

MyTravel

Travel Booking Platform Documentation

Full-Stack Application Documentation

Generated: November 29, 2025

Table of Contents

MyTravel - Overview

Setup Guide

Features

Architecture

Technologies

API Reference

Error Handling

MyTravel - Travel Booking Platform

Screenshot: MyTravel homepage with hero section and destination search

Project Description

MyTravel is a full-stack travel booking platform designed to provide users with a comprehensive travel planning experience. The application serves as a one-stop solution for travelers seeking to discover new destinations, book travel services, and organize their trips efficiently.

The platform enables users to browse an extensive catalog of travel destinations from around the world. Each destination includes detailed information, high-quality imagery, and transparent pricing to help travelers make informed decisions. Users can book flights, hotels, and guided tours directly through the platform, with all selections managed through an integrated shopping cart and checkout system.

A standout feature of MyTravel is the interactive trip planner, which allows users to create custom itineraries using drag-and-drop functionality. This tool empowers travelers to organize their activities day by day, visualize their journey, and adjust plans as needed before finalizing their bookings.

The platform also includes a travel blog section where users can read destination guides, travel tips, and stories from other travelers. Authenticated users have the ability to create and publish their own blog posts, contributing to the community knowledge base.

For returning users, MyTravel provides account management features including booking history, profile customization, and saved preferences. This personalization ensures a streamlined experience for frequent travelers.

The administrative side of the platform offers a comprehensive dashboard for managing the entire system. Administrators can oversee user accounts, process and track bookings, moderate blog content, and monitor platform statistics. This dual-interface approach ensures smooth operation for both end users and platform managers.

Documentation

Document	Description
Setup Guide	Instructions on how to install and run the project
Features	Complete list of pages and features with screenshots
Architecture	Overview of the code structure and organization
Technologies	Libraries and frameworks used in the project
API Reference	Full documentation of all API endpoints
Error Handling	Explanation of error handling strategies

Setup Guide

This guide provides instructions for setting up and running the MyTravel application on your local development environment.

Prerequisites

The following software must be installed on your machine before proceeding with the setup:

- **.NET 10 SDK** - The backend is built on [ASP.NET](#) Core 10. Download from dotnet.microsoft.com
- **Node.js (v18 or higher)** - Required for the frontend build tools. Download from nodejs.org
- **npm** - Comes bundled with Node.js and is used for managing frontend dependencies

To verify your installations, run the following commands in your terminal:

```
dotnet --version      # Should output 10.x.x
node --version        # Should output v18.x.x or higher
npm --version         # Should output 9.x.x or higher
```

Installation

1. Install Frontend Dependencies

Navigate to the client project and install the required npm packages:

```
cd mytravel.client
npm install
cd ..
```

2. Configure the Backend (Optional)

The application comes with default configuration settings that work out of the box for local development. The configuration files are located in the `MyTravel.Server` directory:

- `appsettings.json` - Base configuration

- `appsettings.Development.json` - Development-specific settings

The default admin credentials are configured in `appsettings.json`:

```
{  
    "AdminCredentials": {  
        "Email": "admin@mytravel.com",  
        "Password": "Admin@123!"  
    }  
}
```

For production deployments, these credentials should be changed and stored securely using environment variables or a secrets manager.

Running the Application

The MyTravel solution uses the [ASP.NET](#) Core SPA Proxy feature, which means the frontend development server starts automatically when you run the backend. This provides a seamless development experience where both the API and client are available from a single command.

Using Visual Studio

1. Open `MyTravel.slnx` in Visual Studio
2. Set `MyTravel.Server` as the startup project
3. Press `F5` to run with debugging, or `Ctrl+F5` to run without debugging

Visual Studio will automatically start both the backend server and the frontend development server.

Using the Command Line

Navigate to the server project directory and run:

```
cd MyTravel.Server  
dotnet run
```

For a more interactive development experience, use `dotnet watch` which provides hot reload capabilities. This automatically rebuilds and restarts the application when code changes are detected:

```
cd MyTravel.Server  
dotnet watch
```

The `dotnet watch` command is recommended during active development as it significantly speeds up the feedback loop when making changes to the backend code.

Accessing the Application

Once the application is running, you can access it at the following URLs:

Service	URL	Description
Frontend	http://localhost:49764	Main application interface
Backend API	http://localhost:5083	API endpoints
Swagger UI	http://localhost:5083/swagger	API documentation (development only)

The frontend development server proxies API requests to the backend, so you can interact with the application entirely through the frontend URL during development.

Database

MyTravel uses SQLite as its database, which requires no additional setup. The database file is created automatically when the application first runs and is stored in the `MyTravel.Server` directory as `mytravel.db`.

The database is seeded with sample data on first run, including:

- Sample destinations and activities
- Demo blog posts
- The admin user account

To reset the database, simply delete the `mytravel.db` file and restart the application.

Troubleshooting

Port Already in Use

If you encounter an error indicating that a port is already in use, another application may be using ports 5083 or 49764. You can either stop the conflicting application or modify the ports in the following files:

- `MyTravel.Server/Properties/launchSettings.json` - Backend port configuration
- `mytravel.client/vite.config.ts` - Frontend port configuration

Frontend Not Starting Automatically

If the frontend does not start automatically with the backend, you can run it manually in a separate terminal:

```
cd mytravel.client
npm run dev
```

Certificate Errors

The development server uses HTTPS with a self-signed certificate. If your browser shows a certificate warning, you can proceed by accepting the risk (for development purposes only) or trust the .NET development certificate:

```
dotnet dev-certs https --trust
```

Features

This document provides a comprehensive overview of all pages and features available in the MyTravel application. The platform is divided into two main sections: the public-facing website for travelers and the administrative dashboard for platform management.

Public Pages

Homepage

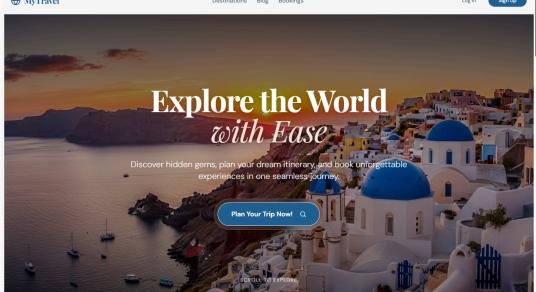
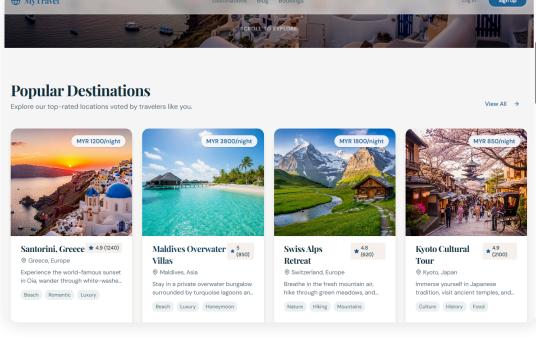
Route: /

“Explore the World with Ease”

The homepage serves as the main entry point for visitors and establishes the MyTravel brand identity. A prominent hero section displays the website's name alongside a captivating tagline that invites users to begin their travel journey. The hero includes a search interface that allows users to quickly find destinations by location or keyword.

Below the hero, the page highlights top travel destinations with visually appealing cards showcasing popular locations. A testimonials section features reviews from satisfied travelers, building trust with new visitors. Promotional offers and seasonal deals are prominently displayed to encourage immediate engagement. A clear call-to-action button labeled “Plan Your Trip Now!” guides users toward the booking flow.

Screenshots

Hero Section	Featured Destinations
 <p>Hero with tagline and search</p>	 <p>Top destination cards</p>

Testimonials

The screenshot shows a grid of travel packages on the MyTravel website. Each package card includes a thumbnail image, the destination name, a rating (e.g., 4.5 stars), and a price per night (e.g., MYR 1000/night). Buttons for 'View Details' and 'Book Now' are present, along with a 'View things to do on Google Maps' link.

Destination	Rating	Price
Costa Rica Rainforest Adventure	4.5 (98)	MYR 1000/night
Panama Canal & City Explorer	4.7 (840)	MYR 1000/night
New Zealand South Island	4.0 (980)	MYR 2200/night
Australian Outback Safari	4.7 (880)	MYR 1800/night
Egypt Pyramids & Nile Cruise	4.3 (2240)	MYR 900/night
Kenya Safari Experience	4.9 (1200)	MYR 2800/night
Rio de Janeiro Carnival	4.8 (880)	MYR 1500/night
Buenos Aires Tango & Culture	4.7 (800)	MYR 2000/night

Promotional Offers

The screenshot shows a grid of promotional travel offers on the MyTravel website. Each offer card includes a thumbnail image, the destination name, a rating, and a price per night. Buttons for 'View Details' and 'Book Now' are present, along with a 'View things to do on Google Maps' link.

Offer	Rating	Price
Costa Rica North Adventure	4.5 (1200)	MYR 900/night
Discover the engineering marvel of the Panama Canal and explore its ancient culture.	4.5 (1200)	MYR 1800/night
Discover dramatic fjords, snow-capped mountains, and adventure.	4.5 (1200)	MYR 1800/night
Experience the vast red desert, ancient Aboriginal culture, and...	4.5 (1200)	MYR 1800/night
Witness the Great Migration, spot exotic birds like...	4.5 (1200)	MYR 1800/night
Explore lush rainforests, spot exotic wildlife, and zip-line through the...	4.5 (1200)	MYR 1800/night
Discover the vibrant energy of Rio's iconic beaches like...	4.5 (1200)	MYR 1800/night
Discover the passionate soul of Argentina through tango, world-class...	4.5 (1200)	MYR 1800/night

User reviews and ratings

Seasonal deals and CTAs

User Registration and Login

Routes: `/login`, `/register`

"Create an account to access personalized travel deals."

The authentication system provides secure access to personalized features and booking capabilities. The registration form collects essential information including email address, first name, last name, and password with confirmation. Form validation is handled client-side using react-hook-form integrated with Zod schemas, providing immediate feedback on input errors.

The login form accepts email and password credentials with options for password visibility toggle. Account verification is supported through email confirmation to ensure valid user accounts. A forgot password feature allows users to reset their credentials by receiving a password reset link via email. User sessions are managed through **ASP.NET Core Identity** with secure cookie-based authentication.

Features:

- Email-based registration with validation
- Account verification via email confirmation
- Forgot password functionality with email reset
- Secure session management
- Protected route redirection after login

Screenshots

Login Form	Registration Form
Email and password login	New account creation
Password Recovery	Form Validation
Password reset request	Real-time validation feedback

Destination Browsing Page

Route: `/destinations`

"Discover hidden gems across the world."

The destinations page presents a comprehensive catalog of all available travel locations with rich visual content. Each destination is displayed as a card featuring high-quality images, descriptive text highlighting key attractions, and aggregated user reviews. Pricing information shows the starting cost to help users quickly assess options within their budget.

The page offers robust filtering capabilities to help users find their ideal destination. Filters are available for continent selection, activity type (beaches, hiking, cultural experiences, adventure sports), and budget range. Users can toggle between a traditional grid view for browsing and an interactive map view powered by Leaflet for geographical exploration. The search functionality allows users to find specific destinations by name or keyword.

Filter Options:

- Continent/Region selection
- Activity type (beaches, hiking, cultural, adventure)
- Budget range (economy, mid-range, luxury)
- Rating and review scores
- Seasonal availability

Screenshots

Grid View	Map View
<i>Card-based destination browsing</i>	<i>Interactive Leaflet map view</i>

Filter Panel

The filter panel includes a search bar, dropdown menus for continent (Europe, Asia, North America, South America, Africa, Oceania), activity (Beaches, Hiking, Culture, City Break, Wildlife, Relaxation), and budget (Max Price: MYR 10,000). It also shows three destination cards: Santorini, Greece (MYR 1200/night); Maldives Overwater Villas (MYR 3800/night); and Swiss Alps Retreat (MYR 1800/night).

Search Results

The search results for Paris show a message: "No destinations found. Try adjusting your filters or search query." There is a "Close all filters" button.

Continent, activity, and budget filters
Filtered destination results

Destination Details

Route: `/destinations/:id`

Individual destination pages provide in-depth information about a specific location. The page opens with a gallery of high-quality images showcasing the destination's highlights. Detailed descriptions cover the location's history, culture, climate, and best times to visit. User reviews and ratings provide social proof and authentic traveler perspectives.

Available activities are listed with descriptions and pricing, allowing users to plan their experiences. Accommodation options range from budget-friendly to luxury establishments. Users can add flights, hotels, or guided tours to their cart directly from this page, streamlining the booking process.

Screenshots

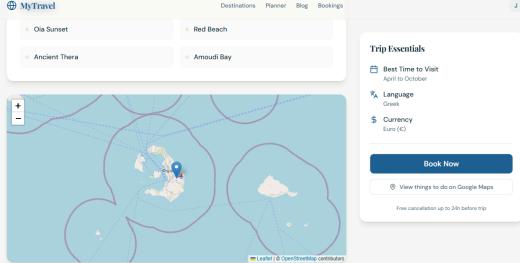
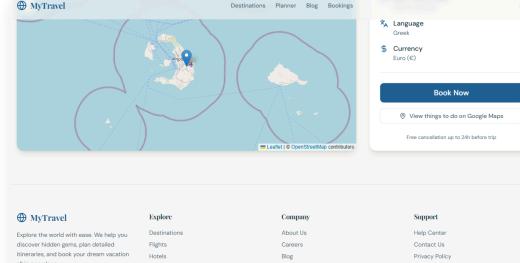
Image Gallery

The image gallery shows a large main image of Santorini at sunset and a smaller inset image of a church. Below the images are sections for "About this place" (rating 4.9, 1240 reviews) and "Trip Essentials" (starting from MYR 1200).

Description & Reviews

The description & reviews section includes a summary of Santorini's history and landscape, a "About this place" card (rating 4.9, 1240 reviews), a "Trip Essentials" card (starting from MYR 1200), and a "Highlights" section listing "Oia Sunset", "Red Beach", "Ancient Thera", and "Ammoudi Bay".

High-quality destination images
Details and user reviews

Available Activities	Booking Options
 <p data-bbox="399 705 632 736"><i>Tours and experiences</i></p>	 <p data-bbox="950 705 1257 736"><i>Flight, hotel, and tour selection</i></p>

Itinerary Planner

Route: [/planner](#)

"Plan your dream trip effortlessly."

The itinerary planner is an interactive tool that empowers authenticated users to create custom travel itineraries tailored to their preferences. The interface enables users to select places, accommodations, and activities, then organize them into a coherent travel plan.

The drag-and-drop functionality, powered by the dnd-kit library, allows users to easily arrange activities across different days of their trip. Users can reorder items, move activities between days, and visualize their complete journey timeline. The planner calculates estimated travel times between locations and provides running cost totals as items are added or removed.

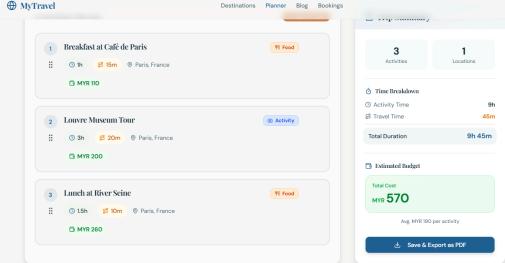
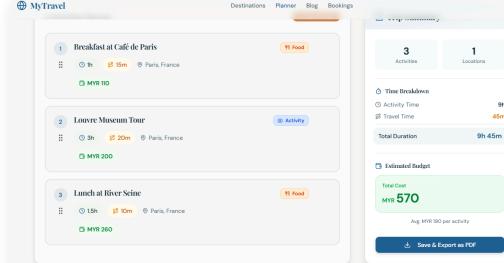
Features:

- Drag-and-drop itinerary creation
- Day-by-day activity organization
- Estimated travel time calculations
- Running cost totals and budget tracking
- Integration with shopping cart
- Save and edit itineraries

Screenshots

Empty Planner	Adding Activities
<p>Initial planner state</p>	<p>Trip Summary</p> <p>Estimated Budget</p> <p>Total Cost: MYR 570</p> <p>Ang. MYR 180 per activity</p> <p>Add Activity</p> <p>Time Breakdown</p> <ul style="list-style-type: none"> Activity Time: 9h 45m Travel Time: 45m <p>Activities Planned</p> <ol style="list-style-type: none"> 1 Breakfast at Café de Paris (Food) 2 Louvre Museum Tour (Activity) 3 Lunch at River Seine (Food) <p>Your Itinerary</p> <p>Drag and drop to reorder your plan. Changes are saved automatically.</p> <p>Destinations Planner Blog Bookings</p>

Drag and Drop	Cost Summary
<p>Save & Export as PDF</p> <p>Estimated Budget</p> <p>Total Cost: MYR 570</p> <p>Ang. MYR 180 per activity</p> <p>Time Breakdown</p> <ul style="list-style-type: none"> Activity Time: 9h 45m Travel Time: 45m <p>Activities</p> <ol style="list-style-type: none"> 1 Breakfast at Café de Paris (Food) 2 Louvre Museum Tour (Activity) 3 Lunch at River Seine (Food) <p>Locations</p> <p>Paris, France</p> <p>Destinations Planner Blog Bookings</p>	<p>Save & Export as PDF</p> <p>Estimated Budget</p> <p>Total Cost: MYR 570</p> <p>Ang. MYR 180 per activity</p> <p>Time Breakdown</p> <ul style="list-style-type: none"> Activity Time: 9h 45m Travel Time: 45m <p>Activities</p> <ol style="list-style-type: none"> 1 Breakfast at Café de Paris (Food) 2 Louvre Museum Tour (Activity) 3 Lunch at River Seine (Food) <p>Locations</p> <p>Paris, France</p> <p>Destinations Planner Blog Bookings</p>

Multi-Day Itinerary	Completed Plan
 <p>Activities across multiple days</p>	 <p>Final itinerary ready for booking</p>

Booking System

Routes: `/booking/:id` , `/checkout`

"Book everything you need for your trip in one place."

The booking system provides a unified interface for reserving all travel services. The booking page presents detailed forms for configuring reservations, including travel dates, number of travelers, and specific options such as room types, flight classes, or tour packages. Real-time availability checking ensures users only see bookable options.

The checkout page consolidates all items from the user's cart into a comprehensive order summary. An itemized breakdown displays each service with its details and pricing. The system calculates total costs including any applicable taxes and fees. Customer information collection includes contact details and any special requirements.

Payment processing is handled securely with real-time confirmation. Upon successful payment, users receive immediate booking confirmation displayed on screen and sent via email. The confirmation includes booking reference numbers, detailed itinerary information, and relevant contact details for each service provider.

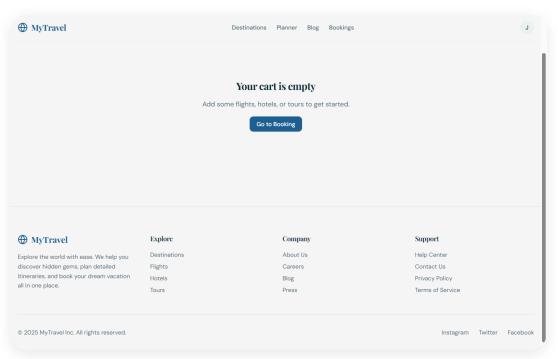
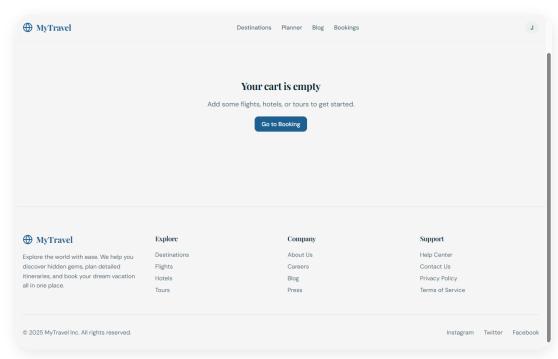
Features:

- Integrated booking for flights, hotels, and tours
- Real-time availability and pricing
- Secure payment processing
- Instant booking confirmation
- Email confirmation with booking details
- Booking reference numbers for each service

Screenshots

Booking Form	Date Selection
<p>Ready for Adventure? Book everything you need for your trip in one place. Flights, hotels, and tours with real-time confirmation.</p> <p>Book Your Trip Secure your flights, hotels, and tours instantly.</p> <p>From: [Where from?] To: [Where to?] Departure: [Pick a date] Class: [Select class]</p> <p>Trip Summary Your itinerary so far Your trip is empty. Start by searching for flight!</p> <p>Search Flights</p> <p>Why book with us?</p> <ul style="list-style-type: none"> Best price guarantee 24/7 customer support Free cancellation on most hotels 	<p>Ready for Adventure? Book everything you need for your trip in one place. Flights, hotels, and tours with real-time confirmation.</p> <p>Book Your Trip Secure your flight(s). Dates from 26/10/2015 to 03/11/2015</p> <p>From: [Where from?] To: [Where to?] Departure: [Pick a date] Class: [Select class]</p> <p>Trip Summary Your itinerary so far Your trip is empty. Start by searching for flight!</p> <p>Search Flights</p> <p>Why book with us?</p> <ul style="list-style-type: none"> Best price guarantee 24/7 customer support Free cancellation on most hotels
<i>Service configuration options</i>	<i>Travel date selection</i>

Cart Summary	Checkout Page
<p>Your cart is empty Add some flights, hotels, or tours to get started.</p> <p>Go to Booking</p> <p>MyTravel Explore the world with ease. We help you discover hidden gems, plan detailed itineraries, and book your dream vacation all in one place.</p> <p>Explore Destinations, Flights, Hotels, Tours</p> <p>Company About Us, Careers, Blog, Press</p> <p>Support Help Center, Contact Us, Privacy Policy, Terms of Service</p>	<p>Your cart is empty Add some flights, hotels, or tours to get started.</p> <p>Go to Booking</p> <p>MyTravel Explore the world with ease. We help you discover hidden gems, plan detailed itineraries, and book your dream vacation all in one place.</p> <p>Explore Destinations, Flights, Hotels, Tours</p> <p>Company About Us, Careers, Blog, Press</p> <p>Support Help Center, Contact Us, Privacy Policy, Terms of Service</p> <p>© 2015 MyTravel Inc. All rights reserved.</p> <p>Instagram · Twitter · Facebook</p>
<i>Itemized booking summary</i>	<i>Customer information entry</i>

Payment Processing	Confirmation
 <p>The screenshot shows the MyTravel website's payment processing page. At the top, there's a navigation bar with links to Destinations, Planner, Blog, and Bookings. Below the navigation, a message says "Your cart is empty" with a sub-instruction "Add some flights, hotels, or tours to get started." A blue "Go to Booking" button is centered below the message. The main content area features the MyTravel logo and a brief tagline: "Explore the world with ease. We help you discover hidden gems, plan detailed itineraries, and book your dream vacation all in one place." Below this, there are four sections: "Explore" (Destinations, Flights, Hotels, Tours), "Company" (About Us, Careers, Blog, Press), "Support" (Help Center, Contact Us, Privacy Policy, Terms of Service), and social media links for Instagram, Twitter, and Facebook.</p>	 <p>The screenshot shows the confirmation page after a booking. It has a similar layout to the payment processing page, with the MyTravel logo and tagline at the top. The main message is "Your cart is empty" with the same sub-instruction. A blue "Go to Booking" button is present. The "Explore", "Company", and "Support" sections are identical to the payment processing page. The bottom of the page includes copyright information: "© 2020 MyTravel Inc. All rights reserved." and social media links for Instagram, Twitter, and Facebook.</p>
<i>Secure payment form</i>	<i>Booking success with reference</i>

Interactive Map Integration

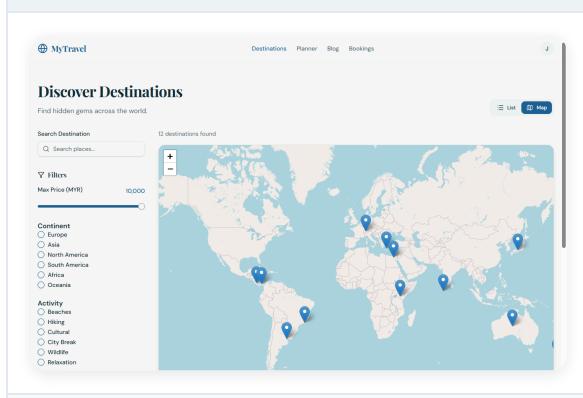
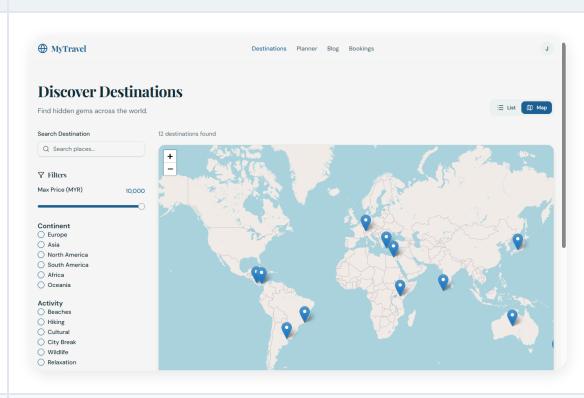
The application integrates interactive mapping capabilities using the Leaflet library with React-Leaflet bindings. Maps are featured prominently on the destinations browsing page, allowing users to explore travel locations geographically.

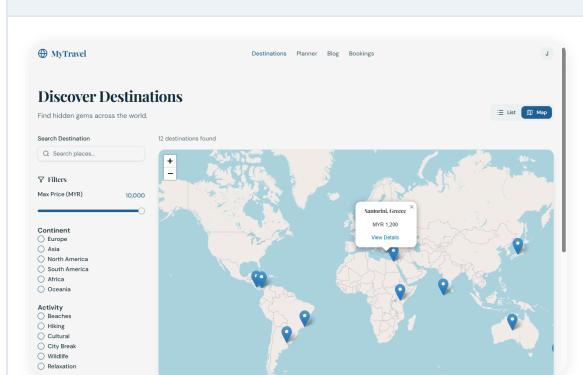
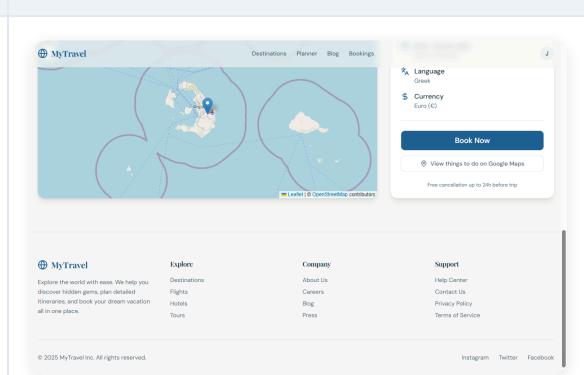
"Explore destinations and travel routes on an interactive map."

Users can pan and zoom across the world map, with destination markers indicating available locations. Clicking a marker reveals a popup with destination summary information and a link to the full details page. The map view provides an intuitive way to discover destinations by region and understand geographical relationships between locations.

On individual destination pages, maps display the specific location along with nearby points of interest, helping users understand the area and plan local exploration.

Screenshots

World Map View	Destination Markers
	
<i>Global destination overview</i>	<i>Clickable location markers</i>

Marker Popup	Destination Map
	
<i>Quick destination info</i>	<i>Detailed location view</i>

Travel Blog

Routes: `/blog`, `/blog/:slug`, `/blog/new`, `/blog/edit/:slug`

"Get inspired by amazing travel experiences."

The blog section serves as a community-driven content hub where users can read and contribute travel stories, tips, and destination guides. The main blog page displays posts in a paginated list with preview cards showing titles, excerpts, featured images, and publication dates.

Content is organized into categories to help readers find relevant articles. Categories include solo travel adventures, family trip planning, luxury travel experiences, budget

backpacking, cultural immersion, and adventure activities. Users can browse all posts or filter by their preferred category.

Individual blog posts are rendered with full rich text content using the Lexical editor's read mode. Articles support formatted text, headings, embedded images, and other media elements. Related posts are suggested at the bottom of each article to encourage continued exploration.

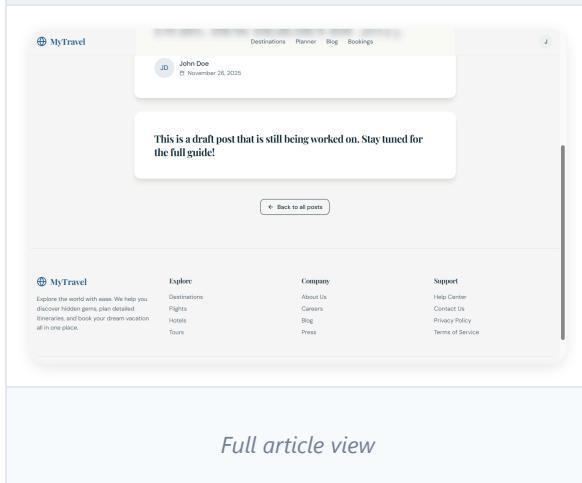
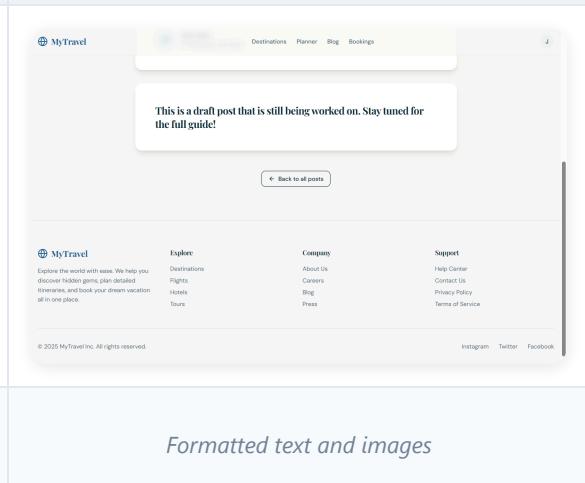
Authenticated users can contribute their own travel stories through the blog editor. The editor provides a rich text interface supporting formatted content, image uploads, and category selection. Authors can save drafts, preview their work, and publish when ready to share with the community.

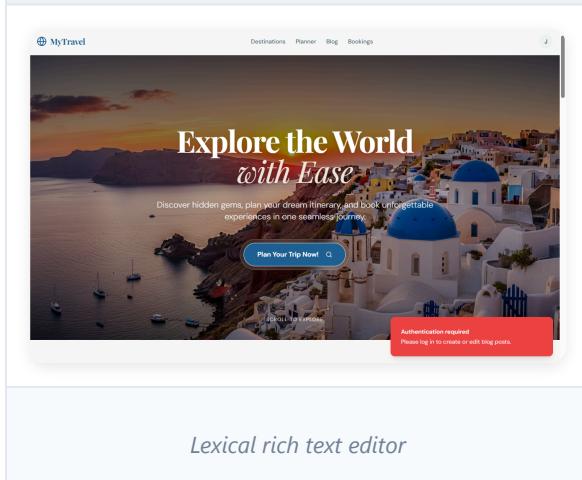
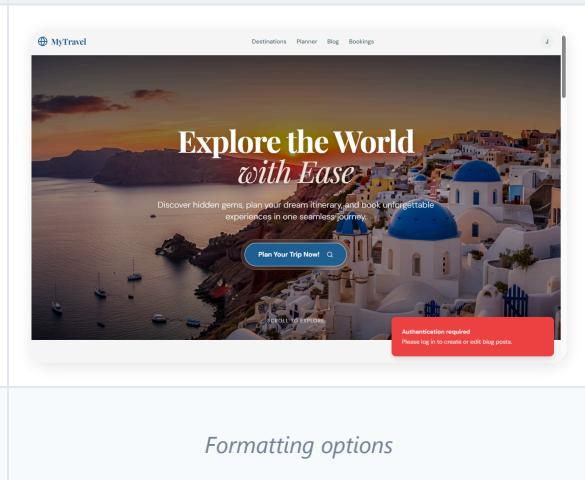
Content Categories:

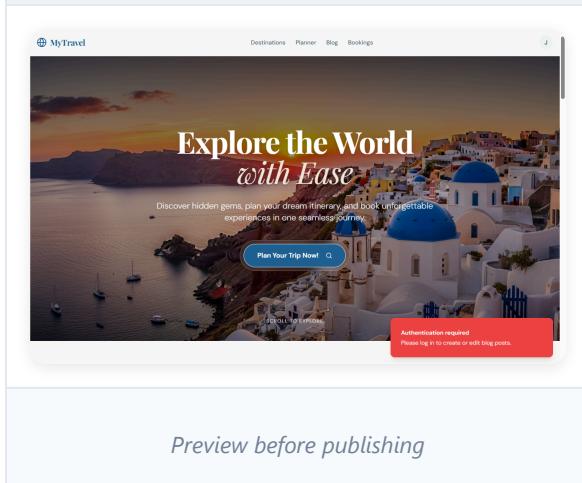
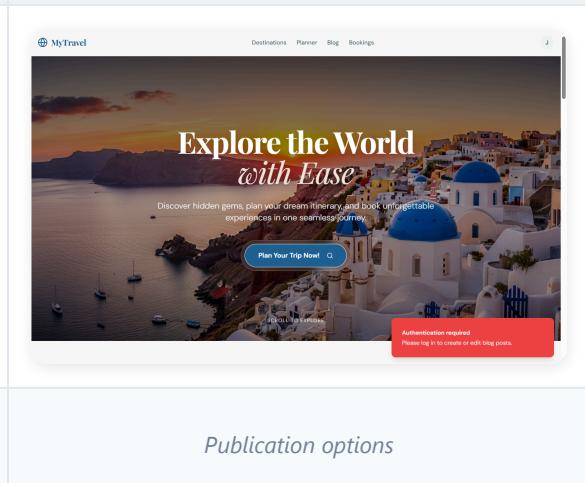
- Solo travel
- Family trips
- Luxury travel
- Budget backpacking
- Cultural experiences
- Adventure activities

Screenshots

Blog Listing	Category Filter
	
<p>Paginated post cards</p>	<p>Filter by travel category</p>

Blog Post	Rich Content
 <p>This is a draft post that is still being worked on. Stay tuned for the full guide!</p> <p>Back to all posts</p> <p>MyTravel</p> <p>Destinations Planner Blog Bookings</p> <p>John Doe November 26, 2025</p> <p>Explore Destinations Flights Hotels Tours</p> <p>Company About Us Careers Blog Press</p> <p>Support Help Center Contact Us Privacy Policy Terms of Service</p> <p>© 2025 MyTravel Inc. All rights reserved.</p>	 <p>This is a draft post that is still being worked on. Stay tuned for the full guide!</p> <p>Back to all posts</p> <p>MyTravel</p> <p>Destinations Planner Blog Bookings</p> <p>Explore Destinations Flights Hotels Tours</p> <p>Company About Us Careers Blog Press</p> <p>Support Help Center Contact Us Privacy Policy Terms of Service</p> <p>Instagram Twitter Facebook</p>
<i>Full article view</i>	<i>Formatted text and images</i>

Blog Editor	Editor Toolbar
 <p>Explore the World with Ease</p> <p>Discover hidden gems, plan your dream itinerary, and book unforgettable experiences in one seamless journey.</p> <p>Plan Your Trip Now! <input type="button"/></p> <p>ROLL TO EXPAND <input type="button"/></p> <p>Authentication required Please log in to create or edit blog posts.</p>	 <p>Explore the World with Ease</p> <p>Discover hidden gems, plan your dream itinerary, and book unforgettable experiences in one seamless journey.</p> <p>Plan Your Trip Now! <input type="button"/></p> <p>ROLL TO EXPAND <input type="button"/></p> <p>Authentication required Please log in to create or edit blog posts.</p>
<i>Lexical rich text editor</i>	<i>Formatting options</i>

Draft Preview	Publishing
 <p>Explore the World with Ease</p> <p>Discover hidden gems, plan your dream itinerary, and book unforgettable experiences in one seamless journey.</p> <p>Plan Your Trip Now! <input type="button"/></p> <p>ROLL TO EXPAND <input type="button"/></p> <p>Authentication required Please log in to create or edit blog posts.</p>	 <p>Explore the World with Ease</p> <p>Discover hidden gems, plan your dream itinerary, and book unforgettable experiences in one seamless journey.</p> <p>Plan Your Trip Now! <input type="button"/></p> <p>ROLL TO EXPAND <input type="button"/></p> <p>Authentication required Please log in to create or edit blog posts.</p>
<i>Preview before publishing</i>	<i>Publication options</i>

Not Found

Route: `*` (catch-all)

A custom 404 page is displayed when users navigate to a route that does not exist. The page provides a friendly message and links back to the homepage and other main sections of the site, helping users recover from navigation errors gracefully.

Administrative Pages

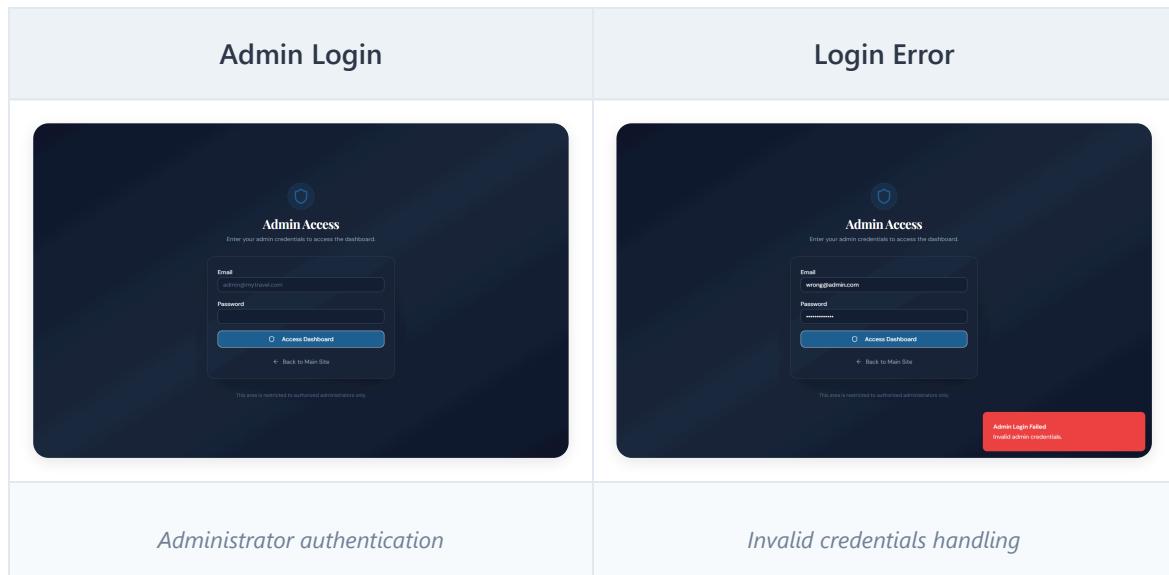
The admin section is protected and requires administrator credentials to access. It provides comprehensive tools for managing all aspects of the platform.

Admin Login

Route: `/admin/login`

The administrative login page is separate from the public user login to maintain security isolation. It accepts the admin email and password configured in the application settings. Successful authentication creates an admin session cookie that grants access to all administrative features. Failed login attempts are logged for security monitoring.

Screenshots



Admin Dashboard

Route: `/admin`

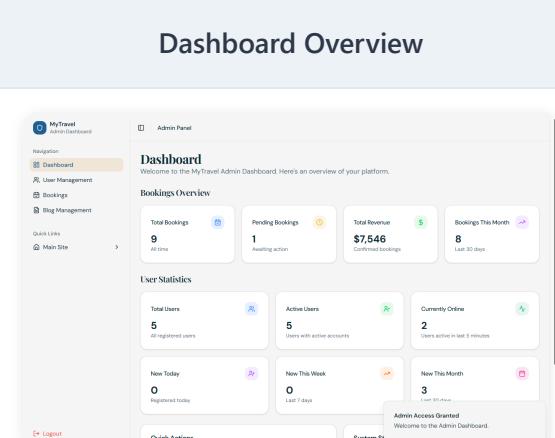
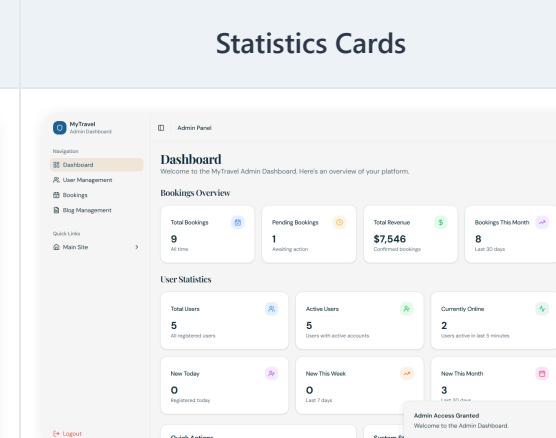
The main dashboard provides a centralized overview of platform statistics, analytics, and recent activity. Key performance metrics are displayed prominently, including total registered users, active bookings, published blog posts, and revenue summaries.

Analytics and Metrics:

- Visitor traffic statistics
- User registration trends
- Booking volume and revenue charts
- Blog engagement metrics
- Active user sessions

Charts built with Recharts visualize trends over time, helping administrators identify patterns and make data-driven decisions. Quick action buttons provide shortcuts to common administrative tasks such as approving pending bookings or reviewing new user registrations. Recent activity feeds show the latest user actions and system events.

Screenshots

Dashboard Overview	Statistics Cards
 <p>This screenshot shows the main Admin Dashboard. It features a top navigation bar with 'MyTravel Admin Dashboard' and links for 'Dashboard', 'User Management', 'Bookings', and 'Blog Management'. Below the navigation is a 'Quick Links' section with 'Main Site'. The main content area is titled 'Dashboard' with a sub-section 'Bookings Overview'. It displays four cards: 'Total Bookings' (9 All time), 'Pending Bookings' (1 Awaiting action), 'Total Revenue' (\$7,546 Confirmed bookings), and 'Bookings This Month' (8 Last 30 days). Below this is a 'User Statistics' section with three cards: 'Total Users' (5 All registered users), 'Active Users' (5 Users with active accounts), and 'Currently Online' (2 Users active in last 5 minutes). At the bottom are 'Quick Actions' (Logout) and 'System Status' (Admin Access Granted).</p>	 <p>This screenshot focuses on the 'Statistics Cards' section of the dashboard. It shows the same four cards from the main overview: Total Bookings (9 All time), Pending Bookings (1 Awaiting action), Total Revenue (\$7,546 Confirmed bookings), and Bookings This Month (8 Last 30 days). Below these are the 'User Statistics' cards: Total Users (5 All registered users), Active Users (5 Users with active accounts), and Currently Online (2 Users active in last 5 minutes). At the bottom are 'Quick Actions' (Logout) and 'System Status' (Admin Access Granted).</p>
<i>Main admin interface</i>	<i>Key metrics at a glance</i>

The image displays two side-by-side screenshots of the MyTravel Admin Dashboard. Both screenshots feature a header with the 'MyTravel Admin Dashboard' logo and navigation links for 'Dashboard', 'User Management', 'Bookings', 'Blog Management', 'Quick Links', and 'Main Site'. Below the header, each screenshot is divided into two main sections: 'Analytics Charts' and 'Recent Activity'.

Analytics Charts Section:

- User Statistics:** Shows metrics like Total Bookings (9), Pending Bookings (1), Total Revenue (\$7,546), and Bookings This Month (8).
- System Status:** Provides an overview of platform health with three operational components: API Server, Database, and Authentication.
- Quick Actions:** Includes links for 'Manage Users' (View, search, and manage user accounts) and 'View Bookings' (Manage travel bookings and update statuses).

Recent Activity Section:

- User Statistics:** Shows metrics like Total Bookings (9), Pending Bookings (1), Total Revenue (\$7,546), and Bookings This Month (8).
- System Status:** Provides an overview of platform health with three operational components: API Server, Database, and Authentication.
- Quick Actions:** Includes links for 'Manage Users' (View, search, and manage user accounts) and 'View Bookings' (Manage travel bookings and update statuses).

Below these sections, there are descriptive labels: 'Trend visualizations' under the Analytics Charts section and 'Latest user actions' under the Recent Activity section.

User Management

Route: [/admin/users](#)

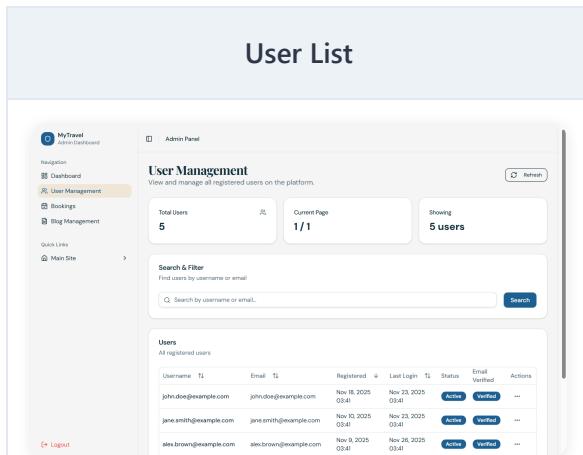
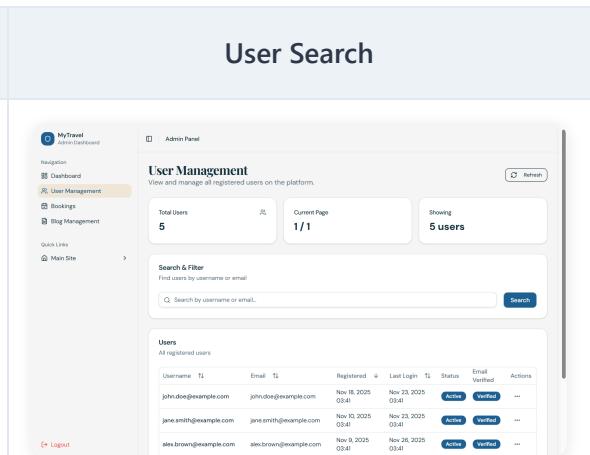
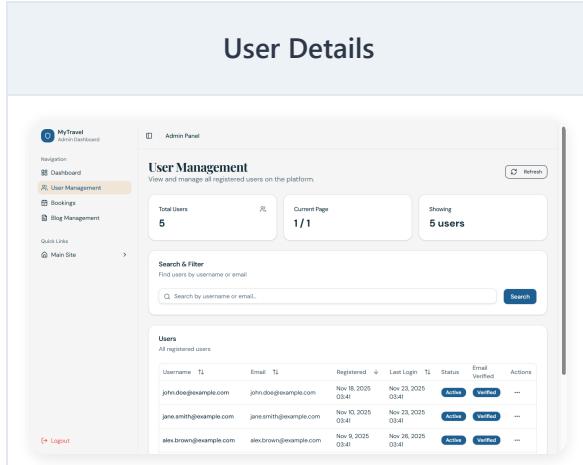
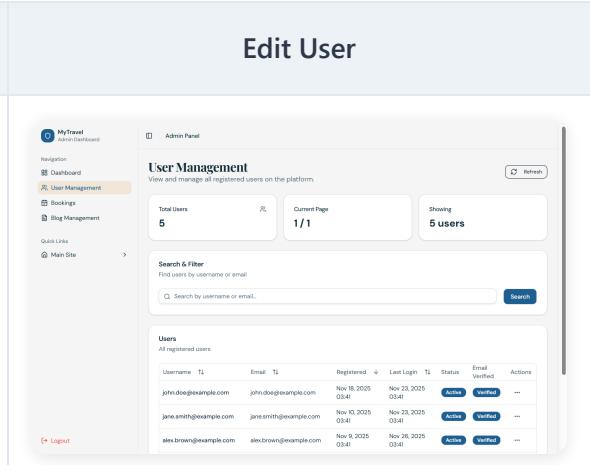
The user management page displays a searchable, paginated table of all registered users. The search functionality allows administrators to find users by name, email, or other attributes. Pagination controls handle large user bases efficiently.

Management Capabilities:

- View detailed user profiles
- Edit user information
- Toggle account active/inactive status
- Delete user accounts
- View registration dates and last login times
- Monitor user activity patterns

Administrators can activate or deactivate accounts as needed for moderation purposes. The interface provides clear visual indicators of account status and important dates.

Screenshots

 <p>User List</p> <p>Paginated user table</p>	 <p>User Search</p> <p>Search by name or email</p>
 <p>User Details</p> <p>Full user profile view</p>	 <p>Edit User</p> <p>Modify user information</p>

Toggle Status

The screenshot shows the 'User Management' section of the admin dashboard. It displays a table of registered users with columns for Username, Email, Registered, Last Login, Status, and Actions. The 'Actions' column contains buttons for 'Active' and 'Verified'. A search bar at the top allows filtering by username or email.

Delete Confirmation

The screenshot shows the same 'User Management' section, but with a modal dialog open. The dialog title is 'Delete Confirmation' and it asks 'Are you sure you want to delete this user?'. It includes a 'Cancel' button and a 'Delete' button.

Activate/deactivate accounts

Delete confirmation dialog

Booking Management

Route: </admin/bookings>

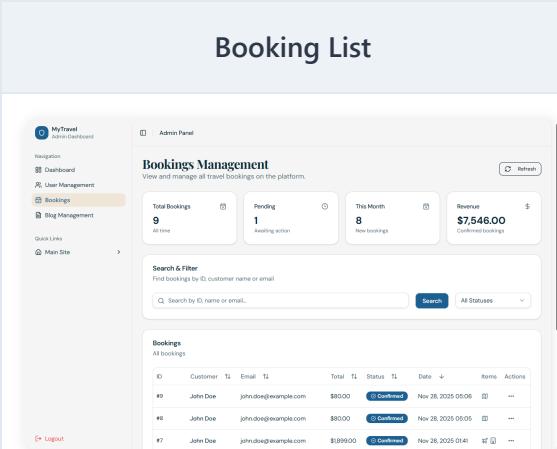
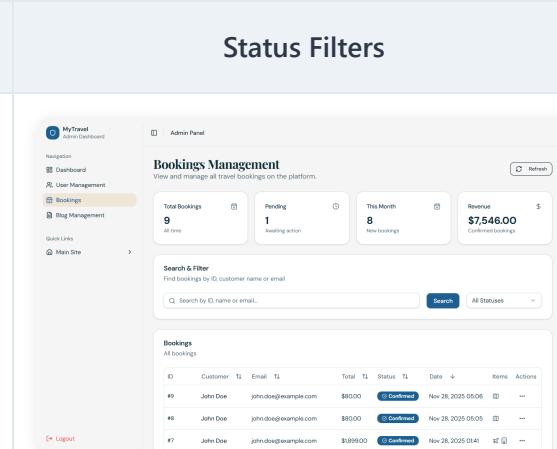
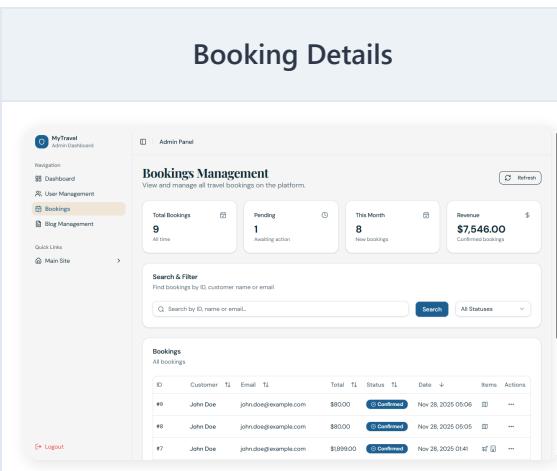
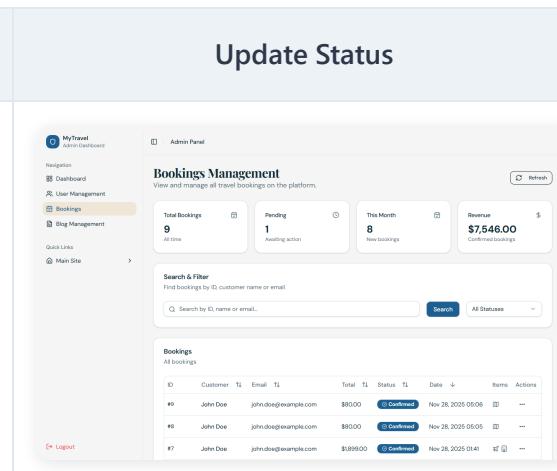
This page provides complete visibility into all bookings made through the platform. The booking list can be filtered by status categories: pending (awaiting confirmation), confirmed (approved and scheduled), cancelled (user or admin cancelled), and completed (past bookings).

Management Features:

- Search bookings by customer name or email
- Filter by booking status
- View detailed booking information
- Update booking status
- Track revenue by period
- Export booking data

Administrators can view comprehensive details for each booking including customer information, itemized services, payment status, and timeline. Status updates trigger notification emails to customers, keeping them informed of any changes.

Screenshots

 <p>Booking List</p> <p>All bookings overview</p>	 <p>Status Filters</p> <p>Filter by status</p>
 <p>Booking Details</p> <p>Full booking information</p>	 <p>Update Status</p> <p>Change booking status</p>

Revenue Stats

Bookings Management

Total Bookings: 9, Pending: 1, This Month: 8, Revenue: \$7,546.00

ID	Customer	T1	Email	T2	Total	T3	Status	T4	Date	Actions
#9	John Doe		john.doe@example.com		\$80.00		Confirmed		Nov 28, 2020 05:06	View Edit Delete
#10	John Doe		john.doe@example.com		\$80.00		Confirmed		Nov 28, 2020 05:05	View Edit Delete
#7	John Doe		john.doe@example.com		\$199.00		Confirmed		Nov 26, 2020 01:41	View Edit Delete

Search Bookings

Bookings Management

Total Bookings: 9, Pending: 1, This Month: 8, Revenue: \$7,546.00

ID	Customer	T1	Email	T2	Total	T3	Status	T4	Date	Actions
#9	John Doe		john.doe@example.com		\$80.00		Confirmed		Nov 28, 2020 05:06	View Edit Delete
#10	John Doe		john.doe@example.com		\$80.00		Confirmed		Nov 28, 2020 05:05	View Edit Delete
#7	John Doe		john.doe@example.com		\$199.00		Confirmed		Nov 26, 2020 01:41	View Edit Delete

Financial metrics
Find specific bookings

Blog Management

Route: </admin/blog>

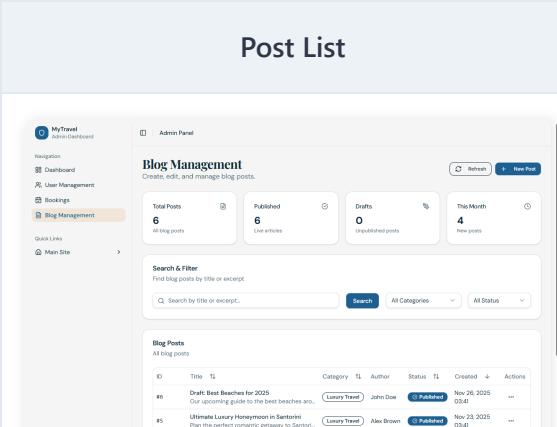
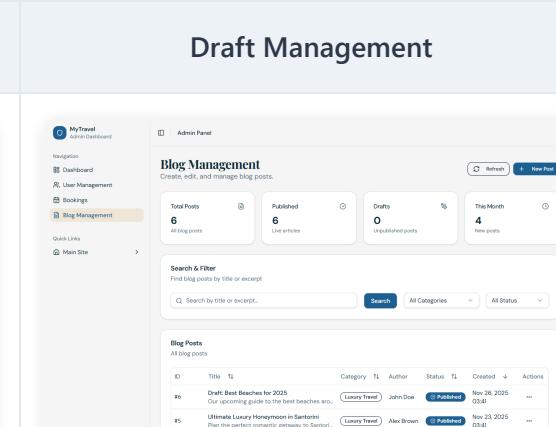
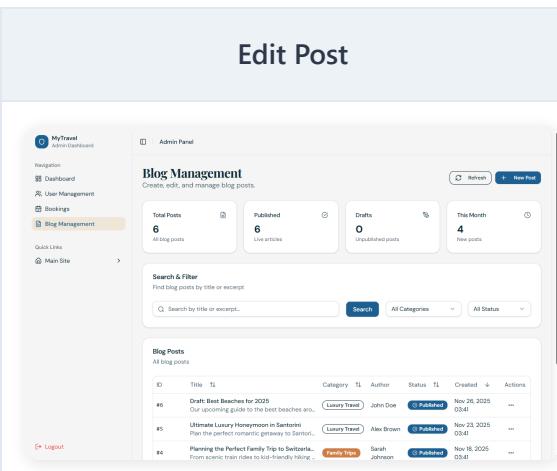
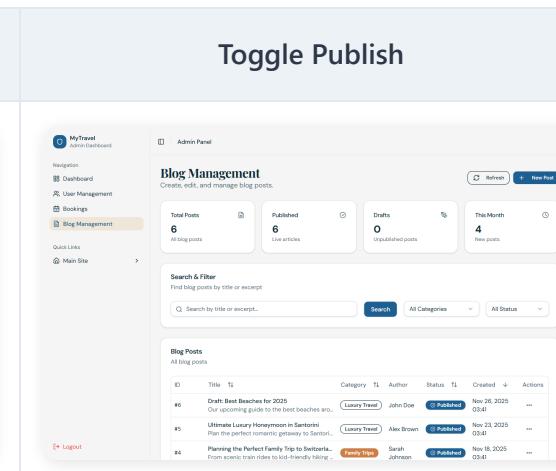
The blog management interface provides administrative control over all blog content. The page lists all posts regardless of publication status, giving administrators visibility into both published articles and pending drafts.

Management Capabilities:

- View all posts (published and drafts)
- Edit existing content
- Toggle publication status
- Delete posts
- Review content before publication
- Monitor blog statistics

Statistics display engagement metrics for blog content, helping administrators understand which topics resonate with readers. Content moderation tools ensure quality and appropriateness of community contributions.

Screenshots

	
Post List <i>All posts overview</i>	Draft Management <i>Pending draft posts</i>
	
Edit Post <i>Modify post content</i>	Toggle Publish <i>Publish/unpublish posts</i>

Blog Stats

Delete Confirmation

Engagement metrics

Delete confirmation

Search and Filtering

The application provides comprehensive search and filtering capabilities throughout the platform. A global search bar in the navigation allows users to find destinations and blog articles from any page.

Search Features:

- Destination search by name or keyword
- Blog article search by title or content
- Real-time search suggestions

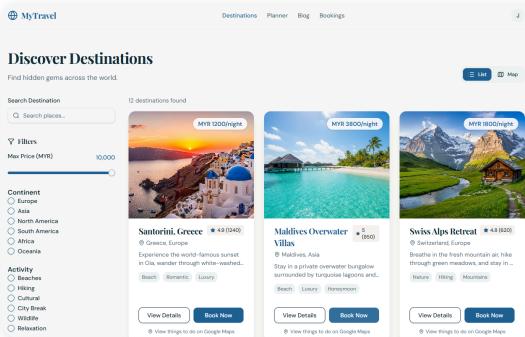
Advanced Filters:

- Price range sliders
- User rating thresholds
- Seasonal availability (best time to visit)
- Activity type categorization
- Continent and region selection

Filters can be combined to narrow results precisely. Filter selections are preserved during navigation, allowing users to refine their search incrementally.

Screenshots

Global Search	Search Suggestions
	
Search from navigation	Real-time suggestions

Advanced Filters	Combined Filters
	
Multiple filter options	Refined search results

Core Features

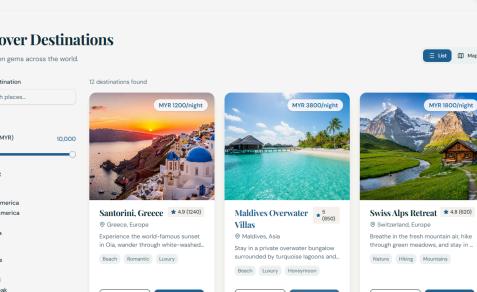
Shopping Cart

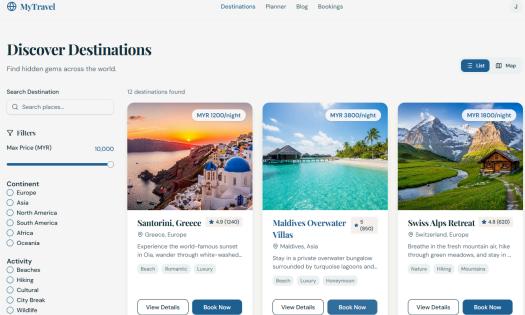
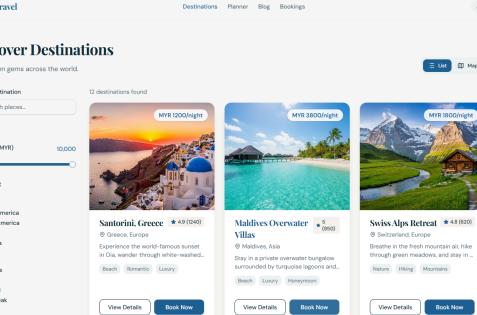
The shopping cart is accessible from any page via a slide-out sheet component triggered by the cart icon in the navigation. It displays all items the user has added, including flights, hotels, and tours with their respective details, dates, and prices.

Users can review their selections, adjust quantities where applicable, remove unwanted items, and see a running total of their planned expenditure. The cart persists across

sessions for authenticated users, ensuring selections are not lost. A prominent checkout button guides users to complete their booking when ready.

Screenshots

Cart Closed	Cart Open
	
<i>Cart indicator in nav</i>	<i>Slide-out cart panel</i>

Cart Items	Cart Actions
	
<i>Itemized service list</i>	<i>Remove and adjust items</i>

Responsive Design and Styling

The entire application is built with a visually appealing and user-friendly layout designed to enhance the user experience. High-quality images showcase destinations and create visual appeal throughout the site. Clear typography using modern font stacks ensures readability across all devices.

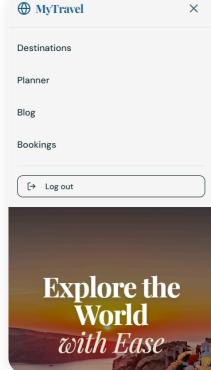
Responsive design principles implemented with Tailwind CSS ensure the interface adapts seamlessly to different screen sizes. On mobile devices, navigation collapses into an

accessible hamburger menu, images resize appropriately, and layouts stack vertically for comfortable scrolling. Tablet and desktop views utilize available screen space with multi-column layouts and expanded navigation.

Intuitive navigation patterns guide users through the site logically. Consistent visual hierarchy helps users understand page structure and find information quickly. Interactive elements provide clear feedback through hover states, focus indicators, and transition animations powered by Framer Motion.

Screenshots

Desktop Layout	Mobile Layout
	
<i>Full desktop experience</i>	<i>Mobile-optimized view</i>

Mobile Navigation	Tablet View
	
<i>Hamburger menu expanded</i>	<i>Tablet layout</i>

Error Handling

"Ensure smooth user experiences by providing meaningful error messages."

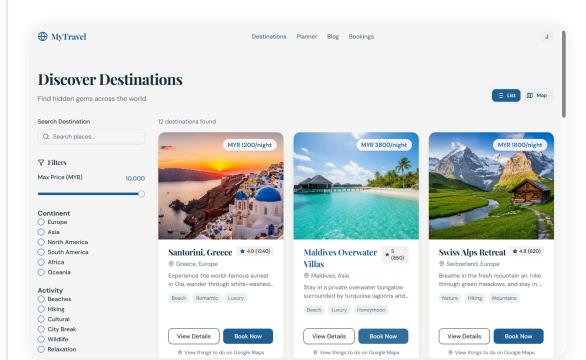
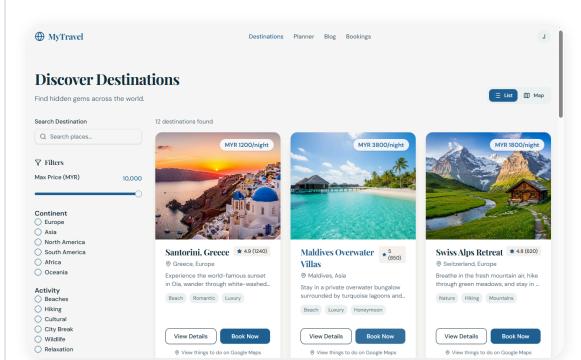
The application implements comprehensive error handling to maintain a smooth user experience even when issues occur. Payment failures display clear messages explaining the issue and suggesting resolution steps. Unavailable services are communicated with helpful alternatives when possible.

Network errors are caught and displayed with retry options. Form validation errors appear inline next to the relevant fields with specific guidance on how to correct the input. Server errors are logged for debugging while users see friendly messages that avoid technical jargon.

Toast notifications provide non-intrusive feedback for both successful actions and error conditions. The notification system uses the Sonner library to display consistent, accessible messages that automatically dismiss after an appropriate duration.

For detailed information about error handling implementation, see [Error Handling](#).

Screenshots

Form Validation	Network Error
 <p>The screenshot shows the 'Discover Destinations' section of the MyTravel website. It features a search bar, filters for continent and activity, and three destination cards: Santorini, Greece; Maldives Overwater Villas; and Swiss Alps Retreat. Each card includes a price (e.g., MYR 1200/night), a rating (e.g., 4.9 (140)), and a 'View Details' or 'Book Now' button. Below each card is an inline validation message: '12 destinations found' (for the search bar) and '12 destinations found' (for the filters). A note at the bottom says 'View things to do on Google Maps'.</p> <p><i>Inline validation messages</i></p>	 <p>The screenshot shows the same 'Discover Destinations' section as the first screenshot, but with a network error. The destination cards are identical, but the validation messages are replaced by error messages: 'An error occurred while loading the results' (for the search bar) and 'An error occurred while loading the filters' (for the filters). A note at the bottom says 'View things to do on Google Maps'.</p> <p><i>Connection error handling</i></p>

<h2 style="text-align: center;">Toast Success</h2>		<h2 style="text-align: center;">Toast Error</h2>	
<p>Discover Destinations</p> <p>Find hidden gems across the world.</p> <p>Search Destination: Q. Search places... 12 destinations found</p> <p>Filters: Max Price (MYR) 10,000</p> <p>Continent: Europe, Asia, North America, South America, Africa, Oceania</p> <p>Activity: Beaches, Hiking, Culture, City Break, Wildlife, Relaxation</p> <p>Santorini, Greece ★ 4.9 (540) MYR 1200/night Experience the world-famous sunset in Oia, wander through white-washed...</p> <p>Maldives Overwater Villas ★ 4.8 (950) MYR 3800/night Stay in a private overwater bungalow surrounded by turquoise lagoons and...</p> <p>Swiss Alps Retreat ★ 4.9 (620) MYR 1800/night Breathe in the fresh mountain air, hike through green meadows, and stay in...</p>		<p>Discover Destinations</p> <p>Find hidden gems across the world.</p> <p>Search Destination: Q. Search places... 12 destinations found</p> <p>Filters: Max Price (MYR) 10,000</p> <p>Continent: Europe, Asia, North America, South America, Africa, Oceania</p> <p>Activity: Beaches, Hiking, Culture, City Break, Wildlife, Relaxation</p> <p>Santorini, Greece ★ 4.9 (540) MYR 1200/night Experience the world-famous sunset in Oia, wander through white-washed...</p> <p>Maldives Overwater Villas ★ 4.8 (950) MYR 3800/night Stay in a private overwater bungalow surrounded by turquoise lagoons and...</p> <p>Swiss Alps Retreat ★ 4.8 (620) MYR 1800/night Breathe in the fresh mountain air, hike through green meadows, and stay in...</p>	
<i>Action confirmation</i>		<i>Error notification</i>	

<h2 style="text-align: center;">404 Page</h2>		<h2 style="text-align: center;">Server Error</h2>	
<p>Discover Destinations</p> <p>Find hidden gems across the world.</p> <p>Search Destination: Q. Search places... 12 destinations found</p> <p>Filters: Max Price (MYR) 10,000</p> <p>Continent: Europe, Asia, North America, South America, Africa, Oceania</p> <p>Activity: Beaches, Hiking, Culture, City Break, Wildlife, Relaxation</p> <p>Santorini, Greece ★ 4.9 (540) MYR 1200/night Experience the world-famous sunset in Oia, wander through white-washed...</p> <p>Maldives Overwater Villas ★ 4.8 (950) MYR 3800/night Stay in a private overwater bungalow surrounded by turquoise lagoons and...</p> <p>Swiss Alps Retreat ★ 4.8 (620) MYR 1800/night Breathe in the fresh mountain air, hike through green meadows, and stay in...</p>		<p>Discover Destinations</p> <p>Find hidden gems across the world.</p> <p>Search Destination: Q. Search places... 12 destinations found</p> <p>Filters: Max Price (MYR) 10,000</p> <p>Continent: Europe, Asia, North America, South America, Africa, Oceania</p> <p>Activity: Beaches, Hiking, Culture, City Break, Wildlife, Relaxation</p> <p>Santorini, Greece ★ 4.9 (540) MYR 1200/night Experience the world-famous sunset in Oia, wander through white-washed...</p> <p>Maldives Overwater Villas ★ 4.8 (950) MYR 3800/night Stay in a private overwater bungalow surrounded by turquoise lagoons and...</p> <p>Swiss Alps Retreat ★ 4.8 (620) MYR 1800/night Breathe in the fresh mountain air, hike through green meadows, and stay in...</p>	
<i>Page not found</i>		<i>Server error message</i>	

Analytics Integration

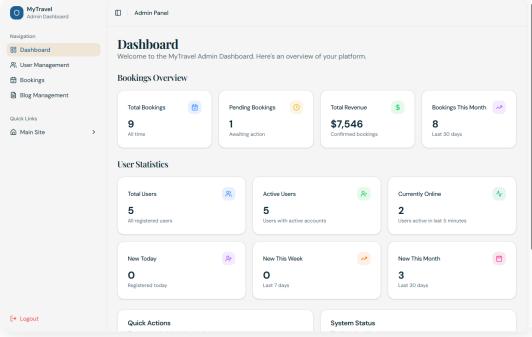
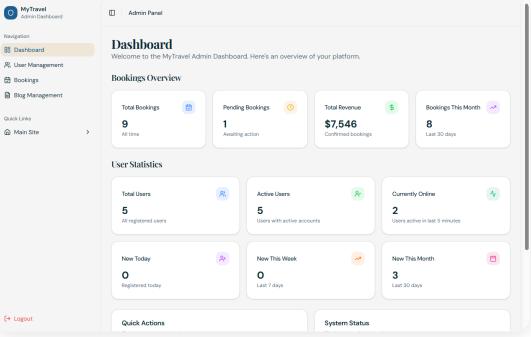
The application integrates PostHog analytics to track user behavior and optimize the platform experience. Analytics capture page views, feature usage patterns, and conversion metrics while respecting user privacy.

Tracked Metrics:

- Page view counts and user flows
- Feature engagement (planner usage, blog reads)
- Booking funnel conversion rates
- Search and filter usage patterns
- Error occurrence frequency

Analytics data informs decisions about content prioritization, feature development, and user experience improvements. The integration is configured to comply with privacy requirements and can be adjusted based on user consent preferences.

Screenshots

Page Analytics	User Flow
 <p>The screenshot shows the MyTravel Admin Dashboard under the 'Page Analytics' section. It features a navigation bar with links for Admin Panel, Dashboard, User Management, Bookings, and Blog Management. Below the navigation is a 'Bookings Overview' section with four cards: 'Total Bookings' (9), 'Pending Bookings' (1), 'Total Revenue' (\$7,546), and 'Bookings This Month' (8). The 'User Statistics' section follows, with three cards: 'Total Users' (5), 'Active Users' (5), and 'Currently Online' (2). At the bottom are 'Quick Actions' and 'System Status' buttons.</p>	 <p>The screenshot shows the MyTravel Admin Dashboard under the 'User Flow' section. The layout is identical to the 'Page Analytics' section, displaying the same overview and statistics. The only difference is the heading at the top.</p>
Page view tracking	User journey visualization

Real-time Form Validation

Forms throughout the application provide immediate feedback using react-hook-form integrated with Zod validation schemas. Users see validation errors as they type, helping them correct issues before submission. This client-side validation reduces server load and improves the user experience by providing instant feedback.

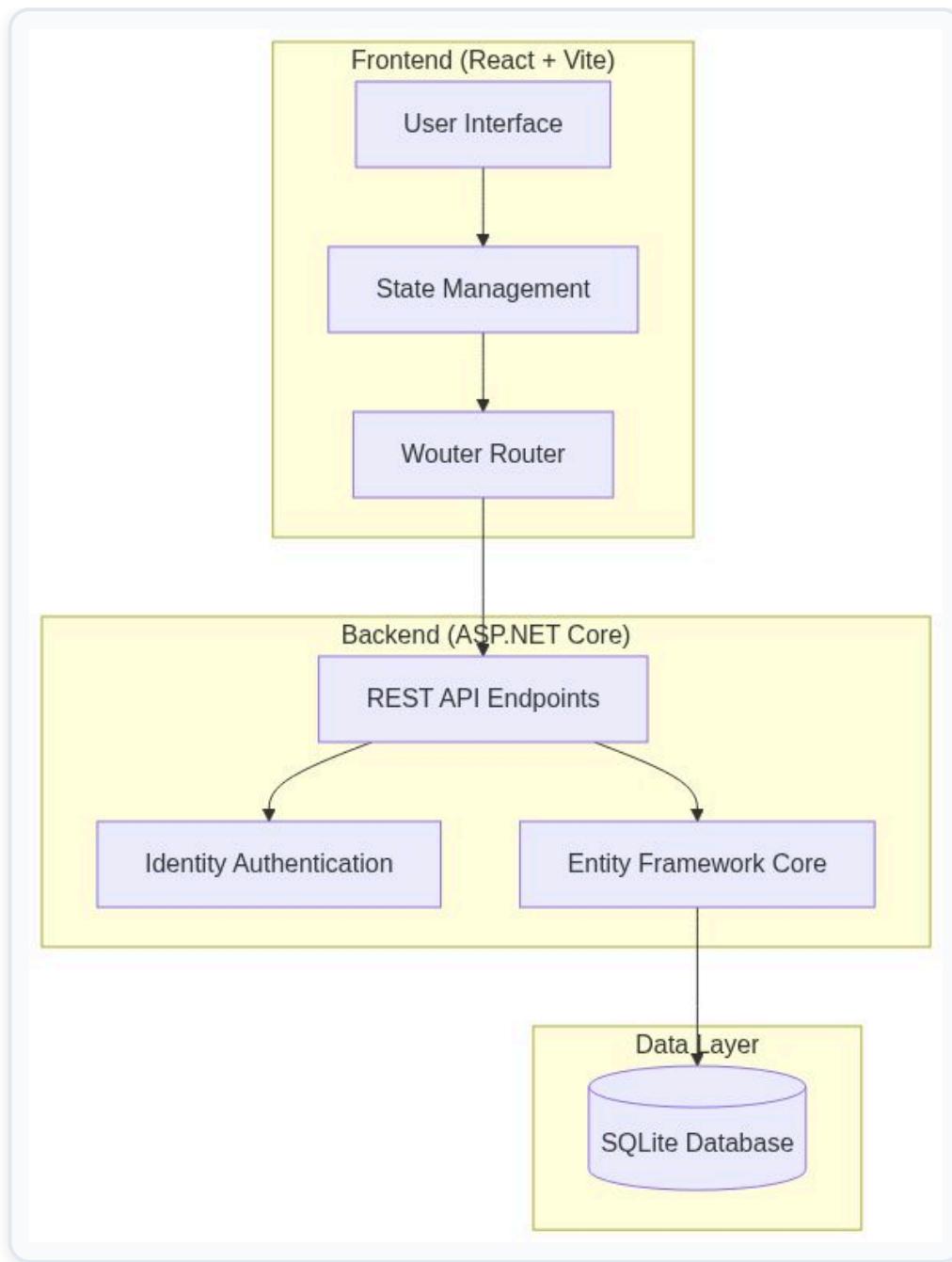
Validation rules enforce data quality requirements including email format verification, password strength requirements, required field completion, and logical constraints such as departure dates preceding return dates.

Architecture

This document provides an overview of the MyTravel application's code structure and organization. The project follows a modern full-stack architecture with a clear separation between the frontend React application and the backend [ASP.NET Core API](#).

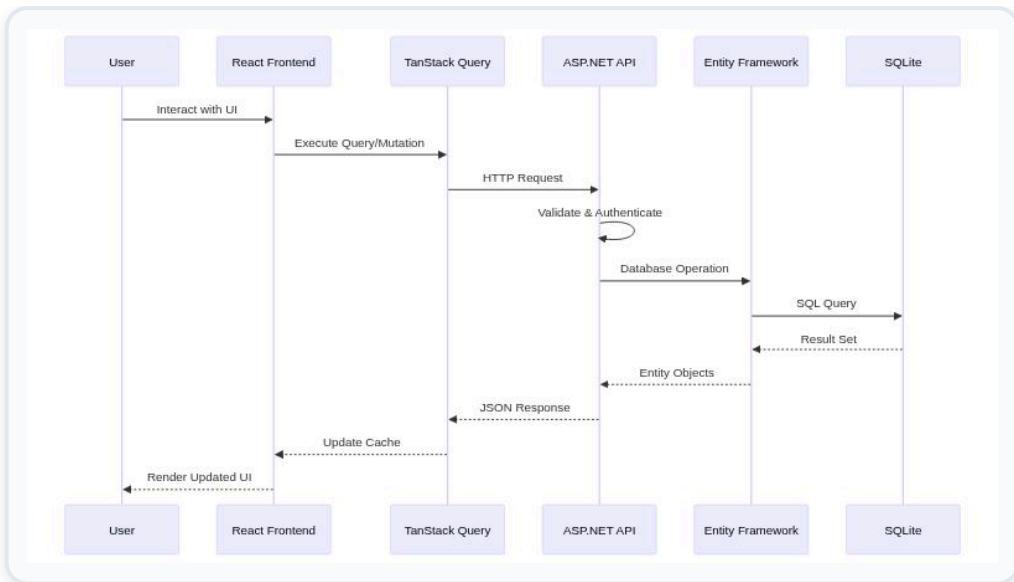
High-Level Architecture

The application consists of two main components that communicate over HTTP. The frontend handles all user interface concerns while the backend manages data persistence, business logic, and authentication.



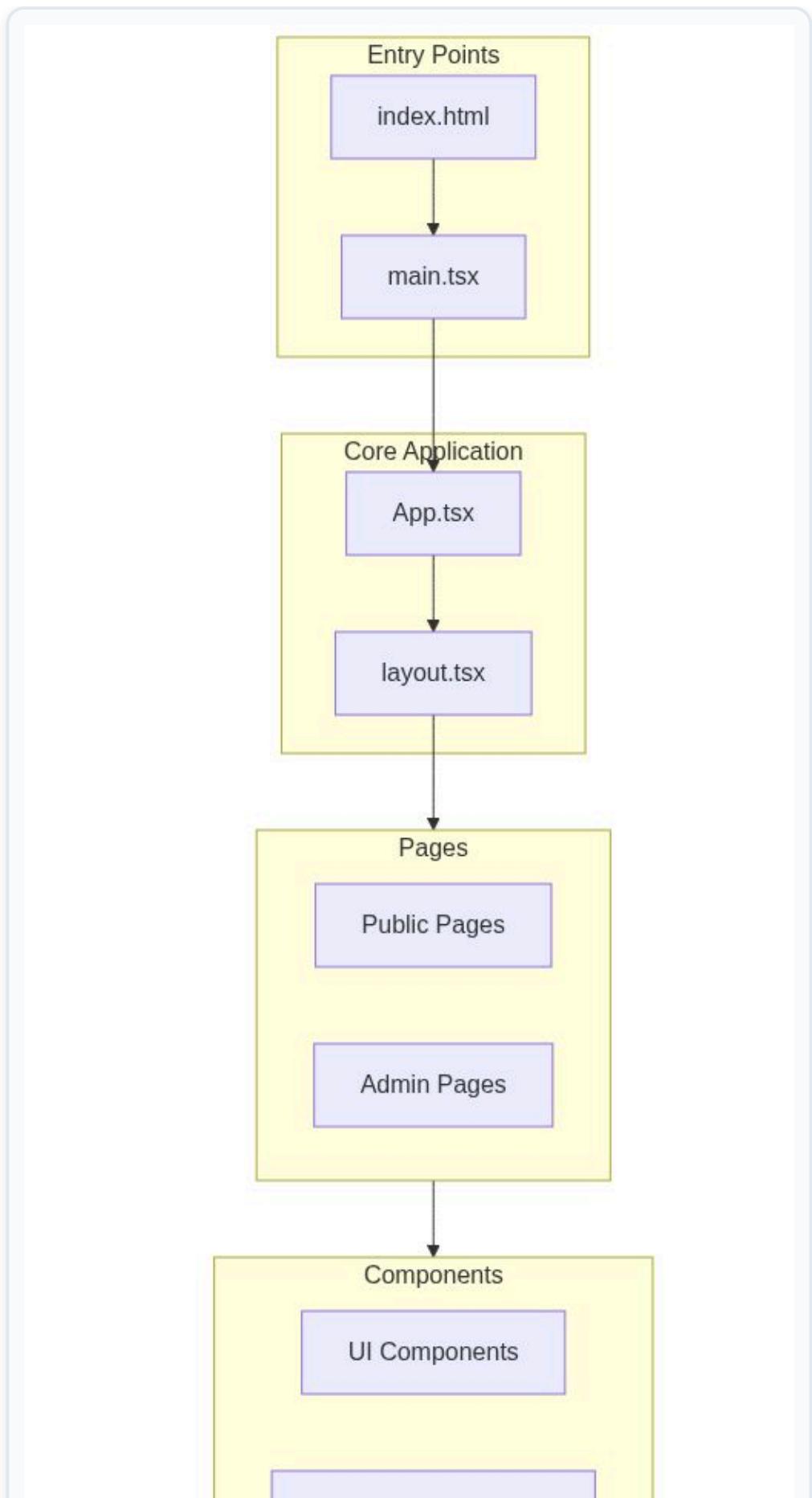
Request Flow

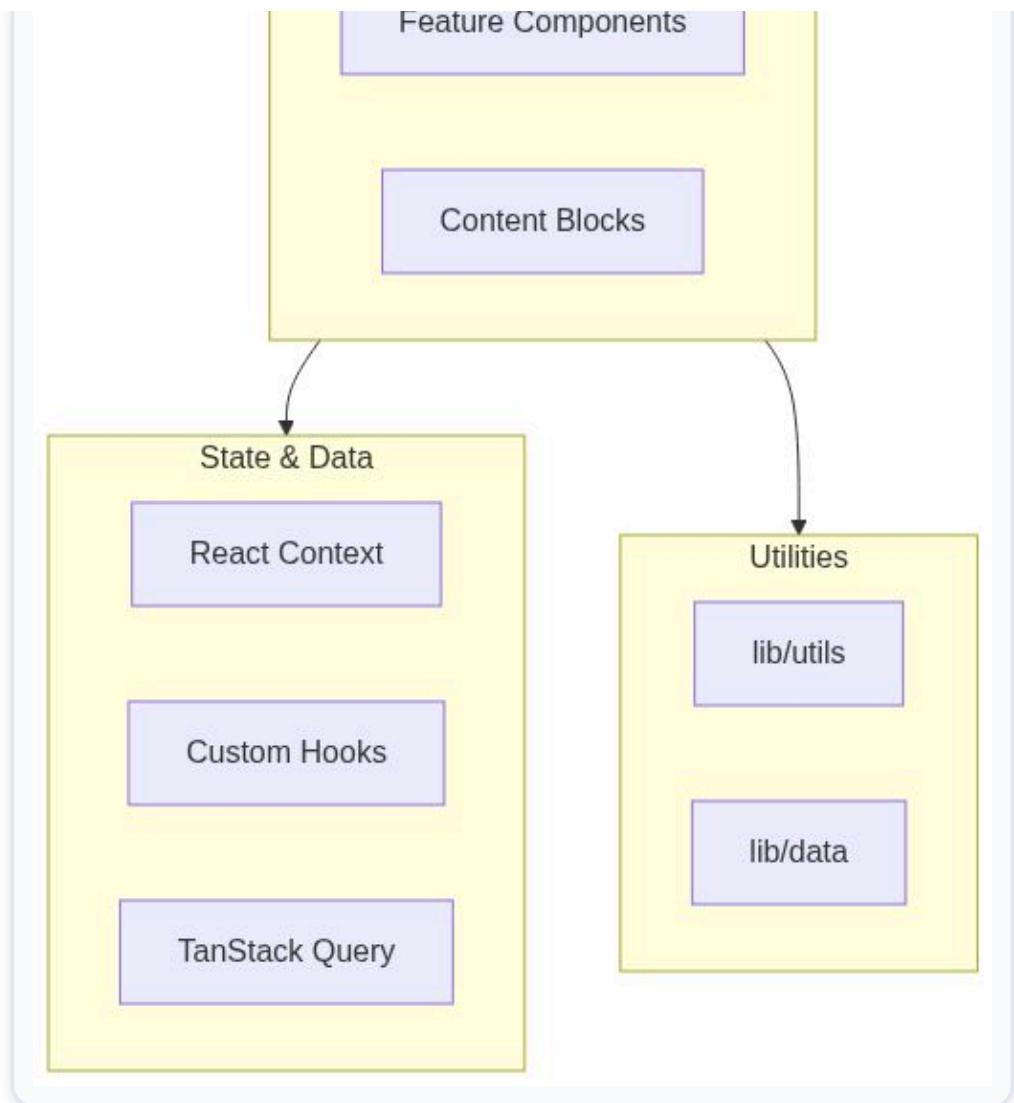
When a user interacts with the application, requests flow through several layers before reaching the database. This diagram illustrates the typical request lifecycle.



Frontend Architecture

The frontend is built with React and TypeScript, using Vite as the build tool. The codebase follows a feature-based organization pattern with clear separation of concerns.





Frontend Directory Structure

```

mytravel.client/
├── client/
│   ├── index.html          # HTML entry point
│   └── public/              # Static assets (images, fonts)
└── src/
    ├── App.tsx             # Root component with routing
    ├── main.tsx             # React entry point
    └── index.css            # Global styles (Tailwind)

    ├── components/          # Reusable components
    │   ├── admin/             # Admin-specific components
    │   │   ├── admin-layout.tsx
    │   │   ├── protected-admin-route.tsx
    │   │   └── ...
    │   ├── blocks/            # Content block components
    │   ├── editor/             # Lexical editor components
    │   └── ui/                # Base UI components (55+)
    │       └── button.tsx
  
```

```
    |   |   |   └── card.tsx
    |   |   └── dialog.tsx
    |   |   └── ...
    |   ├── booking-form.tsx
    |   ├── cart-sheet.tsx
    |   ├── destination-card.tsx
    |   ├── hero.tsx
    |   ├── itinerary-planner.tsx
    |   ├── layout.tsx
    |   └── trip-summary.tsx

    └── context/          # React context providers
        ├── blog-context.tsx  # Blog state management
        └── cart-context.tsx  # Shopping cart state

    └── hooks/           # Custom React hooks
        ├── use-admin-auth.tsx # Admin authentication
        ├── use-auth.tsx       # User authentication
        ├── use-mobile.tsx     # Responsive detection
        ├── use-ping.tsx       # Activity tracking
        └── use-toast.ts       # Toast notifications

    └── lib/              # Utilities and data
        ├── data.ts          # API client functions
        ├── destinations.ts  # Destination data
        ├── mock-data.ts     # Development mock data
        ├── queryClient.ts   # TanStack Query config
        └── utils.ts          # Helper functions

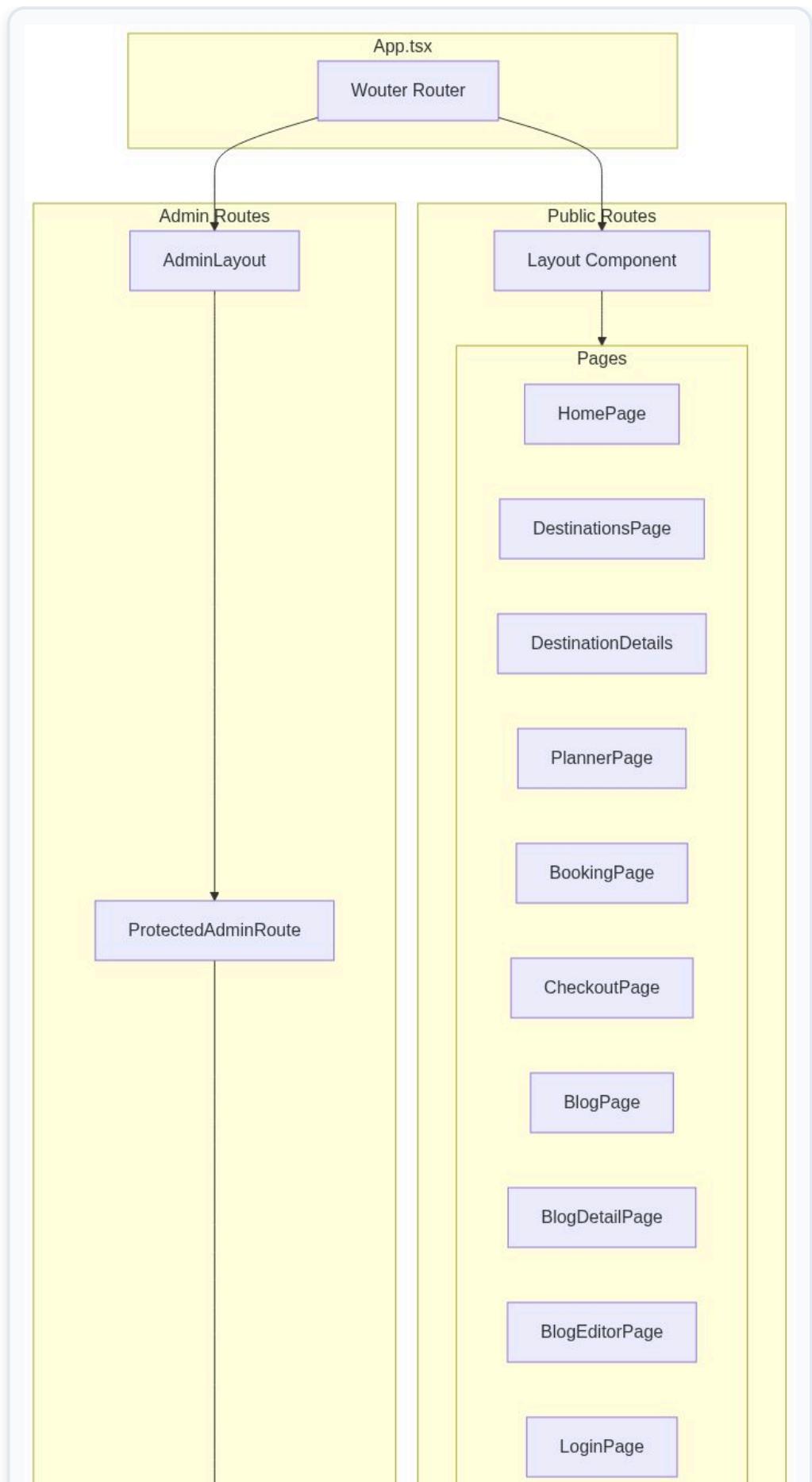
    └── pages/            # Page components
        └── admin/           # Admin pages
            ├── admin-blog.tsx
            ├── admin-bookings.tsx
            ├── admin-dashboard.tsx
            ├── admin-login.tsx
            └── admin-users.tsx
            ├── blog.tsx
            ├── blog-detail.tsx
            ├── blog-editor.tsx
            ├── booking.tsx
            ├── checkout.tsx
            ├── destination-details.tsx
            ├── destinations.tsx
            └── home.tsx
```

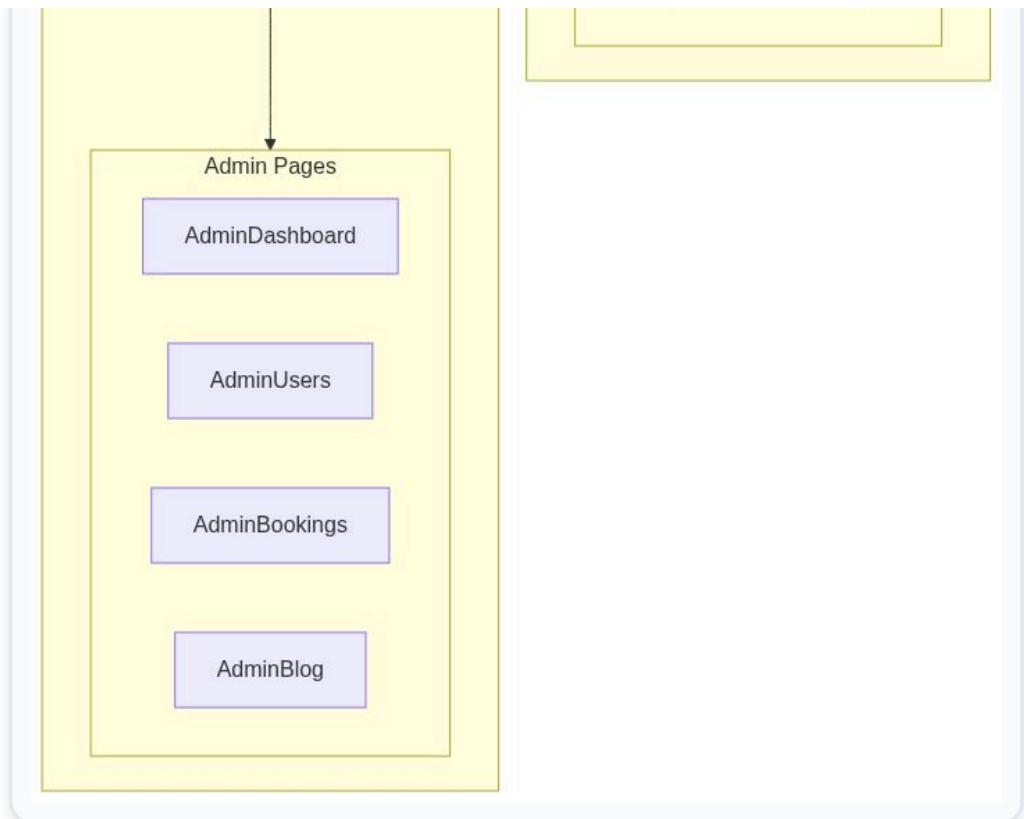
```
|- login.tsx
|- not-found.tsx
└── planner.tsx

├── attached_assets/          # Generated images
├── components.json           # shadcn/ui configuration
├── eslint.config.js          # ESLint configuration
├── package.json               # Dependencies and scripts
├── postcss.config.js         # PostCSS configuration
├── tailwind.config.js        # Tailwind CSS configuration
├── tsconfig.json              # TypeScript configuration
└── vite.config.ts            # Vite build configuration
```

Component Hierarchy

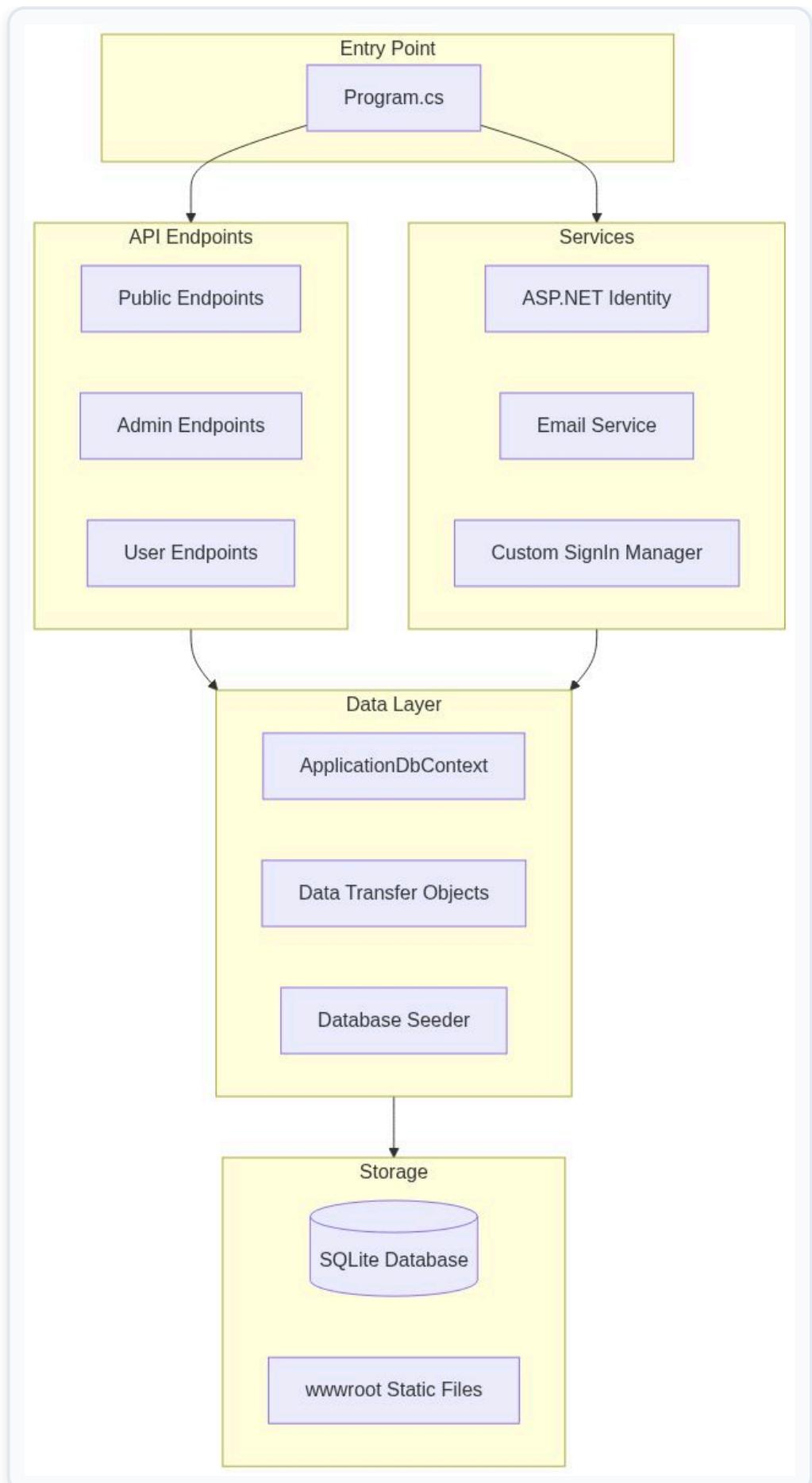
This diagram shows the relationship between major frontend components and how they compose to form the user interface.





Backend Architecture

The backend is built with [ASP.NET](#) Core using a minimal API approach. It follows a layered architecture with clear separation between endpoints, data access, and business logic.

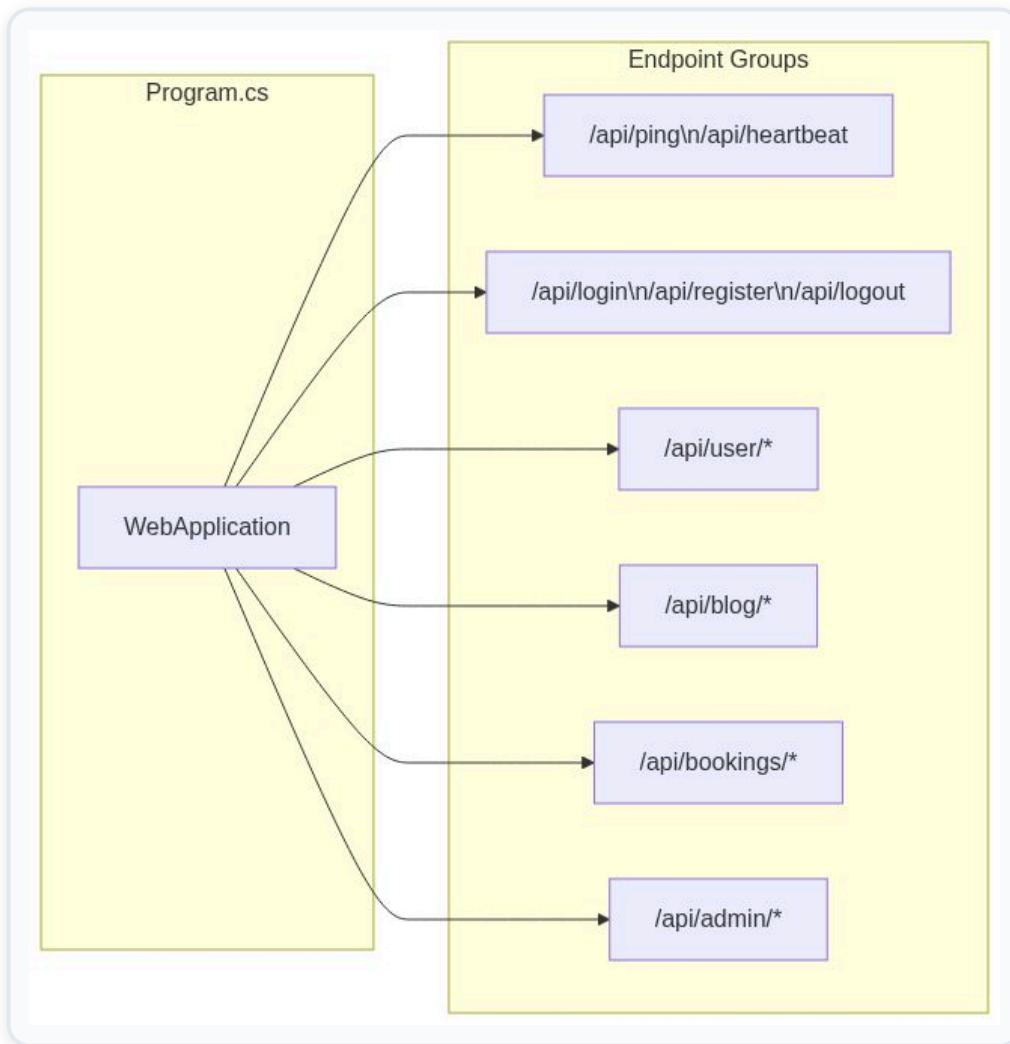


Backend Directory Structure

```
MyTravel.Server/
├── Program.cs          # Application entry point
|   |                   # - Service configuration
|   |                   # - Middleware pipeline
|   |                   # - Endpoint registration
|
├── Data/
|   └── ApplicationDbContext.cs    # EF Core context
|       |                   # - DbSet definitions
|       |                   # - Entity configurations
|       |                   # - Model definitions
|       └── DbSeeder.cs        # Database seeding
|           |                   # - Sample destinations
|           |                   # - Demo blog posts
|           |                   # - Admin user creation
|
├── DTOs/
|   ├── BlogDtos.cs      # Blog request/response models
|   ├── BookingDtos.cs   # Booking request/response models
|   ├── UserDtos.cs      # User request/response models
|   └── WeatherForecast.cs # Sample DTO
|
├── Endpoints/
|   ├── ActivityEndpoints.cs    # API endpoint definitions
|   ├── AdminBlogEndpoints.cs   # Ping and heartbeat tracking
|   ├── AdminBookingEndpoints.cs # Admin blog CRUD operations
|   ├── AdminEndpoints.cs       # Admin booking management
|   ├── BlogEndpoints.cs       # Admin auth and user management
|   ├── BookingEndpoints.cs    # Public blog endpoints
|   ├── UserEndpoints.cs       # Public booking endpoints
|   └── WeatherEndpoints.cs    # User profile endpoints
|       |                   # Sample weather endpoint
|
├── Services/
|   ├── CustomSignInManager.cs  # Business logic services
|   └── EmailSender.cs         # Extended sign-in logic
|       |                   # Email sending service
|
├── Properties/
|   └── launchSettings.json    # Email sending service
|
├── Views/
|   └── PageBlockTypes/        # Development launch profiles
|
├── wwwroot/                # Server-side views (if any)
|
└── appsettings.json         # Static file serving
└── appsettings.Development.json # Base configuration
└── MyTravel.Server.csproj    # Development configuration
└── MyTravel.Server.http      # Project file
                            # HTTP request testing
```

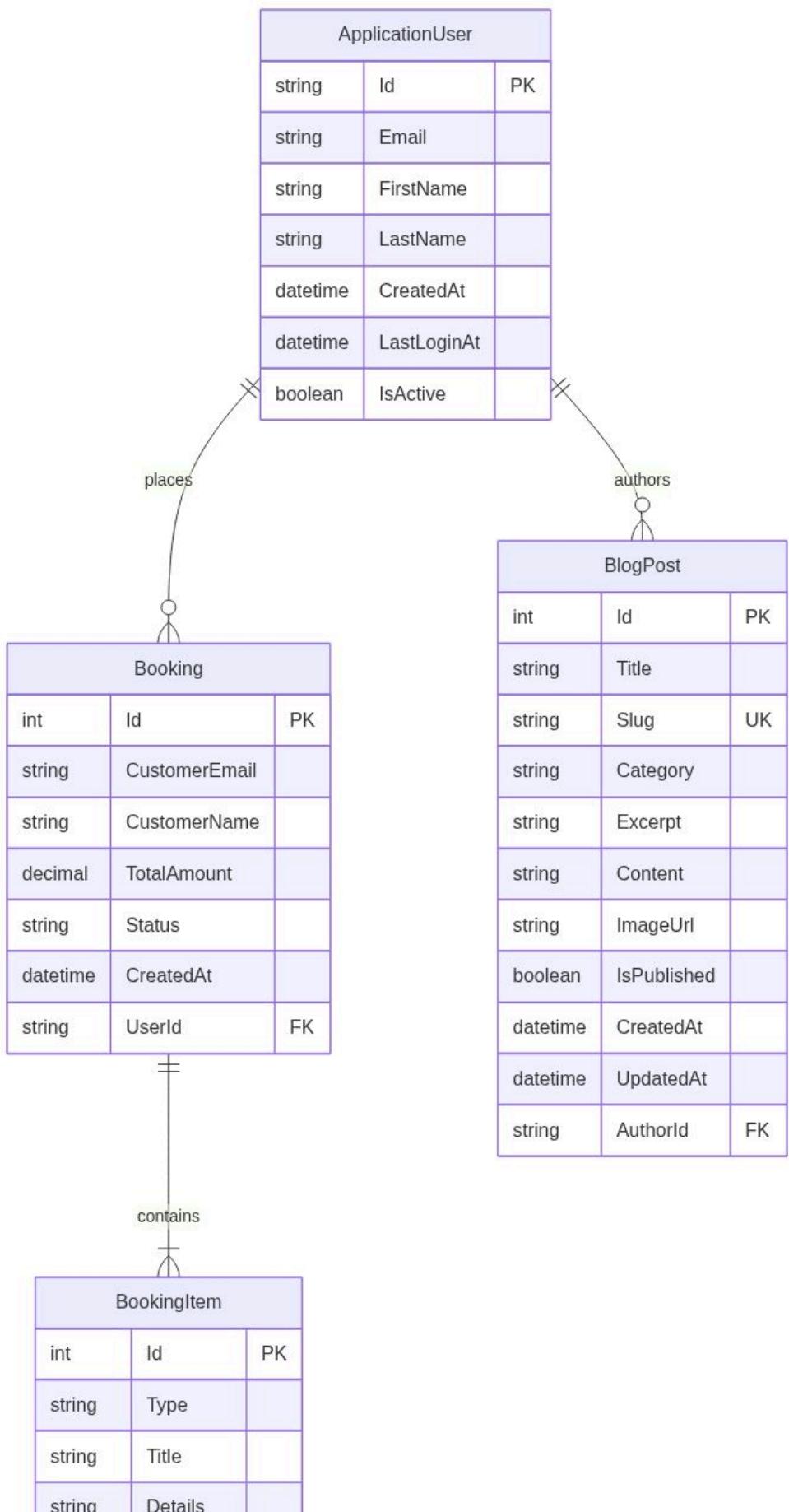
API Endpoint Organization

The API endpoints are organized into logical groups using [ASP.NET Core's minimal API](#) pattern with extension methods.



Data Model

The application uses Entity Framework Core with SQLite for data persistence. The data model centers around users, bookings, and blog content.

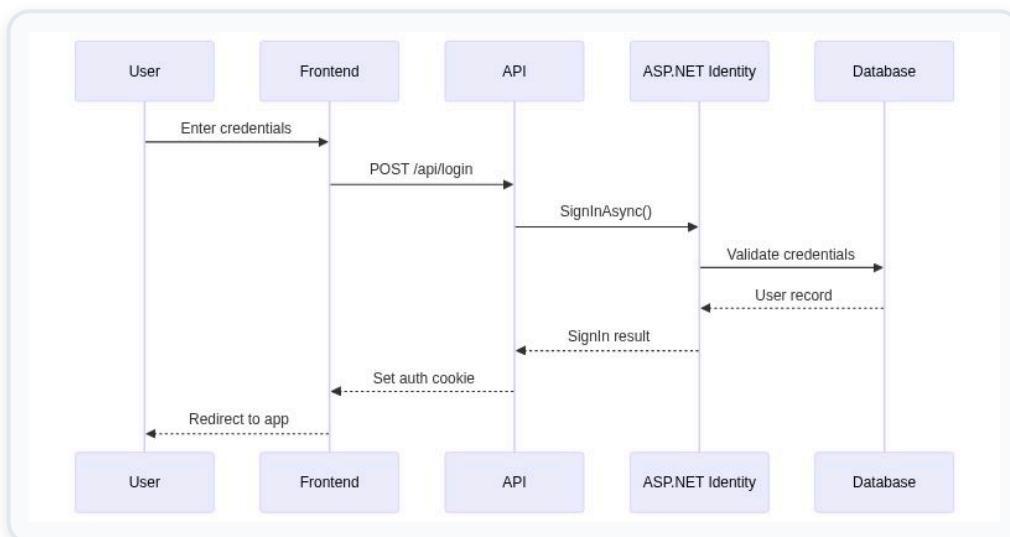


decimal	Price	
int	BookingId	FK

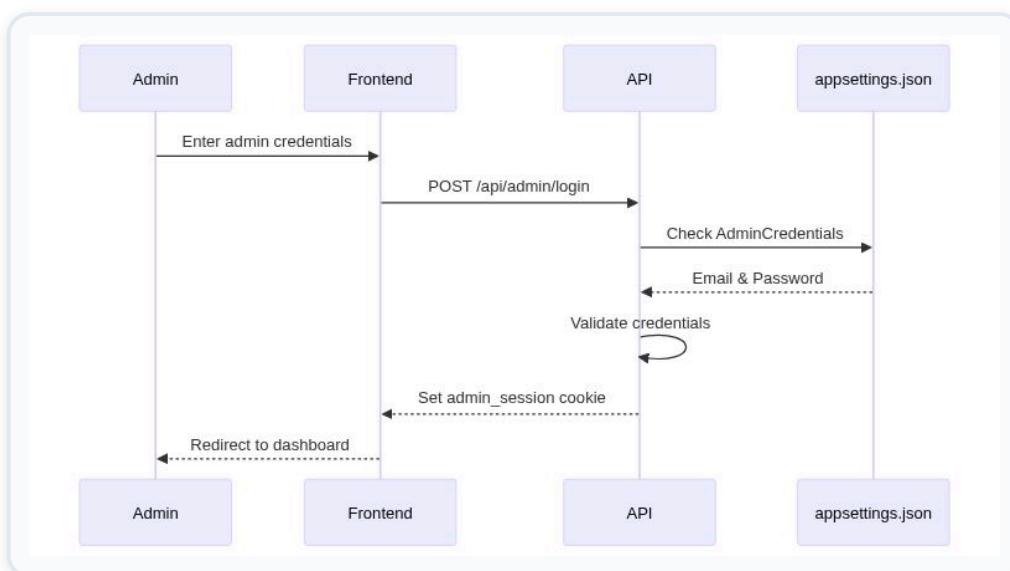
Authentication Flow

The application implements two separate authentication flows: one for regular users and one for administrators.

User Authentication

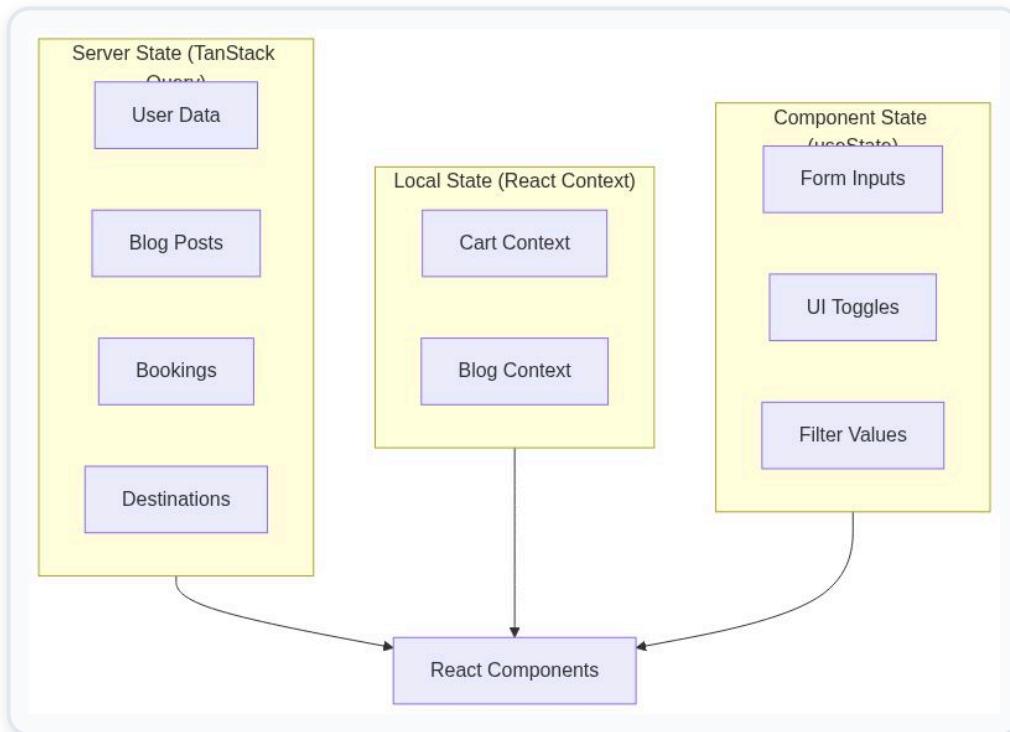


Admin Authentication



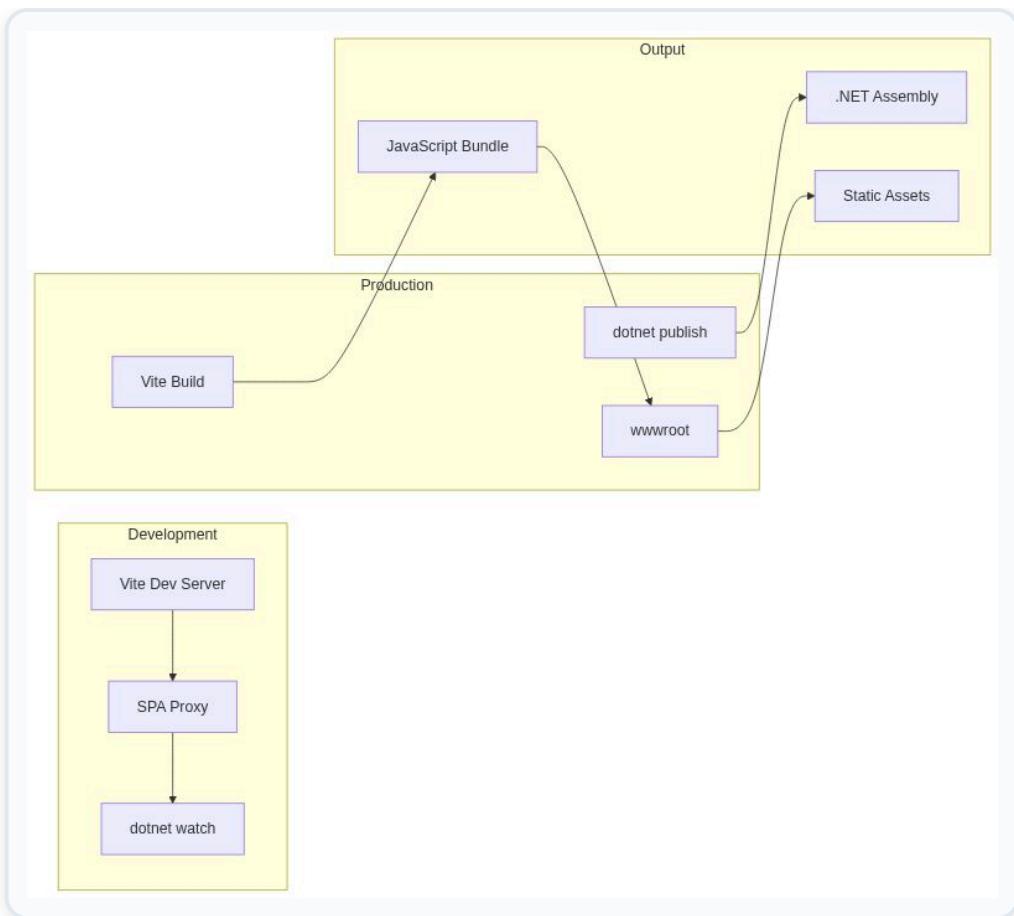
State Management

The frontend uses a combination of React Context and TanStack Query for state management. Local UI state is managed with React hooks, while server state is cached and synchronized using TanStack Query.



Build and Deployment

The application uses different build configurations for development and production environments.



Development Mode

In development, the Vite development server runs on port 49764 and proxies API requests to the [ASP.NET](#) Core server on port 5083. The SPA Proxy middleware automatically starts the Vite server when the .NET application launches.

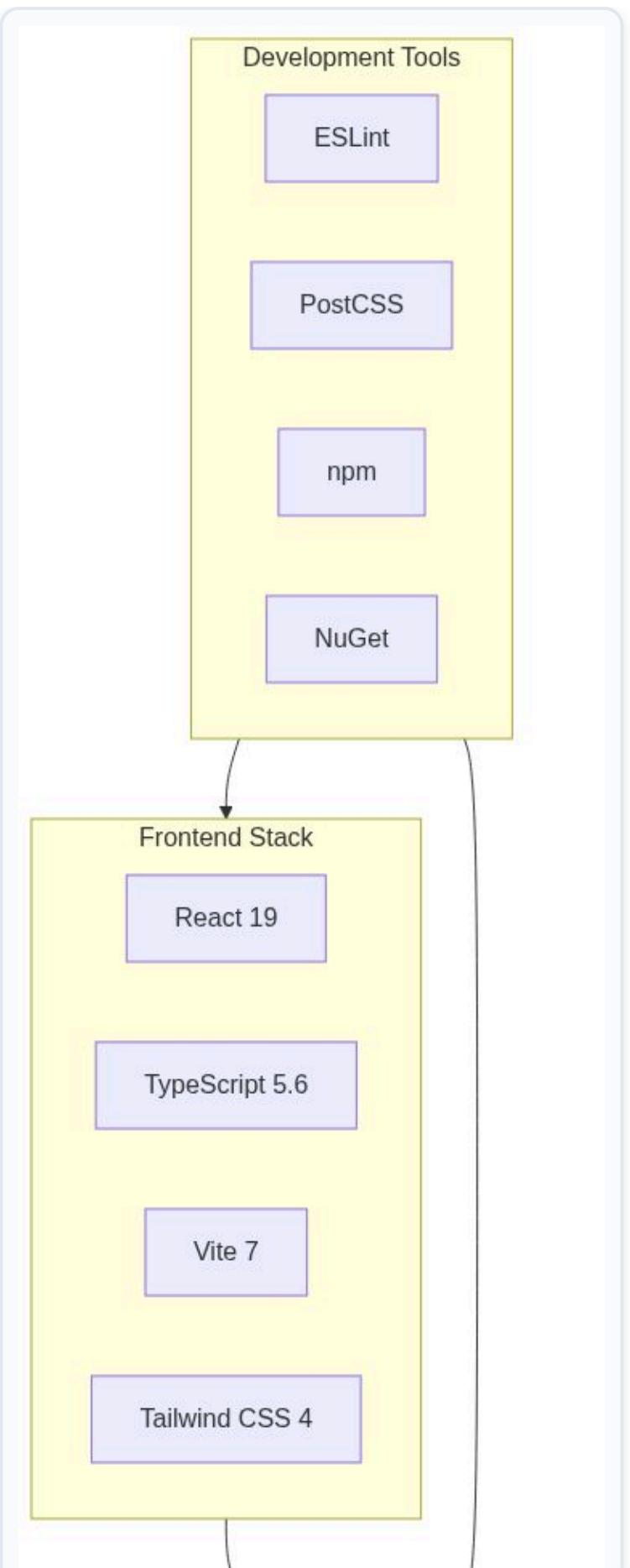
Production Mode

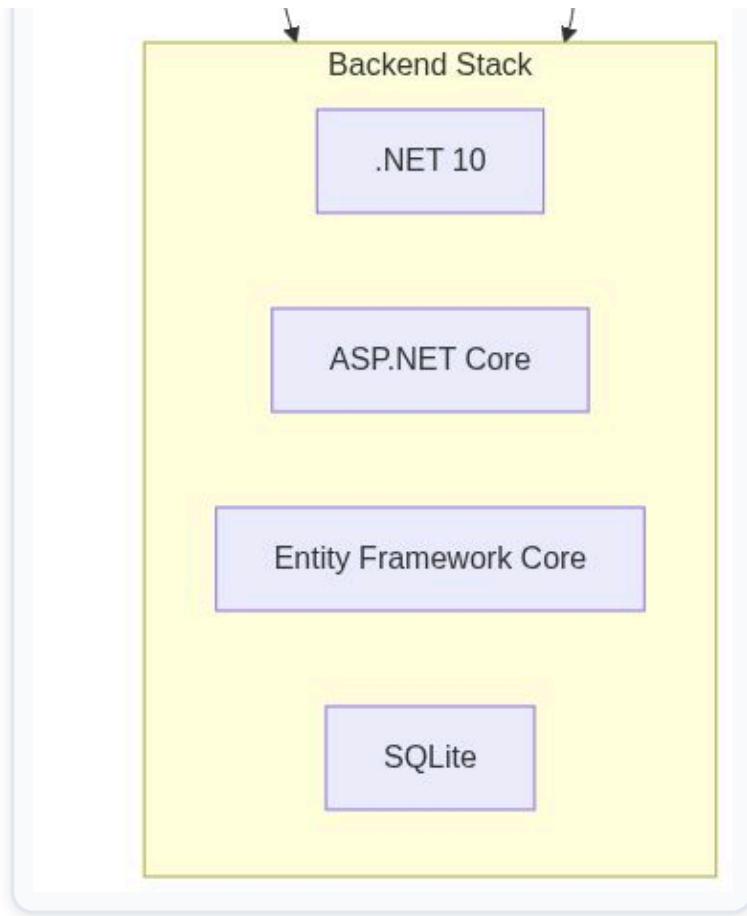
For production, Vite builds optimized static assets that are served from the [ASP.NET](#) Core wwwroot directory. The .NET application serves both the API and the static frontend files.

Technologies

This document provides a comprehensive overview of the libraries, frameworks, and tools used in the MyTravel application. The project leverages modern technologies for both frontend and backend development to deliver a robust and maintainable codebase.

Technology Stack Overview





Frontend Technologies

The frontend is built with React and TypeScript, utilizing a modern toolchain for development and production builds.

Core Framework

Library	Version	Purpose
React	[^] 19.2.0	The core UI library for building component-based user interfaces. React 19 introduces improved performance and new features like automatic batching and concurrent rendering.
React DOM	[^] 19.2.0	Provides DOM-specific methods for rendering React components to the browser.
TypeScript	5.6.3	Adds static typing to JavaScript, enabling better tooling, code navigation, and compile-time error detection.

Build Tools

Library	Version	Purpose
Vite	^7.1.9	A modern build tool that provides extremely fast development server startup and hot module replacement (HMR). Vite uses native ES modules during development for instant updates.
@vitejs/plugin-react	^4.5.2	Official Vite plugin for React that enables Fast Refresh and JSX transformation.
ESLint	^9.28.0	A pluggable linting utility for JavaScript and TypeScript. Identifies and reports on patterns found in code to maintain code quality.
PostCSS	(bundled)	A tool for transforming CSS with JavaScript plugins. Used by Tailwind CSS for processing utility classes.

Styling

Library	Version	Purpose
Tailwind CSS	^4.1.14	A utility-first CSS framework that provides low-level utility classes for building custom designs. Enables rapid UI development without leaving HTML.
tailwindcss-animate	^1.0.7	Adds animation utilities to Tailwind CSS for creating smooth transitions and keyframe animations.
class-variance-authority	^0.7.1	A utility for creating variant-based component styles. Simplifies conditional class application.
clsx	^2.1.1	A utility for constructing className strings conditionally. Works well with Tailwind's utility classes.
tailwind-merge	^3.3.1	Intelligently merges Tailwind CSS classes without style conflicts. Prevents duplicate or conflicting utilities.

Routing

Library	Version	Purpose
wouter	^3.3.5	A minimalist routing library for React. Provides hooks-based routing with a small footprint (~1.5KB). Chosen for its simplicity and performance over React Router.

State Management and Data Fetching

Library	Version	Purpose
@tanstack/react-query	^5.60.5	Powerful data synchronization library for fetching, caching, and updating server state. Handles loading states, error handling, and cache invalidation automatically.
React Context	(built-in)	Used for local application state like shopping cart and blog editor state. Provides component-level state sharing without external dependencies.

UI Components

The application uses Radix UI primitives as the foundation for accessible, unstyled UI components. These are styled with Tailwind CSS to match the application design.

Library	Version	Purpose
<code>@radix-ui/react-accordion</code>	[^] 1.2.11	Collapsible content sections with accessibility support.
<code>@radix-ui/react-alert-dialog</code>	[^] 1.1.14	Modal dialogs for important confirmations and alerts.
<code>@radix-ui/react-aspect-ratio</code>	[^] 1.1.7	Maintains consistent aspect ratios for images and media.
<code>@radix-ui/react-avatar</code>	[^] 1.1.10	User avatar display with fallback support.
<code>@radix-ui/react-checkbox</code>	[^] 1.3.2	Accessible checkbox inputs with custom styling.
<code>@radix-ui/react-collapse</code>	[^] 1.1.11	Expandable/collapsible content areas.
<code>@radix-ui/react-dialog</code>	[^] 1.1.14	Modal dialog windows for forms and content.
<code>@radix-ui/react-dropdown-menu</code>	[^] 2.1.15	Dropdown menus with keyboard navigation.
<code>@radix-ui/react-hover-card</code>	[^] 1.1.14	Content cards that appear on hover.
<code>@radix-ui/react-label</code>	[^] 2.1.7	Accessible form labels.
<code>@radix-ui/react-menubar</code>	[^] 1.1.14	Application menu bars with submenus.
<code>@radix-ui/react-navigation-menu</code>	[^] 1.2.13	Site navigation with dropdowns.
<code>@radix-ui/react-popover</code>	[^] 1.1.14	Floating content panels anchored to triggers.
<code>@radix-ui/react-progress</code>	[^] 1.1.7	Progress indicators and loading bars.

Library	Version	Purpose
<code>@radix-ui/react-radio-group</code>	<code>^1.3.7</code>	Radio button groups with accessibility.
<code>@radix-ui/react-scrollbar-area</code>	<code>^1.2.9</code>	Custom scrollable areas with styled scrollbars.
<code>@radix-ui/react-select</code>	<code>^2.2.5</code>	Custom select dropdowns with search support.
<code>@radix-ui/react-separator</code>	<code>^1.1.7</code>	Visual dividers between content sections.
<code>@radix-ui/react-slider</code>	<code>^1.3.5</code>	Range sliders for numeric input.
<code>@radix-ui/react-slot</code>	<code>^1.2.3</code>	Utility for component composition.
<code>@radix-ui/react-switch</code>	<code>^1.2.5</code>	Toggle switches for boolean settings.
<code>@radix-ui/react-tabs</code>	<code>^1.1.12</code>	Tabbed interfaces for content organization.
<code>@radix-ui/react-toast</code>	<code>^1.2.14</code>	Toast notification system.
<code>@radix-ui/react-toggle</code>	<code>^1.1.9</code>	Toggle buttons for binary states.
<code>@radix-ui/react-toggle-group</code>	<code>^1.1.10</code>	Groups of mutually exclusive toggles.
<code>@radix-ui/react-tooltip</code>	<code>^1.2.7</code>	Informational tooltips on hover.

Forms and Validation

Library	Version	Purpose
<code>react-hook-form</code>	<code>^7.66.0</code>	Performant form library with minimal re-renders. Provides form state management, validation, and submission handling.
<code>@hookform/resolvers</code>	<code>^5.0.1</code>	Validation resolvers for react-hook-form. Connects external validation libraries like Zod.
<code>zod</code>	<code>^3.25.76</code>	TypeScript-first schema validation library. Defines validation rules with full type inference.

Rich Text Editor

Library	Version	Purpose
<code>lexical</code>	<code>^0.38.2</code>	An extensible text editor framework by Meta. Provides the foundation for the blog post editor with support for rich formatting.
<code>@lexical/react</code>	<code>^0.38.2</code>	React bindings for Lexical editor.
<code>@lexical/rich-text</code>	<code>^0.38.2</code>	Rich text editing capabilities including bold, italic, headings.
<code>@lexical/list</code>	<code>^0.38.2</code>	Ordered and unordered list support.
<code>@lexical/link</code>	<code>^0.38.2</code>	Hyperlink editing and display.
<code>@lexical/code</code>	<code>^0.38.2</code>	Code block formatting.
<code>@lexical/table</code>	<code>^0.38.2</code>	Table editing support.
<code>@lexical/selection</code>	<code>^0.38.2</code>	Text selection utilities.
<code>@lexical/utils</code>	<code>^0.38.2</code>	Common utility functions.

Maps and Geolocation

Library	Version	Purpose
leaflet	^1.9.4	Open-source JavaScript library for interactive maps. Provides map rendering, markers, and user interaction.
react-leaflet	^5.0.0	React components for Leaflet maps. Enables declarative map configuration in JSX.
@types/leaflet	^1.9.18	TypeScript type definitions for Leaflet.

Drag and Drop

Library	Version	Purpose
@dnd-kit/core	^6.3.1	Core drag and drop functionality. Provides the foundation for the itinerary planner's drag-and-drop interface.
@dnd-kit/sortable	^10.0.0	Sortable list utilities built on dnd-kit. Enables reordering of itinerary items.
@dnd-kit/utilities	^3.2.2	Helper utilities for dnd-kit implementations.

Animations

Library	Version	Purpose
framer-motion	^12.23.24	Production-ready animation library for React. Powers page transitions, micro-interactions, and complex animations throughout the application.

Charts and Data Visualization

Library	Version	Purpose
recharts	^2.15.4	Composable charting library built on React and D3. Used in the admin dashboard for displaying statistics and trends.

Date and Time

Library	Version	Purpose
date-fns	^3.6.0	Modern JavaScript date utility library. Provides functions for parsing, formatting, and manipulating dates.
react-day-picker	^8.10.1	Flexible date picker component. Used for selecting travel dates in booking forms.

Icons

Library	Version	Purpose
lucide-react	^0.545.0	Beautiful, consistent icon set with React components. Provides 1000+ icons used throughout the interface.

Notifications

Library	Version	Purpose
sonner	^2.0.7	Toast notification library with a clean API. Displays success, error, and informational messages to users.

Analytics

Library	Version	Purpose
posthog-js	^1.298.1	Product analytics platform integration. Tracks user behavior, feature usage, and conversion metrics.

Utilities

Library	Version	Purpose
cmdk	^1.1.1	Command menu component for keyboard-driven interfaces.
embla-carousel-react	^8.6.0	Carousel/slider component with touch support.
input-otp	^1.4.2	One-time password input component.
next-themes	^0.4.6	Theme switching utility (light/dark mode).
react-resizable-pans	^3.0.2	Resizable panel layouts for complex interfaces.
vaul	^1.1.2	Drawer component for mobile-friendly slide-out panels.

Backend Technologies

The backend is built on [ASP.NET](#) Core with Entity Framework Core for data access.

Core Framework

Package	Version	Purpose
.NET	10.0	The runtime and SDK for building the server application. .NET 10 provides the latest performance improvements and language features.
ASP.NET Core	10.0	Web framework for building APIs and web applications. Provides routing, middleware, dependency injection, and HTTP handling.

Authentication and Authorization

Package	Version	Purpose
Microsoft.AspNetCore.Identity.EntityFrameworkCore	10.0.0	Identity system for user authentication. Provides user management, password hashing, role-based authorization, and session management.

Data Access

Package	Version	Purpose
<code>Microsoft.EntityFrameworkCore.Sqlite</code>	10.0.0	SQLite database provider for Entity Framework Core. Enables file-based database storage without requiring a separate database server.
<code>Microsoft.EntityFrameworkCore.Tools</code>	10.0.0	Command-line tools for EF Core migrations and database management.

API Documentation

Package	Version	Purpose
<code>Microsoft.AspNetCore.OpenApi</code>	10.0.0	OpenAPI/Swagger support for API documentation. Automatically generates API documentation from endpoint definitions.

Development Tools

Package	Version	Purpose
<code>Microsoft.AspNetCore.SpaProxy</code>	10.*	Development proxy for Single Page Applications. Automatically starts the Vite development server when running the .NET application.

Development Environment

Recommended Tools

Tool	Purpose
Visual Studio 2022	Full-featured IDE for .NET development with integrated debugging, testing, and deployment tools.
Visual Studio Code	Lightweight editor with excellent TypeScript and React support through extensions.
Node.js 18+	JavaScript runtime required for frontend build tools and development server.
.NET 10 SDK	Required for building and running the backend application.

Browser Support

The application targets modern browsers with ES2020+ support:

- Chrome 90+
- Firefox 90+
- Safari 14+
- Edge 90+

Development Dependencies

Library	Version	Purpose
<code>@types/react</code>	<code>^19.1.8</code>	TypeScript definitions for React.
<code>@types/react-dom</code>	<code>^19.1.6</code>	TypeScript definitions for React DOM.
<code>@eslint/js</code>	<code>^9.28.0</code>	ESLint core JavaScript rules.
<code>eslint-plugin-react-hooks</code>	<code>^5.2.0</code>	ESLint rules for React Hooks best practices.
<code>eslint-plugin-react-refresh</code>	<code>^0.4.20</code>	ESLint plugin for React Fast Refresh compatibility.
<code>globals</code>	<code>^16.2.0</code>	Global variable definitions for ESLint.
<code>typescript-eslint</code>	<code>^8.33.1</code>	TypeScript support for ESLint.

API Reference

This document provides comprehensive documentation for all API endpoints available in the MyTravel application. The API follows RESTful conventions and returns JSON responses.

Base URL

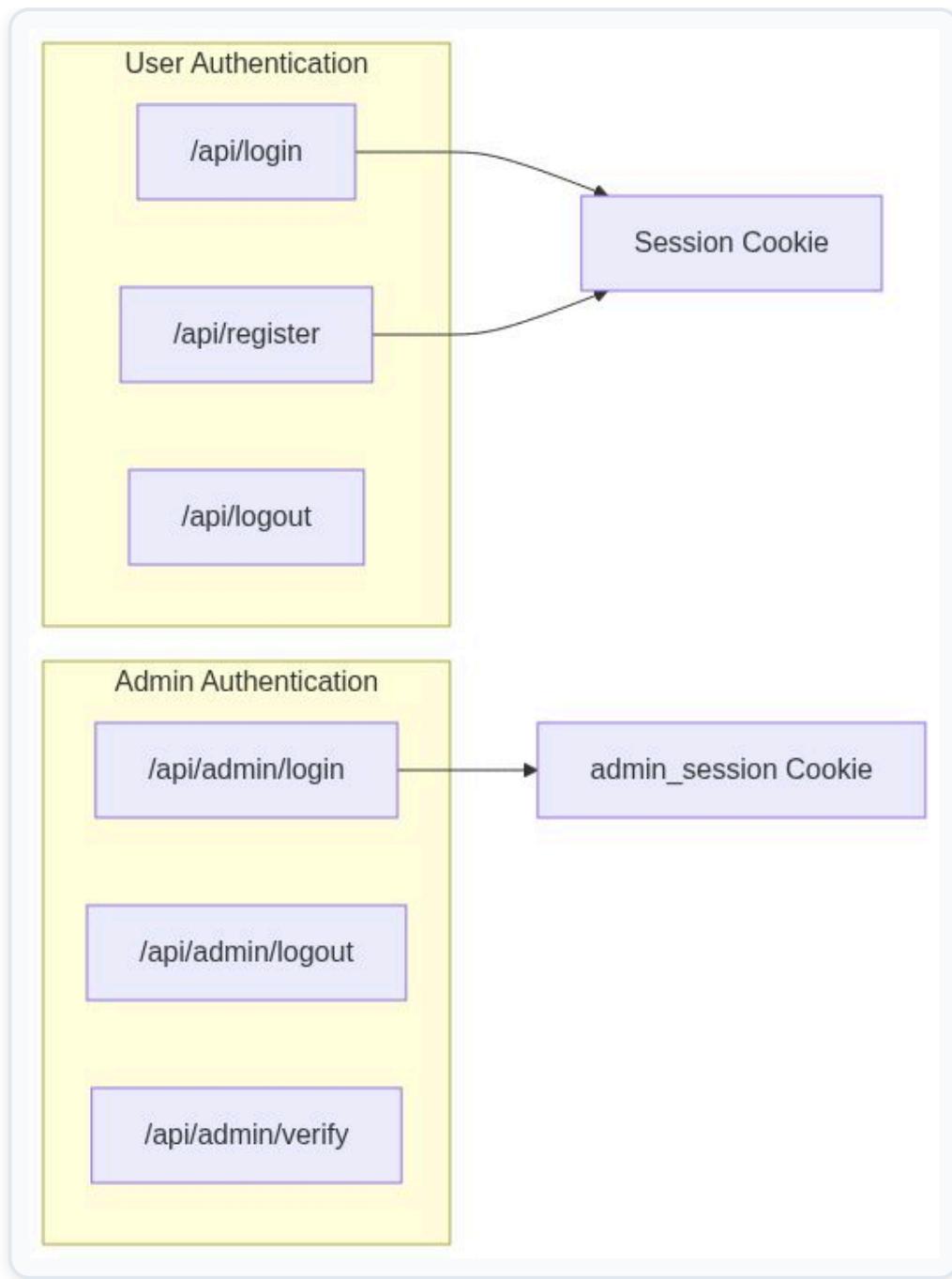
Development: <http://localhost:5083/api>

Production: <https://your-domain.com/api>

Authentication

The API uses cookie-based authentication. There are two authentication mechanisms:

1. **User Authentication** - Managed by [ASP.NET](#) Core Identity with session cookies
2. **Admin Authentication** - Uses a separate `admin_session` cookie



Response Format

All API responses follow a consistent format:

Success Response

```
{  
  "data": { ... },  
  "message": "Operation successful"  
}
```

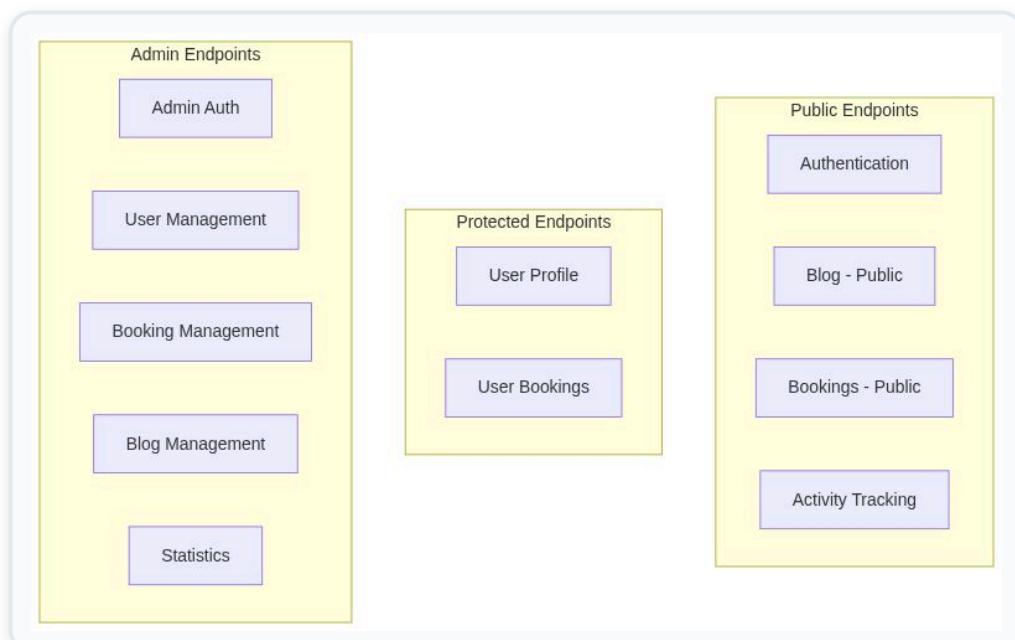
Error Response

```
{  
  "message": "Error description"  
}
```

Paginated Response

```
{  
  "items": [ ... ],  
  "totalCount": 100,  
  "page": 1,  
  "pageSize": 10,  
  "totalPages": 10  
}
```

API Endpoint Groups



Authentication Endpoints

Register User

Creates a new user account.

```
POST /api/register
```

Request Body:

Field	Type	Required	Description
<code>email</code>	string	Yes	User's email address
<code>password</code>	string	Yes	Password (min 6 characters, requires uppercase, lowercase, digit, special char)
<code>firstName</code>	string	No	User's first name
<code>lastName</code>	string	No	User's last name

Example Request:

```
{
  "email": "user@example.com",
  "password": "SecurePass123!",
  "firstName": "John",
  "lastName": "Doe"
}
```

Success Response (200):

```
{
  "message": "Registration successful"
}
```

Error Response (400):

```
{  
  "message": "Email already registered"  
}
```

Login User

Authenticates a user and creates a session.

```
POST /api/login
```

Request Body:

Field	Type	Required	Description
<code>email</code>	string	Yes	User's email address
<code>password</code>	string	Yes	User's password

Example Request:

```
{  
  "email": "user@example.com",  
  "password": "SecurePass123!"  
}
```

Success Response (200):

Sets authentication cookie and returns:

```
{  
  "message": "Login successful"  
}
```

Error Response (401):

```
{  
  "message": "Invalid credentials"  
}
```

Logout User

Ends the current user session.

```
POST /api/logout
```

Authentication: Required

Success Response (200):

```
{}
```

User Endpoints

Get Current User

Returns basic information about the authenticated user.

```
GET /api/user
```

Authentication: Required

Success Response (200):

```
{
  "name": "user@example.com",
  "isAuthenticated": true
}
```

Error Response (401):

```
{
  "message": "Unauthorized"
}
```

Get User Profile

Returns detailed profile information for the authenticated user.

```
GET /api/user/profile
```

Authentication: Required

Success Response (200):

```
{
  "id": "user-guid",
  "email": "user@example.com",
  "firstName": "John",
  "lastName": "Doe",
  "fullName": "John Doe"
}
```

Update User Profile

Updates the authenticated user's profile information.

```
PUT /api/user/profile
```

Authentication: Required

Request Body:

Field	Type	Required	Description
<code>firstName</code>	string	No	Updated first name
<code>lastName</code>	string	No	Updated last name

Example Request:

```
{
  "firstName": "Jonathan",
  "lastName": "Smith"
}
```

Success Response (200):

```
{  
  "id": "user-guid",  
  "email": "user@example.com",  
  "firstName": "Jonathan",  
  "lastName": "Smith",  
  "fullName": "Jonathan Smith"  
}
```

Blog Endpoints (Public)

Get All Published Blog Posts

Returns a paginated list of published blog posts.

```
GET /api/blog
```

Query Parameters:

Parameter	Type	Default	Description
<code>page</code>	integer	1	Page number
<code>pageSize</code>	integer	10	Items per page
<code>category</code>	string	null	Filter by category

Example Request:

```
GET /api/blog?page=1&pageSize=10&category=solo-travel
```

Success Response (200):

```
{  
  "posts": [  
    {  
      "id": 1,  
      "title": "10 Tips for Solo Travelers",  
      "slug": "10-tips-for-solo-travelers",  
      "category": "solo-travel",  
      "excerpt": "Discover essential tips for your solo adventure...",  
      "content": "{\"root\":{\"...\"}}",  
      "imageUrl": "/images/blog/solo-travel.jpg",  
      "published": true,  
      "createdAt": "2025-11-01T10:00:00Z",  
      "updatedAt": "2025-11-01T10:00:00Z",  
      "publishedAt": "2025-11-01T10:00:00Z",  
      "author": {  
        "id": "author-guid",  
        "fullName": "Jane Writer"  
      }  
    },  
    {"totalCount": 25,  
     "page": 1,  
     "pageSize": 10,  
     "totalPages": 3  
   }  
]
```

Get Blog Post by Slug

Returns a single published blog post by its URL slug.

```
GET /api/blog/{slug}
```

Path Parameters:

Parameter	Type	Description
slug	string	The URL-friendly post identifier

Example Request:

```
GET /api/blog/10-tips-for-solo-travelers
```

Success Response (200):

```
{  
  "id": 1,  
  "title": "10 Tips for Solo Travelers",  
  "slug": "10-tips-for-solo-travelers",  
  "category": "solo-travel",  
  "excerpt": "Discover essential tips for your solo adventure...",  
  "content": "{\"root\":{\"...\"}}",  
  "imageUrl": "/images/blog/solo-travel.jpg",  
  "published": true,  
  "createdAt": "2025-11-01T10:00:00Z",  
  "updatedAt": "2025-11-01T10:00:00Z",  
  "publishedAt": "2025-11-01T10:00:00Z",  
  "author": {  
    "id": "author-guid",  
    "fullName": "Jane Writer"  
  }  
}
```

Error Response (404):

```
{  
  "message": "Blog post not found"  
}
```

Booking Endpoints

Create Booking

Creates a new booking with one or more items. Supports both authenticated users and guest checkout.

```
POST /api/bookings
```

Authentication: Optional (guest checkout supported)

Request Body:

Field	Type	Required	Description
<code>customerEmail</code>	string	Yes*	Customer email (*required for guests)
<code>customerName</code>	string	Yes*	Customer name (*required for guests)
<code>paymentReference</code>	string	No	External payment reference
<code>items</code>	array	Yes	Array of booking items

Booking Item Object:

Field	Type	Required	Description
<code>type</code>	string	Yes	Item type: <code>Flight</code> , <code>Hotel</code> , or <code>Tour</code>
<code>title</code>	string	Yes	Item title/name
<code>details</code>	string	No	Additional details
<code>price</code>	decimal	Yes	Item price
<code>imageUrl</code>	string	No	Item image URL

Example Request:

```
{  
    "customerEmail": "guest@example.com",  
    "customerName": "Guest User",  
    "paymentReference": "PAY-123456",  
    "items": [  
        {  
            "type": "Flight",  
            "title": "New York to Paris",  
            "details": "Round trip, Economy class",  
            "price": 850.00,  
            "imageUrl": "/images/flights/nyc-paris.jpg"  
        },  
        {  
            "type": "Hotel",  
            "title": "Hotel Le Marais",  
            "details": "3 nights, Double room",  
            "price": 450.00,  
            "imageUrl": "/images/hotels/le-maraais.jpg"  
        }  
    ]  
}
```

Success Response (201):

```
{  
    "id": 42,  
    "message": "Booking created successfully",  
    "totalAmount": 1300.00,  
    "status": "Confirmed"  
}
```

Error Response (400):

```
{  
    "message": "At least one booking item is required"  
}
```

Get User's Bookings

Returns all bookings for the authenticated user.

```
GET /api/bookings
```

Authentication: Required

Success Response (200):

```
{  
  "bookings": [  
    {  
      "id": 42,  
      "customerEmail": "user@example.com",  
      "customerName": "John Doe",  
      "totalAmount": 1300.00,  
      "status": "Confirmed",  
      "createdAt": "2025-11-15T14:30:00Z",  
      "confirmedAt": "2025-11-15T14:30:00Z",  
      "cancelledAt": null,  
      "paymentReference": "PAY-123456",  
      "items": [  
        {  
          "id": 1,  
          "type": "Flight",  
          "title": "New York to Paris",  
          "details": "Round trip, Economy class",  
          "price": 850.00,  
          "imageUrl": "/images/flights/nyc-paris.jpg"  
        }  
      ]  
    }  
  ]  
}
```

Get Booking by ID

Returns a specific booking by ID.

```
GET /api/bookings/{id}
```

Authentication: Required (user can only view their own bookings)

Path Parameters:

Parameter	Type	Description
<code>id</code>	integer	Booking ID

Success Response (200):

```
{  
  "id": 42,  
  "customerEmail": "user@example.com",  
  "customerName": "John Doe",  
  "totalAmount": 1300.00,  
  "status": "Confirmed",  
  "createdAt": "2025-11-15T14:30:00Z",  
  "confirmedAt": "2025-11-15T14:30:00Z",  
  "cancelledAt": null,  
  "paymentReference": "PAY-123456",  
  "items": [ ... ]  
}
```

Error Response (404):

```
{  
  "message": "Booking not found"  
}
```

Error Response (403):

Returns **Forbid** if user tries to access another user's booking.

Activity Tracking Endpoints

Ping (Visitor Tracking)

Tracks anonymous visitor activity. Creates a visitor ID cookie if not present.

```
GET /api/ping
```

Success Response (200):

```
{  
  "status": "ok"  
}
```

Sets **visitor_id** cookie if not already present.

Heartbeat (User Activity)

Tracks authenticated user activity for real-time online status.

```
POST /api/heartbeat
```

Authentication: Optional (only tracks authenticated users)

Success Response (200):

```
{}
```

Admin Authentication Endpoints

Admin Login

Authenticates an administrator using credentials from configuration.

```
POST /api/admin/login
```

Request Body:

Field	Type	Required	Description
email	string	Yes	Admin email
password	string	Yes	Admin password

Example Request:

```
{
  "email": "admin@mytravel.com",
  "password": "Admin@123!"
}
```

Success Response (200):

Sets `admin_session` cookie (HttpOnly, Secure, 8-hour expiry) and returns:

```
{  
  "message": "Admin login successful",  
  "isAdmin": true  
}
```

Error Response (401):

Returns `Unauthorized` for invalid credentials.

Admin Logout

Ends the admin session.

```
POST /api/admin/logout
```

Success Response (200):

```
{  
  "message": "Admin logged out"  
}
```

Verify Admin Session

Checks if the current session has admin privileges.

```
GET /api/admin/verify
```

Success Response (200):

```
{  
  "isAdmin": true  
}
```

Error Response (401):

Returns `Unauthorized` if no valid admin session.

Admin User Management Endpoints

Get All Users

Returns a paginated, searchable list of all users.

```
GET /api/admin/users
```

Authentication: Admin required

Query Parameters:

Parameter	Type	Default	Description
<code>page</code>	integer	1	Page number
<code>pageSize</code>	integer	10	Items per page
<code>search</code>	string	null	Search by username or email
<code>sortBy</code>	string	"createdAt"	Sort field: <code>username</code> , <code>email</code> , <code>createdAt</code> , <code>lastLoginAt</code>
<code>sortOrder</code>	string	"desc"	Sort direction: <code>asc</code> or <code>desc</code>

Example Request:

```
GET /api/admin/users?page=1&pageSize=20&search=john&sortBy=createdAt&sortOrder=desc
```

Success Response (200):

```
{  
  "users": [  
    {  
      "id": "user-guid",  
      "userName": "john@example.com",  
      "email": "john@example.com",  
      "createdAt": "2025-10-01T08:00:00Z",  
      "lastLoginAt": "2025-11-28T09:30:00Z",  
      "isActive": true,  
      "emailConfirmed": true  
    }  
  ],  
  "totalCount": 150,  
  "page": 1,  
  "pageSize": 20,  
  "totalPages": 8  
}
```

Get User by ID

Returns detailed information for a specific user.

```
GET /api/admin/users/{id}
```

Authentication: Admin required

Path Parameters:

Parameter	Type	Description
<code>id</code>	string	User ID (GUID)

Success Response (200):

```
{  
  "id": "user-guid",  
  "userName": "john@example.com",  
  "email": "john@example.com",  
  "createdAt": "2025-10-01T08:00:00Z",  
  "lastLoginAt": "2025-11-28T09:30:00Z",  
  "isActive": true,  
  "emailConfirmed": true  
}
```

Error Response (404):

```
{  
  "message": "User not found"  
}
```

Update User

Updates a user's information.

```
PUT /api/admin/users/{id}
```

Authentication: Admin required

Request Body:

Field	Type	Required	Description
userName	string	No	New username
isActive	boolean	No	Active status
emailConfirmed	boolean	No	Email confirmation status

Example Request:

```
{  
  "userName": "john.doe@example.com",  
  "isActive": true,  
  "emailConfirmed": true  
}
```

Success Response (200):

```
{  
  "message": "User updated successfully",  
  "user": {  
    "id": "user-guid",  
    "userName": "john.doe@example.com",  
    "email": "john@example.com",  
    "createdAt": "2025-10-01T08:00:00Z",  
    "lastLoginAt": "2025-11-28T09:30:00Z",  
    "isActive": true,  
    "emailConfirmed": true  
  }  
}
```

Toggle User Status

Toggles a user's active/inactive status.

```
PATCH /api/admin/users/{id}/toggle-status
```

Authentication: Admin required

Success Response (200):

```
{  
  "message": "User deactivated successfully",  
  "isActive": false  
}
```

Delete User

Permanently deletes a user account.

```
DELETE /api/admin/users/{id}
```

Authentication: Admin required

Success Response (200):

```
{
  "message": "User deleted successfully"
}
```

Error Response (400):

```
{
  "message": "Failed to delete user",
  "errors": [ ... ]
}
```

Get Admin Statistics

Returns platform-wide statistics for the admin dashboard.

```
GET /api/admin/stats
```

Authentication: Admin required

Success Response (200):

```
{
  "totalUsers": 1250,
  "activeUsersCount": 1180,
  "currentlyOnline": 45,
  "newUsersToday": 12,
  "newUsersThisWeek": 87,
  "newUsersThisMonth": 320
}
```

Admin Booking Management Endpoints

Get All Bookings

Returns a paginated list of all bookings with filtering and sorting.

```
GET /api/admin/bookings
```

Authentication: Admin required

Query Parameters:

Parameter	Type	Default	Description
<code>page</code>	integer	1	Page number
<code>pageSize</code>	integer	10	Items per page
<code>search</code>	string	null	Search by email, name, or booking ID
<code>status</code>	string	null	Filter by status: <code>Pending</code> , <code>Confirmed</code> , <code>Cancelled</code> , <code>Completed</code>
<code>sortBy</code>	string	"createdAt"	Sort field: <code>customerName</code> , <code>customerEmail</code> , <code>totalAmount</code> , <code>status</code> , <code>createdAt</code>
<code>sortOrder</code>	string	"desc"	Sort direction: <code>asc</code> or <code>desc</code>

Success Response (200):

```
{  
  "bookings": [  
    {  
      "id": 42,  
      "customerEmail": "user@example.com",  
      "customerName": "John Doe",  
      "totalAmount": 1300.00,  
      "status": "Confirmed",  
      "createdAt": "2025-11-15T14:30:00Z",  
      "confirmedAt": "2025-11-15T14:30:00Z",  
      "cancelledAt": null,  
      "paymentReference": "PAY-123456",  
      "items": [ ... ]  
    }  
  ],  
  "totalCount": 500,  
  "page": 1,  
  "pageSize": 10,  
  "totalPages": 50  
}
```

Get Booking by ID (Admin)

Returns detailed booking information.

```
GET /api/admin/bookings/{id}
```

Authentication: Admin required

Success Response (200):

Same structure as user booking response.

Update Booking Status

Updates the status of a booking.

```
PATCH /api/admin/bookings/{id}/status
```

Authentication: Admin required

Request Body:

Field	Type	Required	Description
status	string	Yes	New status: Pending, Confirmed, Cancelled, Completed

Example Request:

```
{
  "status": "Completed"
}
```

Success Response (200):

```
{
  "message": "Booking status updated successfully",
  "id": 42,
  "status": "Completed"
}
```

Error Response (400):

```
{
  "message": "Invalid status value"
}
```

Get Booking Statistics

Returns booking-related statistics.

```
GET /api/admin/bookings/stats
```

Authentication: Admin required

Success Response (200):

```
{  
    "totalBookings": 500,  
    "pendingBookings": 25,  
    "confirmedBookings": 350,  
    "cancelledBookings": 50,  
    "completedBookings": 75,  
    "totalRevenue": 425000.00,  
    "bookingsToday": 8,  
    "bookingsThisWeek": 45,  
    "bookingsThisMonth": 180  
}
```

Admin Blog Management Endpoints

Get All Blog Posts (Admin)

Returns all blog posts including drafts with filtering and sorting.

```
GET /api/admin/blog
```

Authentication: Admin required

Query Parameters:

Parameter	Type	Default	Description
<code>page</code>	integer	1	Page number
<code>pageSize</code>	integer	10	Items per page
<code>search</code>	string	null	Search in title or excerpt
<code>category</code>	string	null	Filter by category
<code>status</code>	string	null	Filter: <code>published</code> or <code>draft</code>
<code>sortBy</code>	string	"createdAt"	Sort field: <code>title</code> , <code>category</code> , <code>published</code> , <code>publishedAt</code> , <code>createdAt</code>
<code>sortOrder</code>	string	"desc"	Sort direction: <code>asc</code> or <code>desc</code>

Success Response (200):

```
{
  "posts": [
    {
      "id": 1,
      "title": "Draft Post",
      "slug": "draft-post",
      "category": "adventure",
      "excerpt": "This is a draft...",
      "content": "...",
      "imageUrl": null,
      "published": false,
      "createdAt": "2025-11-20T10:00:00Z",
      "updatedAt": "2025-11-20T10:00:00Z",
      "publishedAt": null,
      "author": null
    }
  ],
  "totalCount": 30,
  "page": 1,
  "pageSize": 10,
  "totalPages": 3
}
```

Get Blog Post by ID (Admin)

Returns a specific blog post by ID (including drafts).

```
GET /api/admin/blog/{id}
```

Authentication: Admin required

Success Response (200):

Same structure as public blog post response.

Create Blog Post

Creates a new blog post.

```
POST /api/admin/blog
```

Authentication: Admin required

Request Body:

Field	Type	Required	Description
<code>title</code>	string	Yes	Post title
<code>authorId</code>	string	No	Author user ID
<code>category</code>	string	No	Category (default: "solo-travel")
<code>excerpt</code>	string	No	Short description
<code>content</code>	string	Yes	Lexical editor JSON content
<code>imageUrl</code>	string	No	Featured image URL
<code>published</code>	boolean	No	Publish immediately (default: false)

Example Request:

```
{  
  "title": "New Adventure Guide",  
  "category": "adventure",  
  "excerpt": "Discover the best adventure destinations...",  
  "content": "{\"root\":{\"children\":[]}}",  
  "imageUrl": "/images/blog/adventure.jpg",  
  "published": false  
}
```

Success Response (201):

```
{  
  "id": 15,  
  "slug": "new-adventure-guide",  
  "message": "Blog post created successfully"  
}
```

Error Response (400):

```
{  
  "message": "Title is required"  
}
```

Update Blog Post

Updates an existing blog post.

```
PUT /api/admin/blog/{id}
```

Authentication: Admin required

Request Body:

Field	Type	Required	Description
<code>title</code>	string	No	Updated title (regenerates slug)
<code>category</code>	string	No	Updated category
<code>excerpt</code>	string	No	Updated excerpt
<code>content</code>	string	No	Updated content
<code>imageUrl</code>	string	No	Updated image URL
<code>published</code>	boolean	No	Publication status

Success Response (200):

```
{
  "message": "Blog post updated successfully",
  "id": 15,
  "slug": "updated-adventure-guide"
}
```

Delete Blog Post

Permanently deletes a blog post.

```
DELETE /api/admin/blog/{id}
```

Authentication: Admin required

Success Response (200):

```
{
  "message": "Blog post deleted successfully"
}
```

Toggle Blog Post Publication

Toggles the published/draft status of a blog post.

```
PATCH /api/admin/blog/{id}/toggle-publish
```

Authentication: Admin required

Success Response (200):

```
{
  "message": "Blog post published successfully",
  "published": true
}
```

Get Blog Statistics

Returns blog-related statistics.

```
GET /api/admin/blog/stats
```

Authentication: Admin required

Success Response (200):

```
{
  "totalPosts": 30,
  "publishedPosts": 25,
  "draftPosts": 5,
  "postsThisWeek": 3,
  "postsThisMonth": 12,
  "postsByCategory": [
    { "category": "solo-travel", "count": 10 },
    { "category": "adventure", "count": 8 },
    { "category": "family-trips", "count": 7 },
    { "category": "luxury-travel", "count": 5 }
  ]
}
```

HTTP Status Codes

Code	Description
200	OK - Request successful
201	Created - Resource created successfully
400	Bad Request - Invalid input or validation error
401	Unauthorized - Authentication required or failed
403	Forbidden - Access denied to resource
404	Not Found - Resource does not exist
500	Internal Server Error - Server-side error

Rate Limiting

The API does not currently implement rate limiting. For production deployments, consider implementing rate limiting at the reverse proxy level.

CORS

Cross-Origin Resource Sharing (CORS) is configured to allow requests from the frontend development server during development. In production, CORS should be configured to allow only trusted origins.

Error Handling

This document explains the error handling strategies implemented throughout the MyTravel application. Comprehensive error handling ensures users receive meaningful feedback when issues occur, maintaining a smooth experience even during failures.

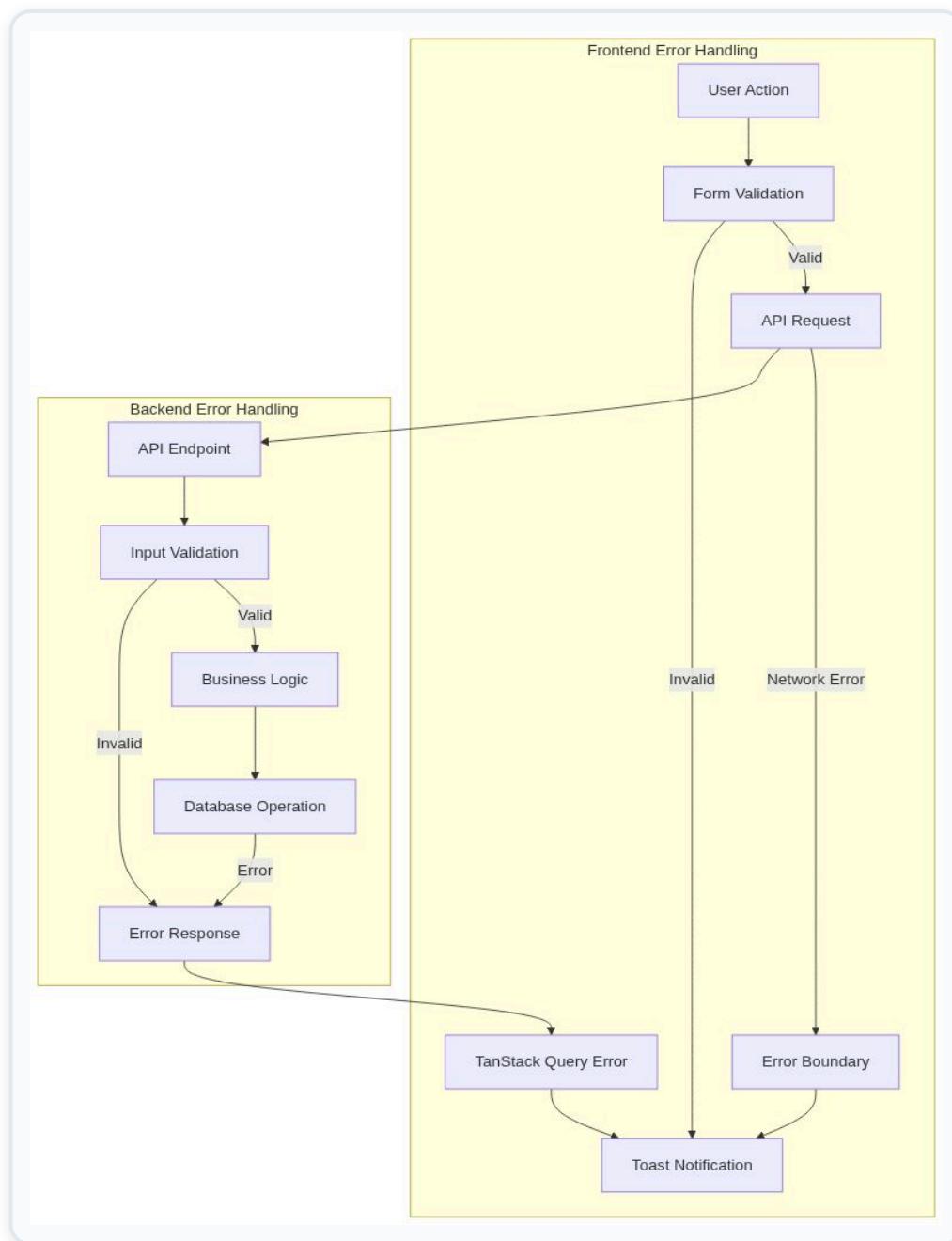
Error Handling Philosophy

"Ensure smooth user experiences by providing meaningful error messages."

The application follows these core principles for error handling:

1. **User-Friendly Messages** - Technical errors are translated into understandable language
2. **Graceful Degradation** - The application remains functional even when individual features fail
3. **Appropriate Feedback** - Users receive timely notifications about the success or failure of their actions
4. **Logging for Debugging** - Errors are captured for developer troubleshooting without exposing sensitive details to users

Error Flow Architecture



Frontend Error Handling

API Request Helper

The application uses a centralized API request function that standardizes error handling for all HTTP requests. This is implemented in `lib/queryClient.ts`.

```
async function throwIfResNotOk(res: Response) {
  if (!res.ok) {
    const text = (await res.text()) || res.statusText;
    throw new Error(`[${res.status}]: ${text}`);
  }
}

export async function apiRequest(
  method: string,
  url: string,
  data?: unknown | undefined,
): Promise<Response> {
  const res = await fetch(url, {
    method,
    headers: data ? { "Content-Type": "application/json" } : {},
    body: data ? JSON.stringify(data) : undefined,
    credentials: "include",
  });

  await throwIfResNotOk(res);
  return res;
}
```

This helper function:

- Checks if the response status indicates an error (`!res.ok`)
- Extracts the error message from the response body or uses the status text
- Throws an `Error` with both the status code and message for handling upstream

TanStack Query Error Handling

TanStack Query is configured with custom error handling behavior in `queryClient.ts`:

```

type UnauthorizedBehavior = "returnNull" | "throw";

export const getQueryFn: <T>(options: {
  on401: UnauthorizedBehavior;
}) => QueryFunction<T> =
  ({ on401: unauthorizedBehavior }) =>
    async ({ queryKey }) => {
      const res = await fetch(queryKey.join("/")) as string, {
        credentials: "include",
      };

      if (unauthorizedBehavior === "returnNull" && res.status === 401) {
        return null;
      }

      await throwIfResNotOk(res);
      return await res.json();
    };
}

export const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      queryFn: getQueryFn({ on401: "throw" }),
      refetchInterval: false,
      refetchOnWindowFocus: false,
      staleTime: Infinity,
      retry: false,
    },
    mutations: {
      retry: false,
    },
  },
});

```

Key configuration choices:

- **401 Handling** - Configurable behavior for unauthorized responses (return null or throw)
- **No Automatic Retry** - Failed requests are not automatically retried to prevent redundant server load
- **No Refetch on Focus** - Prevents unexpected data refreshes when switching browser tabs

Mutation Error Handling Pattern

When performing mutations (create, update, delete operations), errors are handled in the `onError` callback:

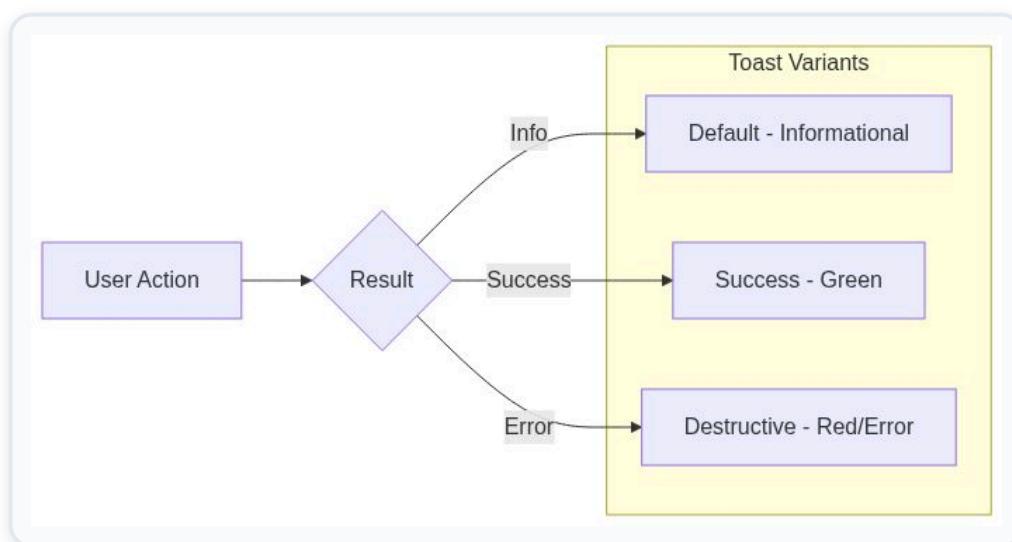
```

const mutation = useMutation({
  mutationFn: async (data: FormData) => {
    const response = await apiRequest("POST", "/api/bookings", data);
    return response.json();
  },
  onSuccess: (data) => {
    toast({
      title: "Success",
      description: "Your booking has been confirmed!",
    });
    // Navigate or update UI
  },
  onError: (error: Error) => {
    toast({
      title: "Booking Failed",
      description: error.message || "Unable to complete your booking. Please try again.",
      variant: "destructive",
    });
  },
});

```

Toast Notification System

The application uses a toast notification system (Sonner/Radix Toast) for user feedback. Toasts support multiple variants for different message types.



Toast Usage Examples:

```
// Success notification
toast({
  title: "Profile Updated",
  description: "Your changes have been saved successfully.",
});

// Error notification
toast({
  title: "Error",
  description: "Failed to save changes. Please try again.",
  variant: "destructive",
});

// Informational notification
toast({
  title: "Session Expiring",
  description: "Your session will expire in 5 minutes.",
});
```

Form Validation with Zod

Forms use react-hook-form with Zod schemas for client-side validation. This catches errors before they reach the server.

```

import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { z } from "zod";

const loginSchema = z.object({
  email: z.string().email("Please enter a valid email address"),
  password: z.string().min(6, "Password must be at least 6 characters"),
});

type LoginForm = z.infer<typeof loginSchema>

function LoginPage() {
  const form = useForm<LoginForm>({
    resolver: zodResolver(loginSchema),
    defaultValues: {
      email: "",
      password: "",
    },
  });

  const onSubmit = async (data: LoginForm) => {
    try {
      await apiRequest("POST", "/api/login", data);
      // Handle success
    } catch (error) {
      form.setError("root", {
        message: "Invalid email or password",
      });
    }
  };
}

return (
  <Form {...form}>
    <FormField
      control={form.control}
      name="email"
      render={({ field }) => (
        <FormItem>
          <FormLabel>Email</FormLabel>
          <FormControl>
            <Input {...field} />
          </FormControl>
          <FormMessage /> {/* Displays validation errors */}
        </FormItem>
      )}
    />
    {/* ... */}
  </Form>
);
}

```

Validation Error Display:

State	User Experience
Empty required field	"This field is required" appears below input
Invalid email format	"Please enter a valid email address"
Password too short	"Password must be at least 6 characters"
Server validation error	Error message displayed at form level

404 Not Found Page

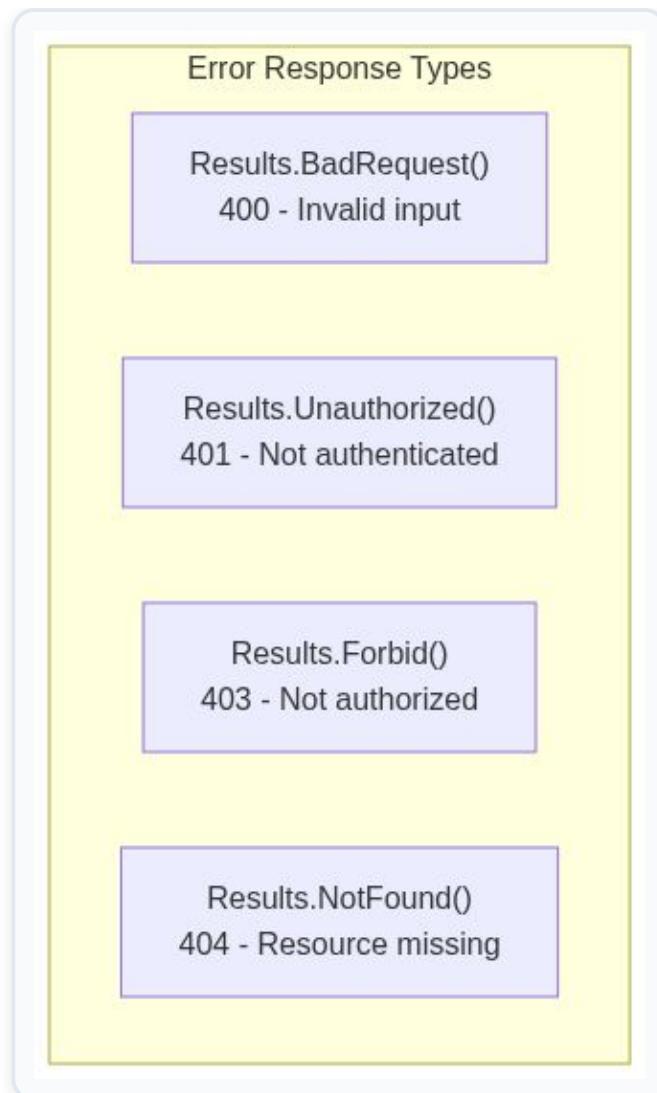
The application includes a custom 404 page for handling invalid routes:

```
// pages/not-found.tsx
export default function NotFound() {
  return (
    <div className="flex flex-col items-center justify-center min-h-screen">
      <h1 className="text-6xl font-bold text-gray-900">404</h1>
      <p className="text-xl text-gray-600 mt-4">Page not found</p>
      <p className="text-gray-500 mt-2">
        The page you're looking for doesn't exist or has been moved.
      </p>
      <Link href="/" className="mt-6">
        <Button>Return to Homepage</Button>
      </Link>
    </div>
  );
}
```

Backend Error Handling

Standardized Error Responses

The backend uses [ASP.NET Core's `Results`](#) class to return consistent error responses:



Error Response Examples:

```
// 400 Bad Request - Validation errors
if (string.IsNullOrWhiteSpace(request.Title))
{
    return Results.BadRequest(new { message = "Title is required" });
}

if (request.Items == null || request.Items.Count == 0)
{
    return Results.BadRequest(new { message = "At least one booking item is required" });
}

// 401 Unauthorized - Authentication required
var adminCookie = httpContext.Request.Cookies["admin_session"];
if (adminCookie != "authenticated")
{
    return Results.Unauthorized();
}

// 403 Forbidden - Access denied
if (booking.UserId != null && booking.UserId != userId)
{
    return Results.Forbid();
}

// 404 Not Found - Resource doesn't exist
var post = await db.BlogPosts.FindAsync(id);
if (post == null)
{
    return Results.NotFound(new { message = "Blog post not found" });
}
```

Input Validation

All API endpoints validate input before processing:

```

// Booking endpoint validation
app.MapPost("/api/bookings", async (CreateBookingRequest request, ...) =>
{
    // Validate required items
    if (request.Items == null || request.Items.Count == 0)
    {
        return Results.BadRequest(new { message = "At least one booking item is required" });
    }

    // Validate guest checkout requirements
    if (user.Identity?.IsAuthenticated != true)
    {
        if (string.IsNullOrWhiteSpace(request.CustomerEmail) ||
            string.IsNullOrWhiteSpace(request.CustomerName))
        {
            return Results.BadRequest(new {
                message = "Customer email and name are required for guest checkout"
            });
        }
    }

    // Validate enum values
    if (!Enum.TryParse<BookingStatus>(request.Status, true, out var statusEnum))
    {
        return Results.BadRequest(new { message = "Invalid status value" });
    }

    // ... proceed with valid request
});

```

Non-Critical Operation Error Handling

For operations that shouldn't block the main request (like sending emails), errors are caught and logged but don't fail the overall operation:

```
// Create booking
db.Bookings.Add(booking);
await db.SaveChangesAsync();

// Try to send confirmation email (non-critical)
try
{
    await emailSender.SendConfirmationLinkAsync(
        new ApplicationUser { Email = booking.CustomerEmail },
        booking.CustomerEmail,
        emailBody);
}
catch
{
    // Email sending failed, but booking was created successfully
    // Log the error for investigation but don't fail the request
}

// Return success - booking was created
return Results.Created($"api/bookings/{booking.Id}", new
{
    id = booking.Id,
    message = "Booking created successfully"
});
```

Authentication Error Handling

The admin authentication system validates credentials against configuration:

```

app.MapPost("/api/admin/login", async (
    AdminLoginRequest request,
    IConfiguration config,
    HttpContext httpContext) =>
{
    var adminEmail = config["AdminCredentials:Email"];
    var adminPassword = config["AdminCredentials:Password"];

    if (request.Email == adminEmail && request.Password == adminPassword)
    {
        // Set secure cookie and return success
        httpContext.Response.Cookies.Append("admin_session", "authenticated",
            new CookieOptions
            {
                HttpOnly = true,
                Secure = true,
                SameSite = SameSiteMode.Strict,
                Expires = DateTimeOffset.UtcNow.AddHours(8)
            });
    }

    return Results.Ok(new { message = "Admin login successful", isAdmin = true });
}

// Invalid credentials - return 401 without revealing which field was wrong
return Results.Unauthorized();
});

```

Database Error Handling

Entity Framework operations are wrapped in appropriate error handling:

```

// Delete user with Identity Manager (handles cascading constraints)
var result = await userManager.DeleteAsync(user);
if (!result.Succeeded)
{
    return Results.BadRequest(new {
        message = "Failed to delete user",
        errors = result.Errors
    });
}

return Results.Ok(new { message = "User deleted successfully" });

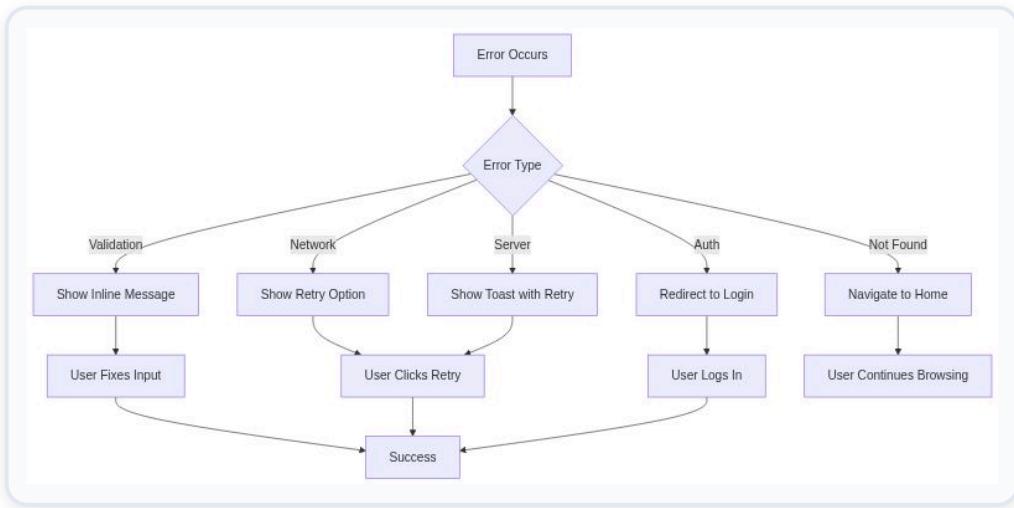
```

Error Scenarios and User Experience

Common Error Scenarios

Scenario	Frontend Handling	Backend Response	User Message
Invalid form input	Inline validation message	N/A (client-side)	"Please enter a valid email address"
Wrong password	Toast notification	401 Unauthorized	"Invalid email or password"
Session expired	Redirect to login	401 Unauthorized	"Your session has expired. Please log in again."
Resource not found	404 page or toast	404 Not Found	"The requested item could not be found"
Network error	Toast notification	N/A	"Unable to connect. Please check your internet connection."
Server error	Toast notification	500 Internal Error	"Something went wrong. Please try again later."
Payment failure	Toast + form state	400 Bad Request	"Payment could not be processed. Please verify your payment details."
Booking unavailable	Toast notification	400 Bad Request	"This booking is no longer available"

Error Recovery Patterns



Best Practices Summary

Frontend

1. Always handle errors in mutations - Use `onError` callbacks to display user feedback
2. Validate before submitting - Use Zod schemas with react-hook-form
3. Provide specific messages - Tell users what went wrong and how to fix it
4. Use appropriate toast variants - `destructive` for errors, default for success
5. Handle loading states - Show spinners/skeletons during async operations

Backend

1. Return appropriate status codes - 400 for validation, 401 for auth, 404 for missing resources
2. Include helpful messages - Explain what went wrong in the response body
3. Don't expose sensitive information - Avoid revealing system internals in error messages
4. Handle non-critical failures gracefully - Don't fail the whole request if email sending fails
5. Validate all input - Never trust client-side validation alone