the application data in MongoDB. It simplifies database interactions with built-in type casting, validation, and query building, enhancing data integrity and operational efficiency.

Schema Details

Below is a table summarizing the key aspects of each schema in the MongoDB database:

Collection	Attributes	Description	Relationships
User	_id, username, email, password, profilePictureURL, firstName, lastName, phoneNumber, country, city, birthDate, gender, languagesSpoken, interests, bio, favoriteTours	Stores comprehensive user-related data including authentication and personal preferences.	Links to **Tour** via favoriteTours.
Tour	name, country, city, maxPeople, typeOfAvailability, availability, date, from, to, description, photos, author, reviewCount	Details about tours including logistics and descriptions.	Linked from **Booking**, **Review**. Author linked to **User**.
Booking	tour, userId, date, status	Manages tour reservations with status tracking.	Links to **Tour** and **User**.
Chat	chatName, isGroupChat, users, latestMessage,	Manages chat sessions and tracks	**Users** and latestMessage link

	groupAdmin	messages.	to **User** and **Message** respectively.
Message	sender, content, chat	Stores messages within chat sessions.	Links to **Chat** and **User**
Review	tour, user, rating, comment	Captures user feedback on tours.	Links to **Tour** and **User**.

Table 1: MongoDB Schemas

Entity Relationship Diagram (ERD)

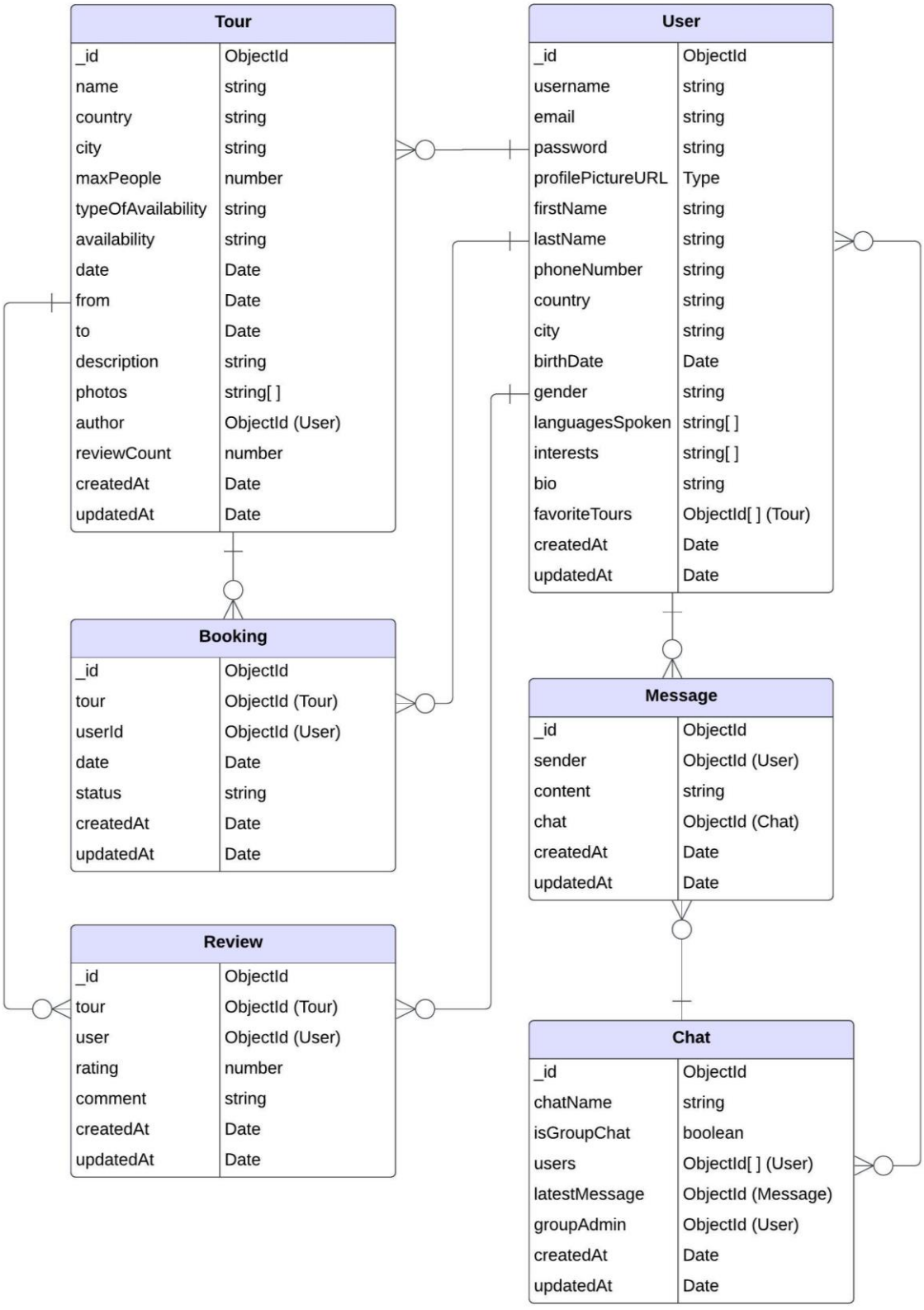


Figure 46: Entity Relationship Diagram

Entity Relationship Diagram (ERD) Description

1. User Entity:

- **Description**: Represents the users of the navM8 application.
- **Attributes:** Includes basic authentication and profile and a link to the user's profile image (profilePictureURL).
- **Profile Image Storage:** The profilePictureURL holds a secure URL pointing to an image stored on Cloudinary.
- **Relationships:**
    - **Tours (OneToMany):** A user can create multiple tours, represented by the author field in the Tour entity.
    - **Bookings (OneToMany):** A user can make multiple bookings.
    - **Reviews (OneToMany):** A user can write multiple reviews.
    - **Messages (OneToMany):** A user can send multiple messages within chat sessions.
    - **Chats (ManyToMany):** Users can belong to multiple chats, and each chat can involve multiple users.

2. Tour Entity:

- **Description:** Represents the tours available in the application.
- **Attributes:** Contains details about the tour and an array of photo URLs (photos)
- **Photo Storage:** The photos attribute stores an array of URLs pointing to images hosted on Cloudinary, illustrating various aspects of the tour.
- **Relationships:**
    - **User (ManyToOne):** Each tour is authored by a single user.
    - **Bookings (OneToMany):** Each tour can have multiple bookings.
    - **Reviews (OneToMany):** Each tour can be reviewed by multiple users.

3. Booking Entity:

- **Description:** Manages the reservations users make for tours.
- **Attributes:** Includes references to the tour and user, the date of the booking, and the status of the booking (e.g., pending, confirmed, completed).

- **Relationships:**
    - **Tour (ManyToOne):** Each booking is associated with a single tour.
    - **User (ManyToOne):** Each booking is made by a single user.

4. Review Entity:

- **Description:** Holds reviews that users leave for tours.
- **Attributes:** Consists of a rating, a textual comment, and links to the user who wrote the review and the tour that was reviewed.
- **Relationships:**
    - **Tour (ManyToOne):** Each review is associated with a single tour.
    - **User (ManyToOne):** Each review is written by a single user.

5. Message Entity:

- **Description:** Represents messages sent by users within chat sessions.
- **Attributes:** Includes the content of the message, the sender, and the chat session in which the message was sent.
- **Relationships:**
    - **Chat (ManyToOne):** Each message is part of one chat session.
    - **User (ManyToOne):** Each message is sent by a single user.

6. Chat Entity:

- **Description:** Manages chat sessions between users.
- **Attributes:** Contains the name of the chat, a boolean indicating if it's a group chat, and a reference to the latest message sent within the chat.
- **Relationships:**
    - **Messages (OneToMany):** A chat contains multiple messages.
    - **Users (ManyToMany):** A chat can include multiple users, and each user can be part of multiple chats.

3.4.2 Backend Layer

Overview

The Server Layer is pivotal in the navM8 architecture, bridging the frontend and the database while ensuring efficient data processing and API management. It is built using Express.js on a Node.js runtime, with TypeScript enhancing development by adding strong typing.

Quality Description

- **TypeScript Integration:** Utilizes TypeScript for improved development experiences with static typing, which helps in catching errors early in the development process, making the codebase more robust and easier to manage.
- **Middleware Integration:** Implements essential middleware for authentication, data validation, and CORS setup, ensuring secure and flexible interactions across different origins.
- **Business Logic Execution:** Controllers written in TypeScript handle specific business tasks, providing clarity and reducing runtime errors.
- **Data Management:** Uses Mongoose for structured data interaction with MongoDB, ensuring robust data handling.

Key Functionalities

Middleware Integration:

- **Authentication (requireAuth):** Secures routes by verifying user tokens; essential for maintaining secure user data access.

```
const requireAuth = async (req: Request, res: Response, next:
NextFunction) => {
  const token = req.cookies.token
  const refreshToken = req.cookies.refreshToken
  if (!token && !refreshToken) {
    return res.status(401).json({ error: 'Authorization token required'
})
  }
```

```
  try {
    const JWT_SECRET = process.env.JWT_SECRET as Secret
    if (!JWT_SECRET) {
      throw new Error('JWT secret is not defined.')
    }
    let decoded: { _id: string }
    try {
      decoded = jwt.verify(token, JWT_SECRET) as { _id: string }
    } catch (error) {
      if (error instanceof jwt.TokenExpiredError && refreshToken) {
        const refreshDecoded = jwt.verify(refreshToken, JWT_SECRET) as {
          _id: string
        }
        const newToken = createToken(refreshDecoded._id.toString(), '3d')
        res.cookie('token', newToken, {
          httpOnly: true,
          secure: true,
          maxAge: 3 * 24 * 60 * 60 * 1000,
        })
        decoded = jwt.verify(newToken, JWT_SECRET) as { _id: string }
      } else {
        throw error
      }
    }
    const user = await User.findById(decoded._id).select('-password')
    if (!user) {
      return res.status(404).json({ error: 'User not found' })
    }
    req.user = user
    next()
  } catch (error) {
    return res.status(401).json({ error: 'Request is not authorized' })
  }
}
```

- **CORS Configuration:** Allows secure, cross-origin requests from designated origins, crucial for client-server interactions.

```
app.use(
  cors({
    origin: 'http://localhost:3001',
    methods: ['GET', 'POST', 'PUT', 'PATCH', 'DELETE'],
    credentials: true,
  }),
)
```

- **Error Handling:** Manages application errors and undefined routes, ensuring graceful degradation and user feedback on failures.

```
const notFound = (req: Request, res: Response, next: NextFunction) => {
  const error = new Error(`Not found - ${req.originalUrl}`)
  res.status(404)
  next(error)
}
```

Real-time Interaction with Socket.IO

Socket.IO is integrated for real-time communications, crucial for features like live chat and instant notifications.

```
const server = http.createServer(app)
const io = new SocketIOServer(server, {
  pingTimeout: 60000,
  cors: {
    origin: 'http://localhost:3001',
  },
})
io.on('connection', (socket) => {
  console.log('a user connected:', socket.id)

  socket.on('setup', (userData) => {
    socket.join(userData._id)
```

```
    socket.emit('connected')

    socket.on('disconnect', () => {
      console.log('USER DISCONNECTED')
      socket.leave(userData._id)
    })
  })
  socket.on('join chat', (room) => {
    socket.join(room)
    console.log('User Joined Room:', room)
  })

  socket.on('typing', ({ chatId }) =>
    socket.to(chatId).emit('typing', { chatId }),
  )
  socket.on('stop typing', ({ chatId }) =>
    socket.to(chatId).emit('stop typing', { chatId }),
  )
  socket.on('new message', (newMessageReceived) => {
    let chat = newMessageReceived.chat
    if (!chat.users) return console.log('Chat.users not defined')
    chat.users.forEach((user) => {
      if (user._id == newMessageReceived.sender._id) return
      socket.in(user._id).emit('message received', newMessageReceived)
    })
  })
})
```

Database Connection:

Manages the MongoDB connection, ensuring the database is connected before server activities commence.

```
mongoose
    .connect(MONGODB_URL)
    .then(() => {
```

```
      server.listen(BACKEND_PORT, () => {
        console.log('=================================================')
        console.log(
        `Connected to MongoDB and listening on port ${BACKEND_PORT} :) `,
        )
        console.log('=================================================')
      })
    })
    .catch((err: unknown) => {
      if (err instanceof Error) {
        console.error('Error connecting to MongoDB:', err.message)
      } else {
        console.error('Error connecting to MongoDB with an unexpected
type')
      }
    })
```

API Endpoints

This section catalogs all API endpoints, organized by functionality. Each endpoint is detailed in a table that includes its HTTP method, a brief description, parameters it accepts, expected returns, and an "Auth Required" status. The "Auth Required" column clearly indicates whether authentication is necessary:

- **Yes:** The endpoint is secured by the **requireAuth** middleware, requiring user authentication.
- **No:** The endpoint is accessible without authentication.

Authentication

Base URL: localhost:3000/api/auth

Endpoint	Method	Description	Parameters	Returns	Auth Required
/signup	POST	Allows users to register for an account.	username, email, password	User's email, token, and id	No
/login	POST	Allows users to login	email, password	User's email, token, and id	No
/logout	POST	Logs out user	–	Success message	Yes
/refresh_token	GET	Refreshes user token	–	New access token	Yes
/user	GET	Retrieves user information	–	User's email, token, and id	Yes

Table 2: Authentication API

User Management

Base URL: localhost:3000/api/users

Endpoint	Method	Description	Parameters	Returns	Auth Required
/	GET	Retrieves all users	–	Array of user objects	No
/:id	GET	Retrieves user profile	–	User profile data	Yes
/:id	PATCH	Updates user profile	User profile fields to update	Success message	Yes
/:id/photo	POST	Uploads user profile photo	Image file	Success message with photo URL	Yes
/:id/photo	GET	Retrieves user profile photo	–	Image URL	Yes
/:id/photo	DELETE	Deletes user profile photo	–	Success message	Yes
/:id/favoriteTours	POST	Adds a tour to user's favorite tours	tourId	Success message	Yes
/:id/favoriteTours	GET	Retrieves user's favorite	–	Array of favorite tour	Yes

		tours		objects	
/:id/favoriteTours/ :tourId	DELETE	Deletes tour from user's favorite tours	–	Success message	Yes

Table 3: User Management API

Tour Management

Base URL: localhost:3000/api/tours

Endpoint	Method	Description	Parameters	Returns	Auth Required
/	GET	Retrieves all tours	–	Array of tour objects	No
/:id	GET	Retrieves a specific tour	tourId	Tour object	Yes
/mytours	POST	Creates a new tour	Tour details, image files	Success message	Yes
/mytours	GET	Retrieves tours created by the current user	Image file	Array of tour objects	Yes
/mytours/:id	PUT	Updates an existing tour	tourId, tour details	Updated tour object	Yes
/mytours/:id	DELETE	Deletes an existing tour	tourId	Success message	Yes
/user/:id	GET	Retrieves tours created by a specific user.	userId	Array of tour objects	Yes

Table 4: Tour Management API

Booking System

Base URL: localhost:3000/api/bookings

Endpoint	Method	Description	Parameters	Returns	Auth Required
/	POST	Creates a new booking	tourId, date	Created booking object	Yes
/mybookings	GET	Retrieves bookings for the current user	–	Array of booking object	Yes
/tours/:tourId	GET	Retrieves bookings for specific tour	tourId	Array of booking objects	Yes
/:bookingId	PATCH	Updates the status of a booking	bookingId, status	Updated booking object	Yes
/:bookingId	DELETE	Deletes a booking	bookingId	Success message	Yes

Table 5: Booking System API

Reviews

Base URL: localhost:3000/api/reviews

Endpoint	Method	Description	Parameters	Returns	Auth Required
/	POST	Creates a review for a tour	tourId, rating, comment	Created review object	Yes
/:tourId	GET	Retrieves reviews for a specific tour	tourId	Array of review object	Yes

Table 6: Reviews API

Chats

Base URL: localhost:3000/api/chats

Endpoint	Method	Description	Parameters	Returns	Auth Required
/	POST	Access chat	userId	Chat object	Yes
/	GET	Get chats	–	Array of chat objects	Yes
/group	POST	Create group chat	name, users	Group chat object	Yes
/group/rename	PUT	Rename group	chatId, chatName	Updated group chat object	Yes
/group/addUser	PUT	Add user to group	chatId, userId	Updated group chat object	Yes
/group/ removeUser	PUT	Remove user from group	chatId, userId	Updated group chat object	Yes

Table 7: Chats API

Messages

Base URL: localhost:3000/api/messages

Endpoint	Method	Description	Parameters	Returns	Auth Required
/	POST	Send message	content, chatId	Message object	Yes
/:chatId	GET	Get all messages for a chat	chatId	Array of message objects	Yes

Table 8: Messages API

3.4.3 Frontend Layer

Overview:

The client layer of the "navM8" application is developed using React with TypeScript, integrated with Chakra UI for styling and Vite for efficient build management. This layer handles the user interface, offering responsive and interactive components for a seamless user experience.

Quality Description

- **React with TypeScript:** Offers strong typing and component-oriented architecture, improving maintainability and reducing bugs.
- **Performance Optimizations:** Uses Vite for fast cold starts and HMR (Hot Module Replacement), optimizing both development and production builds. React's efficient DOM updating algorithms (Virtual DOM) minimize re-renders, enhancing the application's responsiveness and fluidity.
- **Chakra UI:** Ensures accessibility and responsiveness across all components, maintaining a high standard of user experience regardless of device type. Theming capabilities allow for easy customization and scalability in design.

Key Components:

1. State Management:

This component manages the authentication state using React's context API and hooks (useState, useContext, useReducer). This approach avoids the overhead of external state management libraries, keeping the bundle size small and performance high.

- **AuthContext:** Creates a context for authentication.

```
export const AuthContext = createContext<AuthContextType |
undefined>(undefined)
```

- **authReducer:** A reducer function that handles the state transitions based on the action type (LOGIN or LOGOUT).

```
export const authReducer = (state: AuthState, action: AuthAction):
AuthState => {
  switch (action.type) {
    case 'LOGIN':
      return { user: action.payload}
    case 'LOGOUT':
      return { user: null}
    default:
      return state
  }
}
```

- **AuthContextProvider:** A context provider component that uses the useReducer hook to manage the authentication state and provides it to the rest of the app.

```
export const AuthContextProvider: React.FC<ChildrenProps> = ({ children
}) => {
  const [state, dispatch] = useReducer(authReducer, { user: null })

  useEffect(() => {
    const fetchUser = async () => {
      try {
        const user = await getUser()
        if (user) {
          dispatch({ type: 'LOGIN', payload: user })
        } else {
          dispatch({ type: 'LOGOUT', payload: null })
        }
      } catch (error) {
        dispatch({ type: 'LOGOUT', payload: null })
      }
    }
    fetchUser()
```

```
  }, [])

  return (
    <AuthContext.Provider value={{ state, dispatch }}>
      {children}
    </AuthContext.Provider>
  )
}
```

2. Routing and Navigation:

This component manages the application's routing using React Router. It supports dynamic route loading and lazy-loaded components to decrease initial load time and improve the user experience.

- **App Component:** Defines the main application routes.

```
const App = () => {
  return (
      <BrowserRouter>
        <Routes>
          <Route path="/" element={<HomePage />} />
          <Route path="/signup" element={<Signup />} />
          <Route path="/login" element={<Login />} />
          <Route path="/about" element={<About />} />

          <Route
            path="/profile"
            element={<PrivateRoute component={Profile} />}
          />
          <Route
            path="/chat"
            element={<PrivateRoute component={Messages} />} />
          <Route
            path="/bookings"
            element={<PrivateRoute component={MyBookings} />}
          />
```

```
          <Route
            path="/favorites"
            element={<PrivateRoute component={Favorites} />}
          />
          <Route
            path="/mytours"
            element={<PrivateRoute component={MyTours} />}
          />
          <Route
            path="/tours/:id"
            element={<PrivateRoute component={TourDetails} />}
          />
          <Route
            path="/users/:id"
            element={<PrivateRoute component={UserDetails} />}
          />
        </Routes>
      </BrowserRouter>
  )
}
```

- **PrivateRoute Component:** Protects routes that require authentication by checking the user's state. If the user is not authenticated, it redirects to the login page.

```
const PrivateRoute = ({ component: Component, ...rest }) => {
  const { state } = useAuthContext()
  const { user } = state

    return user ? <Component {...rest} /><Navigate to="/login" replace />
  }
```

3. Real-Time Communication:

This component establishes a connection to the Socket.IO server, handling events such as connection, message received, typing, and stop typing.

- **Establish Connection and Event Handling:**

```
  useEffect(() => {
    socket = io(ENDPOINT)
    socket.emit('setup', user)
    socket.on('connected', () => setSocketConnected(true))

    return () => {
      socket.off('connected')
      socket.disconnect()
    }
  }, [])
```

- **Handle Typing Events:**

```
useEffect(() => {
    fetchMessagesHandler()

    socket.on('typing', ({ chatId }) => {
      if (selectedChat?._id === chatId) {
        setIsTyping(true)
      }
    })

    socket.on('stop typing', ({ chatId }) => {
      if (selectedChat?._id === chatId) {
        setIsTyping(false)
      }
    })

    return () => {
      socket.off('typing')
      socket.off('stop typing')
    }
  }, [selectedChat])
```

4. User Interface Components:

This component uses Chakra UI to build a responsive and accessible interface. Custom hooks and utility functions further enhance UI flexibility and maintainability.

- **Button Component:** A styled button component using Chakra UI with responsive design.

```
onst Button: React.FC<ButtonProps> = ({ children, ...props }) => {
  const theme = useTheme()
  const primaryColor = theme.colors.primary
  const whiteColor = theme.colors.white

  const fontSize = useBreakpointValue({ base: 'md', ls: 'lg' })
  const paddingX = useBreakpointValue({ base: 4, md: 5 })
  const paddingY = useBreakpointValue({ base: 2, md: 3 })
  const borderRadius = useBreakpointValue({ base: 'md', md: 'xl' })

  return (
    <ChakraButton
      fontSize={fontSize}
      bg={primaryColor}
      color={whiteColor}
      _hover={{
        bg: primaryColor,
      }}
      borderRadius={borderRadius}
      px={paddingX}
      py={paddingY}
      {...props}
    >
      {children}
    </ChakraButton>
  )
}
```

Quantity Description:

- **Scalability:** The frontend architecture is designed to handle a growing number of users and interactions gracefully. The modular design allows for easy scaling of features and components.
- **Robustness:** Advanced error handling mechanisms are in place to capture and manage errors effectively, ensuring the application remains stable and reliable.
- **Security:** Implements industry-standard security measures to protect user data and interactions. This includes secure handling of authentication tokens, HTTPS communication.

Interaction Flow Example: User Login

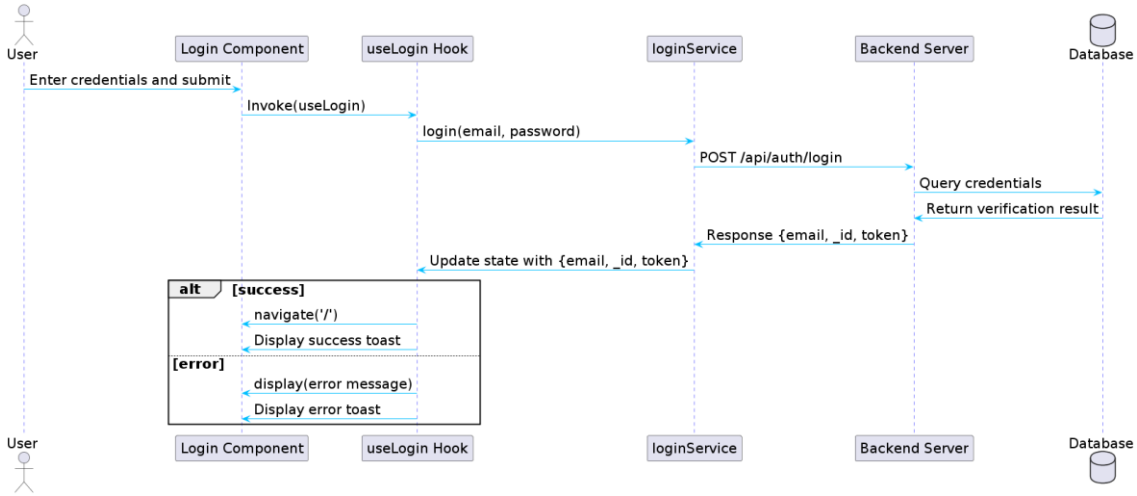


Figure 47: Interaction Flow Example: Login

1. Frontend Interaction (React Components and Hooks)

Component: Login.tsx

- The user is presented with a login form built using Chakra UI components for a responsive and visually appealing interface.
- The user enters their email and password and submits the form.

Hook: useLogin

- On form submission, the useLogin hook is triggered.
- This hook manages the login state, including loading and error states.
- It calls the loginService, which sends a request to the backend.

```
const useLogin = () => {
  const [error, setError] = useState<string | null>(null)
  const [isLoading, setIsLoading] = useState<boolean>(false)
  const { dispatch } = useAuthContext()
  const toast = useToast()

  const login = async (email: string, password: string) => {
    setIsLoading(true)
    setError(null)
    try {
      const json = await loginService(email, password)
      dispatch({ type: 'LOGIN', payload: json })
      return true
    } catch (catchError) {
      if (catchError instanceof Error) {
        setError(catchError.message)
      }
      return false
  }
  return { isLoading, login, error }
}
```

2. Service Layer (Axios HTTP Requests)

Service: loginService

- The loginService function constructs an HTTP request using Axios, targeting the /api/auth/login endpoint.
- It passes the user's email and password in the request body.
- Axios is configured to handle cookies for authentication tokens securely.

```
const loginService = async (email: string, password: string):
Promise<any> => {
  try {
    const response = await axios.post(`${BASE_API_URL}/api/auth/login`,
      { email,password },
      { withCredentials: true },
```

```
    )
    Cookies.set('token', response.data.token)
    return response.data
  } catch (error) {
    if (axios.isAxiosError(error)) {
      if (error.response?.status === 401) {
        const refreshToken = Cookies.get('refreshToken')
        if (refreshToken) {
          try {
            const refreshResponse = await axios.post(
              `${BASE_API_URL}/api/auth/refresh_token`,
              {},
              { withCredentials: true },
            )
            Cookies.set('token', refreshResponse.data.token, {})
            return await loginService(email, password)
          } catch (refreshError: any) {
            throw new Error(
              refreshError.response?.data.error || 'Failed to refresh
token',
            )
          }
        } else {
          throw new Error('Refresh token is missing')
        }
      }
      throw new Error(
        error.response?.data.error || 'An error occurred during login',
      )
    }
    throw new Error('An error occurred during login')
  }
}
```

3. Backend API (Express.js and Middleware)

Route: /api/auth/login in authRoutes.ts

- The login route receives the request and invokes the loginUser controller.

Controller: loginUser in authControllers.ts

- The loginUser function validates the email and password against the database using Mongoose models.
- If the credentials are valid, it generates JWT access and refresh tokens.
- These tokens are set in HTTP-only cookies to enhance security.

```
const loginUser = async (req: Request, res: Response): Promise<void> => {
  const { email, password } = req.body
  try {
    const user = await User.login(email, password)
    const token = createToken(user._id.toString(), '3d')
    const refreshToken = createToken(user._id.toString(), '7d')
    res.cookie('token', token, {
      httpOnly: true,
      secure: true,
      maxAge: 3 * 24 * 60 * 60 * 1000,
    })
    res.cookie('refreshToken', refreshToken, {
      httpOnly: true,
      secure: true,
      maxAge: 7 * 24 * 60 * 60 * 1000,
    })
    res.status(200).json({ email, token, _id: user._id })
  } catch (error: unknown) {
    if (error instanceof Error) {
      res.status(400).json({ error: error.message })
    } else {
      res.status(400).json({ error: 'An unknown error occurred' })
    }
  }
}
```

Model: User

- The User model includes methods for checking user credentials and hashing passwords.

```
const userSchema: MongooseSchema = new Schema(
  {
    username: { type: String, required: true, unique: true, minlength: 3,
      trim: true,},
    email: { type: String, required: true, unique: true, lowercase: true,
      validate: {
        validator: (str: string) => validator.isEmail(str),
        message: (props: { value: string }) =>
          `${props.value} is not a valid email address!`,
      },
    },
    // Other fields…
  }
)
```

```
userSchema.statics.login=async function(email: string, password: string,
): Promise<IUser> {
  if (!email || !password) {
    throw Error('All fields must be filled')
  }
  const user = await this.findOne({ email })
  if (!user) {
    throw Error('Incorrect email')
  }
  const match = await bcrypt.compare(password, user.password)
  if (!match) {
    throw Error('Incorrect password')
  }
  return user
}
```

4. Response Handling

Success:

- If the login is successful, the backend sends a response with the user data and sets the authentication cookies.
- The frontend useLogin hook receives this response and updates the global authentication context using AuthContext, indicating the user is logged in.
- A toast message is displayed to the user indicating a successful login.
- The user is redirected to the homepage of the app.

Error:

- If there's an error (e.g., incorrect credentials), the backend sends an error response.
- The frontend useLogin hook handles this by setting an error state and displaying an error message to the user via a toast and message under input.

Other Functionalities Overview:

The navM8 application includes various functionalities such as user registration, profile management, tour bookings, reviews, and chat. The structure and workflow for these functionalities follow a consistent pattern similar to the login process, ensuring maintainability and scalability across the application.

General Structure:

1. Frontend Interaction:

- Components built using Chakra UI for user interactions where applicable.

2. State Management:

- React hooks and context API manage the state and often invoke service functions.

3. Service Layer:

- Axios is used for sending HTTP requests to backend endpoints.

4. Backend API:

- Express.js routes and controllers handle requests, perform validations, and return responses.

5. Response Handling:

- The frontend updates the global state and UI based on responses, displaying success or error messages as needed.

3.4.4 Cloud Service Layer

Media Handling with Cloudinary:

- **Overview:** Cloudinary is used for handling media uploads, including user profile and tour images, which is integral for media-rich content management in "navM8".
- **Functionality:** Configures and executes media uploads, returns secure URLs for stored media which are used within the app.

```
cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
})
```

```
async function uploadTourImageToCloudinary(filePath: string):
Promise<string> {
  try {
    const result = await cloudinary.uploader.upload(filePath, {
      folder: 'tourPhotos',
    })
    return result.secure_url
  } catch (error) {
    console.error('Cloudinary Upload Error:', error)
    throw new Error(`Failed to upload image: ${error.message}`)
  }
}
```

```
try {
    const result = await cloudinary.uploader.upload(req.file.path, {
      folder: 'userProfilePhotos',
      secure: true,
      resource_type: 'image',
    })
```

```
    const updatedUser = await User.findByIdAndUpdate(
      req.user._id,
      { profilePictureURL: result.secure_url },
      { new: true },
    )
    res.json({
      message: 'Profile photo uploaded successfully',
      profilePictureURL: result.secure_url,
    })
  } catch (error) {
    console.error('Error uploading profile photo:', error)
    res.status(500).json({ error: 'Internal server error' })
  }
```

Quality Description:

- **Reliability:** Cloudinary offers high reliability for media uploads, ensuring that images are stored securely and are readily available through URLs.
- **Scalability:** The cloud service is scalable and can handle increasing amounts of media without impacting performance. Cloudinary's infrastructure supports auto-scaling, making it suitable for applications with growing media needs.
- **Security:** Uses secure URLs and HTTPS to ensure media is accessed and transmitted securely. Access control settings in Cloudinary prevent unauthorized access to stored media.

Quantity Description:

- **Volume:** Cloudinary can handle a high volume of media files, making it suitable for applications with extensive media requirements. This includes storing user profile pictures and tour-related images.
- **Performance:** Optimized for fast media retrieval, which enhances the user experience by ensuring images load quickly. Cloudinary's CDN (Content Delivery Network) ensures media is delivered swiftly across different geographical locations.
- **Maintenance:** Cloudinary manages the underlying infrastructure, reducing the

maintenance burden on the development team. This includes automatic backups and redundancy, ensuring media is not lost.

3.5 Testing

This section of the documentation is dedicated to manual, functional end-to-end testing of the navM8 application. These tests are designed to verify that all features and functionalities of the web application work as intended in a simulated real-world usage scenario.

3.5.1 Test Setup

Before initiating the tests, ensure the following setup conditions are met to accurately simulate the user environment and interactions:

Environment Setup:

- Ensure that both the application server and the database are actively running.

Tools Required:

- A stable internet connection.
- A modern web browser (Chrome, Firefox, Safari).

Test Data:

- Pre-populate the database with a variety of tours, user profiles, and other relevant data.

3.5.2 Test Cases

User Authentication:

Test Case: Sign Up

- **Objective:** Ensure new users can register using valid credentials.
- **Steps:**
  1. Navigate to the Signup page.
  2. Input valid username, email, and password.
  3. Submit the form.
- **Expected Outcome:** The system registers the user, confirms the registration onscreen.

Test Case: Sign Up with Invalid Credentials

- **Objective:** Verify that the system prevents registration with invalid credentials.
- **Steps:**
  1. Navigate to the Signup page.
  2. Enter a username that is too short, less than 3 characters (e.g., "ab").
  3. Enter an invalid email address (e.g., "user@@example.com").
  4. Enter a weak password (e.g., "123").
  5. Attempt to submit the form.
- **Expected Outcome:**
  - The system should display error messages indicating the reason for rejection (invalid email, weak password, short username).
  - The system should not register the user.

Test Case: Log In

- **Objective:** Verify that existing users can log in with correct credentials.
- **Steps:**
  1. Navigate to the Login page.
  2. Enter correct email and password.

3. Click on the 'Log In' button.

- **Expected Outcome:** User is redirected to homepage, and we can see successful login messages on the bottom of the screen.

Test Case: Log In with Incorrect Credentials

- **Objective:** Ensure that the system does not allow users to log in with incorrect email or password.
- **Steps:**
    1. Navigate to the Login page.
    2. Enter an incorrect email address that is not registered (e.g., "unknown@example.com").
    3. Enter an incorrect password.
    4. Attempt to log in.
- **Expected Outcome:**
    - The system should display an error message indicating that the login credentials are incorrect.
    - The user should not be granted access to the system.

Test Case: Log In with Empty Credentials

- **Objective:** Confirm that the system handles empty login fields properly.
- **Steps:**
    1. Navigate to the Login page.
    2. Leave the email and password fields empty.
    3. Attempt to log in.
- **Expected Outcome:**
- The system should display error messages indicating that all fields must be filled.
- The user should not be able to proceed with the login.

Test Case: Log Out

- **Objective:** Confirm users can log out securely.
- **Steps:**
  1. Log in to the application.
  2. Click on the 'Logout' button on the top right side of navbar.
- **Expected Outcome:** User is logged out and redirected to the home page.

Tour Management:

Test Case: Create a Tour

- **Objective:** Ensure that guides can list new tours.
- **Steps:**
  1. Log in to the application.
  2. Go to My Tours from sidebar.
  3. Click on ‘Add Tour’ button to open modal window with form.
  4. Fill in the tour details and click on ‘Save’.
- **Expected Outcome:** Tour is added to the user’s My Tours page and can be booked by travelers.

Test Case: Edit a Tour

- **Objective:** Ensure that user can update details of an existing tour correctly.
- **Steps:**
  1. Log in to the application.
  2. Go to My Tours from sidebar.
  3. If user has created tours, we will see tour card.
  4. Click on ‘Edit’ button on the right bottom side of the card to open modal window with a form.
  5. Edit tour details (e.g., name, date, description, etc.) and click on ‘Save’.
- **Expected Outcome:** The tour details are updated successfully. The changes should reflect immediately in the "My Tours" page, and a confirmation message may appear.

Test Case: Delete a Tour

- **Objective:** Verify that user can successfully delete their tours, ensuring the deletion process includes a confirmation step to prevent accidental removals.
- **Steps:**
  1. Log in to the application.
  2. Go to My Tours page from sidebar.
  3. If the user has created tours, a list of tour cards will be displayed.
  4. Click on ‘Delete button located on the right bottom side of the tour card.
  5. A confirmation window pops up. Confirm the deletion by clicking the 'Confirm' button in the dialog.
- **Expected Outcome:** The tour is permanently deleted from the database. The tour card is removed from the "My Tours" page, and a confirmation message of deletion appears.

Test Case: Search for Tours

- Objective: Test the search functionality for finding tours by city.
- Steps:
  1. Access the search interface in Home Page.
  2. Input a city name and initiate the search.
- **Expected Outcome:** Tours available in the specified city are displayed.

Test Case: Book a Tour

- **Objective:** Verify that travelers can book tours and that the booking interface respects the availability settings of the tour.
- **Steps:**
  1. Log in to the application.
  2. Navigate to the homepage and use the search functionality to find a specific tour or browse the available tours.
  3. Click on a tour card from the search results to view detailed information about the tour.
  4. Review the tour details, focusing on the "Availability Type" which dictates

when the tour can be booked:

5. Recurring on weekends: Ensure the date is a Saturday or Sunday.
6. Daily: Any day of the week is selectable.
7. Weekdays: Ensure the date is between Monday and Friday.
8. Access the booking calendar within the tour detail page.
9. Select an available date based on the tour's availability type.
10. Click the 'Book' button.
11. Confirm the booking details below the calendar, showing the chosen date and booking status as 'PENDING'.

- **Expected Outcome:**
  - Upon clicking 'Book', the tour is successfully booked for the selected date.
  - The booking status should initially display as 'PENDING' and is visible to the user immediately after booking.
  - The user should see an on-screen confirmation, with details of the booking date and current status.
  - The booking status changes to 'CONFIRMED' once the tour author reviews and approves the booking request.
- **Postconditions**
  - Ensure the booking entry is added to the user's 'My Bookings' section with the correct status.
  - Verify that no bookings are possible on days not permitted by the tour's availability settings.

Test Case: Book a Tour on Invalid Date

- **Objective:** Verify the system prevents booking on dates not allowed by the tour's availability type.
- **Steps:**
  1. Log in to the application.
  2. Navigate to the homepage and select a tour.
  3. Access the booking calendar within the tour detail page.
  4. Attempt to select a date that contradicts the availability type (e.g., trying to

book a weekday tour on a weekend).

- **Expected Outcome:** The system should prevent selection of the date or show an error message upon attempt to book, stating invalid booking date for the selected tour's availability.

Test Case: Tour Author Attempts to Book Own Tour

- **Objective**: Ensure that tour authors cannot book their own tours.
- **Steps:**
  1. Log in to the application.
  2. Navigate to the tour user created and attempt to book it.
- **Expected Outcome:** The system should display an error message: Users cannot book their own tours.

Profile Management:

Test Case: Update Profile Information

- **Objective:** Verify that users can update their personal details and password on their profile page.
- **Steps:**
  1. Log in to the application.
  2. Click on 'Profile' from the sidebar to access the profile page.
  3. User can upload a new profile photo by clicking on the green camera icon or delete the existing photo by clicking on the red bin icon at the top of the profile photo.
  4. Click on the 'Edit' button to enable input fields if they are disabled.
  5. Update or add fields such as first name, last name, phone number, country, city, birth date, gender, languages spoken, interests, and bio.
  6. To change the password:
     - Enter the current password in the 'Current Password' field.
     - Enter the new password in the 'New Password' field.
  7. Click 'Save' to submit the changes.

- **Expected Outcome:**
    - All changes to the profile details are saved successfully.
    - The user should receive a confirmation message that profile updates and all the changes have been successful.
    - The input fields become disabled to indicate that the editing mode has been exited, ensuring changes are locked until 'Edit' is clicked again.

Test Case: Enter Invalid City or Country in Profile

- **Objective:** Ensure the system validates city and country inputs against a valid list.
- **Steps:**
    1. Log in to the application.
    2. Navigate to the 'Profile' section.
    3. Click 'Edit' to modify the profile details.
    4. Enter an invalid city or country (e.g., "Budapes ", “Hangary”).
    5. Attempt to save the changes.
- **Expected Outcome:** The system should display an error message indicating the invalid input for city or country.

Test Case: Enter Too Short Bio in Profile

- **Objective:** Verify that the bio must meet a minimum character requirement.
- **Steps:**
    1. Log in to the application.
    2. Navigate to the 'Profile' section.
    3. Enter a bio that is fewer than 10 characters long.
    4. Attempt to save the changes.
- **Expected Outcome:** The system should not save the changes and should display a message indicating the bio is too short.

Chat and Messaging:

Test Case: Send a Message

- **Objective:** Ensure users can send messages to other users/groups in real-time.
- **Steps:**
  1. Log in to the application.
  2. Click on Messages from the sidebar to access Messages Page
  3. Choose an existing chat from the chat list or search for a user using the 'Search User' field and initiate a new chat.
  4. Type a message in the input field at the bottom of the chat window.
  5. Press 'Enter' or click the send icon to send the message.
- **Expected Outcome:**
  - The message appears immediately in the chat window.
  - The recipient receives the message in real-time, reflecting in their chat window without needing to refresh the page.

Test Case: Create a Group Chat

- **Objective:** Verify that users can create group chats by specifying a chat name and adding users.
- **Preconditions:** User must be logged in.
- **Steps:**
  1. Navigate to 'Messages' from the sidebar.
  2. Click on the 'New Group' button.
  3. In the 'Create Group Chat' modal, enter a name for the chat in the 'Chat Name' field.
  4. Use the 'Add Users' field to search and select users to add to the group chat.
  5. Click 'Create' to establish the group chat.
- **Expected Outcome:**
  1. A new group chat is created with the specified users.
  2. The group chat appears in the chat list and is accessible to all added members.
  3. Members can immediately begin exchanging messages within the group.

Chapter 4

4. Conclusion

The development of navM8, a web application designed to enhance travel experiences through local guide engagement, demonstrates the potential of technology to transform tourism. navM8 makes authentic local experiences accessible to travelers by connecting them with local guides who offer free tours, promoting cultural exchange and deeper connections with new places.

The thesis explored key features of navM8, such as user registration, profile management, tour booking, and creation, along with real-time messaging and review systems to enhance user interaction and trust. Utilizing the MERN stack (MongoDB, Express.js, React, Node.js), Socket.io for real-time communication, and Cloudinary for media management, navM8's architecture ensures scalability, performance, and security.

Extensive testing has validated navM8's functionality and reliability, ensuring a seamless user experience. navM8 not only meets its technical goals but also fosters a community of travelers and local guides, enriching the travel experience.

In summary, navM8 addresses the challenge of making genuine travel experiences accessible and affordable. By connecting travelers with locals, it enriches the journey, making travel more meaningful and inclusive.

Bibliography

[1] *React JS*. 2024. URL: https://react.dev

[2] *TypeScript*. 2024. URL: https://www.typescriptlang.org/docs/

[3] *Vite*. 2024. URL https://vitejs.dev/guide/

[4] *Node.js*. 2024. URL: https://nodejs.org/en/docs/

[5] *Express.js*. 2024. URL: https://expressjs.com/en/starter/installing.html

[6] *MongoDB*. 2024. URL: https://docs.mongodb.com/manual/

[7] *Mongoose*. 2024. URL: https://mongoosejs.com/docs/

[8] *Cloudinary*. 2024. URL: https://cloudinary.com/documentation

[9] *Socket.io*. 2024. URL: https://socket.io/docs/

[10] *Chakra UI*. 2024. URL: https://chakra-ui.com/docs/getting-started

List of Figures

List of Tables