Design and Implementation Documentation

1. Introduction and Overview

The project is a **Travel Planning System** designed to help users create, manage, and collaborate on travel plans. The system allows users to:

- Create travel plans with detailed itineraries.
- Invite collaborators to contribute to travel plans.
- View daily itineraries and manage travel details.
- Authenticate and authorize users securely.

The system is built using a **Django REST Framework (DRF)** backend and a **React.js** frontend. It leverages RESTful APIs for communication between the frontend and backend.

---

2. System Architecture

The system follows a **client-server architecture**:

- **Frontend:** Built with React.js, it provides an interactive user interface for managing travel plans and itineraries.
- **Backend:** Built with Django and Django REST Framework, it handles business logic, data persistence, and API endpoints.
- **Database:** SQLite is used for data storage during development.
- **Deployment:** Docker is used for containerization, ensuring consistent environments for development and deployment.

Key Components:

1. **Frontend (React.js):**

    - Pages: `TravelPlanner`, `TravelPlannerHome`, `Login`, `SignUp`.
    - Components: `DailyView`, `ItineraryCard`, `Panel`, `LocationSearch`.

2. **Backend (Django):**

    - Apps: `travel`, `users`.
    - Models: `Travel`, `Itinerary`, `CustomUser`.
    - Views: `TravelViewSet`, `ItineraryViewSet`.
    - Serializers: `TravelSerializer`, `ItinerarySerializer`.

---

3. Data Design

Database Models:

1. **Travel:**

   - Fields: `id`, `title`, `start_date`, `end_date`, `description`, `destination`, `user`, `collaborators`.
   - Relationships:
     - `user`: ForeignKey to `CustomUser` (owner of the travel plan).
     - `collaborators`: ManyToManyField to `CustomUser`.

2. **Itinerary:**

   - Fields: `id`, `travel`, `start_date`, `end_date`, `start_time`, `end_time`, `title`, `notes`, `location`, `location_lat`, `location_lon`, `location_url`, `tag`.
   - Relationships:
     - `travel`: ForeignKey to `Travel`.

3. **CustomUser:**

   - Fields: `id`, `username`, `email`, `password`, etc.
   - Relationships:
     - Related to `Travel` and `Itinerary`.

---

4. Interface Design

API Endpoints:

1. **Travel Endpoints:**

   - `GET /api/travel/`: List all travels for the authenticated user.
   - `POST /api/travel/`: Create a new travel plan.
   - `GET /api/travel/<id>/`: Retrieve details of a specific travel plan.
   - `PATCH /api/travel/<id>/`: Update a travel plan.
   - `POST /api/travel/<id>/invite_collaborator/`: Invite a collaborator.

2. **Itinerary Endpoints:**

   - `GET /api/travel/<travel_id>/itineraries/`: List itineraries for a specific travel plan.
   - `POST /api/travel/<travel_id>/itineraries/`: Create a new itinerary.

---

5. Component Design

Backend Components:

1. **Models:**

   - `Travel`: Represents a travel plan.
   - `Itinerary`: Represents an itinerary item within a travel plan.

2. **Views:**

   - `TravelViewSet`: Handles CRUD operations for travel plans and collaborator invitations.
   - `ItineraryViewSet`: Handles CRUD operations for itineraries.

3. **Serializers:**

- `TravelSerializer`: Serializes travel plans and nested itineraries.
- `ItinerarySerializer`: Serializes individual itineraries.

Frontend Components:

1. **Pages:**
    - `TravelPlanner`: Displays and manages a single travel plan.
    - `TravelPlannerHome`: Lists all travel plans for the user.
2. **Reusable Components:**
    - `DailyView`: Displays daily itineraries.
    - `ItineraryCard`: Represents a single itinerary item.
    - `Panel`: Sidebar navigation.
    - `LocationSearch`: Location input with autocomplete.

---

6. User Interface Design

Key Screens:

1. **Login Page:**
    - Allows users to log in with email and password.
2. **Sign-Up Page:**
    - Allows new users to register.
3. **Travel Planner Home:**
    - Lists all travel plans.
    - Provides options to create a new travel plan.
4. **Travel Planner:**
    - Displays details of a travel plan.
    - Allows users to add/edit itineraries and invite collaborators.
5. **Daily View:**
    - Displays itineraries for a specific day.

---

7. Assumptions and Dependencies

Assumptions:

1. Users must be authenticated to access travel plans and itineraries.
2. Collaborators can only view or edit travel plans they are invited to.
3. The system assumes valid date and time inputs for itineraries.

Dependencies:

1. **Backend:**
   - Django 4.x
   - Django REST Framework
   - SQLite (development) or PostgreSQL (production)
2. **Frontend:**
   - React.js
   - Axios for API requests
   - Radix UI for components
3. **Deployment:**
   - Docker for containerization.

---

8. Glossary of Terms

1. **Travel Plan:** A collection of itineraries and details for a specific trip.
2. **Itinerary:** A single activity or event within a travel plan.
3. **Collaborator:** A user invited to contribute to a travel plan.
4. **Daily View:** A view that displays itineraries for a specific day.
5. **API (Application Programming Interface):** A set of endpoints for communication between the frontend and backend.
6. **Serializer:** A Django REST Framework component that converts complex data types (e.g., models) into JSON and vice versa.
