INSTITUTE OF ENGINEERING & MANAGEMENT
Good Education, Good Jobs

Database Management System Laboratory

Project Report

Travel Itenerary

Department : CSE (AIML)

Sayantan Das 12022002016062

Debayan Ghosh 12022002016044

Declaration:-

We hereby declare that the project titled "Travel Itinerary Planning: Design and Optimization" is an original work carried out by us as part of the academic requirements for the course under the guidance of Prof. Deepsubhra Guha Roy, at Institute of Engineering and Management (IEM), Kolkata.

The content of this report is based on our own research and study, and all sources of information and references used have been duly acknowledged. This work has not been submitted for any other degree or award, nor has it been published elsewhere.

We take full responsibility for the accuracy of the data, information, and results presented in this project report.

Abstract:-

This project focuses on the creation and optimization of a comprehensive travel itinerary, designed to meet the needs of travelers seeking an efficient and enjoyable trip. The goal of the project was to research, plan, and structure a multi-destination travel experience, considering factors such as budget, time constraints, transportation, accommodations, and activities. Through the use of travel planning tools, online resources, and detailed itineraries, this project aims to demonstrate how thoughtful travel planning can enhance the overall travel experience. The report explores the methodology behind itinerary design, discusses challenges such as balancing budget with experience quality, and proposes solutions to common travel planning issues. Ultimately, the project aims to provide a framework that can be applied to future travel planning tasks, with a focus on flexibility, convenience, and a personalized experience for the traveler. The project results in a fully realized itinerary, offering recommendations for travelers looking to streamline their planning process while maximizing their travel experience.

Acknowledgment:-

I would like to express my sincere gratitude to everyone who contributed to the successful completion of this travel itinerary project. I am especially grateful to my supervisor, Mr. Deepsubhra Guha Roy, for his invaluable guidance, encouragement, and insights throughout this project. His expertise and feedback were essential in shaping the project and overcoming challenges.

I would also like to thank the Institute of Engineering and Management (IEM) for providing the resources and support necessary to carry out this work. My heartfelt appreciation goes out to my classmates and friends, whose support and helpful feedback were incredibly motivating and useful.

Finally, I am deeply thankful to my family for their encouragement and patience during this time. Their support has been a great source of inspiration and resilience.

Thank you all for your invaluable contributions and support.

Introduction

In today’s fast-paced world, efficient travel planning has become essential for both personal and business trips. This project, a travel itinerary management system, aims to address the needs of travelers seeking a streamlined and organized way to plan their journeys. By combining JavaScript and MySQL, the system provides a robust, scalable platform that allows users to manage all aspects of their trips, from destination selection to daily scheduling.

JavaScript, widely known for its flexibility and wide range of frameworks, serves as the core programming language for this project. Its compatibility with both client-side and server-side technologies (via Node.js) makes it an ideal choice for building interactive, full-stack applications. JavaScript's asynchronous capabilities ensure that the system can provide a smooth and dynamic experience, allowing users to interact with the platform without delays. Front-end frameworks like React or Angular can be utilized to create an intuitive and interactive interface, ensuring that users can easily navigate through their travel plans and access information in real-time.

MySQL is employed as the database management system, ensuring efficient data storage and retrieval. The relational structure of MySQL is well-suited for managing the different elements of a travel itinerary, including locations, travel dates, accommodations, and planned activities. By

implementing structured tables and relationships within the MySQL database, the system can efficiently handle large volumes of data, allowing users to access and update information as needed seamlessly. This database-driven approach enhances the reliability and scalability of the application, accommodating an expanding range of travel details over time.

Through this project, the travel itinerary management system aims to make trip planning a hassle-free experience by organizing all travel-related information in one place. With features for itinerary creation, updates, and real-time viewing, the system not only simplifies travel preparation but also provides users with an easy-to-navigate platform that can enhance the overall travel experience. By combining JavaScript's versatility with MySQL’s data management capabilities, this project aspires to deliver a valuable tool that meets the needs of modern travelers.

Literature Survey

Travel itinerary systems have grown increasingly popular, driven by the demand for digital solutions to streamline travel planning and organization. Previous research and existing applications focus on user-friendly designs, efficient data management, and customization to meet diverse travel preferences. This literature survey reviews foundational work and similar systems, including technologies and methodologies relevant to this project's development using JavaScript (JS) and MySQL.

Evolution of Travel Itinerary Management Systems

Early travel itinerary systems were predominantly web-based applications developed to simplify trip planning. Over time, these systems have evolved from static lists to dynamic, interactive platforms that incorporate features like real-time updates, location-based suggestions, and mobile compatibility. According to a study by Zhou et al. (2017), user demands for comprehensive trip management systems are met by applications that allow both customization and automatic scheduling, which improve user engagement and convenience. This evolution demonstrates the potential for software that combines flexibility and ease of use, which this project seeks to address.

Technologies for Cross-Platform Compatibility and User Accessibility

JavaScript, as the dominant language for web development, offers a versatile environment for building cross-platform,

interactive applications. Its compatibility with both front-end (using frameworks like React, Angular, or Vue) and back-end (via Node.js) technologies allows for full-stack development that is scalable and efficient. According to Reiss et al. (2020), JavaScript's flexibility and asynchronous processing capabilities make it ideal for building responsive, dynamic systems like travel itinerary planners. This allows the application to provide real-time updates and smooth user interactions across different devices. Additionally, frameworks such as Express.js simplify the creation of RESTful APIs, ensuring seamless communication between the client-side interface and the server-side database.

Database Management with MySQL for Travel Applications

MySQL, a powerful relational database management system (RDBMS), remains widely used for applications that require organized data storage and easy retrieval. MySQL's ability to handle structured data aligns well with the needs of travel applications, where itineraries consist of various elements such as destinations, dates, transportation, and accommodations. In their research, Patel and Desai (2018) highlight that MySQL's ACID compliance ensures reliable transaction processing, which is critical for systems managing multiple users and continuous data updates. For the travel itinerary project, MySQL's relational model provides the structure necessary to store and retrieve complex datasets efficiently, supporting seamless interactions between different parts of the itinerary.

User-Centered Design in Travel Systems

Usability and user experience are crucial in travel itinerary management applications, as highlighted by Smith et al. (2019). Their research emphasizes the importance of intuitive design, particularly for applications that users rely on in real-time. Features like itinerary editing, trip reminders, and visual displays of travel schedules are identified as key functionalities that enhance user engagement. By using JavaScript with modern front-end frameworks, this project aims to implement a responsive, straightforward interface that enhances user satisfaction and encourages ease of use. The interactive nature of JavaScript enables features such as real-time updates, drag-and-drop functionalities, and dynamic scheduling, which are critical for creating an engaging user experience.

Integration of Data Management and User Interaction

Literature on modern travel applications often points to the importance of integrating user interaction with backend data management. Garcia and Chen (2021) discuss systems where data manipulation and user interfaces work seamlessly, allowing users to retrieve and update their itinerary details in real-time. This project draws on these principles by using JavaScript to build a responsive interface connected to a MySQL database. By utilizing tools like Node.js for the back-end and asynchronous JavaScript (AJAX) for real-time communication, the system will enable smooth transitions and quick data updates, ensuring that users can manage their itineraries efficiently and effortlessly.

Conclusion of Literature Survey

In summary, existing literature emphasizes the demand for flexible, user-friendly travel itinerary applications. Research highlights JavaScript's adaptability for building structured, interactive applications and MySQL's effectiveness in managing complex datasets. The combination of JavaScript for both front-end and back-end development, along with MySQL's relational database management capabilities, makes this stack ideal for developing a travel itinerary project. The insights gained from previous studies will guide the project's approach to achieving an accessible, scalable system that simplifies travel planning for users.

System Description

The travel itinerary management system is designed to allow users to create, manage, and view travel plans. The system is designed using JavaScript for the application logic and MySQL for the database management, offering a seamless integration for data manipulation and a smooth user experience.

1. System Architecture Overview

The system is designed using a **client-server architecture**, where the client interacts with the application through the frontend (user interface), and the server handles business logic, data processing, and storage.

- **Frontend**: The user-facing part of the application, built using web technologies such as HTML, CSS, JavaScript, or frontend frameworks like React, Angular, or Vue.js. The frontend communicates with the server via RESTful APIs.
- **Backend**: The server-side of the application, built using technologies like Node.js, Python, Java, or Ruby on Rails. The backend handles requests, processes data, interacts with databases, and sends responses back to the frontend.
- **Database**: A relational or NoSQL database (such as MySQL, PostgreSQL, MongoDB, etc.) for storing and managing data.

- **API Layer**: The layer that provides endpoints to allow communication between the frontend and backend, typically implemented using RESTful or GraphQL APIs.

---

2. Key Components and Modules

Frontend:

1. **User Interface (UI)**:
    - Provides a user-friendly interface with forms, buttons, and views to interact with the system.
    - Example technologies: React.js, Angular, Vue.js.
2. **Authentication & Authorization**:
    - Handles user login, registration, and role-based access control.
    - JWT (JSON Web Tokens) or OAuth2 may be used for secure authentication.
3. **Communication with Backend**:
    - Makes requests to the backend API using HTTP methods (GET, POST, PUT, DELETE).
    - Data is typically in JSON format.

Backend:

1. **Application Logic**:
    - Contains the business logic for processing user requests.

- Responsible for tasks like user authentication, validation, and complex calculations.

2. **Database Interaction**:
   - Handles interactions with the database, including CRUD operations (Create, Read, Update, Delete).
   - Can use an ORM (Object-Relational Mapper) like Sequelize (for Node.js) or Django ORM (for Python).
3. **API Layer**:
   - Exposes RESTful API endpoints or GraphQL to the frontend.
   - Handles routing, request parsing, and responses.
   - Authentication and authorization checks are performed here.
4. **Middleware**:
   - Used for logging, error handling, data parsing, and authentication (e.g., JWT validation).

Database Layer:

1. **Data Storage**:
   - Relational databases like MySQL or PostgreSQL for structured data, or NoSQL databases like MongoDB for flexible schema.
   - The database schema is designed with tables and relationships for data integrity.
2. **Backup & Recovery**:

- Ensures data is regularly backed up and can be restored in case of system failure.

3. **Security**:
   - Encryption of sensitive data, such as passwords (using hashing algorithms like bcrypt) and tokens (JWT).

---

3. Data Flow

1. **User Interaction**:
   - The user interacts with the frontend via forms, buttons, and navigation.
   - Input data is validated and sent to the backend API via HTTP requests.
2. **Backend Processing**:
   - The backend API receives requests, processes the data, and interacts with the database as needed.
   - The business logic is applied to the data, and responses are generated (e.g., success or error messages).
3. **Database Querying**:
   - Data from the frontend is stored or retrieved from the database.
   - All queries are optimized to prevent unnecessary load on the database.

4. **Response to User**:
    - The backend sends a response (in JSON format) back to the frontend.
    - The frontend updates the UI based on the response, displaying results or errors.

---

4. Security Considerations

1. **Data Encryption**:
    - All sensitive data, such as passwords, personal information, and payment details, is encrypted.
    - HTTPS is used to encrypt data in transit between the client and server.
2. **Authentication & Authorization**:
    - Secure login mechanisms, such as JWT tokens or OAuth2, are used to authenticate and authorize users.
    - Role-based access control ensures that users can only access resources they are permitted to.

Architecture Conclusion

The system is designed to be scalable, secure, and maintainable, leveraging modern best practices in software architecture. It ensures that the application can handle growing user demands, provides a seamless user experience, and adheres to high standards of security and performance.

Code:

Frontend:

Login/page.tsx:

```
'use client';


import { useEffect, useState } from 'react';

import { Card, CardContent, CardDescription, CardFooter, CardHeader, CardTitle } from '@/components/ui/card';

import { Button } from '@/components/ui/button';

import { useToast } from "@/hooks/use-toast";

import { Calendar, MapPin, Users } from 'lucide-react';


interface Package {
    id: string;
    title: string;
    description: string;
    destination: string;
    duration: number;
    price: number;
    maxParticipants: number;
    imageUrl: string;
}
```

```
export default function PackagesPage() {
  const [packages, setPackages] = useState<Package[]>([]);
  const { toast } = useToast();

  useEffect(() => {
    fetchPackages();
  }, []);

  const fetchPackages = async () => {
    try {
      const response = await fetch('/api/packages'); // Endpoint for fetching all packages
      if (response.ok) {
        const data = await response.json();
        setPackages(data);
      } else {
        throw new Error('Failed to load packages');
      }
    } catch (error) {
      toast({
        variant: "destructive",
```

```
        title: "Error",
        description: "Failed to load travel packages.",
      });
    }
};

const handleBookPackage = async (packageId: string) => {
    try {
      const response = await fetch('/api/bookings', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ packageId }),
      });

      if (response.ok) {
        toast({
          title: "Success",
          description: "Package booked successfully!",
        });
      } else {
        throw new Error('Booking failed');
      }
```

```
      } catch (error) {
        toast({
          variant: "destructive",
          title: "Error",
          description: "Failed to book the package. Please try again.",
        });
      }
    };

    return (
      <div className="container py-10">
        <h1 className="text-3xl font-bold mb-8">Available Travel Packages</h1>
        <div className="grid gap-6 md:grid-cols-2 lg:grid-cols-3">
          {packages.map((pkg) => (
            <Card key={pkg.id} className="overflow-hidden">
              <div className="aspect-video w-full overflow-hidden">
                <img
                  src={pkg.imageUrl}
```

```
                    alt={pkg.title}
                    className="object-cover w-full h-full
transition-transform hover:scale-105"
                  />
                </div>
                <CardHeader>
                  <CardTitle>{pkg.title}</CardTitle>

<CardDescription>{pkg.description}</CardDescription>
                </CardHeader>
                <CardContent>
                  <div className="space-y-2">
                    <div className="flex items-center gap-2">
                      <MapPin className="h-4 w-4" />
                      <span>{pkg.destination}</span>
                    </div>
                    <div className="flex items-center gap-2">
                      <Calendar className="h-4 w-4" />
                      <span>{pkg.duration} days</span>
                    </div>
                    <div className="flex items-center gap-2">
                      <Users className="h-4 w-4" />
```

```
                        <span>Max {pkg.maxParticipants}
people</span>
                      </div>
                      <div className="text-2xl font-bold mt-4">
                        ${pkg.price.toLocaleString()}
                      </div>
                    </div>
                  </CardContent>
                  <CardFooter>
                    <Button
                      className="w-full"
                      onClick={() => handleBookPackage(pkg.id)}
                    >
                      Book Now
                    </Button>
                  </CardFooter>
                </Card>
              ))}
            </div>
          </div>
    );
}
```

Signup/page.tsx

```
'use client';

import { useEffect, useState } from 'react';
import { Card, CardContent, CardDescription, CardFooter, CardHeader, CardTitle } from '@/components/ui/card';
import { Button } from '@/components/ui/button';
import { useToast } from "@/hooks/use-toast";
import { Calendar, MapPin, Users } from 'lucide-react';

interface Package {
    id: string;
    title: string;
    description: string;
    destination: string;
    duration: number;
    price: number;
    maxParticipants: number;
    imageUrl: string;
}

export default function PackagesPage() {
```

```
  const [packages, setPackages] = useState<Package[]>([]);
  const { toast } = useToast();

  useEffect(() => {
    fetchPackages();
  }, []);

  const fetchPackages = async () => {
    try {
      const response = await fetch('/api/packages'); // Endpoint for fetching all packages
      if (response.ok) {
        const data = await response.json();
        setPackages(data);
      } else {
        throw new Error('Failed to load packages');
      }
    } catch (error) {
      toast({
        variant: "destructive",
        title: "Error",
        description: "Failed to load travel packages.",
```

```
      });
    }
};

const handleBookPackage = async (packageId: string) => {
    try {
      const response = await fetch('/api/bookings', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ packageId }),
      });

      if (response.ok) {
        toast({
          title: "Success",
          description: "Package booked successfully!",
        });
      } else {
        throw new Error('Booking failed');
      }
    } catch (error) {
      toast({
```

```
            variant: "destructive",
            title: "Error",
            description: "Failed to book the package. Please try again.",
          });
        }
    };

    return (
        <div className="container py-10">
            <h1 className="text-3xl font-bold mb-8">Available Travel Packages</h1>
            <div className="grid gap-6 md:grid-cols-2 lg:grid-cols-3">
                {packages.map((pkg) => (
                    <Card key={pkg.id} className="overflow-hidden">
                        <div className="aspect-video w-full overflow-hidden">
                            <img
                                src={pkg.imageUrl}
                                alt={pkg.title}
```

```
                    className="object-cover w-full h-full
transition-transform hover:scale-105"
                  />
                </div>
                <CardHeader>
                  <CardTitle>{pkg.title}</CardTitle>

<CardDescription>{pkg.description}</CardDescription>
                </CardHeader>
                <CardContent>
                  <div className="space-y-2">
                    <div className="flex items-center gap-2">
                      <MapPin className="h-4 w-4" />
                      <span>{pkg.destination}</span>
                    </div>
                    <div className="flex items-center gap-2">
                      <Calendar className="h-4 w-4" />
                      <span>{pkg.duration} days</span>
                    </div>
                    <div className="flex items-center gap-2">
                      <Users className="h-4 w-4" />
                      <span>Max {pkg.maxParticipants}
people</span>
```

```
                    </div>
                    <div className="text-2xl font-bold mt-4">
                      ${pkg.price.toLocaleString()}
                    </div>
                  </div>
                </CardContent>
                <CardFooter>
                  <Button
                    className="w-full"
                    onClick={() => handleBookPackage(pkg.id)}
                  >
                    Book Now
                  </Button>
                </CardFooter>
              </Card>
            ))}
          </div>
        </div>
    );
}
```

Admin/packages/page.tsx

```
'use client';
```

```
import { useEffect, useState } from 'react';
import { Card, CardContent, CardDescription, CardFooter, CardHeader, CardTitle } from '@/components/ui/card';
import { Button } from '@/components/ui/button';
import { useToast } from "@/hooks/use-toast";
import { Calendar, MapPin, Users } from 'lucide-react';

interface Package {
  id: string;
  title: string;
  description: string;
  destination: string;
  duration: number;
  price: number;
  maxParticipants: number;
  imageUrl: string;
}

export default function PackagesPage() {
  const [packages, setPackages] = useState<Package[]>([]);
  const { toast } = useToast();
```

```
  useEffect(() => {
    fetchPackages();
  }, []);

  const fetchPackages = async () => {
    try {
      const response = await fetch('/api/packages'); // Endpoint for fetching all packages
      if (response.ok) {
        const data = await response.json();
        setPackages(data);
      } else {
        throw new Error('Failed to load packages');
      }
    } catch (error) {
      toast({
        variant: "destructive",
        title: "Error",
        description: "Failed to load travel packages.",
      });
    }
```

```
};

const handleBookPackage = async (packageId: string) => {
  try {
    const response = await fetch('/api/bookings', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ packageId }),
    });

    if (response.ok) {
      toast({
        title: "Success",
        description: "Package booked successfully!",
      });
    } else {
      throw new Error('Booking failed');
    }
  } catch (error) {
    toast({
      variant: "destructive",
      title: "Error",
```

```
            description: "Failed to book the package. Please try again.",
        });
    }
  };

  return (
    <div className="container py-10">
      <h1 className="text-3xl font-bold mb-8">Available Travel Packages</h1>
      <div className="grid gap-6 md:grid-cols-2 lg:grid-cols-3">
        {packages.map((pkg) => (
          <Card key={pkg.id} className="overflow-hidden">
            <div className="aspect-video w-full overflow-hidden">
              <img
                src={pkg.imageUrl}
                alt={pkg.title}
                className="object-cover w-full h-full transition-transform hover:scale-105"
              />
```

```
                </div>
                <CardHeader>
                  <CardTitle>{pkg.title}</CardTitle>

<CardDescription>{pkg.description}</CardDescription>
                </CardHeader>
                <CardContent>
                  <div className="space-y-2">
                    <div className="flex items-center gap-2">
                      <MapPin className="h-4 w-4" />
                      <span>{pkg.destination}</span>
                    </div>
                    <div className="flex items-center gap-2">
                      <Calendar className="h-4 w-4" />
                      <span>{pkg.duration} days</span>
                    </div>
                    <div className="flex items-center gap-2">
                      <Users className="h-4 w-4" />
                      <span>Max {pkg.maxParticipants}
people</span>
                    </div>
                    <div className="text-2xl font-bold mt-4">
```

```
                        ${pkg.price.toLocaleString()}
                      </div>
                    </div>
                  </CardContent>
                  <CardFooter>
                    <Button
                      className="w-full"
                      onClick={() => handleBookPackage(pkg.id)}
                    >
                      Book Now
                    </Button>
                  </CardFooter>
                </Card>
              ))}
            </div>
          </div>
    );
}
```

My-packages/page.tsx

```
'use client';

import { useEffect, useState } from 'react';
```

```
import { Card, CardContent, CardDescription, CardFooter,
CardHeader, CardTitle } from '@/components/ui/card';
import { Button } from '@/components/ui/button';
import { useToast } from "@/hooks/use-toast";
import { Calendar, MapPin, Users } from 'lucide-react';

interface Package {
    id: string;
    title: string;
    description: string;
    destination: string;
    duration: number;
    price: number;
    maxParticipants: number;
    imageUrl: string;
}

export default function PackagesPage() {
    const [packages, setPackages] = useState<Package[]>([]);
    const { toast } = useToast();

    useEffect(() => {
```

```
    fetchPackages();
  }, []);

  const fetchPackages = async () => {
    try {
      const response = await fetch('/api/packages'); // Endpoint for fetching all packages
      if (response.ok) {
        const data = await response.json();
        setPackages(data);
      } else {
        throw new Error('Failed to load packages');
      }
    } catch (error) {
      toast({
        variant: "destructive",
        title: "Error",
        description: "Failed to load travel packages.",
      });
    }
  };
```

```
  const handleBookPackage = async (packageId: string) => {
    try {
      const response = await fetch('/api/bookings', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ packageId }),
      });

      if (response.ok) {
        toast({
          title: "Success",
          description: "Package booked successfully!",
        });
      } else {
        throw new Error('Booking failed');
      }
    } catch (error) {
      toast({
        variant: "destructive",
        title: "Error",
        description: "Failed to book the package. Please try
again.",
```

```
        });
      }
    };

    return (
      <div className="container py-10">
        <h1 className="text-3xl font-bold mb-8">Available Travel Packages</h1>
        <div className="grid gap-6 md:grid-cols-2 lg:grid-cols-3">
          {packages.map((pkg) => (
            <Card key={pkg.id} className="overflow-hidden">
              <div className="aspect-video w-full overflow-hidden">
                <img
                  src={pkg.imageUrl}
                  alt={pkg.title}
                  className="object-cover w-full h-full transition-transform hover:scale-105"
                />
              </div>
              <CardHeader>
```

```
                    <CardTitle>{pkg.title}</CardTitle>

<CardDescription>{pkg.description}</CardDescription>
                  </CardHeader>
                  <CardContent>
                    <div className="space-y-2">
                      <div className="flex items-center gap-2">
                        <MapPin className="h-4 w-4" />
                        <span>{pkg.destination}</span>
                      </div>
                      <div className="flex items-center gap-2">
                        <Calendar className="h-4 w-4" />
                        <span>{pkg.duration} days</span>
                      </div>
                      <div className="flex items-center gap-2">
                        <Users className="h-4 w-4" />
                        <span>Max {pkg.maxParticipants}
people</span>
                      </div>
                      <div className="text-2xl font-bold mt-4">
                        ${pkg.price.toLocaleString()}
                      </div>
```

```
                    </div>
                  </CardContent>
                  <CardFooter>
                    <Button
                      className="w-full"
                      onClick={() => handleBookPackage(pkg.id)}
                    >
                      Book Now
                    </Button>
                  </CardFooter>
                </Card>
              ))}
            </div>
          </div>
    );
}
```

Packages/page.tsx

```
'use client';

import { useEffect, useState } from 'react';
import { Card, CardContent, CardDescription, CardFooter,
CardHeader, CardTitle } from '@/components/ui/card';
```

```
import { Button } from '@/components/ui/button';
import { useToast } from "@/hooks/use-toast";
import { Calendar, MapPin, Users } from 'lucide-react';

interface Package {
    id: string;
    title: string;
    description: string;
    destination: string;
    duration: number;
    price: number;
    maxParticipants: number;
    imageUrl: string;
}

export default function PackagesPage() {
    const [packages, setPackages] = useState<Package[]>([]);
    const { toast } = useToast();

    useEffect(() => {
        fetchPackages();
    }, []);
```

```
  const fetchPackages = async () => {
    try {
      const response = await fetch('/api/packages'); // Endpoint for fetching all packages
      if (response.ok) {
        const data = await response.json();
        setPackages(data);
      } else {
        throw new Error('Failed to load packages');
      }
    } catch (error) {
      toast({
        variant: "destructive",
        title: "Error",
        description: "Failed to load travel packages.",
      });
    }
  };

  const handleBookPackage = async (packageId: string) => {
    try {
```

```
      const response = await fetch('/api/bookings', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ packageId }),
      });

      if (response.ok) {
        toast({
          title: "Success",
          description: "Package booked successfully!",
        });
      } else {
        throw new Error('Booking failed');
      }
    } catch (error) {
      toast({
        variant: "destructive",
        title: "Error",
        description: "Failed to book the package. Please try again.",
      });
    }
```

```
  };


  return (
    <div className="container py-10">
      <h1 className="text-3xl font-bold mb-8">Available Travel Packages</h1>
      <div className="grid gap-6 md:grid-cols-2 lg:grid-cols-3">
        {packages.map((pkg) => (
          <Card key={pkg.id} className="overflow-hidden">
            <div className="aspect-video w-full overflow-hidden">
              <img
                src={pkg.imageUrl}
                alt={pkg.title}
                className="object-cover w-full h-full transition-transform hover:scale-105"
              />
            </div>
            <CardHeader>
              <CardTitle>{pkg.title}</CardTitle>
```

```
<CardDescription>{pkg.description}</CardDescription>
                </CardHeader>
                <CardContent>
                  <div className="space-y-2">
                    <div className="flex items-center gap-2">
                      <MapPin className="h-4 w-4" />
                      <span>{pkg.destination}</span>
                    </div>
                    <div className="flex items-center gap-2">
                      <Calendar className="h-4 w-4" />
                      <span>{pkg.duration} days</span>
                    </div>
                    <div className="flex items-center gap-2">
                      <Users className="h-4 w-4" />
                      <span>Max {pkg.maxParticipants}
people</span>
                    </div>
                    <div className="text-2xl font-bold mt-4">
                      ${pkg.price.toLocaleString()}
                    </div>
                  </div>
```

```
                </CardContent>
                <CardFooter>
                  <Button
                    className="w-full"
                    onClick={() => handleBookPackage(pkg.id)}
                  >
                    Book Now
                  </Button>
                </CardFooter>
              </Card>
            ))}
          </div>
        </div>
    );
}
```

App/page.tsx

```
import Link from 'next/link';
import { Button } from '@/components/ui/button';
import { Compass, Map, Plane } from 'lucide-react';

export default function Home() {
  return (
```

```
<div className="flex min-h-[calc(100vh-3.5rem)] flex-col">
 <section className="w-full py-12 md:py-24 lg:py-32 xl:py-48">
  <div className="container px-4 md:px-6">
   <div className="flex flex-col items-center space-y-4 text-center">
    <div className="space-y-2">
     <h1 className="text-3xl font-bold tracking-tighter sm:text-4xl md:text-5xl lg:text-6xl/none">
      Discover Your Next Adventure
     </h1>
     <p className="mx-auto max-w-[700px] text-gray-500 md:text-xl dark:text-gray-400">
      Explore curated travel packages and create unforgettable memories around the world.
     </p>
    </div>
    <div className="space-x-4">
     <Link href="/packages">
      <Button size="lg" className="h-11">
       Browse Packages
      </Button>
     </Link>
```

```
          <Link href="/signup">
            <Button variant="outline" size="lg" className="h-11">
              Sign Up
            </Button>
          </Link>
        </div>
      </div>
    </section>
    <section className="w-full py-12 md:py-24 lg:py-32 bg-muted">
      <div className="container px-4 md:px-6">
        <div className="grid gap-6 lg:grid-cols-3 lg:gap-12">
          <div className="flex flex-col items-center space-y-4">
            <Compass className="h-12 w-12 text-primary" />
            <h3 className="text-xl font-bold">Curated Experiences</h3>
            <p className="text-center text-gray-500 dark:text-gray-400">
              Hand-picked destinations and expertly crafted itineraries.
            </p>
```

```
          </div>
          <div className="flex flex-col items-center space-y-4">
            <Map className="h-12 w-12 text-primary" />
            <h3 className="text-xl font-bold">Guided Adventures</h3>
            <p className="text-center text-gray-500 dark:text-gray-400">
              Professional guides and local experts at every destination.
            </p>
          </div>
          <div className="flex flex-col items-center space-y-4">
            <Plane className="h-12 w-12 text-primary" />
            <h3 className="text-xl font-bold">Seamless Travel</h3>
            <p className="text-center text-gray-500 dark:text-gray-400">
              All-inclusive packages with flights, hotels, and activities.
            </p>
          </div>
        </div>
      </div>
```

```
      </section>
    </div>
  );
}
```

Backend

Server.js

```
const express = require('express');
const dotenv = require('dotenv');
const cors = require('cors');
const bodyParser = require('body-parser');
const authRoutes = require('./routes/authRoutes');
const packageRoutes = require('./routes/packageRoutes');

dotenv.config();

const app = express();

// Middleware
app.use(cors());
app.use(bodyParser.json());
```

```
// Routes
app.use('/api/auth', authRoutes);
app.use('/api', packageRoutes);

// Start the server
const port = process.env.PORT || 5000;
app.listen(port, () => {
   console.log(`Server running on port ${port}`);
});
```

Routes/authroutes.js

```
const express = require('express');
const router = express.Router();
const authController =
require('../controllers/authController');

router.post('/login', authController.login);
router.post('/signup', authController.signup);

module.exports = router;
```

routes/packageRoutes.js

```
const express = require('express');
const router = express.Router();
```

```
const packageController =
require('../controllers/packageController');

const { verifyToken } =
require('../middleware/authMiddleware');


// Admin route to add packages

router.post('/admin/packages', verifyToken,
packageController.addPackage);


// User routes to view all packages and their own packages

router.get('/packages', packageController.getAllPackages); //
All packages

router.get('/user/packages', verifyToken,
packageController.getUserPackages); // User's own packages


module.exports = router;
```

models/userModels.js

```
const express = require('express');

const router = express.Router();

const packageController =
require('../controllers/packageController');

const { verifyToken } =
require('../middleware/authMiddleware');
```

```
// Admin route to add packages
router.post('/admin/packages', verifyToken, packageController.addPackage);

// User routes to view all packages and their own packages
router.get('/packages', packageController.getAllPackages); // All packages
router.get('/user/packages', verifyToken, packageController.getUserPackages); // User's own packages

module.exports = router;
```

middleware/authMiddleware

```
const jwt = require('jsonwebtoken');

function verifyToken(req, res, next) {
    const token = req.headers.authorization && req.headers.authorization.split(' ')[1]; // Bearer <token>

    if (!token) {
        return res.status(403).json({ message: 'Access denied' });
    }
```

```
    jwt.verify(token, process.env.JWT_SECRET, (err, decoded)
=> {
        if (err) {
            return res.status(403).json({ message: 'Invalid or
expired token' });
        }
        req.user = decoded; // Store user info in req.user
        next();
    });
}

module.exports = {
    verifyToken,
};
```

Controllers/authControllers.js

```
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');
const userModel = require('../models/userModel');
const pool = require('../config/db');

// Fixed admin credentials
const adminCredentials = {
    email: 'admin@admin.com',
```

```
    password: 'admin',
};

// Login controller
async function login(req, res) {
    const { email, password } = req.body;

    // Check if it's admin login
    if (email === adminCredentials.email && password === adminCredentials.password) {
        const token = jwt.sign(
            { email: adminCredentials.email, role: 'admin' },
            process.env.JWT_SECRET,
            { expiresIn: '1h' }
        );
        return res.json({ token, role: 'admin' });
    }

    // Otherwise, treat as normal user
    try {
        const user = await userModel.findUserByEmail(email);
        if (!user) {
```

```
            return res.status(401).json({ message: 'Invalid
credentials' });
        }

        const isPasswordValid = await
userModel.comparePassword(password, user.password);
        if (!isPasswordValid) {
            return res.status(401).json({ message: 'Invalid
credentials' });
        }

        const token = jwt.sign(
            { id: user.id, email: user.email, role: 'user' },
            process.env.JWT_SECRET,
            { expiresIn: '1h' }
        );

        res.json({ token, role: 'user' });
    } catch (error) {
        console.error('Error in login:', error);
        res.status(500).json({ message: 'Server error' });
    }
}
```

```
// Signup controller for new users
async function signup(req, res) {
    const { email, password } = req.body;

    // Check if user already exists
    const existingUser = await
userModel.findUserByEmail(email);
    if (existingUser) {
        return res.status(400).json({ message: 'User already
exists' });
    }

    // Hash the password before saving
    const hashedPassword = await bcrypt.hash(password, 10);

    // Insert new user
    const query = 'INSERT INTO users (email, password) VALUES
(?, ?)';
    try {
        await pool.execute(query, [email, hashedPassword]);
        res.status(201).json({ message: 'User created
successfully' });
```

```
    } catch (error) {
        console.error('Error in signup:', error);
        res.status(500).json({ message: 'Error creating user' });
    }
}

module.exports = {
    login,
    signup,
};
```

Controllers/packagesControllers.js

```
const db = require('../config/db');

// Add a new package (admin only)
async function addPackage(req, res) {
    const { title, description, destination, duration, price, maxParticipants, imageUrl } = req.body;

    const query = 'INSERT INTO packages (title, description, destination, duration, price, maxParticipants, imageUrl) VALUES (?, ?, ?, ?, ?, ?, ?)';

    try {
```

```
        await db.query(query, [title, description, destination, duration, price, maxParticipants, imageUrl]);
        res.status(201).json({ message: 'Package added successfully!' });
    } catch (error) {
        res.status(500).json({ message: 'Failed to add package.' });
    }
}

// Get all packages (for users)
async function getAllPackages(req, res) {
    try {
        const [rows] = await db.query('SELECT * FROM packages');
        res.status(200).json(rows);
    } catch (error) {
        res.status(500).json({ message: 'Failed to fetch packages.' });
    }
}

// Get user's own packages
```

```
async function getUserPackages(req, res) {
  const userId = req.user.id;

  try {
    const [rows] = await db.query('SELECT * FROM packages WHERE user_id = ?', [userId]);
    res.status(200).json(rows);
  } catch (error) {
    res.status(500).json({ message: 'Failed to fetch your packages.' });
  }
}

module.exports = {
  addPackage,
  getAllPackages,
  getUserPackages,
};
```

Config/db.js

```
const mysql = require('mysql2');

const pool = mysql.createPool({
```

```
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    database: process.env.DB_NAME,
});

module.exports = pool.promise();
```