TravelMate.lk - Technical Documentation

Table of Contents

1. Project Overview

1.1 Application Name

TravelMate.lk - Comprehensive Travel Guide Web Application for Sri Lanka

1.2 Project Description

TravelMate.lk is a modern, full-stack web application designed to provide comprehensive travel guidance for Sri Lanka. The platform features an AI-powered chatbot, interactive maps, personalized itinerary planning, and multilingual support to enhance the travel experience for both local and international visitors.

1.3 Key Features

- Interactive destination discovery with detailed information
- AI-powered travel assistant chatbot
- Personalized itinerary planning and management
- Interactive maps with geolocation services
- User reviews and ratings system
- Multilingual support (English, Sinhala, Tamil)
- Offline functionality for downloaded guides

- Social features for travel community building

1.4 Target Audience

- International tourists visiting Sri Lanka
- Local travelers exploring domestic destinations
- Travel enthusiasts and photographers
- Travel agencies and tour operators

2. Team Structure and Role Assignments

2.1 Team Composition

Name	Primary Role	Secondary Role	Key Responsibilities
Shalon	Software Architect & Full Stack Lead	Project Manager	Architecture design, code reviews, technical decisions, team coordination
Pamindu Hennadige	Frontend Lead Developer	Backend Developer	UI/UX implementation, React components, API integration
Chandima Nanayakkara	Backend Developer	Database Administrator	Server-side logic, API development, database optimization
Rakhitha Weerasinghe	Database Lead & Project Coordinator	Full Stack Developer	Database design, project planning, quality assurance
Mohammed Shaabik	Frontend Developer	Testing Lead	Component development, testing implementation, user experience

2.2 Role-Based Responsibilities

2.2.1 Shalon - Software Architect & Full Stack Lead

- **Architecture & Design**: System architecture planning, technology stack decisions
- **Code Quality**: Code reviews, design patterns implementation, best practices enforcement
- **Technical Leadership**: Mentoring team members, resolving complex technical issues
- **Integration**: Ensuring seamless integration between frontend and backend systems
- **Documentation**: Technical documentation oversight and review

2.2.2 Pamindu Hennadige - Frontend Lead Developer

- **UI Development**: Lead React.js development using ShadCN UI and Tailwind CSS
- **Component Architecture**: Reusable component design and implementation
- **State Management**: Implementation of global state management solutions
- **API Integration**: Frontend-backend communication and data handling

- **Responsive Design**: Ensuring mobile-first responsive design principles

2.2.3 Chandima Nanayakkara - Backend Developer

- **API Development**: RESTful API design and implementation using Node.js/Express
- **Business Logic**: Server-side application logic and data processing
- **Third-party Integration**: AI chatbot, payment gateway, and external API integrations
- **Performance Optimization**: Backend performance tuning and optimization
- **Security Implementation**: Authentication, authorization, and security measures

2.2.4 Rakhitha Weerasinghe - Database Lead & Project Coordinator

- **Database Design**: MongoDB schema design and optimization strategies
- **Data Management**: Database indexing, query optimization, and data integrity
- **Project Coordination**: Sprint planning, task tracking, and timeline management
- **Quality Assurance**: Testing coordination and quality control processes
- **Documentation**: Database documentation and project management artifacts

2.2.5 Mohammed Shaabik - Frontend Developer & Testing Lead

- **Component Development**: React component implementation and styling
- **User Experience**: Frontend user experience optimization and testing
- **Testing Strategy**: Unit testing, integration testing, and test automation
- **Browser Compatibility**: Cross-browser testing and compatibility assurance
- **Performance Testing**: Frontend performance monitoring and optimization

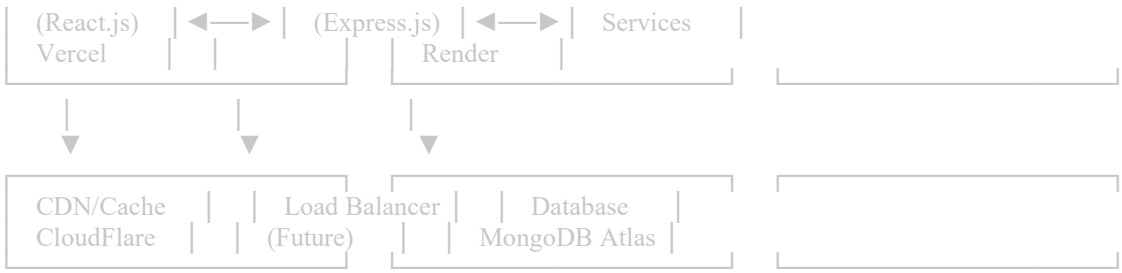
2.3 Communication Structure

- **Daily Standups**: 15-minute daily meetings for progress updates
- **Weekly Sprint Reviews**: Comprehensive progress review and planning sessions
- **Technical Discussions**: Ad-hoc technical meetings as needed
- **Code Reviews**: Mandatory peer code reviews for all commits
- **Documentation Reviews**: Weekly documentation updates and reviews

3. System Architecture

3.1 High-Level Architecture Overview

```
text
```



3.2 Microservices Architecture

```
text
Frontend Layer
├── User Interface (React.js + Tailwind CSS + ShadCN UI)
├── State Management (Zustand/Context API)
└── Service Workers (PWA Support)

API Gateway Layer
├── Authentication Middleware
├── Rate Limiting
├── Request Routing
└── Response Caching

Backend Services Layer
├── Auth Service (User Management)
├── Destination Service (Places & Attractions)
├── Itinerary Service (Trip Planning)
├── Review Service (User Feedback)
├── Chatbot Service (AI Integration)
├── Map Service (Geolocation)
└── Notification Service (Future)

Data Layer
├── MongoDB (Primary Database)
├── Redis (Caching - Future)
└── File Storage (Cloudinary)
```

3.3 Technology Integration Flow

```
text
graph TD
    A[User Request] --> B[React Frontend]
    B --> C[API Gateway]
    C --> D[Authentication Check]
    D --> E[Route to Service]
    E --> F[Business Logic]
    F --> G[Database Query]
    G --> H[Response Processing]
    H --> I[Frontend Update]
    I --> J[User Interface]
```

4. Technology Stack

4.1 Frontend Technologies

Technology	Version	Purpose	Owner
React.js	18.2+	Core frontend framework	Pamindu, Mohammed
Tailwind CSS	3.3+	Utility-first CSS framework	Pamindu, Mohammed
ShadCN UI	Latest	Pre-built component library	Pamindu, Mohammed
React Router	6.0+	Client-side routing	Pamindu
Zustand	4.0+	State management	Pamindu
React Query	3.39+	Server state management	Pamindu
Mapbox GL JS	2.15+	Interactive maps	Mohammed
Framer Motion	10.0+	Animations	Mohammed

4.2 Backend Technologies

Technology	Version	Purpose	Owner
Node.js	18+	Runtime environment	Chandima, Rakhitha
Express.js	4.18+	Web application framework	Chandima
MongoDB	6.0+	Primary database	Rakhitha
Mongoose	7.0+	MongoDB ODM	Rakhitha
JWT	9.0+	Authentication	Chandima
Bcrypt	5.1+	Password hashing	Chandima
Joi	17.9+	Input validation	Chandima
Winston	3.9+	Logging	Chandima

4.3 Development & Deployment Tools

Category	Technology	Purpose	Owner
Version Control	Git + GitHub	Code management	All
Frontend Deployment	Vercel	Static site hosting	Shalon
Backend Deployment	Render	Server hosting	Shalon
Database Hosting	MongoDB Atlas	Cloud database	Rakhitha
File Storage	Cloudinary	Image/media storage	Chandima
API Testing	Postman	API development	All
Package Manager	npm	Dependency management	All

4.4 External APIs & Services

Service	Purpose	Integration Owner
OpenAI API	AI-powered chatbot	Chandima
Mapbox API	Interactive maps and geocoding	Mohammed
Cloudinary API	Image upload and optimization	Chandima
Google Translate API	Multi-language support	Pamindu

5. Database Design

5.1 Database Schema Overview

5.1.1 Users Collection

```
javascript
{
  _id: ObjectId,
  email: String, // Unique, required
  password: String, // Hashed using bcrypt
  profile: {
    firstName: String,
    lastName: String,
    dateOfBirth: Date,
    nationality: String,
    preferences: [String], // travel interests
    language: String, // preferred language
    avatar: String // Cloudinary URL
  },
  bookmarks: [ObjectId], // destination IDs
  itineraries: [ObjectId], // itinerary IDs
  socialProfile: {
    isPublic: Boolean,
    followers: [ObjectId],
    following: [ObjectId]
  },
  settings: {
    emailNotifications: Boolean,
    pushNotifications: Boolean,
    privacy: String // public, private, friends
  },
  lastActive: Date,
  createdAt: Date,
  updatedAt: Date
}
```

5.1.2 Destinations Collection

```
javascript
{
  _id: ObjectId,
  name: String, // required
```

```
  slug: String, // URL-friendly name
  category: String, // attraction, hotel, restaurant, activity
  subcategory: String, // beach, temple, museum, etc.
  location: {
    type: "Point",
    coordinates: [Number, Number], // [longitude, latitude]
    address: String,
    city: String,
    province: String,
    zipCode: String
  },
  description: {
    en: String,
    si: String,
    ta: String
  },
  images: [{
    url: String, // Cloudinary URL
    alt: String,
    caption: String
  }],
  amenities: [String], // wifi, parking, etc.
  openingHours: {
    monday: { open: String, close: String },
    tuesday: { open: String, close: String },
    // ... other days
  },
  pricing: {
    currency: String,
    adult: Number,
    child: Number,
    foreigner: Number
  },
  ratings: {
    average: Number, // 0-5 scale
    count: Number,
    breakdown: {
      5: Number,
      4: Number,
      3: Number,
      2: Number,
      1: Number
    }
  },
  reviews: [ObjectId], // review IDs
  tags: [String], // searchable tags
  isActive: Boolean,
  featured: Boolean,
  createdBy: ObjectId, // admin user ID
  createdAt: Date,
  updatedAt: Date
}
```

5.1.3 Itineraries Collection

```javascript
{
  _id: ObjectId,
  userId: ObjectId, // creator
  title: String,
  description: String,
  coverImage: String, // Cloudinary URL
  duration: Number, // days
  days: [{
    dayNumber: Number,
    date: Date,
    destinations: [{
      destinationId: ObjectId,
      arrivalTime: String,
      departureTime: String,
      notes: String,
      transportMode: String // car, bus, train, walk
    }],
    accommodation: {
      destinationId: ObjectId,
      checkIn: String,
      checkOut: String
    },
    totalBudget: Number,
    notes: String
  }],
  metadata: {
    totalBudget: Number,
    currency: String,
    travelStyle: String, // budget, mid-range, luxury
    groupSize: Number,
    transportation: [String]
  },
  sharing: {
    isPublic: Boolean,
    allowComments: Boolean,
    allowFork: Boolean // allow others to copy
  },
  stats: {
    views: Number,
    likes: Number,
    forks: Number,
    comments: Number
  },
  createdAt: Date,
  updatedAt: Date
}
```

5.1.4 Reviews Collection

```javascript
{
  _id: ObjectId,
  userId: ObjectId,
  destinationId: ObjectId,
```

```
  rating: Number, // 1-5 scale
  title: String,
  content: String,
  images: [String], // Cloudinary URLs
  visitDate: Date,
  travelType: String, // solo, couple, family, friends
  isVerified: Boolean,
  likes: Number,
  helpful: [ObjectId], // user IDs who found helpful
  reported: [ObjectId], // user IDs who reported
  moderationStatus: String, // approved, pending, rejected
  createdAt: Date,
  updatedAt: Date
}
```

5.1.5 Chatbot Conversations Collection

```
javascript
{
  _id: ObjectId,
  userId: ObjectId, // null for anonymous users
  sessionId: String, // unique session identifier
  messages: [{
    role: String, // user, assistant
    content: String,
    timestamp: Date,
    metadata: {
      queryType: String, // destination, weather, transport
      confidence: Number,
      suggestedActions: [String]
    }
  }],
  context: {
    currentLocation: String,
    travelDates: {
      start: Date,
      end: Date
    },
    preferences: [String],
    language: String
  },
  isActive: Boolean,
  createdAt: Date,
  lastActivity: Date
}
```

5.2 Database Indexes

```
javascript
// Performance-critical indexes
db.destinations.createIndex({ "location": "2dsphere" }); // Geospatial queries
db.destinations.createIndex({ "name": "text", "description.en": "text", "tags": "text" }); // Full-text search
db.destinations.createIndex({ "category": 1, "ratings.average": -1 }); // Category filtering with ratings
```

```
db.destinations.createIndex({ "featured": -1, "createdAt": -1 }); // Featured destinations

db.users.createIndex({ "email": 1 }, { unique: true }); // Unique email constraint
db.users.createIndex({ "profile.preferences": 1 }); // Recommendation queries

db.itineraries.createIndex({ "userId": 1, "createdAt": -1 }); // User's itineraries
db.itineraries.createIndex({ "sharing.isPublic": 1, "stats.likes": -1 }); // Public popular itineraries

db.reviews.createIndex({ "destinationId": 1, "createdAt": -1 }); // Destination reviews
db.reviews.createIndex({ "userId": 1, "createdAt": -1 }); // User's reviews
db.reviews.createIndex({ "rating": -1, "createdAt": -1 }); // Top-rated reviews

db.chatbot_conversations.createIndex({ "userId": 1, "lastActivity": -1 }); // User conversations
db.chatbot_conversations.createIndex({ "sessionId": 1 }, { unique: true }); // Session lookup
```

5.3 Data Relationships

```
text
erDiagram
    USERS ||--o{ ITINERARIES : creates
    USERS ||--o{ REVIEWS : writes
    USERS ||--o{ CHATBOT_CONVERSATIONS : participates
    USERS }o--o{ DESTINATIONS : bookmarks

    DESTINATIONS ||--o{ REVIEWS : receives
    DESTINATIONS }o--o{ ITINERARIES : includes

    ITINERARIES ||--o{ ITINERARY_DAYS : contains
    ITINERARY_DAYS }o--o{ DESTINATIONS : visits
```

6. API Documentation

6.1 API Architecture Overview

The TravelMate.lk API follows RESTful principles with a clear URL structure and standardized HTTP methods. All APIs return JSON responses and use consistent error handling.

6.1.1 Base URL Structure

```
text
Production: https://api.travelmate.lk/v1
Development: http://localhost:3001/api/v1
```

6.1.2 Authentication

- **Type**: JWT Bearer Token
- **Header**: `Authorization: Bearer <token>`
- **Token Expiry**: 24 hours (access token), 7 days (refresh token)

6.2 Authentication Endpoints

6.2.1 User Registration

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

{
  "email": "user@example.com",
  "password": "SecurePassword123!",
  "firstName": "John",
  "lastName": "Doe",
  "nationality": "LK",
  "dateOfBirth": "1990-01-01"
}

Response: 201 Created
{
  "success": true,
  "message": "User registered successfully",
  "data": {
    "user": {
      "id": "64a7b8c9d1e2f3g4h5i6j7k8",
      "email": "user@example.com",
      "profile": {
        "firstName": "John",
        "lastName": "Doe"
      }
    },
    "tokens": {
      "accessToken": "eyJhbGciOiJIUzI1NiIs...",
      "refreshToken": "eyJhbGciOiJIUzI1NiIs..."
    }
  }
}
```

6.2.2 User Login

```
text
POST /auth/login
Content-Type: application/json

{
  "email": "user@example.com",
  "password": "SecurePassword123!"
}

Response: 200 OK
{
  "success": true,
  "message": "Login successful",
  "data": {
```

```
    "user": {
      "id": "64a7b8c9d1e2f3g4h5i6j7k8",
      "email": "user@example.com",
      "profile": {
        "firstName": "John",
        "lastName": "Doe"
      }
    },
    "tokens": {
      "accessToken": "eyJhbGciOiJIUzI1NiIs...",
      "refreshToken": "eyJhbGciOiJIUzI1NiIs..."
    }
  }
}
```


6.3 Destination Endpoints

6.3.1 Get All Destinations

```
text
GET /destinations?page=1&limit=20&category=attraction&city=Colombo&search=temple

Response: 200 OK
{
  "success": true,
  "data": {
    "destinations": [
      {
        "id": "64a7b8c9d1e2f3g4h5i6j7k8",
        "name": "Temple of the Sacred Tooth Relic",
        "slug": "temple-sacred-tooth-relic",
        "category": "attraction",
        "location": {
          "city": "Kandy",
          "province": "Central",
          "coordinates": [80.641, 7.294]
        },
        "ratings": {
          "average": 4.8,
          "count": 1245
        },
        "images": [
          {
            "url": "https://res.cloudinary.com/...",
            "alt": "Temple exterior view"
          }
        ]
      }
    ],
    "pagination": {
      "page": 1,
      "limit": 20,
      "total": 156,
      "pages": 8
```

```
    }
  }
}
```

6.3.2 Get Destination by ID

```
text
GET /destinations/:id

Response: 200 OK
{
  "success": true,
  "data": {
    "destination": {
      "id": "64a7b8c9d1e2f3g4h5i6j7k8",
      "name": "Temple of the Sacred Tooth Relic",
      "description": {
        "en": "A Buddhist temple in Kandy...",
        "si": "කන්දෙ‍ඹ්ගේ බුද්ධ විහාරයක්...",
        "ta": "கண்டியில் உள்ள பௌத்த கோவில்..."
      },
      "location": {
        "address": "Sri Dalada Veediya, Kandy 20000",
        "coordinates": [80.641, 7.294]
      },
      "openingHours": {
        "monday": { "open": "05:30", "close": "20:00" }
      },
      "pricing": {
        "adult": 1500,
        "child": 750,
        "foreigner": 2000
      }
    }
  }
}
```

6.4 Itinerary Endpoints

6.4.1 Create Itinerary

```
text
POST /itineraries
Authorization: Bearer <token>
Content-Type: application/json

{
  "title": "3-Day Kandy Cultural Tour",
  "description": "Explore the cultural heart of Sri Lanka",
  "duration": 3,
  "days": [
    {
```

```
      "dayNumber": 1,
      "destinations": [
        {
          "destinationId": "64a7b8c9d1e2f3g4h5i6j7k8",
          "arrivalTime": "09:00",
          "departureTime": "11:00",
          "notes": "Early morning visit recommended"
        }
      ]
    }
  ]
}

Response: 201 Created
{
  "success": true,
  "message": "Itinerary created successfully",
  "data": {
    "itinerary": {
      "id": "64a7b8c9d1e2f3g4h5i6j7k9",
      "title": "3-Day Kandy Cultural Tour",
      "userId": "64a7b8c9d1e2f3g4h5i6j7k8"
    }
  }
}
```

6.5 Chatbot Endpoints

6.5.1 Send Message to Chatbot

```
text
POST /chatbot/message
Content-Type: application/json

{
  "message": "What are the best beaches in Sri Lanka?",
  "sessionId": "session_123456789",
  "context": {
    "language": "en",
    "currentLocation": "Colombo"
  }
}

Response: 200 OK
{
  "success": true,
  "data": {
    "response": "Sri Lanka has many beautiful beaches! Here are some top recommendations...",
    "suggestions": [
      "Tell me about Mirissa Beach",
      "What activities are available at Hikkaduwa?",
      "Best time to visit beaches in Sri Lanka"
    ],
    "confidence": 0.95,
```

```
    "queryType": "destination_recommendation"
  }
}
```

6.6 Error Handling

6.6.1 Standard Error Response Format

```
json
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid input data",
    "details": [
      {
        "field": "email",
        "message": "Email is required"
      }
    ]
  },
  "timestamp": "2024-01-15T10:30:00Z"
}
```

6.6.2 HTTP Status Codes

- `200` - Success
- `201` - Created
- `400` - Bad Request
- `401` - Unauthorized
- `403` - Forbidden
- `404` - Not Found
- `429` - Too Many Requests
- `500` - Internal Server Error

7. Frontend Architecture

7.1 Project Structure

```
text
src/
├── components/
│   ├── ui/                 # ShadCN UI components
│   │   ├── button.jsx
│   │   ├── input.jsx
│   │   ├── card.jsx
│   │   └── ...
│   ├── layout/             # Layout components
```

```
        ├── Header.jsx
        ├── Footer.jsx
        ├── Navigation.jsx
        └── Sidebar.jsx
    ├── features/              # Feature-specific components
    │   ├── auth/
    │   │   ├── LoginForm.jsx
    │   │   ├── RegisterForm.jsx
    │   │   └── ProtectedRoute.jsx
    │   ├── destinations/
    │   │   ├── DestinationCard.jsx
    │   │   ├── DestinationList.jsx
    │   │   ├── DestinationDetails.jsx
    │   │   └── SearchFilters.jsx
    │   ├── itinerary/
    │   │   ├── ItineraryBuilder.jsx
    │   │   ├── DayPlanner.jsx
    │   │   └── ItineraryCard.jsx
    │   ├── map/
    │   │   ├── InteractiveMap.jsx
    │   │   ├── MapMarker.jsx
    │   │   └── RouteDisplay.jsx
    │   └── chatbot/
    │       ├── ChatWindow.jsx
    │       ├── MessageBubble.jsx
    │       └── ChatInput.jsx
    └── common/                # Reusable components
        ├── LoadingSpinner.jsx
        ├── ErrorBoundary.jsx
        ├── ImageGallery.jsx
        └── ConfirmDialog.jsx
├── pages/                     # Page components
    ├── Home.jsx
    ├── Destinations.jsx
    ├── DestinationDetail.jsx
    ├── Itinerary.jsx
    ├── Map.jsx
    ├── Profile.jsx
    └── auth/
        ├── Login.jsx
        └── Register.jsx
├── hooks/                     # Custom React hooks
    ├── useAuth.js
    ├── useDestinations.js
    ├── useItinerary.js
    ├── useMap.js
    └── useChatbot.js
├── services/                  # API service functions
    ├── api.js                 # Axios configuration
    ├── authService.js
    ├── destinationService.js
    ├── itineraryService.js
    └── chatbotService.js
├── store/                     # State management
    ├── authStore.js           # Zustand store for auth
    ├── destinationStore.js
```

```
│   └── itineraryStore.js
├── utils/                  # Helper functions
│   ├── formatters.js
│   ├── validators.js
│   ├── constants.js
│   └── dateUtils.js
├── assets/                 # Static assets
│   ├── images/
│   ├── icons/
│   └── fonts/
├── styles/                 # Global styles
│   ├── globals.css
│   └── components.css
└── lib/                    # Third-party configurations
    ├── axios.js
    ├── mapbox.js
    └── i18n.js
```

7.2 Component Architecture

7.2.1 Component Hierarchy

```
text
graph TD
    A[App.jsx] --> B[Router]
    B --> C[Layout]
    C --> D[Header]
    C --> E[Main Content]
    C --> F[Footer]

    E --> G[Home Page]
    E --> H[Destinations Page]
    E --> I[Itinerary Page]
    E --> J[Map Page]

    H --> K[SearchFilters]
    H --> L[DestinationList]
    L --> M[DestinationCard]

    I --> N[ItineraryBuilder]
    N --> O[DayPlanner]
    O --> P[DestinationSelector]
```

7.2.2 State Management Strategy

Global State (Zustand)

```
javascript
// authStore.js
import { create } from 'zustand';
import { persist } from 'zustand/middleware';
```

```
export const useAuthStore = create(
  persist(
    (set, get) => ({
      user: null,
      token: null,
      isAuthenticated: false,

      login: (userData, token) => set({
        user: userData,
        token,
        isAuthenticated: true
      }),

      logout: () => set({
        user: null,
        token: null,
        isAuthenticated: false
      }),

      updateProfile: (profileData) => set((state) => ({
        user: { ...state.user, profile: { ...state.user.profile, ...profileData } }
      }))
    }),
    {
      name: 'auth-storage',
      partialize: (state) => ({
        user: state.user,
        token: state.token,
        isAuthenticated: state.isAuthenticated
      })
    }
  )
);
```

Local State (React Hooks)

```
javascript
// useDestinations.js
import { useState, useEffect } from 'react';
import { destinationService } from '../services/destinationService';

export const useDestinations = (filters = {}) => {
  const [destinations, setDestinations] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [pagination, setPagination] = useState({});

  useEffect(() => {
    const fetchDestinations = async () => {
      try {
        setLoading(true);
        const response = await destinationService.getDestinations(filters);
        setDestinations(response.data.destinations);
        setPagination(response.data.pagination);
      } catch (err) {
```

```
      setError(err.message);
    } finally {
      setLoading(false);
    }
  };

  fetchDestinations();
 }, [filters]);

 return { destinations, loading, error, pagination };
};
```

7.3 UI/UX Design System

7.3.1 Design Tokens

```javascript
// tailwind.config.js
module.exports = {
 theme: {
  extend: {
   colors: {
    primary: {
     50: '#f0f9ff',
     500: '#3b82f6',
     600: '#2563eb',
     700: '#1d4ed8',
    },
    secondary: {
     50: '#fdf4ff',
     500: '#a855f7',
     600: '#9333ea',
    },
    accent: {
     orange: '#fb923c',
     green: '#22c55e',
    }
   },
   fontFamily: {
    sans: ['Inter', 'sans-serif'],
    display: ['Poppins', 'sans-serif'],
   },
   spacing: {
    '18': '4.5rem',
    '88': '22rem',
   }
  }
 }
};
```

7.3.2 Component Variants

```javascript
// Button component variants
const buttonVariants = {
  variant: {
    default: "bg-primary-600 text-white hover:bg-primary-700",
    outline: "border border-primary-600 text-primary-600 hover:bg-primary-50",
    ghost: "text-primary-600 hover:bg-primary-50",
    destructive: "bg-red-600 text-white hover:bg-red-700"
  },
  size: {
    sm: "h-8 px-3 text-sm",
    md: "h-10 px-4",
    lg: "h-12 px-6 text-lg"
  }
};
```

7.4 Performance Optimization

7.4.1 Code Splitting

```javascript
// Lazy loading for routes
import { lazy, Suspense } from 'react';

const Destinations = lazy(() => import('./pages/Destinations'));
const Itinerary = lazy(() => import('./pages/Itinerary'));
const Map = lazy(() => import('./pages/Map'));

// Route configuration
const routes = [
  {
    path: '/destinations',
    element: (
      <Suspense fallback={<LoadingSpinner />}>
        <Destinations />
      </Suspense>
    )
  }
];
```

7.4.2 Image Optimization

```javascript
// Optimized image component
import { useState } from 'react';

export const OptimizedImage = ({ src, alt, className, ...props }) => {
  const [isLoading, setIsLoading] = useState(true);
  const [hasError, setHasError] = useState(false);

  return (
    <div className={`relative ${className}`}>
```

```
      {isLoading && (
        <div className="absolute inset-0 bg-gray-200 animate-pulse rounded" />
      )}
      <img
        src={src}
        alt={alt}
        loading="lazy"
        onLoad={() => setIsLoading(false)}
        onError={() => {
          setHasError(true);
          setIsLoading(false);
        }}
        className={`transition-opacity duration-300 ${
          isLoading ? 'opacity-0' : 'opacity-100'
        }`}
        {...props}
      />
      {hasError && (
        <div className="absolute inset-0 bg-gray-100 flex items-center justify-center">
          <span className="text-gray-400">Image not available</span>
        </div>
      )}
    </div>
  );
};
```

8. Backend Architecture

8.1 Project Structure

```
text
backend/
├── src/
│   ├── controllers/          # Request handlers
│   │   ├── authController.js
│   │   ├── destinationController.js
│   │   ├── itineraryController.js
│   │   ├── reviewController.js
│   │   └── chatbotController.js
│   ├── middleware/           # Custom middleware
│   │   ├── auth.js           # Authentication middleware
│   │   ├── validation.js     # Input validation
│   │   ├── rateLimit.js      # Rate limiting
│   │   ├── errorHandler.js   # Global error handling
│   │   └── logger.js         # Request logging
│   ├── models/               # Database models
│   │   ├── User.js
│   │   ├── Destination.js
│   │   ├── Itinerary.js
│   │   ├── Review.js
│   │   └── ChatbotConversation.js
│   ├── routes/               # API routes
│   │   ├── auth.js
│   │   ├── destinations.js
```

```
        ├── itineraries.js
        ├── reviews.js
        └── chatbot.js
    ├── services/           # Business logic
    │   ├── authService.js
    │   ├── destinationService.js
    │   ├── itineraryService.js
    │   ├── reviewService.js
    │   ├── chatbotService.js
    │   ├── emailService.js
    │   └── uploadService.js
    ├── utils/              # Utility functions
    │   ├── passwordHash.js
    │   ├── jwtUtils.js
    │   ├── validation.js
    │   ├── dateUtils.js
    │   └── constants.js
    ├── config/             # Configuration files
    │   ├── database.js
    │   ├── jwt.js
    │   ├── cloudinary.js
    │   └── openai.js
    └── tests/              # Test files
        ├── unit/
        ├── integration/
        └── fixtures/
├── .env                    # Environment variables
├── .env.example            # Environment template
├── package.json
└── server.js               # Entry point
```

8.2 Service Layer Architecture

8.2.1 Authentication Service

```
javascript
// authService.js
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const User = require('../models/User');
const { generateTokens, verifyRefreshToken } = require('../utils/jwtUtils');

class AuthService {
  async register(userData) {
    const { email, password, firstName, lastName, nationality, dateOfBirth } = userData;

    // Check if user already exists
    const existingUser = await User.findOne({ email });
    if (existingUser) {
      throw new Error('User already exists with this email');
    }

    // Hash password
    const hashedPassword = await bcrypt.hash(password, 12);

    // Create user
    const user = new User({
      email,
      password: hashedPassword,
      profile: {
        firstName,
        lastName,
        nationality,
        dateOfBirth: new Date(dateOfBirth),
        language: 'en'
      }
    });

    await user.save();

    // Generate tokens
    const tokens = generateTokens(user._id);

    return {
      user: {
        id: user._id,
        email: user.email,
        profile: user.profile
      },
      tokens
    };
  }

  async login(email, password) {
    // Find user
```

```
    const user = await User.findOne({ email });
    if (!user) {
      throw new Error('Invalid credentials');
    }

    // Verify password
    const isValidPassword = await bcrypt.compare(password, user.password);
    if (!isValidPassword) {
      throw new Error('Invalid credentials');
    }

    // Update last active
    user.lastActive = new Date();
    await user.save();

    // Generate tokens
    const tokens = generateTokens(user._id);

    return {
      user: {
        id: user._id,
        email: user.email,
        profile: user.profile
      },
      tokens
    };
  }

  async refreshToken(refreshToken) {
    const decoded = verifyRefreshToken(refreshToken);
    const user = await User.findById(decoded.userId);

    if (!user) {
      throw new Error('Invalid refresh token');
    }

    return generateTokens(user._id);
  }
}

module.exports = new AuthService();
```

8.2.2 Destination Service

```
javascript
// destinationService.js
const Destination = require('../models/Destination');
const Review = require('../models/Review');

class DestinationService {
  async getDestinations(filters = {}) {
    const { page = 1, limit = 20, category, city, search, sortBy = 'ratings.average' } = filters;

    // Build query
    const query = {};
```

```
    if (category) query.category = category;
    if (city) query['location.city'] = new RegExp(city, 'i');
    if (search) {
      query.$text = { $search: search };
    }

    // Execute query with pagination
    const destinations = await Destination
      .find(query)
      .sort({ [sortBy]: -1 })
      .limit(limit * 1)
      .skip((page - 1) * limit)
      .select('-__v')
      .lean();

    const total = await Destination.countDocuments(query);

    return {
      destinations,
      pagination: {
        page: parseInt(page),
        limit: parseInt(limit),
        total,
        pages: Math.ceil(total / limit)
      }
    };
  }

  async getDestinationById(id) {
    const destination = await Destination
      .findById(id)
      .populate('reviews', 'rating title content userId createdAt')
      .lean();

    if (!destination) {
      throw new Error('Destination not found');
    }

    return destination;
  }

  async getNearbyDestinations(longitude, latitude, radius = 5000) {
    const destinations = await Destination.find({
      location: {
        $near: {
          $geometry: {
            type: 'Point',
            coordinates: [longitude, latitude]
          },
          $maxDistance: radius
        }
      }
    }).limit(10);

    return destinations;
```

```
  }

  async searchDestinations(searchTerm, filters = {}) {
    const query = {
      $text: { $search: searchTerm },
      ...filters
    };

    const destinations = await Destination
      .find(query, { score: { $meta: 'textScore' } })
      .sort({ score: { $meta: 'textScore' } })
      .limit(50);

    return destinations;
  }
}

module.exports = new DestinationService();
```

8.2.3 Chatbot Service

```
javascript
// chatbotService.js
const OpenAI = require('openai');
const ChatbotConversation = require('../models/ChatbotConversation');
const destinationService = require('./destinationService');

const openai = new OpenAI({
  apiKey: process.env.OPENAI_API_KEY
});

class ChatbotService {
  constructor() {
    this.systemPrompt = `You are a helpful travel assistant for Sri Lanka. You have extensive knowledge about:
    - Tourist attractions and destinations
    - Local culture and traditions
    - Transportation options
    - Weather patterns
    - Local cuisine and restaurants
    - Hotels and accommodations
    - Travel tips and safety advice

    Always provide accurate, helpful, and culturally sensitive information about Sri Lanka.
    Keep responses conversational and engaging while being informative.`;
  }

  async processMessage(message, sessionId, userId = null, context = {}) {
    try {
      // Get or create conversation
      let conversation = await ChatbotConversation.findOne({ sessionId });

      if (!conversation) {
        conversation = new ChatbotConversation({
          userId,
          sessionId,
```

```
      messages: [],
      context
    });
  }

  // Add user message
  conversation.messages.push({
    role: 'user',
    content: message,
    timestamp: new Date()
  });

  // Prepare messages for OpenAI
  const messages = [
    { role: 'system', content: this.systemPrompt },
    ...conversation.messages.slice(-10).map(msg => ({
      role: msg.role,
      content: msg.content
    }))
  ];

  // Get AI response
  const response = await openai.chat.completions.create({
    model: 'gpt-3.5-turbo',
    messages,
    max_tokens: 500,
    temperature: 0.7
  });

  const aiResponse = response.choices[0].message.content;

  // Add AI response to conversation
  conversation.messages.push({
    role: 'assistant',
    content: aiResponse,
    timestamp: new Date(),
    metadata: {
      queryType: this.detectQueryType(message),
      confidence: 0.9
    }
  });

  conversation.lastActivity = new Date();
  await conversation.save();

  // Generate suggestions
  const suggestions = await this.generateSuggestions(message, aiResponse);

  return {
    response: aiResponse,
    suggestions,
    confidence: 0.9,
    queryType: this.detectQueryType(message)
  };

} catch (error) {
```

```
      console.error('Chatbot service error:', error);
      return {
        response: "I'm sorry, I'm having trouble processing your request right now. Please try again later.",
        suggestions: ["What are popular destinations in Sri Lanka?", "Tell me about Sri Lankan cuisine"],
        confidence: 0.1,
        queryType: 'error'
      };
    }
  }

  detectQueryType(message) {
    const message_lower = message.toLowerCase();

    if (message_lower.includes('beach') || message_lower.includes('sea')) return 'beaches';
    if (message_lower.includes('temple') || message_lower.includes('religious')) return 'culture';
    if (message_lower.includes('food') || message_lower.includes('restaurant')) return 'cuisine';
    if (message_lower.includes('hotel') || message_lower.includes('accommodation')) return 'accommodation';
    if (message_lower.includes('transport') || message_lower.includes('travel')) return 'transportation';
    if (message_lower.includes('weather') || message_lower.includes('climate')) return 'weather';

    return 'general';
  }

  async generateSuggestions(userMessage, aiResponse) {
    const queryType = this.detectQueryType(userMessage);

    const suggestionMap = {
      beaches: [
        "What's the best time to visit beaches?",
        "Which beaches are good for surfing?",
        "Tell me about beach safety in Sri Lanka"
      ],
      culture: [
        "What are the cultural etiquettes to follow?",
        "When are the main festivals celebrated?",
        "Tell me about ancient kingdoms of Sri Lanka"
      ],
      cuisine: [
        "What are must-try Sri Lankan dishes?",
        "Where can I find authentic local food?",
        "Are there vegetarian options available?"
      ],
      general: [
        "What are the top 10 places to visit?",
        "How many days do I need for Sri Lanka?",
        "What's the best way to get around?"
      ]
    };

    return suggestionMap[queryType] || suggestionMap.general;
  }
}

module.exports = new ChatbotService();
```

8.3 Middleware Implementation

8.3.1 Authentication Middleware

```
javascript
// middleware/auth.js
const jwt = require('jsonwebtoken');
const User = require('../models/User');

const authMiddleware = async (req, res, next) => {
  try {
    const token = req.header('Authorization')?.replace('Bearer ', '');

    if (!token) {
      return res.status(401).json({
        success: false,
        error: {
          code: 'NO_TOKEN',
          message: 'Access token is required'
        }
      });
    }

    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    const user = await User.findById(decoded.userId).select('-password');

    if (!user) {
      return res.status(401).json({
        success: false,
        error: {
          code: 'INVALID_TOKEN',
          message: 'Invalid access token'
        }
      });
    }

    req.user = user;
    next();
  } catch (error) {
    res.status(401).json({
      success: false,
      error: {
        code: 'INVALID_TOKEN',
        message: 'Invalid access token'
      }
    });
  }
};

module.exports = authMiddleware;
```

8.3.2 Validation Middleware

```javascript
// middleware/validation.js
const Joi = require('joi');

const validate = (schema) => {
  return (req, res, next) => {
    const { error } = schema.validate(req.body);

    if (error) {
      const details = error.details.map(detail => ({
        field: detail.path.join('.'),
        message: detail.message
      }));

      return res.status(400).json({
        success: false,
        error: {
          code: 'VALIDATION_ERROR',
          message: 'Invalid input data',
          details
        }
      });
    }

    next();
  };
};

// Validation schemas
const schemas = {
  register: Joi.object({
    email: Joi.string().email().required(),
    password: Joi.string().min(8).pattern(new RegExp('^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-
9])(?=.*[!@#\$%\^&\*])')).required(),
    firstName: Joi.string().min(2).max(50).required(),
    lastName: Joi.string().min(2).max(50).required(),
    nationality: Joi.string().length(2).required(),
    dateOfBirth: Joi.date().max('now').required()
  }),

  login: Joi.object({
    email: Joi.string().email().required(),
    password: Joi.string().required()
  }),

  destination: Joi.object({
    name: Joi.string().required().max(100),
    category: Joi.string().valid('attraction', 'hotel', 'restaurant', 'activity').required(),
    location: Joi.object({
      coordinates: Joi.array().items(Joi.number()).length(2).required(),
      address: Joi.string().required(),
      city: Joi.string().required(),
      province: Joi.string().required()
    }).required(),
    description: Joi.object({
      en: Joi.string().required(),
```

```
      si: Joi.string(),
      ta: Joi.string()
    }).required()
  })
};

module.exports = { validate, schemas };
```

9. Security Implementation

9.1 Authentication & Authorization

9.1.1 JWT Implementation

```
javascript
// utils/jwtUtils.js
const jwt = require('jsonwebtoken');

const generateTokens = (userId) => {
  const accessToken = jwt.sign(
    { userId, type: 'access' },
    process.env.JWT_SECRET,
    { expiresIn: '24h' }
  );

  const refreshToken = jwt.sign(
    { userId, type: 'refresh' },
    process.env.REFRESH_SECRET,
    { expiresIn: '7d' }
  );

  return { accessToken, refreshToken };
};

const verifyAccessToken = (token) => {
  return jwt.verify(token, process.env.JWT_SECRET);
};

const verifyRefreshToken = (token) => {
  return jwt.verify(token, process.env.REFRESH_SECRET);
};

module.exports = {
  generateTokens,
  verifyAccessToken,
  verifyRefreshToken
};
```

9.1.2 Rate Limiting

```
javascript
```

```
// middleware/rateLimit.js
const rateLimit = require('express-rate-limit');

const createRateLimiter = (windowMs, max, message) => {
  return rateLimit({
    windowMs,
    max,
    message: {
      success: false,
      error: {
        code: 'RATE_LIMIT_EXCEEDED',
        message
      }
    },
    standardHeaders: true,
    legacyHeaders: false
  });
};

const rateLimiters = {
  general: createRateLimiter(15 * 60 * 1000, 1000, 'Too many requests'),
  auth: createRateLimiter(15 * 60 * 1000, 5, 'Too many authentication attempts'),
  chatbot: createRateLimiter(60 * 1000, 30, 'Too many chatbot requests')
};

module.exports = rateLimiters;
```

9.2 Data Protection

9.2.1 Input Sanitization

```
javascript
// utils/sanitization.js
const mongoSanitize = require('express-mongo-sanitize');
const xss = require('xss');

const sanitizeInput = (input) => {
  if (typeof input === 'string') {
    return xss(input);
  }
  if (typeof input === 'object' && input !== null) {
    const sanitized = {};
    for (const [key, value] of Object.entries(input)) {
      sanitized[key] = sanitizeInput(value);
    }
    return sanitized;
  }
  return input;
};

const sanitizationMiddleware = (req, res, next) => {
  req.body = sanitizeInput(req.body);
  req.query = sanitizeInput(req.query);
  req.params = sanitizeInput(req.params);
```

```
  next();
};

module.exports = { sanitizeInput, sanitizationMiddleware };
```

9.3 Environment Security

```
javascript
// config/security.js
const helmet = require('helmet');
const cors = require('cors');

const securityConfig = {
  helmet: helmet({
    contentSecurityPolicy: {
      directives: {
        defaultSrc: ["'self'"],
        styleSrc: ["'self'", "'unsafe-inline'", "https://fonts.googleapis.com"],
        fontSrc: ["'self'", "https://fonts.gstatic.com"],
        imgSrc: ["'self'", "data:", "https://res.cloudinary.com"],
        scriptSrc: ["'self'"],
        connectSrc: ["'self'", "https://api.mapbox.com"]
      }
    }
  }),

  cors: cors({
    origin: process.env.NODE_ENV === 'production'
      ? ['https://travelmate.lk', 'https://www.travelmate.lk']
      : ['http://localhost:3000', 'http://localhost:5173'],
    credentials: true,
    methods: ['GET', 'POST', 'PUT', 'DELETE', 'PATCH'],
    allowedHeaders: ['Content-Type', 'Authorization']
  })
};

module.exports = securityConfig;
```

10. Deployment Strategy

10.1 Frontend Deployment (Vercel)

10.1.1 Vercel Configuration

```
json
// vercel.json
{
  "builds": [
    {
      "src": "package.json",
      "use": "@vercel/static-build",
```

```
      "config": {
        "distDir": "dist"
      }
    }
  ],
  "routes": [
    {
      "src": "/(.*)",
      "dest": "/index.html"
    }
  ],
  "env": {
    "VITE_API_URL": "@api-url",
    "VITE_MAPBOX_TOKEN": "@mapbox-token",
    "VITE_CLOUDINARY_CLOUD_NAME": "@cloudinary-name"
  }
}
```

10.1.2 Build Configuration

```
json
// package.json (frontend)
{
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview",
    "lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0"
  },
  "devDependencies": {
    "@vitejs/plugin-react": "^4.0.3",
    "vite": "^4.4.5"
  }
}
```

10.2 Backend Deployment (Render)

10.2.1 Render Configuration

```
text
# render.yaml
services:
  - type: web
    name: travelmate-api
    env: node
    plan: starter
    buildCommand: npm install && npm run build
    startCommand: npm start
    envVars:
      - key: NODE_ENV
        value: production
      - key: PORT
```

```
      value: 10000
    - key: MONGODB_URI
      fromDatabase:
        name: travelmate-db
        property: connectionString
    - key: JWT_SECRET
      generateValue: true
    - key: REFRESH_SECRET
      generateValue: true

databases:
  - name: travelmate-db
    databaseName: travelmate
    user: travelmate_user
```

10.2.2 Production Server Setup

```
javascript
// server.js
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const helmet = require('helmet');
const compression = require('compression');

const app = express();

// Security middleware
app.use(helmet());
app.use(cors());
app.use(compression());

// Database connection
mongoose.connect(process.env.MONGODB_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true
});

// Routes
app.use('/api/v1/auth', require('./src/routes/auth'));
app.use('/api/v1/destinations', require('./src/routes/destinations'));
app.use('/api/v1/itineraries', require('./src/routes/itineraries'));
app.use('/api/v1/chatbot', require('./src/routes/chatbot'));

// Health check
app.get('/health', (req, res) => {
  res.json({ status: 'OK', timestamp: new Date().toISOString() });
});

const PORT = process.env.PORT || 3001;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

11. Development Workflow

11.1 Git Workflow

11.1.1 Branch Strategy

```
text
main (production)
├── develop (development)
    ├── feature/user-authentication (Chandima)
    ├── feature/destination-management (Rakhitha)
    ├── feature/frontend-components (Pamindu)
    ├── feature/chatbot-integration (Mohammed)
    └── feature/map-integration (Shalon)
```

11.1.2 Commit Standards

```
text
feat: add user authentication system
fix: resolve destination search bug
docs: update API documentation
style: format code according to prettier
refactor: restructure destination service
test: add unit tests for auth service
chore: update dependencies
```

11.2 Code Review Process

1. **Create Pull Request**: Developer creates PR with detailed description
2. **Automated Checks**: ESLint, Prettier, and tests run automatically
3. **Peer Review**: At least one team member reviews the code
4. **Architect Review**: Shalon reviews for architectural compliance
5. **Merge**: After approval, merge to develop branch

11.3 Quality Assurance

11.3.1 Testing Strategy

```
javascript
// Example test structure
describe('Auth Service', () => {
  describe('register', () => {
    it('should create a new user with valid data', async () => {
      const userData = {
        email: 'test@example.com',
        password: 'SecurePass123!',
        firstName: 'John',
```

```
      lastName: 'Doe',
      nationality: 'LK',
      dateOfBirth: '1990-01-01'
    };

    const result = await authService.register(userData);

    expect(result.user.email).toBe(userData.email);
    expect(result.tokens.accessToken).toBeDefined();
  });
 });
});
```

12. Project Timeline - Detailed 6-Week Plan

Week 1: Foundation & Setup

Owners: All team members

Days 1-2: Project Setup

- **Shalon**: Repository setup, architecture review, team onboarding
- **All**: Environment setup (Node.js, MongoDB, React development environment)
- **Rakhitha**: Database schema design and MongoDB Atlas setup
- **Pamindu**: Frontend project structure and ShadCN UI integration

Days 3-5: Core Infrastructure

- **Chandima**: Basic Express server setup, middleware configuration
- **Rakhitha**: Database models creation and seed data preparation
- **Pamindu & Mohammed**: UI component library setup, basic routing
- **Shalon**: Code review and integration oversight

Days 6-7: Authentication Foundation

- **Chandima**: JWT authentication system implementation
- **Pamindu**: Login/Register UI components
- **Mohammed**: Form validation and error handling
- **Rakhitha**: User model testing and validation

Week 2: Core Backend Development

Lead: Chandima, Support: Rakhitha

Days 1-3: API Development

- **Chandima**: Destination CRUD APIs, search functionality

- **Rakhitha**: Database optimization, indexing strategies
- **Pamindu**: API integration utilities for frontend
- **Mohammed**: API testing with Postman, documentation

Days 4-5: File Upload & External Services

- **Chandima**: Cloudinary integration for image uploads
- **Rakhitha**: Database performance testing
- **Shalon**: Security review and best practices implementation
- **Mohammed**: Error handling and logging setup

Days 6-7: Testing & Documentation

- **All**: Unit testing for API endpoints
- **Mohammed**: API documentation creation
- **Shalon**: Code review and architecture validation

Week 3: Frontend Development Phase 1

Lead: Pamindu, Support: Mohammed

Days 1-3: Core UI Components

- **Pamindu**: Homepage design and destination showcase
- **Mohammed**: Destination listing and filtering components
- **Chandima**: Backend API refinements based on frontend needs
- **Rakhitha**: Database query optimization

Days 4-5: User Interface Polish

- **Pamindu**: User dashboard and profile management
- **Mohammed**: Responsive design implementation
- **Shalon**: UI/UX review and feedback
- **Chandima**: Authentication integration with frontend

Days 6-7: Integration Testing

- **All**: Frontend-backend integration testing
- **Mohammed**: Cross-browser compatibility testing
- **Shalon**: Performance optimization review

Week 4: Advanced Features Phase 1

Distributed ownership based on expertise

Days 1-3: Map Integration

- **Mohammed**: Mapbox integration and interactive maps
- **Pamindu**: Map UI components and controls
- **Rakhitha**: Geospatial database queries
- **Chandima**: Location-based API endpoints

Days 4-5: Itinerary Planning

- **Pamindu**: Itinerary builder UI with drag-and-drop
- **Chandima**: Itinerary management APIs
- **Rakhitha**: Itinerary data model optimization
- **Mohammed**: Itinerary sharing functionality

Days 6-7: Chatbot Foundation

- **Chandima**: OpenAI API integration
- **Mohammed**: Chatbot UI component
- **Shalon**: AI prompt engineering and optimization
- **Pamindu**: Chat interface design

Week 5: Advanced Features Phase 2

Focus on completing core features

Days 1-2: Review System

- **Chandima**: Review APIs and moderation system
- **Mohammed**: Review UI components and rating system
- **Rakhitha**: Review data model and aggregation queries
- **Pamindu**: Review display and interaction components

Days 3-4: Multilingual Support

- **Pamindu**: i18n framework setup (react-i18next)
- **Mohammed**: Language switching functionality
- **Chandima**: Backend localization support
- **All**: Content translation coordination

Days 5-7: Performance & Optimization

- **Shalon**: Overall performance audit and optimization
- **Pamindu**: Frontend performance optimization (lazy loading, code splitting)
- **Chandima**: Backend caching and database optimization
- **Mohammed**: PWA implementation and offline functionality

Week 6: Testing, Deployment & Launch

Lead: Shalon, All hands on deck

Days 1-2: Comprehensive Testing

- **Mohammed**: End-to-end testing with Cypress
- **All**: User acceptance testing and bug fixes
- **Shalon**: Security testing and vulnerability assessment
- **Rakhitha**: Database performance testing under load

Days 3-4: Deployment Preparation

- **Shalon**: Production environment setup
- **Chandima**: Environment configuration and secrets management
- **Pamindu**: Build optimization and asset optimization
- **Rakhitha**: Database migration and production data setup

Days 5-6: Deployment & Monitoring

- **Shalon**: Backend deployment to Render
- **Pamindu**: Frontend deployment to Vercel
- **Chandima**: API monitoring and logging setup
- **Mohammed**: User testing and feedback collection

Day 7: Documentation & Handover

- **All**: Final documentation completion
- **Shalon**: Technical presentation preparation
- **Rakhitha**: User manual and API documentation finalization
- **Mohammed**: Demo preparation and testing scenarios

13. Risk Management & Contingency Plans

13.1 Technical Risks

Risk	Probability	Impact	Mitigation Strategy	Owner
API Rate Limits	Medium	High	Implement caching, fallback mechanisms	Chandima
Database Performance	Low	High	Proper indexing, query optimization	Rakhitha
Third-party Service Downtime	Low	Medium	Graceful degradation, status pages	Shalon

Risk	Probability	Impact	Mitigation Strategy	Owner
Security Vulnerabilities	Medium	High	Regular security audits, best practices	All

13.2 Project Risks

Risk	Probability	Impact	Mitigation Strategy	Owner
Scope Creep	High	Medium	Clear requirements, change control	Shalon
Team Member Unavailability	Medium	High	Cross-training, documentation	All
Timeline Delays	Medium	High	Buffer time, priority management	Rakhitha
Integration Issues	Medium	High	Early integration, regular testing	Shalon

14. Monitoring & Maintenance

14.1 Performance Monitoring

- **Frontend**: Vercel Analytics, Core Web Vitals
- **Backend**: Custom metrics, response time monitoring
- **Database**: MongoDB Atlas monitoring, query performance
- **External APIs**: Rate limit monitoring, error tracking

14.2 Error Tracking

```
javascript
// Error tracking setup
const Sentry = require('@sentry/node');

Sentry.init({
  dsn: process.env.SENTRY_DSN,
  environment: process.env.NODE_ENV
});

// Error handling middleware
const errorHandler = (err, req, res, next) => {
  Sentry.captureException(err);

  res.status(500).json({
    success: false,
    error: {
      code: 'INTERNAL_ERROR',
      message: 'Something went wrong'
    }
  });
};
```

14.3 Backup Strategy

- **Database**: Daily automated backups via MongoDB Atlas
- **Code**: Version control with Git, multiple remote repositories
- **Assets**: Cloudinary automatic backup and CDN distribution
- **Environment**: Infrastructure as Code documentation

15. Future Enhancements (Post-MVP)

Phase 2 Features (Months 2-3)

1. **Social Features**: User profiles, travel buddy matching
2. **Booking Integration**: Hotel and activity booking APIs
3. **Weather Integration**: Real-time weather information
4. **Push Notifications**: Travel alerts and recommendations

Phase 3 Features (Months 4-6)

1. **Mobile App**: React Native mobile application
2. **AR Features**: Augmented reality for landmark identification
3. **Advanced AI**: Machine learning recommendations
4. **Analytics Dashboard**: Business intelligence and user analytics