Travel Planner - Complete Project Documentation

-Shivam Kumar Choudhary (30100838)

📋 Table of Contents

🎯 **Project Overview**

The Travel Planner is a full-stack web application that allows users to create, manage, and visualize their travel itineraries. Built with modern web technologies, it provides a seamless experience for planning trips with multiple destinations, dates, and optional map integration.

Key Features

- ✅ User authentication (JWT-based)
- ✅ CRUD operations for itineraries
- ✅ Multiple destinations per itinerary
- ✅ Google Maps integration with markers
- ✅ Responsive design with modern UI
- ✅ Protected routes and secure API endpoints
- ✅ Real-time form validation
- ✅ Edit and delete functionality

• Prompt Documentation:

I want to build a full-stack itinerary planner web app using React for frontend, Node.js with Express for backend, and MongoDB for the database.

Requirements:

- User authentication with JWT (signup, login, protected routes).

- Backend REST API with routes to create, read, and delete itineraries.

- Each itinerary has a title, multiple destinations with location name, start/end dates, and optional notes.

- Locations will be saved as text only.

- Frontend React app with pages for login, registration, dashboard, and itinerary management.

- Itinerary page should display a list of itineraries and a form to add new ones.

- Include a Google Map that shows markers only if latitude and longitude are available.

- Use axios or fetch for API calls.

- Proper error handling and loading states.

- Use environment variables for API keys and backend URLs.

- Use React Router for navigation with protected routes.

- Provide schema/model definitions for MongoDB with Mongoose.

- Backend should validate required fields but allow null lat/lng.

- Use best practices for security and code structure.

Please provide me:

1. Backend folder structure and code examples for:
   - Express server setup
   - User model and itinerary model
   - Auth middleware for JWT
   - Routes for auth and itineraries

2. Frontend folder structure and React code examples for:
   - Auth pages (Login/Register)
   - Protected routes and routing setup
   - Navbar with login/logout and navigation links
   - Itineraries page with form and Google Map integration

3. Instructions for setting up environment variables and running the app locally.

Start with frontend and backend code with folder structure.

🏗️ **Architecture & Tech Stack**

Frontend

- ****React 18**** - UI library

- ****React Router DOM v6**** - Client-side routing

- ****Axios**** - HTTP client for API calls

- ****Google Maps React Wrapper**** - Map integration

- ****Context API**** - State management

- **CSS-in-JS** - Inline styling approach

Backend

- **Node.js** - Runtime environment
- **Express.js** - Web framework
- **MongoDB** - NoSQL database
- **Mongoose** - ODM for MongoDB
- **JWT** - Authentication tokens
- **bcryptjs** - Password hashing
- **express-validator** - Input validation
- **CORS** - Cross-origin resource sharing

Development Tools

- **Nodemon** - Development server auto-restart
- **dotenv** - Environment variable management

📁 Project Structure

```
Travel-planner-project/
├── frontend/                # React frontend application
│   ├── public/
│   │   └── index.html       # HTML template
│   ├── src/
│   │   ├── components/      # Reusable React components
│   │   │   ├── GoogleMap.js    # Google Maps integration
```

```
│   │   │   ├── Navbar.js       # Navigation component
│   │   │   └── ProtectedRoute.js # Route protection wrapper
│   │   ├── context/           # React Context providers
│   │   │   └── AuthContext.js   # Authentication state management
│   │   ├── pages/             # Page-level components
│   │   │   ├── Dashboard.js     # User dashboard
│   │   │   ├── Itineraries.js   # Main itinerary management
│   │   │   ├── Login.js         # User login form
│   │   │   └── Register.js      # User registration form
│   │   ├── services/          # API service layer
│   │   │   └── api.js         # Axios configuration and API calls
│   │   ├── App.js             # Main application component
│   │   └── index.js           # Application entry point
│   ├── .env                 # Frontend environment variables
│   └── package.json           # Frontend dependencies
├── backend/                 # Node.js/Express backend
│   ├── middleware/            # Custom middleware functions
│   │   └── auth.js            # JWT authentication middleware
│   ├── models/                # Mongoose data models
│   │   ├── Itinerary.js       # Itinerary schema definition
│   │   └── User.js            # User schema definition
│   ├── routes/                # Express route handlers
│   │   ├── auth.js            # Authentication routes
│   │   └── itineraries.js     # Itinerary CRUD routes
│   ├── .env                 # Backend environment variables
│   ├── package.json           # Backend dependencies
```

```
│   └── server.js            # Express server configuration
├── DOCUMENTATION.md         # This documentation file
└── README.md                # Project setup instructions
```

🚀 **Setup & Installation**

Prerequisites

- Node.js (v14 or higher)
- MongoDB (local installation or MongoDB Atlas)
- Google Maps API key (optional, for map functionality)

Backend Setup

1. ****Navigate to backend directory:****

```bash
cd backend
```

2. ****Install dependencies:****

```bash
npm install
```

3. ****Configure environment variables in `.env`:****

```env
```

PORT=5000

MONGODB_URI=mongodb://localhost:27017/travel-planner

JWT_SECRET=your_jwt_secret_key_here_change_in_production

```

4. **Start the development server:**

```bash

npm run dev

```

Backend will run on: `http://localhost:5000`

Frontend Setup

1. **Navigate to frontend directory:**

```bash

cd frontend

```

2. **Install dependencies:**

```bash

npm install

```

3. **Configure environment variables in `.env`:**

```env

REACT_APP_API_URL=http://localhost:5000/api

REACT_APP_GOOGLE_MAPS_API_KEY=AIzaSyBxUcVV9M0ZWj-qojyifxWU5pycVE4_Up0

```

4. **Start the development server:**

```bash

npm start

```

Frontend will run on: `http://localhost:3000`

Database Setup

1. **Ensure MongoDB is running locally on port 27017**

2. **The application will automatically create the `travel-planner` database**

📡 API Documentation

Base URL

```

http://localhost:5000/api

```

Authentication Endpoints

Register User
```

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

{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "password123"
}
```

Response:

```json
{
  "token": "jwt_token_here",
  "user": {
    "id": "user_id",
    "name": "John Doe",
    "email": "john@example.com"
  }
}
```

Login User

```http
POST /auth/login
```

Content-Type: application/json

{

"email": "john@example.com",

"password": "password123"

}

```

**Response:**

```json

{

"token": "jwt_token_here",

"user": {

"id": "user_id",

"name": "John Doe",

"email": "john@example.com"

}

}

```

Itinerary Endpoints (Protected)

Get All Itineraries

```http

GET /itineraries

Authorization: Bearer jwt_token_here

```

Response:

```json
[
 {
  "_id": "itinerary_id",
  "title": "Europe Trip",
  "destinations": [
   {
    "location": "Paris, France",
    "startDate": "2024-06-01T00:00:00.000Z",
    "endDate": "2024-06-05T00:00:00.000Z",
    "notes": "Visit Eiffel Tower",
    "latitude": 48.8566,
    "longitude": 2.3522
   }
  ],
  "user": "user_id",
  "createdAt": "2024-01-01T00:00:00.000Z",
  "updatedAt": "2024-01-01T00:00:00.000Z"
 }
]
```

Create Itinerary
```

```http
POST /itineraries
Authorization: Bearer jwt_token_here
Content-Type: application/json

{
  "title": "Asia Adventure",
  "destinations": [
    {
      "location": "Tokyo, Japan",
      "startDate": "2024-07-01",
      "endDate": "2024-07-07",
      "notes": "Cherry blossom season",
      "latitude": 35.6762,
      "longitude": 139.6503
    }
  ]
}
```

Update Itinerary

```http
PUT /itineraries/:id
Authorization: Bearer jwt_token_here
Content-Type: application/json
```

{

"title": "Updated Trip Name",

"destinations": [...]

}

```

#### **Delete Itinerary**

```http

DELETE /itineraries/:id

Authorization: Bearer jwt_token_here

```

🧩 **Frontend Components**

AuthContext.js

****Purpose:**** Manages global authentication state

****Key Features:****

- User login/logout functionality

- Token persistence in localStorage

- Loading state management

- Context provider for child components

ProtectedRoute.js

****Purpose:**** Protects routes that require authentication

****Key Features:****

- Checks user authentication status
```

- Redirects to login if not authenticated

- Shows loading state during auth check

Navbar.js

****Purpose:**** Navigation and user interface

****Key Features:****

- Responsive navigation menu

- User greeting with name display

- Conditional rendering based on auth status

- Modern styling with dark theme

GoogleMap.js

****Purpose:**** Displays interactive map with destination markers

****Key Features:****

- Google Maps integration

- Dynamic marker placement

- Error handling for missing API key

- Responsive map container

Itineraries.js

****Purpose:**** Main itinerary management interface

****Key Features:****

- CRUD operations for itineraries

- Dynamic form for multiple destinations

- Edit mode with pre-populated data

- Modern card-based UI design

- Real-time form validation

🔧 **Backend Services**

server.js

****Purpose:**** Express server configuration and startup

****Key Features:****

- Middleware setup (CORS, JSON parsing)

- Route registration

- MongoDB connection

- Error handling

auth.js (middleware)

****Purpose:**** JWT token validation middleware

****Key Features:****

- Token extraction from headers

- Token verification

- User authentication

- Request context enhancement

User.js (model)

****Purpose:**** User data schema and methods

****Key Features:****

- User schema definition

- Password hashing pre-save hook

- Password comparison method

- Email uniqueness validation

Itinerary.js (model)

****Purpose:**** Itinerary and destination data schema

****Key Features:****

- Nested destination schema

- User reference relationship

- Optional coordinate fields

- Timestamp tracking

🗄️ **Database Schema**

User Collection

```javascript

{

  _id: ObjectId,

  name: String (required, trimmed),

  email: String (required, unique, lowercase),

  password: String (required, hashed),

  createdAt: Date,

  updatedAt: Date

}

```

Itinerary Collection

```javascript
```

```
{
  _id: ObjectId,
  title: String (required, trimmed),
  destinations: [
    {
      location: String (required, trimmed),
      startDate: Date (required),
      endDate: Date (required),
      notes: String (optional, trimmed),
      latitude: Number (optional),
      longitude: Number (optional)
    }
  ],
  user: ObjectId (required, ref: 'User'),
  createdAt: Date,
  updatedAt: Date
}
```

🔐 **Authentication Flow**

Registration Process

1. User submits registration form
2. Frontend validates input fields
3. Backend validates data using express-validator
4. Password is hashed using bcryptjs

5. User is saved to MongoDB

6. JWT token is generated and returned

7. Frontend stores token and updates auth state

Login Process

1. User submits login credentials

2. Backend finds user by email

3. Password is compared using bcrypt

4. JWT token is generated if valid

5. Token is returned to frontend

6. Frontend stores token and updates auth state

Protected Route Access

1. Frontend checks for stored token

2. Token is sent in Authorization header

3. Backend middleware validates token

4. User object is attached to request

5. Route handler processes authenticated request

💻 **Development Prompts Used**

Initial Project Setup Prompt

```

I want to build a full-stack itinerary planner web app using React for frontend, Node.js with Express for backend, and MongoDB for the database.
```

Requirements:

- User authentication with JWT (signup, login, protected routes).

- Backend REST API with routes to create, read, and delete itineraries.

- Each itinerary has a title, multiple destinations with location name, start/end dates, and optional notes.

- Locations will be saved as text only (no geocoding required).

- Frontend React app with pages for login, registration, dashboard, and itinerary management.

- Itinerary page should display a list of itineraries and a form to add new ones.

- Include a Google Map that shows markers only if latitude and longitude are available.

- Use axios or fetch for API calls.

- Proper error handling and loading states.

- Use environment variables for API keys and backend URLs.

- Use React Router for navigation with protected routes.

- Provide schema/model definitions for MongoDB with Mongoose.

- Backend should validate required fields but allow null lat/lng.

- Use best practices for security and code structure.

```

**### UI Enhancement Prompt**

```

can you make visible the name of the user logged in example "Hello User_name" on top right with logout option and please make ui more attractive while enter itinerary creation and there is no option for edit itinerary please make it too

```

**### Google Maps API Key Update Prompt**

```

map api key : AIzaSyBxUcVV9M0ZWj-qojyifxWU5pycVE4_Up0

```

### Documentation Request Prompt

```

create a very good and full fledge documentation on this whole project with prompts that are used

```

## ✨ Features & Functionality

### User Management

- **Registration:** New user account creation with validation

- **Login:** Secure authentication with JWT tokens

- **Profile Display:** User name shown in navigation

- **Session Management:** Automatic token handling and logout

### Itinerary Management

- **Create:** Add new itineraries with multiple destinations

- **Read:** View all user itineraries in organized cards

- **Update:** Edit existing itineraries with pre-populated forms

- **Delete:** Remove itineraries with confirmation dialog

### Destination Features

- **Multiple Destinations:** Add unlimited destinations per itinerary

- **Date Management:** Start and end dates for each destination
```

- **Notes:** Optional notes for each destination

- **Coordinates:** Optional latitude/longitude for map markers

- **Dynamic Forms:** Add/remove destinations dynamically

Map Integration

- **Google Maps:** Interactive map display

- **Markers:** Automatic markers for destinations with coordinates

- **Responsive:** Adapts to different screen sizes

- **Error Handling:** Graceful fallback when API key missing

User Interface

- **Modern Design:** Card-based layout with shadows and rounded corners

- **Responsive:** Works on desktop, tablet, and mobile

- **Visual Feedback:** Loading states, error messages, success indicators

- **Intuitive Navigation:** Clear menu structure and breadcrumbs

- **Accessibility:** Proper form labels and keyboard navigation

🔒 Security Implementation

Password Security

- **Hashing:** bcryptjs with salt rounds

- **Validation:** Minimum length requirements

- **Storage:** Never store plain text passwords

JWT Security

- **Token Generation:** Secure random tokens

- **Expiration:** 7-day token lifetime

- **Validation:** Middleware verification on protected routes

- **Storage:** Client-side localStorage (consider httpOnly cookies for production)

Input Validation

- **Frontend:** Real-time form validation

- **Backend:** express-validator for all inputs

- **Sanitization:** Trim whitespace and normalize data

- **Type Checking:** Proper data type validation

API Security

- **CORS:** Configured for specific origins

- **Rate Limiting:** Consider implementing for production

- **Error Handling:** Generic error messages to prevent information leakage

- **Authorization:** User-specific data access only

🔧 Troubleshooting

Common Issues

MongoDB Connection Error

Problem: Cannot connect to MongoDB

Solution:

1. Ensure MongoDB is running: `mongod`

2. Check connection string in `.env`

3. Verify port 27017 is available

Google Maps Not Loading

Problem: Map shows error or doesn't load

Solution:

1. Verify API key in frontend `.env`
2. Enable Maps JavaScript API in Google Cloud Console
3. Check browser console for specific errors

JWT Token Issues

Problem: Authentication not working

Solution:

1. Check JWT_SECRET in backend `.env`
2. Verify token format in API requests
3. Check token expiration

CORS Errors

Problem: Frontend cannot access backend API

Solution:

1. Verify CORS configuration in server.js
2. Check API URL in frontend `.env`
3. Ensure both servers are running

Development Tips

1. **Environment Variables:** Always use `.env` files for sensitive data
2. **Error Logging:** Add console.log statements for debugging

3. **Network Tab:** Use browser dev tools to inspect API calls

4. **Database GUI:** Use MongoDB Compass for database inspection

5. **Postman:** Test API endpoints independently

Production Considerations

1. **Environment Variables:** Use production-safe values

2. **HTTPS:** Enable SSL certificates

3. **Database:** Use MongoDB Atlas or similar cloud service

4. **Deployment:** Consider platforms like Heroku, Vercel, or AWS

5. **Monitoring:** Implement logging and error tracking

6. **Performance:** Add caching and optimization

7. **Security:** Implement rate limiting and additional validation

📈 Future Enhancements

Potential Features

- **Image Upload:** Add photos to destinations

- **Weather Integration:** Show weather forecasts

- **Expense Tracking:** Budget management features

- **Social Sharing:** Share itineraries with friends

- **Offline Support:** PWA capabilities

- **Mobile App:** React Native version

- **Advanced Maps:** Route planning and directions

- **Notifications:** Email reminders for trips

Technical Improvements

- **TypeScript:** Add type safety

- **Testing:** Unit and integration tests

- **State Management:** Redux or Zustand

- **Performance:** Code splitting and lazy loading

- **SEO:** Server-side rendering with Next.js

- **Real-time:** WebSocket integration

- **Caching:** Redis implementation

- **Microservices:** Split into smaller services

📞 Support

For issues or questions:

1. Check this documentation first

2. Review error messages in browser console

3. Verify environment variable configuration

4. Test API endpoints with Postman

5. Check MongoDB connection and data

This documentation provides a comprehensive guide to understanding, setting up, and maintaining the Travel Planner application. Keep it updated as the project evolves!