SoulSathi Travel Platform - Comprehensive Technical Documentation

Executive Summary

I've thoroughly reviewed the SoulSathi Travel Platform documentation. This is a **production-ready, full-stack travel booking platform** specializing in South Korea and international destinations, supporting both B2C (direct customers) and B2B (travel agents/partners) operations. The system is 95%+ functional with 231 total files, 200+ API endpoints, and 45+ database collections.

---

1. Project Overview

1.1 What is SoulSathi?

SoulSathi is a comprehensive travel booking platform that handles:

- **B2C Operations**: Direct customer bookings for tour packages
- **B2B Operations**: Agent/partner bookings with commission management
- **Content Management**: Dynamic website content through CMS
- **Payment Processing**: Multiple payment gateways (Razorpay, PayU, Stripe)
- **Financial Management**: Wallet system, commission tracking, tax management
- **Analytics**: Comprehensive tracking and reporting

1.2 Project Structure

```
graph TD
    A[SoulSathi Platform] --> B[Backend - Node.js/Express]
    A --> C[Frontend - Next.js 14]
    A --> D[Database - MongoDB]

    B --> B1[127 Files]
    B --> B2[REST APIs]
    B --> B3[Business Logic]

    C --> C1[104 Files]
    C --> C2[React Components]
    C --> C3[App Router]

    D --> D1[45+ Collections]
    D --> D2[Mongoose ODM]
```

Total System Size:

- **Backend**: 127 files
- **Frontend**: 104 files (90 src + 14 app)
- **Total**: 231 files
- **API Endpoints**: 200+
- **Database Collections**: 45+

---

2. Technology Stack

2.1 Backend Technologies

```
graph LR
    A[Backend Stack] --> B[Runtime: Node.js 18+]
    A --> C[Framework: Express.js]
    A --> D[Database: MongoDB]
    A --> E[ODM: Mongoose]

    F[Key Libraries] --> G[JWT Authentication]
    F --> H[Multer + Cloudinary]
    F --> I[Socket.io]
    F --> J[Nodemailer]
    F --> K[Payment SDKs]
```

Core Technologies:

- **Runtime**: Node.js 18+
- **Framework**: Express.js
- **Database**: MongoDB with Mongoose ODM
- **Authentication**: JWT (jsonwebtoken) + bcrypt
- **File Upload**: Multer (local handling) + Cloudinary (cloud storage)
- **Real-time**: Socket.io for WebSockets
- **Email**: Nodemailer
- **Security**: Helmet, CORS, Rate Limiting
- **Validation**: express-validator, Mongoose validators

Payment Integrations:

- Razorpay (primary)
- PayU (secondary)
- Stripe (international)
- Automatic failover between gateways

2.2 Frontend Technologies

```
graph LR
    A[Frontend Stack] --> B[Framework: Next.js 14]
    A --> C[UI: Tailwind CSS]
    A --> D[State: Context API]

    E[Key Features] --> F[App Router]
    E --> G[Server Components]
    E --> H[Client Components]
    E --> I[API Routes]
```

Core Technologies:

- **Framework**: Next.js 14 (App Router, NOT Pages Router)
- **UI Library**: Tailwind CSS (NO Material-UI)
- **State Management**: React Context API (NO Redux/Zustand)
- **Routing**: Next.js App Router
  - `useRouter()` from `next/navigation`
  - `usePathname()` from `next/navigation`
  - `useSearchParams()` from `next/navigation`
  - `Link` from `next/link`
- **HTTP Client**: Fetch API + custom `apiService` wrapper
- **Forms**: React hooks (useState, useEffect)
- **Notifications**: react-hot-toast

Important Note: All `react-router-dom` imports have been removed and replaced with Next.js routing.

2.3 Database Architecture

```
graph TD
    A[MongoDB Database] --> B[45+ Collections]

    B --> C[Core Collections]
    B --> D[Financial Collections]
    B --> E[Content Collections]
    B --> F[System Collections]

    C --> C1[users]
    C --> C2[packages]
    C --> C3[bookings]
    C --> C4[payments]

    D --> D1[wallets]
    D --> D2[commissions]
    D --> D3[payouts]
    D --> D4[refunds]

    E --> E1[cmscontents]
    E --> E2[popups]
    E --> E3[blogs]

    F --> F1[settings]
    F --> F2[analytics]
    F --> F3[activitylogs]
```

Database: MongoDB

- **Collections**: 45+ models
- **Indexes**: Optimized for common queries
- **Relations**: ObjectId references with Mongoose populate()
- **Validation**: Mongoose schema validators

---

3. System Architecture

3.1 High-Level Architecture

```
graph TB
    subgraph Client
        A[Web Browser]
        B[Mobile Browser]
    end

    subgraph Frontend[Next.js Frontend :3000]
        C[App Router Pages]
        D[React Components]
        E[API Service Layer]
        F[Context Providers]
    end

    subgraph Backend[Express Backend :5000]
        G[API Routes]
        H[Controllers]
        I[Services]
        J[Middleware]
    end

    subgraph Database
        K[(MongoDB)]
    end

    subgraph External
        L[Cloudinary]
        M[Razorpay/PayU/Stripe]
        N[Email Service]
        O[SMS Service]
    end

    A --> C
    B --> C
    C --> E
    E --> G
    G --> H
    H --> I
    I --> K
    I --> L
    I --> M
    I --> N
    I --> O
```

3.2 Request Flow

```
sequenceDiagram
    participant U as User
    participant F as Frontend Component
    participant S as API Service
    participant R as Backend Route
    participant C as Controller
    participant M as Model
    participant DB as MongoDB

    U->>F: User Action (e.g., Login)
    F->>S: apiService.post('/auth/login', data)
    S->>R: POST /api/auth/login
    R->>C: auth.login(req, res)
    C->>M: User.findOne({email})
    M->>DB: Query
    DB-->>M: User Document
    M-->>C: User Object
    C-->>R: Generate JWT + Response
    R-->>S: {token, user}
    S-->>F: Store Token, Update State
    F-->>U: Redirect to Dashboard
```

3.3 Authentication Flow

```
graph TD
    A[User Login Request] --> B{Valid Credentials?}
    B -->|No| C[Return 401 Error]
    B -->|Yes| D[Generate JWT Token]
    D --> E[Store in localStorage/sessionStorage]
    E --> F[Set AuthContext State]
    F --> G[Protected Route Access]

    G --> H{Token Valid?}
    H -->|No| I[Redirect to Login]
    H -->|Yes| J{Role Authorized?}
    J -->|No| K[Show Unauthorized Page]
    J -->|Yes| L[Render Protected Page]
```

---

4. Backend Deep Dive

4.1 Folder Structure

```
backend/
├── src/
│   ├── models/          # 45 Mongoose models
│   ├── controllers/     # 26 business logic files
│   ├── routes/          # 33 route definitions (31 registered)
│   ├── middleware/      # 7 middleware files
│   ├── services/        # 12 external service integrations
│   ├── utils/           # 5 helper utilities
│   ├── config/          # Database & environment config
│   ├── socket/          # Socket.io manager
│   └── server.js        # Main entry point
├── uploads/             # Local file uploads
└── package.json
```

4.2 Models (45 Models)

The database is organized into several categories:

Core Business Models (Active)

```
erDiagram
    User ||--o{ Booking : creates
    User ||--o| Wallet : has
    Package ||--o{ Booking : "booked in"
    Booking ||--o{ Payment : "paid with"
    Booking ||--o| Refund : "may have"
    Payment ||--o| Refund : "refunded to"
    User ||--o{ Commission : earns
    Booking ||--o| Commission : generates

    User {
        string firstName
        string lastName
        string email
        string password
        string role
        boolean isActive
    }

    Package {
        string title
        object destination
        object pricing
        array departures
        array itinerary
    }

    Booking {
        string bookingId
        ObjectId user
        ObjectId package
        object travelDates
        array travelers
        object pricing
        string status
    }

    Payment {
        string paymentId
        ObjectId bookingId
        number amount
        string status
        string gateway
    }
```

Key Models:

1. **User.js** - User accounts
    - Fields: firstName, lastName, email, password, role (user/agent/admin), preferences, address
    - Indexes: email (unique), phone, role
    - Relations: Bookings, Wallet, Commissions
2. **Package.js** - Tour packages (19+ fields)
    - Fields: title, slug, destination, duration, pricing, priceStructure, gstPercentage, tcsPercentage, images, itinerary, inclusions, exclusions, **departures** (array with startDate, endDate, availableSeats, isActive), accommodationDetails, citiesToExplore, mealPlan, termsAndConditions, cancellationPolicy
    - Indexes: title (text), category, type, status, destination.country, isFeatured, isPopular
    - Relations: Bookings, Reviews
3. **Booking.js** - Booking records
    - Fields: bookingId (unique), user, package, travelDates, travelers (array), contactInfo, pricing (breakdown), payment (status, history), status, assignedStaff
    - Indexes: bookingId (unique), user, package, status, payment.status
    - Relations: Payments, Refunds, MoveDepartures
4. **SmartBooking.js** - Extended booking with AI metadata
    - Extends Booking via Mongoose discriminator

    - Additional fields for AI recommendations, preferences
    - Used by `smartBooking.js` controller

5. **Payment.js** - Payment records
    - Fields: paymentId (unique), bookingId, userId, amount, currency, paymentMethod, status, gateway (razorpay/payu/stripe), gatewayPaymentId, transactionId
    - Indexes: paymentId (unique), bookingId, userId, status, gateway
    - Relations: Bookings, Refunds
6. **Wallet.js** - User/Agent wallets
    - Fields: userId (unique), balance, totalEarnings, totalWithdrawals, pendingBalance, commissionBalance, currency, status, bankDetails, upiDetails
    - **Static Method**: `getOrCreateWallet(userId)` - Gets or creates wallet □
    - Indexes: userId (unique), status
    - Relations: WalletTransactions
7. **WalletTransaction.js** - Wallet transactions
    - Fields: walletId, userId, type (credit/debit), amount, status, description, category, method, relatedBooking, relatedPayment
    - Indexes: walletId, userId, type, status, category
8. **Commission.js** - Agent commissions
    - Fields: commissionId (unique), userId, bookingId, paymentId, type (booking_commission/referral/bonus/penalty/adjustment), baseAmount, commissionRate, commissionAmount, status, tier
    - Indexes: commissionId (unique), userId, bookingId, status
9. **Payout.js** - Commission payouts
    - Fields: payoutId (unique), partnerId, amount, status (requested/processing/completed/failed/rejected), method (wallet/bank_transfer/cheque), reference
    - Indexes: payoutId (unique), partnerId
10. **TaxConfig.js** - Tax configurations
    - Fields: name, type (gst/service_charge/fee), calculation (percentage/fixed/slab), value, isActive, applicableFor, slabRates
    - Features: GST, TCS, dynamic tax calculations
    - Indexes: type, isActive, applicableFor
11. **Refund.js** - Refund management
    - Fields: refundId (unique), refundNumber (unique), booking, user, payment, amount (breakdown), refundMethod, status
    - Indexes: refundId (unique), booking, user, status
12. **MoveDeparture.js** - Date change requests
    - Fields: requestId (unique), booking, customer, originalDates, requestedDates, availableDates, requestDetails, financialImpact, status, assignedStaff
    - Indexes: requestId (unique), booking, customer, status

Content Management Models

13. **CMSComplete.js** - Dynamic content
    - Fields: contentId (unique), title, slug (unique), contentType (page/blog_post/testimonial/faq/policy/banner/widget), content (sections), excerpt, status, publishedAt, seo
    - Indexes: contentId, slug (unique), contentType, status
14. **Popup.js** - Popup configurations
    - Fields: title, type (newsletter/discount/announcement), content, targeting (userType, deviceType, pages, countries), behavior (frequency, delay, autoClose), styling, status, startDate, endDate
    - **Static Method**: `getActivePopups(filters)` - Gets active popups
    - Indexes: status, startDate, endDate
15. **SettingsComplete.js** - Website settings
    - Fields: key (unique), value, displayName, category, description, type, access (public/user/admin), validation, defaultValue, status
    - Indexes: key (unique), category, access.level, status

System Models

16. **Admin.js** - Admin accounts
17. **Staff.js** - Staff members
18. **Analytics.js** - Analytics events
19. **ActivityLog.js** - Activity tracking
20. **NotificationComplete.js** - Notifications
21. **ReviewComplete.js** - Reviews/Testimonials
22. **City.js** - City data
23. **Hotel.js** - Hotel data

Legacy/Complete Models

These exist but controllers prefer the core versions:

- UserComplete.js, PackageComplete.js, BookingComplete.js, PaymentComplete.js, AdminComplete.js, etc.

Recommendation: Use core models (User.js, Package.js, Booking.js, Payment.js) for new development.

4.3 Controllers (26 Controllers)

Controllers handle business logic. Each controller exports functions that are called by routes.

```
graph LR
    A[Route] --> B[Controller]
    B --> C[Model]
    B --> D[Service]
    B --> E[Validation]

    C --> F[Database Query]
    D --> G[External API]

    F --> H[Response]
    G --> H
```

Key Controllers:

1. **auth.js** (1,131 lines) - Authentication
    - Functions: register, login, logout, getMe, updateProfile, updatePassword, forgotPassword, resetPassword, verifyEmail, resendVerificationEmail, enableTwoFactor, verifyTwoFactorSetup, getActiveSessions
    - Features: JWT tokens, email verification, 2FA, session management, account lockout
    - Models: User, Wallet, ActivityLog, LoyaltyRewards
    - Routes: /api/auth/*
    - Status: ▯ Fully Functional
2. **user.js** - User management
    - Functions: getUsers, getUser, updateUser, deleteUser, getUserBookings, getUserProfile, updatePassword, uploadProfilePicture, getUserAnalytics
    - Features: User CRUD, advanced filtering, analytics, document management
    - Models: User, Booking, Payment, Wallet, Commission, LoyaltyRewards, ActivityLog, Document
    - Routes: /api/users/*
    - Status: ▯ Fully Functional
3. **package.js** - Package management
    - Functions: getPackages, getPackage, createPackage, updatePackage, deletePackage, uploadPackageImages, getPackageAnalytics, getFeaturedPackages, getPopularPackages, searchPackages, getPackagesByCategory, **addDeparture**, **toggleDepartureStatus**, **deleteDeparture**
    - Features: Full CRUD, image upload, search, filtering, caching, departure management
    - Models: Package, DynamicPricing, ActivityLog, Booking
    - Routes: /api/packages/*
    - Status: ▯ Fully Functional
4. **smartBooking.js** (511 lines) - Smart booking
    - Functions: checkAvailability, createBooking, getBookings, getMyBookings, getBooking, confirmBooking, requestModification, cancelBooking, getBookingAnalytics, getRecommendations, updateBookingStatus
    - Features: AI recommendations, availability checking, booking modifications
    - Models: SmartBooking (⚠ model file missing, uses Booking as fallback), Package
    - Routes: /api/bookings/* (via routes/booking.js ▯ registered)
    - Status: ▯ Fully Functional (SmartBooking model missing but works with Booking)
5. **booking.js** - Legacy booking controller
    - Functions: checkAvailability, createBooking, getBookings, getBooking, updateBookingStatus, cancelBooking, getBookingAnalytics
    - Status: ⚠ **May be unused** (routes/booking.js uses smartBooking.js instead)
    - Note: routes/bookings.js (500 lines) imports this but is NOT registered in server.js
6. **payment.js** - Payment processing
    - Functions: initiatePayment, verifyPayment, getPayment, getPayments, requestRefund, getPaymentAnalytics, processWalletPayment, processPostPaymentActions, **handleRazorpayWebhook**, **handleStripeWebhook**, **handlePayUWebhook**, processRefundRequest
    - Features: Multiple gateways, wallet payments, webhooks, refund processing, commission calculation, loyalty points
    - Models: Payment, Booking, User, Wallet, WalletTransaction, Refund, Commission, LoyaltyRewards, ActivityLog
    - Routes: /api/payments/*
    - Status: ▯ Fully Functional
7. **wallet.js** - Wallet operations
    - Functions: getMyWallet, getWallet, getAllWallets, initiateTopup, processTopupSuccess, initiateWithdrawal, getWalletTransactions, transferBetweenWallets, getWalletStats
    - Features: Wallet creation (calls `Wallet.getOrCreateWallet()`), top-up, withdrawal, transfers
    - Models: Wallet (with `getOrCreateWallet()` ▯), WalletTransaction, User, Payment, ActivityLog
    - Routes: /api/wallet/*
    - Status: ▯ Fully Functional
8. **commission.js** - Commission management
    - Functions: getCommissions, getCommissionSummary, releaseCommission, bulkReleaseCommissions, requestPayout, getPayouts, getCommissionAnalytics
    - Features: Commission tracking, payout requests, bulk operations, analytics
    - Models: Commission, User, Booking, Wallet, WalletTransaction, ActivityLog, Payout
    - Routes: /api/commission/*

- Status: Fully Functional (getPayouts uses Payout model)

9. **admin.js** (694 lines) - Admin operations
   - Functions: adminLogin, adminLogout, getDashboardStats, getAdminProfile, updateAdminProfile, getSystemAnalytics, updateSystemSettings, getSystemHealth
   - Features: Admin auth, dashboard stats, system analytics, health monitoring
   - Models: Admin, User, Booking, Package, Payment, Wallet, Commission, ActivityLog, Refund, Newsletter, Lead
   - Routes: /api/admin/*
   - Status: Fully Functional
10. **cms.js** - CMS operations
    - Functions: getAllContent, getContentBySlug, createContent, updateContent, deleteContent, uploadMedia, getMediaGallery, bulkOperations, getContentAnalytics
    - Features: Content CRUD, media upload, bulk operations, real-time updates via Socket.io
    - Models: CMSComplete, ActivityLog
    - Routes: /api/cms/* (Includes frontend compatibility routes: /api/cms/content/:contentType)
    - Status: Fully Functional (Frontend-backend API mismatch fixed)
11. **settings.js** - Settings management
    - Functions: getPublicSettings, getAllSettings, getSettingByKey, createSetting, updateSetting, bulkUpdateSettings, uploadSettingFile, initializeDefaultSettings
    - Features: Public/private settings, file uploads, bulk updates, real-time updates
    - Models: SettingsComplete
    - Routes: /api/settings/*
    - Status: Fully Functional
12. **tax.js** - Tax management
    - Functions: createTaxConfig, getTaxConfigs, getTaxConfig, updateTaxConfig, deleteTaxConfig, calculateTax, getApplicableTaxes, bulkUpdateTaxConfigs, getTaxAnalytics
    - Features: GST, TCS, slab rates, tax calculations
    - Models: TaxConfig, ActivityLog
    - Routes: /api/taxes/*
    - Status: Fully Functional
13. **analytics.js** - Analytics
    - Class: AnalyticsController
    - Methods: trackEvent, getDashboardMetrics, generateBusinessReport, getRevenueAnalytics, getRevenueTrends, getRevenueByGateway, getBookingAnalytics, getUserBehaviorAnalytics, exportData
    - Features: Event tracking, dashboard metrics, revenue analytics, user behavior, cohort analysis, real-time updates
    - Models: AnalyticsEvent, BusinessReport, RealTimeAnalytics, User, Booking, Payment, Package, ActivityLog
    - Routes: /api/analytics/*
    - Status: Fully Functional
14. **dashboard.js** - Dashboard stats
    - Functions: getDashboardStats, getRecentBookings, getRevenueChart, getActivityFeed, getPerformanceMetrics, getQuickActionsData
    - Features: Role-based stats, recent data, charts, activity feed
    - Models: User, Package, Booking, Payment, Commission, Wallet, WalletTransaction, ActivityLog
    - Routes: /api/dashboard/*
    - Status: Fully Functional
15. **moveDeparture.js** - Move departure requests
    - Functions: createMoveDepartureRequest, approveMoveRequest, processApprovedRequest, etc.
    - Features: Date change requests, approval workflow, availability checking, financial impact
    - Models: MoveDeparture, Booking, Package
    - Routes: /api/move-departure/*
    - Status: Fully Functional
16. **city.js** - City CRUD
17. **hotel.js** - Hotel CRUD
18. **reports.js** - Report generation
19. **notification.js** - Notifications
20. **upload.js** - File uploads
21. **destinations.js** - Destinations
22. **testimonials.js** - Testimonials
23. **contact.js** - Contact form
24. **staffController.js** - Staff management
25. **staffAuthController.js** - Staff auth
26. **dateChangeController.js** - Date change (legacy, use moveDeparture.js)

4.4 Routes (33 Route Files, 31 Registered)

Routes define API endpoints and map them to controllers.

```
graph TD
    A[server.js] --> B[Register Routes]

    B --> C[/api/auth]
    B --> D[/api/users]
    B --> E[/api/packages]
    B --> F[/api/bookings]
    B --> G[/api/payments]
    B --> H[/api/wallet]
    B --> I[/api/commission]
    B --> J[/api/cms]
    B --> K[/api/popups]
    B --> L[/api/settings]
    B --> M[... 20 more routes]

    C --> C1[auth.js controller]
    D --> D1[user.js controller]
    E --> E1[package.js controller]
    F --> F1[smartBooking.js controller]
```

Registered Routes in server.js (lines 114-145):

1. `/api/auth` → routes/auth.js → controllers/auth.js
2. `/api/users` → routes/user.js → controllers/user.js
3. `/api/packages` → routes/package.js → controllers/package.js
4. `/api/bookings` → routes/booking.js → controllers/**smartBooking.js** □
5. `/api/admin` → routes/admin.js → controllers/admin.js
6. `/api/payments` → routes/payment.js → controllers/payment.js
7. `/api/upload` → routes/upload.js → controllers/upload.js
8. `/api/chat/ai` → routes/aiChat.js
9. `/api/wallet` → routes/wallet.js → controllers/wallet.js
10. `/api/commission` → routes/commission.js → controllers/commission.js
11. `/api/reports` → routes/reports.js → controllers/reports.js
12. `/api/dashboard` → routes/dashboard.js → controllers/dashboard.js
13. `/api/notifications` → routes/notification.js → controllers/notification.js
14. `/api/taxes` → routes/tax.js → controllers/tax.js
15. `/api/analytics` → routes/analytics.js → controllers/analytics.js
16. `/api/*` → routes/dateChangeRoutes.js → controllers/dateChangeController.js
17. `/api/communication` → routes/communicationTest.js
18. `/api/staff` → routes/staff.js → controllers/staffController.js
19. `/api/staff/auth` → routes/staffAuth.js → controllers/staffAuthController.js
20. `/api/cms` → routes/cms.js → controllers/cms.js □
21. `/api/popups` → routes/popups.js → Popup model (direct) □
22. `/api/settings` → routes/settings.js → controllers/settings.js □
23. `/api/communications` → routes/communications.js □
24. `/api/b2b` → routes/b2b.js □
25. `/api/email` → routes/email.js □
26. `/api/destinations` → routes/destinations.js → controllers/destinations.js □
27. `/api/testimonials` → routes/testimonials.js → controllers/testimonials.js □
28. `/api/contact` → routes/contact.js → controllers/contact.js □
29. `/api/cities` → routes/city.js → controllers/city.js □
30. `/api/hotels` → routes/hotel.js → controllers/hotel.js □
31. `/api/move-departure` → routes/moveDeparture.js → controllers/moveDeparture.js □

⚠ Unregistered Route Files (NOT used):

- `routes/bookings.js` (500 lines) - NOT registered, uses both booking.js and smartBooking.js controllers
- `routes/index.js` - Centralized routes but NOT used (server.js registers individually)
- `routes/analytics_full.js` - Alternative analytics (not used)
- `routes/analytics_minimal.js` - Alternative analytics (not used)

Important: Use `routes/booking.js` (registered) which uses `smartBooking.js` controller, NOT `routes/bookings.js` (unregistered).

4.5 Middleware (7 Files)

Middleware functions run before/after route handlers.

```
graph LR
    A[Request] --> B[Middleware Chain]

    B --> C[auth.js]
    B --> D[admin.js]
    B --> E[agent.js]
    B --> F[analytics.js]

    C --> G[Controller]
    D --> G
    E --> G
    F --> G

    G --> H[Response]
    H --> I[errorHandler.js]
    I --> J[Client]
```

1. **middleware/auth.js** - Authentication
    - `protect` : Verifies JWT token, attaches `req.user`
    - `authorize(...roles)` : Checks if user has required role(s)
    - `optionalAuth` : Allows optional authentication
    - Usage: `router.get('/protected', protect, controller.fn)`
2. **middleware/admin.js** - Admin authorization
    - `admin` : Checks if user is admin/super-admin
    - `superAdmin` : Checks if user is super-admin
    - `hasPermission(permission)` : Checks specific permission
    - `bookingAccess` , `financialAccess` , `reportAccess` : Role-based access
    - Usage: `router.post('/admin-only', protect, admin, controller.fn)`
3. **middleware/agent.js** - Agent/B2B authorization
    - `agentMiddleware` : Checks if user is agent
    - `protectAgent` : Protects B2B routes
    - Usage: `router.get('/b2b/wallet', protectAgent, controller.fn)`
4. **middleware/errorHandler.js** - Global error handling
    - Catches all errors
    - Formats error responses
    - Logs errors
    - Usage: Added to Express app after all routes
5. **middleware/notFound.js** - 404 handler
    - Returns 404 for unknown routes
    - Usage: Added to Express app after all routes
6. **middleware/analytics.js** - Analytics tracking
    - `trackAPIRequests` : Tracks API calls
    - `trackUserActions` : Tracks user actions
    - Usage: Added to specific routes for tracking
7. **middleware/permissions.js** - Permission checking (if exists)

4.6 Services (12 Files)

Services handle external integrations and complex business logic.

```
graph TD
    A[Controllers] --> B[Services]

    B --> C[paymentGatewayService.js]
    B --> D[emailService.js]
    B --> E[smsService.js]
    B --> F[fileUploadService.js]

    C --> G[Razorpay API]
    C --> H[PayU API]
    C --> I[Stripe API]

    D --> J[SMTP Server]
    E --> K[SMS Gateway]
    F --> L[Cloudinary]
```

Key Services:

1. **paymentGatewayService.js** - Payment integration
    - **Razorpay**: createRazorpayOrder, verifyRazorpayPayment, getRazorpayPaymentDetails
    - **PayU**: createPayUOrder, verifyPayUPayment
    - **Stripe**: createStripePaymentIntent, verifyStripePayment
    - Features: Automatic failover, webhook handling, signature verification
2. **emailService.js** - Email sending
    - Methods: sendEmail, sendBookingConfirmation, sendPaymentConfirmation, sendDepartureDateChange
    - Provider: Nodemailer
    - Features: HTML templates, attachments
3. **smsService.js** - SMS sending
    - Methods: sendSMS, sendOTP, sendBookingConfirmation
    - Provider: Configurable (Twilio, etc.)
4. **whatsappService.js** - WhatsApp integration
    - Methods: sendWhatsAppMessage, sendBookingConfirmation
5. **fileUploadService.js** - File uploads
    - Methods: Upload to Cloudinary or local storage
    - Features: Image optimization, multiple file types
6. **smartBookingService.js** - Smartbooking logic
    - Methods: checkAvailability, createBooking
    - Features: AI recommendations, availability checking
7. **dateChangeService.js** - Date change processing
8. **staffAuthService.js** - Staff authentication logic
9. **cacheService.js** - Caching utilities
10. **dynamicPricingService.js** - Dynamic pricing calculations (if exists)
11. **reportingService.js** - Report generation
12. **uploadService.js** - Upload utilities

4.7 Main Server File (server.js)

The entry point that sets up Express, connects to MongoDB, registers routes, and starts the server.

Key Setup:

```
// 1. Environment variables
require('dotenv').config();

// 2. Database connection
const connectDB = require('./config/database');
connectDB();

// 3. Middleware setup
app.use(helmet());
app.use(cors());
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// 4. Rate limiting
const limiter = rateLimit({...});
app.use('/api/auth/login', limiter);

// 5. Routes registration (lines 114-145)
app.use('/api/auth', require('./routes/auth'));
app.use('/api/users', require('./routes/user'));
// ... 29 more routes

// 6. Error handlers
app.use(notFound);
app.use(errorHandler);

// 7. Start server
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

---

5. Frontend Deep Dive

5.1 Folder Structure

```
frontend/
├── app/                  # Next.js App Router pages (14 pages)
│   ├── page.tsx          # Homepage
│   ├── login/
│   ├── dashboard/
│   ├── admin/
│   ├── b2b/
│   └── layout.tsx        # Root layout
├── src/
│   ├── components/       # 69 React components
│   │   ├── admin/        # Admin components
│   │   ├── auth/         # Auth components
│   │   ├── booking/      # Booking components
│   │   ├── payment/      # Payment components
│   │   ├── popup/        # Popup components
│   │   └── ui/           # UI components
│   ├── services/         # 15 API service files
│   ├── contexts/         # 1 context (AuthContext)
│   ├── hooks/            # 2 custom hooks
│   └── lib/              # Utilities
└── public/               # Static assets
```

5.2 Pages (App Router)

Next.js 14 uses the App Router (NOT Pages Router).

```
graph TD
    A[App Router Structure] --> B[Public Pages]
    A --> C[Protected Pages]
    A --> D[Admin Pages]
    A --> E[B2B Pages]

    B --> B1[/]
    B --> B2[/packages]
    B --> B3[/destinations]
    B --> B4[/login]

    C --> C1[/dashboard - ProtectedRoute]
    C --> C2[/bookings - ProtectedRoute]
    C --> C3[/profile - ProtectedRoute]

    D --> D1[/admin/dashboard - AdminRoute]
    D --> D2[/admin/packages - AdminRoute]
    D --> D3[/admin/bookings - AdminRoute]
    D --> D4[/admin/content - AdminRoute]

    E --> E1[/b2b/dashboard - AgentRoute]
    E --> E2[/b2b/wallet - AgentRoute]
```

Public Pages:

1. `app/page.tsx` - Homepage (uses FeaturedPackages, PopularDestinations, Testimonials)
2. `app/packages/page.tsx` - Package listing (uses SearchPackages, SearchResults)
3. `app/packages/[id]/page.tsx` - Package detail (dynamic route)
4. `app/destinations/page.tsx` - Destinations
5. `app/about/page.tsx` - About (static)
6. `app/contact/page.tsx` - Contact (static form)
7. `app/login/page.tsx` - Login
8. `app/cart/page.tsx` - Cart (placeholder)
9. `app/wishlist/page.tsx` - Wishlist (placeholder)

Protected User Pages (ProtectedRoute): 10. `app/dashboard/page.js` - User dashboard 11. `app/bookings/page.js` - My bookings 12. `app/profile/page.js` - User profile

Admin Pages (AdminRoute): 13. `app/admin/page.js` - Admin redirect 14. `app/admin/dashboard/page.js` - Admin dashboard 15. `app/admin/packages/page.js` - Package management 16. `app/admin/bookings/page.js` - Booking management 17. `app/admin/content/page.js` - CMS management 18. `app/admin/analytics/page.js` - Analytics

B2B Pages (AgentRoute): 19. `app/b2b/dashboard/page.js` - B2B dashboard 20. `app/b2b/wallet/page.js` - B2B wallet

Layout Files:

- `app/layout.tsx` - Root layout (Providers, SiteShell, Toaster)
- `app/providers.tsx` - Context providers wrapper

5.3 Components (69 Components)

Components are organized by feature.

Admin Components (12 files)

```
graph LR
    A[Admin Pages] --> B[AdminLayout]
    B --> C[AdminDashboard]
    B --> D[PackageManagement]
    B --> E[BookingManagement]
    B --> F[AdminContentManagement]
    B --> G[AnalyticsDashboard]
```

1. **AdminDashboard.js** - Main admin dashboard
   - API Calls: `/api/admin/dashboard`, `/api/dashboard/stats`
   - Features: Overview stats, charts, recent activity
   - Status: ▯ Connected
2. **AdminLayout.js** - Admin layout wrapper
   - Features: Sidebar navigation, header, user menu

   - **Uses**: Next.js `Link` (NOT `<a>` tags) □
   - Status: □ Connected

3. **PackageManagement.js** - Package CRUD
   - API Calls: `/api/packages` (GET, POST, PUT, DELETE)
   - Features: Package list, create/edit forms, departure management
   - **Integrated with**: packageApiService.js □
   - Status: □ Connected

4. **BookingManagement.js** - Booking management
   - API Calls: `/api/bookings`, `/api/admin/bookings`
   - Status: □ Connected

5. **AdminContentManagement.js** - CMS management
   - API Calls: `/api/cms/content/:contentType` □ (Fixed)
   - Features: Content CRUD, content type selector, bulk operations
   - **Integrated with**: cmsService.js □
   - Status: □ Connected (Frontend-backend API mismatch fixed)

6. **SettingsManagement.js** - Settings management
   - API Calls: `/api/settings`
   - **Integrated with**: settingsService.js □
   - Status: □ Connected

7. **AnalyticsDashboard.js** - Analytics

8. **PaymentManagement.js** - Payment management

9. **CommunicationManagement.js** - Communication management

10. **AdminReports.js** - Reports

11. **MoveDepartureRequest.js** - Move departure form

12. **MoveDepartureStatus.js** - Move departure status

Auth Components (8 files)

```
graph TD
    A[Auth Flow] --> B[LoginForm]
    A --> C[RegisterForm]

    B --> D[authService.login]
    C --> E[authService.register]

    D --> F[AuthContext.login]
    E --> F

    F --> G[Store Token]
    G --> H[Update User State]
    H --> I[ProtectedRoute]
    I --> J{Authenticated?}
    J -->|Yes| K[Render Page]
    J -->|No| L[Redirect to Login]
```

1. **ProtectedRoute.js** - Route protection
   - **Uses**: Next.js `useRouter` (NOT react-router-dom) □
   - Features: Loading state, role checking, permission checking
   - Variants: AdminRoute, ManagerRoute, EmployeeRoute, AgentRoute, PermissionRoute
   - Status: □ Connected (Next.js routing fixed)

2. **UnauthorizedPage.js** - Access denied page
   - **Uses**: Next.js `useRouter` □
   - Status: □ Connected

3. **LoginForm.js** - Login form
   - API Calls: `/api/auth/login`
   - Status: □ Connected

4. **RegisterForm.js** - Registration form
   - API Calls: `/api/auth/register`
   - Status: □ Connected

5. **ForgotPasswordForm.js** - Password reset

- API Calls: `/api/auth/forgot-password`, `/api/auth/reset-password`
- Status: ✅ Connected

6. **AuthModal.js** - Auth modal wrapper
7. **AuthDemo.js** - Auth demo (unused)
8. **index.js** - Auth exports

Booking Components (10 files)

```
graph LR
   A[User] --> B[SmartBooking.jsx]
   B --> C[Check Availability]
   B --> D[Create Booking]

   E[User] --> F[MyBookings.jsx]
   F --> G[Fetch My Bookings]
   F --> H[View Details]
   F --> I[Cancel Booking]
```

1. **MyBookings.jsx** - User bookings list
   - API Calls: `/api/bookings/my-bookings`
   - **Uses**: Next.js `useRouter` (NOT react-router-dom) ✅
   - **UI**: Tailwind CSS (NO Material-UI) ✅
   - Features: Booking list, filters, status badges, cancel button
   - Status: ✅ Connected (Next.js routing + Tailwind CSS)
2. **SmartBooking.jsx** - Smart booking flow
   - API Calls: `/api/bookings`, `/api/bookings/check-availability`
   - **Uses**: Next.js routing ✅
   - **UI**: Tailwind CSS (NO Material-UI) ✅
   - Features: Multi-step booking, traveler details, payment integration
   - Status: ✅ Connected (Next.js routing + Tailwind CSS)
3. **BookingFlow.js** - Booking wizard
4. **BookingSteps.js** - Booking step indicators
5. **BookingManagement.js** - Admin booking management
6. **BookingModifications.js** - Booking modifications
7. **DateChangeRequestForm.js** - Date change form
8. **DateChangeRequestList.js** - Date change list
9. **MoveDepartureRequest.js** - Move departure form
10. **MoveDepartureStatus.js** - Move departure status

Search Components (4 files)

1. **SearchPackages.js** - Search and filters
   - **Uses**: Next.js `useRouter`, `useSearchParams` ✅
   - API Calls: `/api/packages/search`
   - Features: Filter form, URL param sync, search suggestions
   - Status: ✅ Connected (Next.js routing fixed)
2. **SearchResults.js** - Search results display
   - **Uses**: Next.js `useRouter` ✅
   - Status: ✅ Connected
3. **SavedSearches.js** - Saved searches
   - **Uses**: Next.js `useRouter` ✅
   - Status: ✅ Connected
4. **SearchFilters.js** - Advanced filters

Payment Components (7 files)

1. **PaymentGatewayEnhanced.jsx** - Payment gateway
   - API Calls: `/api/payments/initiate`, `/api/payments/verify`
   - Features: Razorpay, PayU, Stripe integration, failover, webhook handling
   - Status: ✅ Connected
2. **PaymentGateway.js** - Basic payment gateway

3. **PaymentHistory.js** - Payment history

4. **PaymentStatus.js** - Payment status

5. **EMIPayment.js** - EMI options

6. **InternationalPayment.js** - International payment

7. **PaymentTesting.js** - Payment testing (unused)

Wallet Components (3 files)

1. **WalletDashboard.jsx** - Wallet dashboard

    - API Calls: `/api/wallet/my`, `/api/wallet/transactions`
    - **Integrated with**: walletApiService.js □
    - Features: Balance display, transaction history, top-up/withdrawal buttons
    - Status: □ Connected (walletApiService integration)

2. **TopupForm.jsx** - Wallet top-up form

3. **WithdrawalForm.jsx** - Withdrawal form

B2B Components (1 file)

1. **B2BPricingCalculator.jsx** - B2B pricing calculator
    - API Calls: `/api/b2b/calculate-price`, `/api/wallet/balance`
    - **Integrated with**: walletApiService.js □
    - Features: Price calculation, wallet balance check, commission preview
    - Status: □ Connected

Popup Components (3 files)

1. **PopupManager.jsx** - Popup manager

    - API Calls: `/api/popups/active` □ (Fixed endpoint)
    - Features: Fetches active popups, frequency control, context filtering
    - Status: □ Connected (Endpoint fixed from /api/popups to /api/popups/active)

2. **PopupRenderer.jsx** - Popup renderer

3. **PopupSystem.jsx** - Popup system wrapper

Other Components

- **Package Components** (4): PackageDisplay, PackageCreator, PackageCreatorTabs, PackageManagement
- **Review Components** (2): ReviewSystem, ReviewForm
- **Inquiry Components** (2): InquiryForm, InquiryTracker
- **Chat Components** (2): AIChat, RealTimeChat
- **Notification Components** (3): NotificationSystem, EmailNotifications, SMSNotifications
- **UI Components** (5): Button, Card, Input, Modal, Select
- **Layout Components** (1): Header
- **User Components** (1): HomePage (legacy, may be unused)

5.4 Services (15 Files)

Frontend services handle API communication.

```
graph TD
    A[Components] --> B[Services]

    B --> C[apiService.js - Base]
    B --> D[authService.js]
    B --> E[packageApiService.js]
    B --> F[bookingApiService.js]
    B --> G[paymentApiService.js]
    B --> H[walletApiService.js]
    B --> I[cmsService.js]
    B --> J[settingsService.js]

    C --> K[Backend API]
    D --> K
    E --> K
    F --> K
    G --> K
    H --> K
    I --> K
    J --> K
```

Base Service:

1. **apiService.js** - Base API client
   - **Base URL**: `process.env.NEXT_PUBLIC_API_URL` (default: `http://localhost:5000/api`)
   - Methods: `get`, `post`, `put`, `delete`, `patch`
   - Features: Automatic token injection (from localStorage/sessionStorage), error handling, response parsing
   - Usage:

     ```
     import apiService from '@/services/apiService';
     const response = await apiService.get('/packages');
     ```

Feature Services:

2. **authService.js** - Authentication

   - Methods: `login(email, password, rememberMe)`, `register(userData)`, `logout()`, `getCurrentUser()`, `socialLogin(provider)`, `setupTokenRefresh()`
   - Features: Token storage (localStorage or sessionStorage based on rememberMe), user caching
   - Usage:

     ```
     import { authService } from '@/services/authService';
     const { user, token } = await authService.login(email, password, true);
     ```

3. **packageApiService.js** - Package operations

   - Methods:
     - `getPackages(filters)` - Get packages with filters
     - `getPackage(id)` - Get single package
     - `createPackage(data)` - Create package (admin)
     - `updatePackage(id, data)` - Update package (admin)
     - `deletePackage(id)` - Delete package (admin)
     - `getFeaturedPackages()` - Get featured packages
     - `getPopularPackages()` - Get popular packages
     - `searchPackages(query)` - Search packages
     - `getCategories()` - Get package categories
     - `getCities()` - Get cities
     - `getHotelsByCity(cityId)` - Get hotels by city
     - `addDeparture(packageId, departureData)` - Add departure
     - `toggleDepartureStatus(packageId, departureId)` - Toggle departure
     - `deleteDeparture(packageId, departureId)` - Delete departure
   - Usage: Used by PackageManagement, SearchPackages, PackageDisplay

4. **bookingApiService.js** - Booking operations

   - Methods:
     - `checkAvailability(packageId, dates)` - Check availability
     - `createBooking(bookingData)` - Create booking
     - `getUserBookings(filters)` - Get user bookings
     - `getBooking(id)` - Get single booking
     - `confirmBooking(id)` - Confirm booking
     - `cancelBooking(id, reason)` - Cancel booking
     - `requestModification(id, changes)` - Request modification
   - Usage: Used by SmartBooking, MyBookings

5. **paymentApiService.js** - Payment operations
    - Methods:
        - `initializePayment(paymentData)` - Initialize payment
        - `verifyPayment(paymentId, signature)` - Verify payment
        - `processPayment(bookingId, gateway)` - Process payment
        - `requestRefund(paymentId, reason)` - Request refund
    - Usage: Used by PaymentGatewayEnhanced
6. **walletApiService.js** - Wallet operations
    - Methods:
        - `getWallet()` - Get user wallet
        - `getOrCreateWallet()` - Get or create wallet
        - `getBalance()` - Get wallet balance
        - `getTransactions(filters)` - Get transactions
        - `requestWithdrawal(amount, method)` - Request withdrawal
    - Usage: Used by WalletDashboard, B2BPricingCalculator □
7. **cmsService.js** - CMS operations
    - Methods:
        - `getContent(contentType, filters)` - Get content by type
        - `getContentById(id)` - Get content by ID
        - `getContentBySlug(slug)` - Get content by slug
        - `createContent(data)` - Create content (admin)
        - `updateContent(id, data)` - Update content (admin)
        - `deleteContent(id)` - Delete content (admin)
        - `getTestimonials()` - Get testimonials
        - `getBlogPosts()` - Get blog posts
        - `getPages()` - Get pages
    - Usage: Used by AdminContentManagement □
8. **settingsService.js** - Settings operations
    - Methods:
        - `getSettings(category)` - Get settings
        - `updateSettings(key, value)` - Update setting (admin)
        - `getLogo()` - Get site logo
        - `getEmailTemplates()` - Get email templates
        - `getSocialLinks()` - Get social links
        - `getContactInfo()` - Get contact info
    - Usage: Used by SettingsManagement □
9. **popupService.js** - Popup operations
    - Methods:
        - `getPopups()` - Get all popups (admin)
        - `createPopup(data)` - Create popup (admin)
        - `updatePopup(id, data)` - Update popup (admin)
        - `deletePopup(id)` - Delete popup (admin)
        - `togglePopup(id)` - Toggle popup status (admin)
10. **adminApiService.js** - Admin operations
    - Methods:
        - `getDashboardStats()` - Get dashboard stats
        - `getAnalytics()` - Get analytics
        - `getRevenueAnalytics()` - Get revenue analytics
        - `getUserManagement()` - Get user management data
        - `getSystemSettings()` - Get system settings
11. **userApiService.js** - User operations
    - Methods:
        - `getUser(id)` - Get user
        - `updateUser(id, data)` - Update user
12. **searchService.js** - Search operations
13. **notificationService.js** - Notification operations (WebSocket support)
14. **dateChangeApiService.js** - Date change operations
15. **analyticsTracker.js** - Analytics tracking

5.5 Contexts (1 Context)

AuthContext.js - Authentication context

```
graph TD
    A[AuthContext Provider] --> B[State Management]

    B --> C[user: User object or null]
    B --> D[token: JWT string or null]
    B --> E[isAuthenticated: boolean]
    B --> F[isLoading: boolean]
    B --> G[error: string or null]

    H[Methods] --> I[login email password rememberMe]
    H --> J[register userData]
    H --> K[logout]
    H --> L[hasRole role]
    H --> M[hasPermission permission]

    N[Storage] --> O[localStorage - remember=true]
    N --> P[sessionStorage - remember=false]
```

Features:

- **State**: `user`, `token`, `isAuthenticated`, `isLoading`, `error`
- **Methods**: `login`, `register`, `logout`, `hasRole`, `hasPermission`
- **Storage**: localStorage (rememberMe=true) or sessionStorage (rememberMe=false)
- **Initialization**: Checks for token on mount, fetches user profile
- **Usage**: Wrapped in `app/providers.tsx`, used by ProtectedRoute and all pages

Example Usage:

```
import { useAuth } from '@/contexts/AuthContext';

function MyComponent() {
  const { user, isAuthenticated, login, logout, hasRole } = useAuth();

  if (!isAuthenticated) {
    return <LoginForm />;
  }

  return (
    <div>
      <p>Welcome, {user.firstName}!</p>
      {hasRole('admin') && <AdminPanel />}
      <button onClick={logout}>Logout</button>
    </div>
  );
}
```

5.6 Hooks (3 Custom Hooks)

1. **useApi.js** - API hook
2. **useDebounce.js** - Debounce hook
3. **usePackageForm.js** - Package form hook

5.7 Routing (Next.js App Router)

```
graph TD
    A[Next.js App Router] --> B[Navigation Hooks]

    B --> C[useRouter - from next/navigation]
    B --> D[usePathname - from next/navigation]
    B --> E[useSearchParams - from next/navigation]
    B --> F[useParams - from next/navigation]

    G[Link Component] --> H[Link - from next/link]

    I[Route Protection] --> J[ProtectedRoute]
    I --> K[AdminRoute]
    I --> L[AgentRoute]
```

Important: All components use Next.js routing. **NO react-router-dom imports remain**.

Navigation:

- `useRouter()` from `next/navigation` - For navigation (`router.push`, `router.replace`, `router.back`)
- `usePathname()` from `next/navigation` - Current pathname
- `useSearchParams()` from `next/navigation` - URL search params
- `useParams()` from `next/navigation` - Dynamic route params
- `Link` from `next/link` - Client-side navigation

Example:

```
'use client';

import { useRouter, useSearchParams } from 'next/navigation';
import Link from 'next/link';

function MyComponent() {
  const router = useRouter();
  const searchParams = useSearchParams();

  const handleClick = () => {
    router.push('/packages?category=adventure');
  };

  return (
    <div>
      <Link href="/packages">Browse Packages</Link>
      <button onClick={handleClick}>Adventure Packages</button>
    </div>
  );
}
```

---

6. Database Deep Dive

6.1 Database Overview

```
graph TD
    A[MongoDB Database] --> B[soulsathi_travel_12_11_2025]

    B --> C[45+ Collections]

    C --> D[Core: 10 collections]
    C --> E[Financial: 5 collections]
    C --> F[Content: 4 collections]
    C --> G[System: 4 collections]
    C --> H[Operational: 11 collections]
    C --> I[Legacy: 11 collections]
```

Database Name: `soulsathi_travel_12_11_2025` **Total Collections**: 45+ **ODM**: Mongoose **Connection**: Configured in `backend/src/config/database.js`

6.2 Core Collections (10 Collections)

These are the primary collections used by the application.

1. users

```
erDiagram
    users {
        ObjectId _id PK
        string firstName
        string lastName
        string email UK
        string password
        string phone
        string role
        boolean isActive
        boolean isEmailVerified
        object preferences
        object address
        string avatar
        number loyaltyPoints
        date createdAt
        date updatedAt
    }
```

Purpose: User accounts (customers, agents, admins)

Fields:

- `_id` : Primary key
- `firstName` , `lastName` : User name
- `email` : Unique email (indexed)
- `password` : Hashed password (bcrypt)
- `phone` : Phone number
- `role` : `user` , `agent` , `admin` , `super-admin` , `manager` , `employee`
- `isActive` : Account status
- `isEmailVerified` : Email verification status
- `preferences` : User preferences (object)
- `address` : User address (object)
- `avatar` : Profile picture URL
- `loyaltyPoints` : Loyalty program points
- `createdAt` , `updatedAt` : Timestamps

Indexes:

- `email` : Unique index
- `phone` : Index
- `role` : Index
- `isActive` : Index

Relations:

- Has many: `bookings` , `payments` , `commissions` , `walletTransactions`
- Has one: `wallet`

API Endpoints:

- `POST /api/auth/register` - Create user
- `POST /api/auth/login` - Authenticate user
- `GET /api/users` - Get all users (admin)
- `GET /api/users/:id` - Get user by ID
- `PUT /api/users/:id` - Update user

2. packages

```
erDiagram
    packages {
        ObjectId _id PK
        string title
        string slug UK
        string description
        object destination
        object duration
        object pricing
        object priceStructure
        number gstPercentage
        number tcsPercentage
        array images
        string category
        string type
        array itinerary
        array inclusions
        array exclusions
        array departures
        object accommodationDetails
        array citiesToExplore
        object mealPlan
        string whatsappNumber
        string termsAndConditions
        string cancellationAndRefundPolicy
        string status
        boolean isFeatured
        boolean isPopular
        object stats
        date createdAt
        date updatedAt
    }
```

Purpose: Tour packages (19+ fields)

Key Fields:

- `title`, `slug`: Package name and URL slug
- `destination`: {name, country, state, city, coordinates}
- `duration`: {days, nights}
- `pricing`: {basePrice, currency, discountPercentage}
- `priceStructure`: {single, twin, tripleExtraBed, childFare, infantFare}
- `gstPercentage`, `tcsPercentage`: Tax rates
- `images`: Array of image URLs
- `category`: Package category
- `type`: `domestic` or `international`
- `itinerary`: Array of day-wise itinerary
- `inclusions`, `exclusions`: Array of items
- **`departures`**: Array of {startDate, endDate, availableSeats, isActive} ☒
- `accommodationDetails`: Hotel information
- `citiesToExplore`: Array of cities
- `mealPlan`: Meal details
- `whatsappNumber`: WhatsApp contact
- `termsAndConditions`: T&C text
- `cancellationAndRefundPolicy`: Cancellation policy
- `status`: `draft`, `active`, `inactive`, `archived`
- `isFeatured`, `isPopular`: Boolean flags
- `stats`: {views, bookings, rating}

Indexes:

- `title`: Text index (for search)
- `slug`: Unique index
- `category`: Index
- `type`: Index
- `status`: Index
- `destination.country`: Index
- `pricing.basePrice`: Index
- `isFeatured`, `isPopular`: Indexes

Relations:

- Has many: `bookings`, `reviews`

API Endpoints:

- `GET /api/packages` - Get all packages
- `GET /api/packages/:id` - Get package by ID
- `POST /api/packages` - Create package (admin)
- `PUT /api/packages/:id` - Update package (admin)
- `DELETE /api/packages/:id` - Delete package (admin)
- `POST /api/packages/:id/departures` - Add departure (admin)
- `PUT /api/packages/:id/departures/:departureId` - Update departure (admin)
- `PATCH /api/packages/:id/departures/:departureId/toggle-status` - Toggle departure (admin)
- `DELETE /api/packages/:id/departures/:departureId` - Delete departure (admin)

3. bookings

```
erDiagram
    bookings {
        ObjectId _id PK
        string bookingId UK
        ObjectId user FK
        ObjectId package FK
        object travelDates
        array travelers
        object contactInfo
        object pricing
        object payment
        string status
        ObjectId assignedStaff FK
        object preferences
        array documents
        array notes
        object cancellation
        ObjectId agent FK
        object commission
        date createdAt
        date updatedAt
    }
```

Purpose: Booking records

Key Fields:

- `bookingId`: Unique booking ID (e.g., "BK-2025-001")
- `user`: Reference to `users` collection
- `package`: Reference to `packages` collection
- `travelDates`: {startDate, endDate}
- `travelers`: Array of {type, title, firstName, lastName, dateOfBirth, gender, nationality, passportNumber}
- `contactInfo`: {email, phone, emergencyContact}
- `pricing`: {baseAmount, taxes, fees, discount, totalAmount, currency}
- `payment`: {status, method, transactionId, paidAmount, pendingAmount, paymentHistory}
- `status`: `pending`, `confirmed`, `cancelled`, `completed`, `refunded`
- `assignedStaff`: Reference to `staff` collection
- `preferences`: Special requests
- `documents`: Uploaded documents
- `notes`: Internal notes
- `cancellation`: Cancellation details
- `agent`: Reference to `users` (if B2B booking)
- `commission`: Commission details

Indexes:

- `bookingId`: Unique index
- `user`: Index
- `package`: Index
- `status`: Index
- `payment.status`: Index
- `travelDates.startDate`: Index
- `agent`: Index

Relations:

- Belongs to: `user`, `package`, `assignedStaff`, `agent`
- Has many: `payments`, `refunds`, `moveDepartures`

API Endpoints:

- `POST /api/bookings` - Create booking
- `GET /api/bookings/my-bookings` - Get user bookings
- `GET /api/bookings/:id` - Get booking by ID
- `POST /api/bookings/:id/confirm` - Confirm booking
- `POST /api/bookings/:id/cancel` - Cancel booking
- `GET /api/bookings` - Get all bookings (admin)

4. payments

Purpose: Payment records

Key Fields:

- `paymentId` : Unique payment ID
- `bookingId` : Reference to booking
- `userId` : Reference to user
- `amount` , `currency` : Payment amount
- `paymentMethod` : Payment method
- `status` : `pending` , `processing` , `completed` , `failed` , `cancelled` , `refunded`
- `gateway` : `razorpay` , `payu` , `stripe`
- `gatewayPaymentId` , `gatewayOrderId` : Gateway IDs
- `transactionId` : Transaction ID
- `breakdown` : Amount breakdown
- `metadata` : Additional data

Indexes:

- `paymentId` : Unique
- `bookingId` : Index
- `userId` : Index
- `status` : Index
- `gateway` : Index

Relations:

- Belongs to: `booking` , `user`
- Has one: `refund`

5. wallets

Purpose: User/Agent wallets for B2B operations

Key Fields:

- `userId` : Reference to user (unique)
- `balance` : Current balance
- `totalEarnings` : Total earnings
- `totalWithdrawals` : Total withdrawals
- `pendingBalance` : Pending amount
- `commissionBalance` : Commission balance
- `referralEarnings` : Referral earnings
- `currency` : Currency code
- `status` : `active` , `suspended` , `frozen` , `closed`
- `bankDetails` : Bank account details
- `upiDetails` : UPI details
- `creditLimit` : Credit limit

Static Method:

- `getOrCreateWallet(userId)` - Gets or creates wallet □

Indexes:

- `userId` : Unique
- `status` : Index

Relations:

- Belongs to: `user`
- Has many: `walletTransactions`

API Endpoints:

- `GET /api/wallet/my` - Get user wallet
- `GET /api/wallet/balance` - Get balance
- `POST /api/wallet/topup` - Top up wallet
- `POST /api/wallet/withdraw` - Withdraw from wallet
- `GET /api/wallet/transactions` - Get transactions

6. wallettransactions

Purpose: Wallet transaction history

Key Fields:

- `walletId` : Reference to wallet
- `userId` : Reference to user
- `type` : `credit` or `debit`
- `amount` : Transaction amount
- `status` : `pending` , `completed` , `failed`
- `description` : Transaction description
- `category` : Transaction category
- `method` : Payment method
- `relatedBooking` : Reference to booking (if applicable)
- `relatedPayment` : Reference to payment (if applicable)
- `metadata` : Additional data

Indexes:

- `walletId` : Index
- `userId` : Index
- `type` : Index
- `status` : Index
- `category` : Index

Relations:

- Belongs to: `wallet` , `user`

7. commissions

Purpose: Agent commission tracking

Key Fields:

- `commissionId` : Unique commission ID
- `userId` : Reference to agent
- `bookingId` : Reference to booking
- `paymentId` : Reference to payment
- `type` : `booking_commission` , `referral_commission` , `override_commission` , `bonus_commission` , `penalty_deduction` , `adjustment`
- `baseAmount` : Base amount
- `commissionRate` : Commission rate (percentage)
- `commissionAmount` : Calculated commission
- `tier` : Commission tier
- `status` : `pending` , `calculated` , `approved` , `released` , `paid` , `cancelled` , `disputed` , `adjusted`
- `lockPeriod` : Lock period (days)
- `earnedAt` , `releasedAt` , `paidAt` : Timestamps

Indexes:

- `commissionId` : Unique
- `userId` : Index
- `bookingId` : Index
- `status` : Index
- `type` : Index

Relations:

- Belongs to: `user` , `booking` , `payment`

API Endpoints:

- `GET /api/commission` - Get commissions
- `GET /api/commission/summary` - Get commission summary
- `POST /api/commission/payout/request` - Request payout
- `GET /api/commission/payouts` - Get payouts

8. payouts

Purpose: Commission payout records

Key Fields:

- `payoutId` : Unique payout ID
- `partnerId` : Reference to agent
- `amount` : Payout amount
- `status` : `requested` , `processing` , `completed` , `failed` , `rejected`
- `requestedAt` , `processedAt` : Timestamps
- `method` : `wallet` , `bank_transfer` , `cheque` , `other`
- `reference` : Reference number
- `metadata` : Additional data

Indexes:

- `payoutId` : Unique
- `partnerId` : Index

Relations:

- Belongs to: `user` (partnerId)

9. taxconfigs

Purpose: Tax configuration (GST, TCS, etc.)

Key Fields:

- `name` : Tax name
- `type` : `gst` , `service_charge` , `convenience_fee` , `processing_fee` , `other`
- `calculation` : `percentage` , `fixed` , `slab`
- `value` : Tax value
- `isActive` : Active status
- `applicableFor` : `all` , `domestic` , `international` , `specific_packages`
- `minAmount` , `maxAmount` : Amount range
- `slabRates` : Slab rates (array)
- `description` , `displayName` : Display info
- `priority` : Priority order
- `isIncludedInBase` : Whether included in base price
- `compoundTax` : Compound tax flag

Indexes:

- `type` : Index
- `isActive` : Index
- `applicableFor` : Index
- `priority` : Index

API Endpoints:

- `POST /api/taxes/calculate` - Calculate tax
- `GET /api/taxes` - Get tax configs (admin)
- `POST /api/taxes` - Create tax config (admin)

10. refunds

Purpose: Refund management

Key Fields:

- `refundId` : Unique refund ID
- `refundNumber` : Unique refund number
- `booking` : Reference to booking
- `user` : Reference to user
- `payment` : Reference to payment
- `cancellation` : Cancellation details
- `amount` : {total, principal, taxes, fees, penalties, currency}
- `originalPayment` : Original payment details
- `refundMethod` : Refund method
- `refundDetails` : Refund details (gateway info)
- `status` : `requested` , `processing` , `completed` , `failed` , `rejected`
- `requestedAt` , `processedAt` : Timestamps

Indexes:

- `refundId` , `refundNumber` : Unique
- `booking` , `user` , `payment` : Indexes
- `status` : Index

Relations:

- Belongs to: `booking` , `user` , `payment`

API Endpoints:

- `POST /api/payments/:id/refund` - Request refund
- `POST /api/payments/process-refund` - Process refund (admin)

6.3 Financial Collections (5 Collections)

Already covered above:

- `wallets`
- `wallettransactions`
- `commissions`
- `payouts`

- `taxconfigs`

6.4 Content Collections (4 Collections)

1. cmscontents (CMSComplete model)

Purpose: Dynamic website content

Key Fields:

- `contentId` : Unique content ID
- `title` , `slug` : Title and URL slug (unique)
- `contentType` : `page` , `blog_post` , `news_article` , `travel_guide` , `destination_guide` , `testimonial` , `faq` , `policy` , `terms` , `privacy` , `about` , `contact` , `landing_page` , `email_template` , `notification_template` , `banner` , `popup` , `widget`
- `category` , `tags` : Categorization
- `content` : {sections with type, title, content, media, gallery, video, table}
- `excerpt` : Short description
- `status` : `draft` , `published` , `archived`
- `publishedAt` : Publish date
- `author` : Author reference
- `seo` : SEO metadata

Indexes:

- `contentId` , `slug` : Unique
- `contentType` , `category` , `status` , `publishedAt` : Indexes

API Endpoints:

- `GET /api/cms` - Get all content
- `GET /api/cms/content/:contentType` - Get content by type ▯
- `POST /api/cms/content/:contentType` - Create content by type (admin) ▯
- `PUT /api/cms/content/:contentType/:id` - Update content (admin) ▯
- `DELETE /api/cms/content/:contentType/:id` - Delete content (admin) ▯

2. popups

Purpose: Popup configurations

Key Fields:

- `title` : Popup title
- `type` : `newsletter` , `discount` , `announcement` , `feedback` , `promotion` , `custom`
- `content` : {title, subtitle, description, message, image, discount, ctaText}
- `targeting` : {userType, deviceType, pages, countries, languages, newUsers, returningUsers}
- `behavior` : {showFrequency, showDelay, showOnScroll, showOnExitIntent, autoCloseAfter, dismissible}
- `styling` : {position, animation, backgroundColor, textColor}
- `status` : `active` , `inactive`
- `startDate` , `endDate` : Date range
- `priority` : Display priority
- `displayCount` : How many times shown

Static Method:

- `getActivePopups(filters)` - Gets active popups based on targeting

Indexes:

- `status` , `startDate` , `endDate` , `priority` : Indexes

API Endpoints:

- `GET /api/popups/active` - Get active popups (public) ▯

3. settings (SettingsComplete model)

Purpose: Website settings

Key Fields:

- `key` : Unique setting key
- `value` : Setting value
- `displayName` : Display name
- `category` : Setting category
- `description` : Description
- `type` : `string` , `number` , `boolean` , `object` , `array`
- `access` : {level: `public` , `user` , `admin` , roles: array}
- `validation` : Validation rules
- `defaultValue` : Default value
- `status` : `active` , `inactive`

Indexes:

- `key` : Unique
- `category` , `access.level` , `status` : Indexes

API Endpoints:

- `GET /api/settings/public` - Get public settings
- `GET /api/settings` - Get all settings (admin)
- `PUT /api/settings/:key` - Update setting (admin)

4. reviews (ReviewComplete model)

Purpose: Reviews/Testimonials

Key Fields:

- `reviewId` : Unique review ID
- `reviewTarget` : {targetType, targetId}
- `reviewer` : {userId, name, email, avatar, isVerified}
- `booking` : Reference to booking
- `ratings` : {overall, valueForMoney, service, cleanliness, comfort, location, food, guide, transport}
- `content` : {title, text, pros, cons, tips}
- `status` : `pending` , `approved` , `rejected`
- `isPublished` , `isFeatured` : Flags
- `helpfulCount` : Helpful count

Indexes:

- `reviewId` : Unique
- `reviewTarget.targetType` , `reviewTarget.targetId` : Compound index
- `reviewer.userId` , `status` , `isPublished` , `isFeatured` : Indexes

6.5 System Collections (4 Collections)

1. admins

Purpose: Admin accounts

Key Fields:

- `email` : Unique email
- `password` : Hashed password
- `firstName` , `lastName` : Name
- `role` : `admin` , `super-admin` , `manager`
- `permissions` : Array of permissions
- `isActive` : Active status
- `failedLoginAttempts` : Failed login count
- `lockUntil` : Account lock timestamp
- `lastLoginAt` , `lastLoginIP` : Last login info

Indexes:

- `email` : Unique
- `role` , `isActive` : Indexes

2. staff

Purpose: Staff members

Key Fields:

- `employeeId` : Unique employee ID
- `firstName` , `lastName` , `email` : Personal info
- `password` : Hashed password
- `phone` : Phone number
- `role` : Staff role
- `department` : Department
- `specializations` : Array of specializations
- `currentAvailability` : Availability status
- `assignedBookings` : Array of assigned bookings
- `isActive` : Active status

Indexes:

- `employeeId` , `email` : Unique
- `role` , `department` , `isActive` : Indexes

3. analytics (Analytics model)

Purpose: Analytics events

Key Fields:

- `eventType` : Event type
- `eventData` : Event data (object)
- `businessMetrics` : Business metrics (object)
- `timestamp` : Event timestamp

Indexes:

- `eventType` , `timestamp` : Indexes

4. activitylogs

Purpose: Activity tracking

Key Fields:

- `actionType` : Action type
- `actionCategory` : Action category
- `actionDescription` : Action description
- `user` : User reference
- `userType` : User type
- `status` : Status
- `target` : {type, id, name}
- `metadata` : Additional data
- `technical` : {ipAddress, userAgent}

Indexes:

- `user` , `actionType` , `actionCategory` , `createdAt` : Indexes

6.6 Operational Collections (11 Collections)

1. movedepartures

Purpose: Date change requests

Key Fields:

- `requestId` : Unique request ID
- `booking` : Reference to booking
- `customer` : Reference to user
- `originalDates` : {departureDate, returnDate, duration}
- `requestedDates` : {departureDate, returnDate, duration}
- `availableDates` : Array of available dates
- `requestDetails` : {reason, description, urgency, flexibility}
- `financialImpact` : {priceDifference, fees, totalCost}
- `status` : `pending` , `under_review` , `checking_availability` , `approved` , `rejected` , `cancelled`
- `assignedStaff` : Reference to staff
- `timeline` : Array of status changes

Indexes:

- `requestId` : Unique
- `booking` , `customer` , `status` : Indexes

API Endpoints:

- `POST /api/move-departure` - Create request
- `GET /api/move-departure/my-requests` - Get user requests
- `PUT /api/move-departure/:requestId/approve` - Approve request (admin)

2. cities

Purpose: City data

Key Fields:

- `name` , `state` , `country` : Location
- `description` : City description
- `images` : Array of images
- `coordinates` : {latitude, longitude}
- `attractions` : Array of attractions
- `isPopular` , `isActive` : Flags

Indexes:

- `name+country` : Compound unique
- `isPopular` , `isActive` : Indexes

API Endpoints:

- `GET /api/cities` - Get all cities
- `GET /api/cities/:id` - Get city by ID
- `POST /api/cities` - Create city (admin)

3. hotels

Purpose: Hotel data

Key Fields:

- `name` : Hotel name
- `city` : Reference to city (ObjectId)
- `cityName` : City name (string)
- `address` : Hotel address
- `starRating` : Star rating
- `category` : `budget` , `standard` , `deluxe` , `luxury` , `super-luxury`
- `description` : Hotel description
- `images` : Array of images
- `amenities` : Array of amenities
- `roomTypes` : Array of room types
- `contact` : Contact details
- `coordinates` : {latitude, longitude}
- `isActive` : Active status

Indexes:

- `city` , `name` , `starRating` , `isActive` : Indexes

API Endpoints:

- `GET /api/hotels` - Get all hotels
- `GET /api/hotels/city/:cityId` - Get hotels by city
- `GET /api/hotels/:id` - Get hotel by ID

4-11. Other Operational Collections

- **NotificationComplete**: Notifications
- **InquiryComplete**: Contact inquiries
- **DateChangeRequest**: Date change requests (legacy, use MoveDeparture)
- **Document**: Document storage
- **Lead**: Lead management
- **Newsletter**: Newsletter subscriptions
- **LoyaltyRewards**: Loyalty program
- **DynamicPricing**: Dynamic pricing rules
- **ChatSession**, **ChatMessage**, **ChatLog**: Chat system

6.7 Legacy/Complete Models (11 Collections)

These models have "Complete" suffix or are alternative versions:

- UserComplete.js
- PackageComplete.js
- BookingComplete.js
- PaymentComplete.js
- AdminComplete.js
- AnalyticsComplete.js
- CategoryComplete.js
- BlogComplete.js
- PaymentGatewayComplete.js

Status: ⚠ Exist but controllers prefer core models (User.js, Package.js, Booking.js, Payment.js)

Recommendation: Use core models for new development unless specific "Complete" features are needed.

---

7. API Endpoints Reference

7.1 API Endpoint Categories

```
graph TD
    A[200+ API Endpoints] --> B[Authentication]
    A --> C[User Management]
    A --> D[Package Management]
    A --> E[Booking System]
    A --> F[Payment Processing]
    A --> G[Wallet & B2B]
    A --> H[CMS & Content]
    A --> I[Admin & Analytics]
    A --> J[Operational]

    B --> B1[10 endpoints]
    C --> C1[9 endpoints]
    D --> D1[14 endpoints]
    E --> E1[11 endpoints]
    F --> F1[10 endpoints]
    G --> G1[15 endpoints]
    H --> H1[20 endpoints]
    I --> I1[25 endpoints]
    J --> J1[30+ endpoints]
```

7.2 Authentication Endpoints (`/api/auth`)

Method	Endpoint	Auth	Description
POST	`/api/auth/register`	Public	Register new user
POST	`/api/auth/login`	Public	User login (rate limited)
POST	`/api/auth/logout`	Protected	User logout
GET	`/api/auth/me`	Protected	Get current user
PUT	`/api/auth/update-profile`	Protected	Update profile
PUT	`/api/auth/update-password`	Protected	Update password
POST	`/api/auth/forgot-password`	Public	Request password reset
PUT	`/api/auth/reset-password/:resettoken`	Public	Reset password
GET	`/api/auth/verify-email/:token`	Public	Verify email
POST	`/api/auth/resend-verification`	Protected	Resend verification email

Rate Limiting: Login endpoint is rate limited to 5 attempts per 15 minutes.

7.3 Package Endpoints (`/api/packages`)

Method	Endpoint	Auth	Description
GET	`/api/packages`	Optional	Get all packages (with filters)
GET	`/api/packages/featured`	Public	Get featured packages
GET	`/api/packages/popular`	Public	Get popular packages
GET	`/api/packages/search`	Optional	Search packages
GET	`/api/packages/category/:category`	Public	Get packages by category
GET	`/api/packages/:id`	Public	Get single package

Method	Endpoint	Auth	Description
POST	`/api/packages`	Admin	Create package
PUT	`/api/packages/:id`	Admin	Update package
DELETE	`/api/packages/:id`	Admin	Delete package
PATCH	`/api/packages/:id/toggle-status`	Admin	Toggle package status
POST	`/api/packages/:id/departures`	Admin	Add departure
PUT	`/api/packages/:id/departures/:departureId`	Admin	Update departure
PATCH	`/api/packages/:id/departures/:departureId/toggle-status`	Admin	Toggle departure status
DELETE	`/api/packages/:id/departures/:departureId`	Admin	Delete departure

Filters (GET /api/packages):

- `category`, `type`, `destination`, `country`
- `minPrice`, `maxPrice`, `minDuration`, `maxDuration`
- `rating`, `search`, `featured`, `popular`, `status`
- Pagination: `page`, `limit`
- Sorting: `sort` (e.g., `price`, `-createdAt`)

7.4 Booking Endpoints (`/api/bookings`)

Method	Endpoint	Auth	Description
POST	`/api/bookings/check-availability`	Public	Check availability
GET	`/api/bookings/recommendations`	Protected	Get AI recommendations
POST	`/api/bookings`	Protected	Create booking
GET	`/api/bookings/my-bookings`	Protected	Get user bookings
GET	`/api/bookings/:id`	Protected	Get booking by ID
POST	`/api/bookings/:id/confirm`	Protected	Confirm booking
POST	`/api/bookings/:id/modify`	Protected	Request modification
POST	`/api/bookings/:id/cancel`	Protected	Cancel booking
GET	`/api/bookings`	Admin	Get all bookings
PUT	`/api/bookings/:id/status`	Admin	Update booking status
GET	`/api/bookings/analytics`	Admin	Booking analytics

Note: These routes use `smartBooking.js` controller (NOT `booking.js`).

7.5 Payment Endpoints (`/api/payments`)

Method	Endpoint	Auth	Description
POST	`/api/payments/initiate`	Protected	Initiate payment
POST	`/api/payments/verify`	Protected	Verify payment
GET	`/api/payments`	Protected	Get user payments
GET	`/api/payments/:id`	Protected	Get payment by ID
POST	`/api/payments/:id/refund`	Protected	Request refund

Method	Endpoint	Auth	Description
POST	`/api/payments/process-refund`	Admin	Process refund
GET	`/api/payments/analytics`	Admin	Payment analytics
POST	`/api/payments/webhook/razorpay`	Public	Razorpay webhook
POST	`/api/payments/webhook/stripe`	Public	Stripe webhook
POST	`/api/payments/webhook/payu`	Public	PayU webhook

Payment Gateways:

- Razorpay (primary)
- PayU (secondary)
- Stripe (international)
- Automatic failover

7.6 Wallet Endpoints (`/api/wallet`)

Method	Endpoint	Auth	Description
GET	`/api/wallet/my`	Protected	Get user wallet
GET	`/api/wallet/balance`	Protected	Get balance (alias for /my)
POST	`/api/wallet/get-or-create`	Protected	Get or create wallet
GET	`/api/wallet/:id`	Admin	Get wallet by ID
GET	`/api/wallet`	Admin	Get all wallets
GET	`/api/wallet/transactions/:id?`	Protected	Get transactions
POST	`/api/wallet/topup`	Protected	Initiate top-up
POST	`/api/wallet/topup/success`	Protected	Process top-up success
POST	`/api/wallet/withdraw`	Protected	Initiate withdrawal
POST	`/api/wallet/transfer`	Protected	Transfer between wallets
GET	`/api/wallet/stats`	Admin	Wallet statistics

Wallet Creation: Automatic via `Wallet.getOrCreateWallet(userId)` method

7.7 Commission Endpoints (`/api/commission`)

Method	Endpoint	Auth	Description
GET	`/api/commission`	Admin	Get commissions
GET	`/api/commission/summary`	Protected	Get commission summary
POST	`/api/commission/release/:id`	Admin	Release commission
POST	`/api/commission/bulk-release`	Admin	Bulk release
POST	`/api/commission/payout/request`	Protected	Request payout
GET	`/api/commission/payouts`	Protected	Get payouts (uses Payout model)
GET	`/api/commission/analytics`	Admin	Commission analytics

7.8 CMS Endpoints (`/api/cms`)

Method	Endpoint	Auth	Description
GET	`/api/cms`	Public/Admin	Get all content
GET	`/api/cms/:slug`	Public	Get content by slug
POST	`/api/cms`	Admin	Create content
PUT	`/api/cms/:id`	Admin	Update content
DELETE	`/api/cms/:id`	Admin	Delete content
POST	`/api/cms/bulk`	Admin	Bulk operations
GET	`/api/cms/analytics/overview`	Admin	Content analytics

Frontend Compatibility Routes ✓:

Method	Endpoint	Auth	Description
GET	`/api/cms/content/:contentType`	Public/Admin	Get content by type
POST	`/api/cms/content/:contentType`	Admin	Create content by type
PUT	`/api/cms/content/:contentType/:id`	Admin	Update content by type
DELETE	`/api/cms/content/:contentType/:id`	Admin	Delete content by type
POST	`/api/cms/content/:contentType/bulk`	Admin	Bulk operations by type
GET	`/api/cms/analytics/:contentType`	Admin	Analytics by type

Content Type Specific Routes:

- `GET /api/cms/blog/posts` - Get blog posts
- `GET /api/cms/pages/all` - Get all pages
- `GET /api/cms/guides/travel` - Get travel guides
- `GET /api/cms/testimonials/all` - Get testimonials
- `GET /api/cms/faqs/all` - Get FAQs
- `GET /api/cms/gallery/photos` - Get photo gallery
- `GET /api/cms/banners/active` - Get active banners
- `GET /api/cms/popups/active` - Get active popups

7.9 Popup Endpoints (`/api/popups`)

Method	Endpoint	Auth	Description
GET	`/api/popups/active`	Public	Get active popups ✓
GET	`/api/popups`	Admin	Get all popups
POST	`/api/popups`	Admin	Create popup
PUT	`/api/popups/:id`	Admin	Update popup
DELETE	`/api/popups/:id`	Admin	Delete popup
PATCH	`/api/popups/:id/toggle`	Admin	Toggle popup status

Query Params for `/api/popups/active`:

- `userType`: `guest`, `user`, `agent`
- `deviceType`: `desktop`, `mobile`, `tablet`
- `page`: Current page URL

- `country` : User country
- `language` : User language
- `isNewUser` : `true` / `false`
- `sessionId` : Session ID (for frequency control)

Fixed: Frontend now calls `/api/popups/active` instead of `/api/popups` □

7.10 Settings Endpoints (`/api/settings`)

Method	Endpoint	Auth	Description
GET	`/api/settings/public`	Public	Get public settings
GET	`/api/settings`	Admin	Get all settings
GET	`/api/settings/:key`	Admin	Get setting by key
POST	`/api/settings`	Admin	Create setting
PUT	`/api/settings/:key`	Admin	Update setting
DELETE	`/api/settings/:key`	Admin	Delete setting
PUT	`/api/settings/bulk`	Admin	Bulk update settings
POST	`/api/settings/upload`	Admin	Upload setting file
POST	`/api/settings/initialize`	Admin	Initialize default settings

7.11 Other Major Endpoints

Admin (`/api/admin`):

- `POST /api/admin/login` - Admin login
- `GET /api/admin/dashboard` - Dashboard stats
- `GET /api/admin/analytics` - System analytics
- `GET /api/admin/system-health` - System health

Dashboard (`/api/dashboard`):

- `GET /api/dashboard/stats` - Dashboard stats
- `GET /api/dashboard/recent-bookings` - Recent bookings
- `GET /api/dashboard/revenue-chart` - Revenue chart
- `GET /api/dashboard/activity-feed` - Activity feed

Analytics (`/api/analytics`):

- `POST /api/analytics/track` - Track event
- `GET /api/analytics/dashboard` - Dashboard metrics
- `GET /api/analytics/revenue` - Revenue analytics
- `GET /api/analytics/bookings` - Booking analytics

Tax (`/api/taxes`):

- `POST /api/taxes/calculate` - Calculate tax
- `GET /api/taxes/applicable` - Get applicable taxes
- `GET /api/taxes` - Get tax configs (admin)

Move Departure (`/api/move-departure`):

- `POST /api/move-departure` - Create request
- `GET /api/move-departure/my-requests` - Get user requests
- `PUT /api/move-departure/:requestId/approve` - Approve (admin)
- `PUT /api/move-departure/:requestId/reject` - Reject (admin)

Cities & Hotels:

- `GET /api/cities` - Get all cities
- `GET /api/hotels` - Get all hotels
- `GET /api/hotels/city/:cityId` - Get hotels by city

Total Endpoints: 200+ endpoints across 31 route modules

8. Frontend-Backend Connection Flow

8.1 Complete Request-Response Flow

```
sequenceDiagram
    participant U as User
    participant C as Component
    participant S as Service
    participant A as apiService
    participant B as Backend Route
    participant Ctrl as Controller
    participant M as Model
    participant DB as MongoDB

    U->>C: User Action (e.g., Login)
    C->>S: authService.login(email, password)
    S->>A: apiService.post('/auth/login', data)

    Note over A: Add Authorization header<br/>if token exists

    A->>B: POST /api/auth/login

    Note over B: Middleware Chain:<br/>1. express.json()<br/>2. Rate Limiter

    B->>Ctrl: auth.login(req, res)
    Ctrl->>M: User.findOne({email})
    M->>DB: Query users collection
    DB-->>M: User document

    Note over Ctrl: 1. Verify password (bcrypt)<br/>2. Generate JWT token<br/>3. Log activity

    M-->>Ctrl: User object
    Ctrl-->>B: {success, token, user}
    B-->>A: HTTP 200 + JSON
    A-->>S: {token, user}

    Note over S: Store token in<br/>localStorage/sessionStorage

    S-->>C: {token, user}
    C->>C: Update AuthContext
    C-->>U: Redirect to Dashboard
```

8.2 Connection Matrix

```
graph LR
    subgraph Frontend
        P[Pages]
        C[Components]
        S[Services]
    end

    subgraph Backend
        R[Routes]
        Ctrl[Controllers]
        M[Models]
    end

    subgraph Database
        DB[(MongoDB)]
    end

    P --> C
    C --> S
    S --> R
    R --> Ctrl
    Ctrl --> M
    M --> DB
```

Status: ▯ 95%+ Fully Connected

All major flows are connected and functional:

- Authentication ▯
- Package Management ▯
- Booking System ▯
- Payment Processing ▯
- Wallet Operations ▯
- B2B Operations ▯
- CMS Content ▯
- Popup System ▯
- Settings Management ▯

---

9. Known Issues & Recommendations

9.1 Fixed Issues ▯

1. **Wallet.getOrCreateWallet() method** - ▯ Implemented
2. **CMS API routes** - ▯ Added frontend compatibility routes
3. **Popup endpoint** - ▯ Fixed to `/api/popups/active`
4. **React Router** - ▯ All components converted to Next.js routing
5. **Material-UI** - ▯ Removed from MyBookings and SmartBooking
6. **ProtectedRoute** - ▯ Enhanced with RBAC
7. **AdminLayout** - ▯ Uses Next.js Link
8. **Homepage** - ▯ Made dynamic (FeaturedPackages, PopularDestinations)
9. **SmartBooking model** - ▯ Created
10. **Payout model** - ▯ Created and integrated
11. **City/Hotel models** - ▯ Created with CRUD operations
12. **Package Management** - ▯ Integrated with packageApiService
13. **Settings Management** - ▯ Integrated with settingsService
14. **Wallet Dashboard** - ▯ Integrated with walletApiService
15. **B2B Pricing Calculator** - ▯ Integrated with walletApiService

9.2 Unregistered/Unused Files ⚠

1. **routes/bookings.js** (500 lines)

   - Status: ▯ NOT registered in server.js
   - Uses both `booking.js` and `smartBooking.js` controllers
   - Recommendation: Remove (functionality available via `routes/booking.js` which IS registered)

2. **routes/index.js**

   - Status: ▯ NOT used (server.js registers routes individually)
   - Recommendation: Remove or use for route organization

3. **routes/analytics_full.js** & **routes/analytics_minimal.js**

   - Status: □ NOT used (main `analytics.js` is used)
   - Recommendation: Remove

4. **controllers/booking.js**
   - Status: ⚠ May be unused (routes/booking.js uses `smartBooking.js`)
   - Recommendation: Verify usage and remove if unused

5. **Demo/Test Components**
   - `components/demo/ApiIntegrationDemo.js`
   - `components/auth/AuthDemo.js`
   - `components/payment/PaymentTesting.js`
   - Status: ⚠ Unused in production
   - Recommendation: Move to `dev/` folder or remove

6. **components/user/HomePage.js**
   - Status: ⚠ May be unused (prefer `app/page.tsx`)
   - Recommendation: Verify and remove if unused

7. **Legacy "Complete" Models**
   - UserComplete.js, PackageComplete.js, BookingComplete.js, PaymentComplete.js, etc.
   - Status: ⚠ Exist but controllers prefer core models
   - Recommendation: Use core models for new work

9.3 Static/Placeholder Pages □

1. **app/about/page.tsx** - Static content (no CMS integration)
   - Recommendation: Integrate with CMS

2. **app/contact/page.tsx** - Static form (no API submission)
   - Recommendation: Connect to `/api/contact` endpoint

3. **app/cart/page.tsx** - Placeholder (empty cart message)
   - Recommendation: Implement cart system

4. **app/wishlist/page.tsx** - Placeholder (empty wishlist message)
   - Recommendation: Implement wishlist system

9.4 Missing Features / TODO

1. **Token Refresh**
   - Status: Placeholder in `authService.js`, not implemented
   - Recommendation: Implement JWT refresh token mechanism

2. **Cart System**
   - Status: Placeholder page exists
   - Recommendation: Implement full cart functionality

3. **Wishlist**
   - Status: Placeholder page exists
   - Recommendation: Implement wishlist functionality

4. **Model Consolidation**
   - Issue: Many models have both basic and "Complete" versions
   - Recommendation: Standardize on one version or clearly document usage

5. **Real-time Notifications**
   - Status: WebSocket infrastructure exists but may need full implementation
   - Recommendation: Verify and complete real-time notification system

9.5 Security Recommendations

1. **Environment Variables**
   - Ensure all sensitive keys are in `.env` (never committed)
   - Required vars: JWT_SECRET, MONGODB_URI, payment gateway keys, Cloudinary keys, email credentials

2. **Rate Limiting**
   - Currently on login endpoint
   - Recommendation: Add to other sensitive endpoints (register, password reset, payment initiation)

3. **Input Validation**
   - Currently using express-validator and Mongoose validators
   - Recommendation: Review all endpoints for proper validation

4. **CORS Configuration**
    - Currently configured
    - Recommendation: Ensure proper origin restrictions in production
5. **Helmet Security Headers**
    - Currently enabled
    - Recommendation: Review CSP (Content Security Policy) settings

---

10. Deployment & Environment Setup

10.1 Environment Variables

Backend (.env):

```
# Server
PORT=5000
NODE_ENV=production

# Database
MONGODB_URI=mongodb://localhost:27017/soulsathi_travel_12_11_2025

# JWT
JWT_SECRET=your_jwt_secret_key_here
JWT_EXPIRE=30d

# Payment Gateways
RAZORPAY_KEY_ID=your_razorpay_key_id
RAZORPAY_KEY_SECRET=your_razorpay_secret
PAYU_MERCHANT_ID=your_payu_merchant_id
PAYU_MERCHANT_KEY=your_payu_key
PAYU_MERCHANT_SALT=your_payu_salt
STRIPE_SECRET_KEY=your_stripe_secret_key

# Cloudinary
CLOUDINARY_CLOUD_NAME=your_cloud_name
CLOUDINARY_API_KEY=your_api_key
CLOUDINARY_API_SECRET=your_api_secret

# Email
EMAIL_HOST=smtp.gmail.com
EMAIL_PORT=587
EMAIL_USER=your_email@gmail.com
EMAIL_PASS=your_email_password
EMAIL_FROM=noreply@soulsathi.com

# SMS/WhatsApp (optional)
SMS_API_KEY=your_sms_api_key
WHATSAPP_API_KEY=your_whatsapp_api_key

# Frontend URL (for CORS)
FRONTEND_URL=http://localhost:3000
```

Frontend (.env.local):

```
# API URL
NEXT_PUBLIC_API_URL=http://localhost:5000/api

# WebSocket URL (optional, derived from API URL)
NEXT_PUBLIC_WS_URL=ws://localhost:5000
```

10.2 Installation & Running

Backend:

```
cd backend
npm install
npm start           # Development (nodemon)
npm run build       # Production build (if configured)
npm run prod        # Production (node)
```

Frontend:

```
cd frontend
npm install
npm run dev         # Development
npm run build       # Production build
npm start           # Production server
```

Docker (if available):

```
docker-compose up -d
```

10.3 Database Setup

1. **Install MongoDB** (if not using cloud)
2. **Create Database**: `soulsathi_travel_12_11_2025`
3. **Indexes**: Automatically created by Mongoose on first run
4. **Seed Data** (optional): Use admin scripts to create initial data

10.4 Production Checklist

- [ ] Set `NODE_ENV=production`
- [ ] Use production MongoDB (MongoDB Atlas recommended)
- [ ] Configure CORS with production frontend URL
- [ ] Set strong JWT_SECRET
- [ ] Configure payment gateway webhooks
- [ ] Set up Cloudinary for production
- [ ] Configure email service (SendGrid, AWS SES, etc.)
- [ ] Enable HTTPS (SSL/TLS)
- [ ] Set up monitoring (PM2, New Relic, etc.)
- [ ] Configure backup strategy for MongoDB
- [ ] Set up logging (Winston, Loggly, etc.)
- [ ] Review and harden security settings

---

11. Maintenance Guide

11.1 Common Tasks

Adding a New Feature:

1. **Backend**:
   - Create/update model in `backend/src/models/`
   - Create/update controller in `backend/src/controllers/`
   - Create/update route in `backend/src/routes/`
   - Register route in `backend/src/server.js`
   - Test with Postman/Insomnia
2. **Frontend**:
   - Create/update component in `frontend/src/components/`
   - Create/update service in `frontend/src/services/`
   - Create/update page in `frontend/app/`
   - Test in browser

Updating a Package:

1. Use `PackageManagement.js` component (admin)
2. Or call `PUT /api/packages/:id` directly

Managing Departures:

1. Use departure management in `PackageManagement.js`
2. Or call:

    - `POST /api/packages/:id/departures` - Add
    - `PUT /api/packages/:id/departures/:departureId` - Update
    - `PATCH /api/packages/:id/departures/:departureId/toggle-status` - Toggle
    - `DELETE /api/packages/:id/departures/:departureId` - Delete

Processing Refunds:

1. User requests refund via `POST /api/payments/:id/refund`
2. Admin processes via `POST /api/payments/process-refund`
3. System updates `refunds` collection and `payments` collection

Releasing Commissions:

1. Admin views commissions via `GET /api/commission`
2. Admin releases commission via `POST /api/commission/release/:id`
3. Or bulk release via `POST /api/commission/bulk-release`

Handling Date Change Requests:

1. User submits via `POST /api/move-departure`
2. Admin reviews via `GET /api/move-departure/admin/all`
3. Admin approves/rejects via `PUT /api/move-departure/:requestId/approve` or `/reject`

11.2 Debugging Tips

Backend Debugging:

- Check console logs for errors
- Use `console.log()` or debugger in controllers
- Check MongoDB logs
- Use Postman to test API endpoints directly
- Check `activitylogs` collection for user actions

Frontend Debugging:

- Open browser DevTools (F12)
- Check Console for errors
- Check Network tab for failed API calls
- Use React DevTools extension
- Check AuthContext state in Components tab

Common Issues:

1. **"Token invalid" or "Unauthorized"**
    - Check if token is stored in localStorage/sessionStorage
    - Check if token is expired
    - Check if `Authorization` header is being sent
2. **"Cannot connect to backend"**
    - Check if backend server is running (port 5000)
    - Check `NEXT_PUBLIC_API_URL` in frontend `.env.local`
    - Check CORS settings in backend
3. **"Payment failed"**
    - Check payment gateway credentials
    - Check payment gateway webhook URLs
    - Check payment logs in `payments` collection
4. **"Wallet not found"**
    - Ensure `Wallet.getOrCreateWallet()` is being called
    - Check `wallets` collection in MongoDB
5. **"Popup not showing"**
    - Check if popup is active in `popups` collection
    - Check targeting criteria (userType, deviceType, etc.)
    - Check if popup was already shown (frequency control)

11.3 Database Maintenance

Backup:

```
mongodump --db soulsathi_travel_12_11_2025 --out /backup/path
```

Restore:

```
mongorestore --db soulsathi_travel_12_11_2025 /backup/path/soulsathi_travel_12_11_2025
```

Indexes:

- Automatically created by Mongoose
- To manually create: Use MongoDB Compass or Mongo Shell

Cleanup:

- Periodically clean old `activitylogs` (e.g., older than 90 days)
- Archive old `bookings` (e.g., completed > 1 year)
- Clean expired `popups`

---

12. Conclusion

12.1 System Summary

SoulSathi Travel Platform is a production-ready, full-stack travel booking system with:

- **231 files** (127 backend + 104 frontend)
- **200+ API endpoints**
- **45+ database collections**
- **95%+ functionality complete**

Key Strengths:

- ▯ Comprehensive authentication & authorization (RBAC)
- ▯ Full booking system with smart features
- ▯ Multiple payment gateway integration with failover
- ▯ B2B wallet & commission management
- ▯ Dynamic CMS with multiple content types
- ▯ Context-aware popup system
- ▯ Tax management (GST, TCS)
- ▯ Move departure (date change) system
- ▯ Refund management
- ▯ Analytics & reporting
- ▯ Admin panel with full CRUD operations
- ▯ Next.js 14 App Router frontend
- ▯ Tailwind CSS styling (no Material-UI)

System Health: ▯ 95%+ Production Ready

12.2 Next Steps for Maintenance

1. **Review unregistered files** - Remove or archive unused route files
2. **Implement missing features** - Cart system, wishlist, token refresh
3. **Integrate static pages** - Connect about/contact pages to CMS
4. **Security audit** - Review all endpoints, add rate limiting
5. **Performance optimization** - Add caching, optimize queries
6. **Monitoring setup** - PM2, logging, error tracking
7. **Documentation updates** - Keep this doc updated with changes

12.3 Contact & Support

For questions or issues:

- Review this documentation first
- Check code comments in relevant files
- Review API endpoint documentation
- Check database models for field definitions
- Test with Postman/browser DevTools

Appendix: Quick Reference

File Organization

```
Project Root
├── backend/               # Node.js/Express API
│   └── src/
│       ├── models/        # 45 Mongoose models
│       ├── controllers/   # 26 business logic files
│       ├── routes/        # 33 route files (31 registered)
│       ├── middleware/    # 7 middleware files
│       ├── services/      # 12 service integrations
│       └── server.js      # Main entry point
└── frontend/              # Next.js 14 App Router
    ├── app/               # 20+ pages
    └── src/
        ├── components/    # 69 components
        ├── services/      # 15 API services
        ├── contexts/      # AuthContext
        └── hooks/         # 3 custom hooks
```

Tech Stack Summary

- **Backend**: Node.js 18+, Express.js, MongoDB, Mongoose, JWT, Socket.io
- **Frontend**: Next.js 14 (App Router), React 18, Tailwind CSS, Context API
- **Database**: MongoDB (45+ collections)
- **Payments**: Razorpay, PayU, Stripe
- **Storage**: Cloudinary
- **Email**: Nodemailer

Key URLs

- **Backend**: http://localhost:5000
- **Frontend**: http://localhost:3000
- **API Base**: http://localhost:5000/api
- **Database**: mongodb://localhost:27017/soulsathi_travel_12_11_2025

End of Documentation

This comprehensive guide covers every aspect of the SoulSathi Travel Platform. Use it as your primary reference for maintenance, debugging, and feature development. Good luck! 🚀