HomyHive Project

Comprehensive Technical Documentation

For RAG System Implementation

Generated on: October 27, 2025

Project Owner: Nagashree-250804

Table of Contents

1. Project Overview

1.1 Project Description

HomyHive is a comprehensive travel and accommodation platform similar to Airbnb, built with Node.js, Express, and MongoDB. The platform connects travelers with local hosts, providing unique accommodation experiences across India. The project implements modern web development practices including responsive design, secure authentication, real-time features, and comprehensive user management.

1.2 Key Features

Feature	Description	Status
User Authentication	Multi-provider OAuth (Google, Facebook) + Local auth	■ Complete
Property Listings	CRUD operations for accommodations	■ Complete
Booking System	Reservation management with payment integration	■ Complete
Review System	User reviews and ratings	■ Complete
Geolocation	Mapbox integration for location services	■ Complete
Email System	Nodemailer with department routing	■ Complete
Host Onboarding	Comprehensive host registration	■ Complete
Static Pages	FAQ, Careers, Blog, Help Center	■ Complete
Newsletter	Email subscription system	■ Complete
Responsive Design	Mobile-first responsive UI	■ Complete

1.3 Technology Stack

Layer	Technology	Version	Purpose
Backend	Node.js	22.18.0	Runtime environment
Framework	Express.js	5.1.0	Web application framework
Database	MongoDB Atlas	Latest	Cloud database service
ODM	Mongoose	8.17.0	MongoDB object modeling
Authentication	Passport.js	0.7.0	Authentication middleware
Template Engine	EJS	3.1.10	Server-side templating
File Upload	Multer + Cloudinary	2.0.2	Image upload and storage
Email Service	Nodemailer	7.0.10	Email sending capability
Maps	Mapbox SDK	0.16.2	Location and mapping services
SMS	Twilio	5.10.3	SMS and communication services
Validation	Joi	18.0.1	Data validation library

2. Technical Architecture

2.1 MVC Architecture

HomyHive follows the Model-View-Controller (MVC) architectural pattern: • Models: MongoDB schemas defined with Mongoose for User, Listing, Review, and Newsletter • Views: EJS templates with responsive design and component-based structure • Controllers: Express route handlers managing business logic and data flow

2.2 Project Structure

```
HomyHive/ ■■■ app.js # Main application entry point ■■■ package.json # Dependencies and scripts ■■■ .env # Environment variables ■■■ cloudConfig.js # Cloudinary configuration ■■■ middleware.js # Custom middleware functions ■■■ schema.js # Joi validation schemas ■ ■■■ controllers/ # Business logic controllers ■ ■■■ listings.js # Listing CRUD operations ■ ■■■ reviews.js # Review management ■ ■■■ users.js # User management ■ ■■■ otp.js # OTP verification system ■ ■■■ models/ # MongoDB schemas ■ ■■■ listing.js # Property listings model ■ ■■■ review.js # Reviews and ratings model ■ ■■■ user.js # User authentication model ■ ■■■ routes/ # Route definitions ■ ■■■ listing.js # Listing routes ■ ■■■ review.js # Review routes ■ ■■■ user.js # Authentication routes ■ ■■■ newsletter.js # Newsletter subscription ■ ■■■ static.js # Static page routes ■ ■■■ views/ # EJS templates ■ ■■■ layouts/ # Base layouts ■ ■■■ includes/ # Reusable components ■ ■■■ listings/ # Listing-related views ■ ■■■ users/ # Authentication views ■ ■■■ static/ # Static content pages ■ ■■■ public/ # Static assets ■ ■■■ css/ # Stylesheets ■ ■■■ js/ # Client-side JavaScript ■ ■■■ static/ # Images and media ■ ■■■ utils/ # Utility functions ■ ■■■ ExpressError.js # Error handling ■ ■■■ sendMail.js # Email utilities ■ ■■■ wrapAsync.js # Async error wrapper ■ ■■■ init/ # Database initialization ■■■ data.js # Sample data ■■■ index.js # Data seeding script
```

2.3 Key Middleware

Middleware	Purpose	Implementation
Session Management	User session handling	express-session + MongoStore
Authentication	User login/logout	Passport.js with local & OAuth
Flash Messages	User feedback	connect-flash for notifications
File Upload	Image handling	Multer + Cloudinary integration
Error Handling	Global error management	Custom ExpressError class
Validation	Input validation	Joi schemas with custom middleware
Security	Route protection	isLoggedIn, isOwner middleware
CORS	Cross-origin requests	Built-in Express CORS handling

3. Database Models

3.1 User Model

```
const userSchema = new Schema({ email: { type: String, required: true, unique: true }, // Passport-local-mongoose adds username and password displayName: String, googleId: String, facebookId: String, profilePicture: String, phoneNumber: String, isEmailVerified: { type: Boolean, default: false }, isPhoneVerified: { type: Boolean, default: false }, createdAt: { type: Date, default: Date.now } }); userSchema.plugin(passportLocalMongoose, { usernameField: 'email' });
```

3.2 Listing Model

```
const listingSchema = new Schema({ title: { type: String, required: true, }, description: String, image: { url: String, filename: String, }, price: { type: Number, default: 0 }, location: String, country: String, reviews: [{ type: Schema.Types.ObjectId, ref: "Review", }], owner: { type: Schema.Types.ObjectId, ref: "User", }, geometry: { type: { type: String, enum: ['Point'], required: true }, coordinates: { type: [Number], required: true } } }); // Geospatial indexing for location-based queries listingSchema.index({ geometry: '2dsphere' });
```

3.3 Review Model

```
const reviewSchema = new Schema({ comment: String, rating: { type: Number, min: 1, max: 5 }, createdAt: { type: Date, default: Date.now }, author: { type: Schema.Types.ObjectId, ref: "User" } });
```

3.4 Newsletter Model

```
const NewsletterEmailSchema = new Schema({ email: { type: String, required: true, unique: true }, subscribedAt: { type: Date, default: Date.now }, isActive: { type: Boolean, default: true } });
```

4. API Endpoints

4.1 Authentication Endpoints

Method	Endpoint	Description	Middleware
GET	/signup	User registration page	None
POST	/signup	Process user registration	validateUser
GET	/login	User login page	None
POST	/login	Process user login	passport.authenticate
GET	/logout	User logout	isLoggedIn
POST	/send-otp	Send OTP for verification	None
POST	/verify-otp	Verify OTP	None
GET	/auth/google	Google OAuth initiation	passport.authenticate
GET	/auth/google/callback	Google OAuth callback	passport.authenticate
GET	/auth/facebook	Facebook OAuth initiation	passport.authenticate
GET	/auth/facebook/callback	Facebook OAuth callback	passport.authenticate

4.2 Listing Endpoints

Method	Endpoint	Description	Middleware
GET	/listings	Display all listings	None
GET	/listings/new	New listing form	isLoggedIn
POST	/listings	Create new listing	isLoggedIn, validateListing
GET	/listings/:id	Show specific listing	None
GET	/listings/:id/edit	Edit listing form	isLoggedIn, isOwner
PUT	/listings/:id	Update listing	isLoggedIn, isOwner, validateListing
DELETE	/listings/:id	Delete listing	isLoggedIn, isOwner

4.3 Review Endpoints

Method	Endpoint	Description	Middleware
POST	/listings/:id/reviews	Create review	isLoggedIn, validateReview
DELETE	/listings/:id/reviews/:reviewId	Delete review	isLoggedIn, isReviewAuthor

4.4 Static Page Endpoints

Method	Endpoint	Description
GET	/about	About page
GET	/contact	Contact page
POST	/contact	Process contact form
GET	/faq	FAQ page with interactive features
GET	/careers	Careers page with job listings
GET	/blog	Blog page with category filtering
GET	/help	Help center with search functionality
GET	/host	Host onboarding page

POST	/host/signup	Process host signup
GET	/host/resources	Host resources page
GET	/host/support	Host support page
GET	/community	Community hub page
GET	/events	Events page
GET	/stories	Guest stories page
GET	/tips	Travel tips page
GET	/safety	Safety information page
GET	/accessibility	Accessibility features page
POST	/newsletter	Newsletter subscription
GET	/privacy	Privacy policy page
GET	/terms	Terms of service page

5. Authentication System

5.1 Multi-Provider Authentication

HomyHive implements a comprehensive authentication system supporting multiple login methods: • Local Authentication: Username/password with Passport Local Strategy • Google OAuth 2.0: Seamless Google account integration • Facebook OAuth: Facebook account integration • Email Verification: OTP-based email verification system • Phone Verification: SMS-based phone number verification via Twilio

5.2 OAuth Configuration

```
// Google OAuth Strategy passport.use(new GoogleStrategy({ clientID:
process.env.GOOGLE_CLIENT_ID, clientSecret: process.env.GOOGLE_CLIENT_SECRET,
callbackURL: "/auth/google/callback" }, async (accessToken, refreshToken,
profile, done) => { try { const email = profile.emails[0].value; let user = await
User.findOne({ email: email }); if (user) { if (!user.googleId) { user.googleId =
profile.id; user.displayName = profile.displayName; await user.save(); } return
done(null, user); } else { const newUser = new User({ email: email, googleId:
profile.id, displayName: profile.displayName, isEmailVerified: true }); await
newUser.save(); return done(null, newUser); } } catch (error) { return
done(error, null); } }));
```

5.3 Session Management

```
const sessionOptions = { store: MongoStore.create({ mongoUrl: dbUrl, crypto: {
secret: process.env.SECRET, }, touchAfter: 24 * 3600, }), secret:
process.env.SECRET, resave: false, saveUninitialized: true, cookie: { expires:
Date.now() + 7 * 24 * 60 * 60 * 1000, maxAge: 7 * 24 * 60 * 60 * 1000, httpOnly:
true, }, };
```

5.4 Middleware Functions

```
// Authentication middleware module.exports.isLoggedIn = (req, res, next) => { if
(!req.isAuthenticated()) { req.session.redirectUrl = req.originalUrl;
req.flash("error", "You must be logged in to access this page!"); return
res.redirect("/login"); } next(); }; // Owner authorization middleware
module.exports.isOwner = async (req, res, next) => { let { id } = req.params; let
listing = await Listing.findById(id); if
(!listing.owner.equals(res.locals.currUser._id)) { req.flash("error", "You are
not the owner of this listing!"); return res.redirect(`/listings/${id}`); }
next(); };
```

6. Frontend Components

6.1 Template Engine & Layout

HomyHive uses EJS (Embedded JavaScript) as the template engine with a component-based architecture: • Base Layout: Common HTML structure with responsive design • Reusable Components: Header, footer, flash messages, forms • Responsive Design: Mobile-first approach with Bootstrap integration • Interactive Elements: JavaScript-enhanced user experience

6.2 Key Frontend Features

Component	Description	Technologies
Navigation Bar	Responsive navigation with user status	EJS, Bootstrap, JavaScript
Flash Messages	User feedback system	Connect-flash, Custom CSS
Forms	Input validation and submission	HTML5, JavaScript, Joi validation
Image Upload	Drag-and-drop file upload	Multer, Cloudinary, JavaScript
Maps Integration	Interactive property location	Mapbox GL JS
Star Ratings	Interactive rating system	Custom CSS, JavaScript
Search & Filter	Property search functionality	JavaScript, AJAX
Modal Dialogs	Enhanced user interactions	JavaScript, CSS
Responsive Design	Mobile-optimized layouts	CSS Media Queries, Flexbox

6.3 CSS Architecture

```
/* Custom CSS Variables for Consistent Theming */ :root { --primary-color:
#fe424d; --primary-hover: #d7263d; --secondary-color: #ff7e5f;
--background-light: #f8f9fa; --text-dark: #333333; --text-light: #666666;
--border-color: #dee2e6; --success-color: #28a745; --warning-color: #ffc107;
--error-color: #dc3545; } /* Responsive Grid System */ .listing-grid { display:
grid; grid-template-columns: repeat(auto-fit, minmax(300px, 1fr)); gap: 2rem;
padding: 2rem; } /* Interactive Card Components */ .listing-card { background:
white; border-radius: 12px; box-shadow: 0 4px 20px rgba(0,0,0,0.1); transition:
transform 0.3s, box-shadow 0.3s; overflow: hidden; } .listing-card:hover {
transform: translateY(-5px); box-shadow: 0 8px 30px rgba(0,0,0,0.15); }
```

6.4 JavaScript Functionality

```
// Interactive Map Integration function initializeMap(coordinates) {
mapboxgl.accessToken = mapToken; const map = new mapboxgl.Map({ container: 'map',
style: 'mapbox://styles/mapbox/streets-v12', center: coordinates, zoom: 10 });
const marker = new mapboxgl.Marker({ color: '#fe424d' }) .setLngLat(coordinates)
.addTo(map); } // Form Validation function validateForm(formId) { const form =
document.getElementById(formId); const inputs =
form.querySelectorAll('input[required]'); inputs.forEach(input => {
input.addEventListener('blur', function() { if (!this.value.trim()) {
this.classList.add('error'); } else { this.classList.remove('error'); } }); }); }
```

7. Features & Functionality

7.1 Core Platform Features

HomyHive provides a comprehensive set of features for both guests and hosts:

Feature Category	Guest Features	Host Features
Account Management	• Multi-provider authentication\n• Profile management\n• Email/phone verification	• Host onboarding process\n• Identity verification\n• Payout setup
Property Discovery	• Advanced search & filters\n• Map-based browsing\n• Property details & photos	• Property listing creation\n• Photo upload & management\n• Property analytics
Booking System	• Instant booking\n• Booking history\n• Cancellation management	• Booking requests\n• Calendar management\n• Guest communication
Communication	• Host messaging\n• Support chat\n• Email notifications	• Guest messaging\n• Host support\n• Notification preferences
Reviews & Ratings	• Submit reviews\n• View host ratings\n• Property feedback	• Respond to reviews\n• Guest ratings\n• Performance metrics
Support System	• 24/7 help center\n• FAQ system\n• Contact forms	• Host resources\n• Community support\n• Educational content

7.2 Advanced Functionality

Email System

Comprehensive email system with department-based routing: • Guest Support: guest-support@homyhive.com • Host Support: host-support@homyhive.com • Payments: payments@homyhive.com • Emergency: emergency@homyhive.com • Partnerships: partnerships@homyhive.com • Press Inquiries: press@homyhive.com • Trust & Safety: trust-safety@homyhive.com

Geolocation Services

Mapbox integration for location-based features: • Interactive property maps • Geocoding for address conversion • Distance-based search • Neighborhood information • 2D sphere indexing for efficient queries

7.3 Content Management

Page Type	Features	Interactive Elements
FAQ Page	Categorized questions, search functionality	Accordion UI, filtering, search
Blog Page	Content categories, featured posts	Category filtering, modal previews
Careers Page	Job listings, application system	Application forms, email integration
Help Center	Comprehensive support articles	Search, categorization, quick help
Host Resources	Educational content, guides	Resource downloads, tutorials
Contact Page	Multi-department routing	Smart forms, emergency contacts

8. Configuration & Environment

8.1 Environment Variables

```
# Database Configuration
ATLASDB_URL=mongodb+srv://username:password@cluster.mongodb.net/homyhive #
Session Security SECRET=your-super-secret-session-key # Cloudinary Configuration
(Image Storage) CLOUD_NAME=your-cloudinary-cloud-name
CLOUD_API_KEY=your-cloudinary-api-key CLOUD_API_SECRET=your-cloudinary-api-secret
# Mapbox Configuration MAP_TOKEN=your-mapbox-access-token # OAuth Configuration
GOOGLE_CLIENT_ID=your-google-oauth-client-id
GOOGLE_CLIENT_SECRET=your-google-oauth-client-secret
FACEBOOK_APP_ID=your-facebook-app-id FACEBOOK_APP_SECRET=your-facebook-app-secret
# Email Configuration GMAIL_USER=your-gmail-username
GMAIL_APP_PASSWORD=your-gmail-app-password # SMS Configuration (Twilio)
TWILIO_ACCOUNT_SID=your-twilio-account-sid
TWILIO_AUTH_TOKEN=your-twilio-auth-token
TWILIO_PHONE_NUMBER=your-twilio-phone-number
```

8.2 Cloudinary Configuration

```
const cloudinary = require('cloudinary').v2; const { CloudinaryStorage } =
require('multer-storage-cloudinary'); cloudinary.config({ cloud_name:
process.env.CLOUD_NAME, api_key: process.env.CLOUD_API_KEY, api_secret:
process.env.CLOUD_API_SECRET }); const storage = new CloudinaryStorage({
cloudinary: cloudinary, params: { folder: 'homyhive_listings', allowedFormats:
["png", "jpg", "jpeg"], transformation: [ { width: 800, height: 600, crop: "fill"
}, { quality: "auto" }, { fetch_format: "auto" } ] }, });
```

8.3 Database Configuration

```
const mongoose = require("mongoose"); const dbUrl = process.env.ATLASDB_URL;
async function main() { await mongoose.connect(dbUrl, { useNewUrlParser: true,
useUnifiedTopology: true, }); console.log("Connected to MongoDB Atlas"); } //
MongoDB Atlas Connection with Error Handling main().then(() => {
console.log("Database connection established"); }).catch((err) => {
console.error("Database connection failed:", err); });
```

8.4 Security Configuration

Security Feature	Implementation	Purpose
Session Management	express-session + MongoStore	Secure user sessions with database persistence
Password Hashing	passport-local-mongoose	Automatic password hashing with salt
CSRF Protection	Built-in Express protection	Prevent cross-site request forgery
Input Validation	Joi validation schemas	Sanitize and validate all user inputs
File Upload Security	Multer + Cloudinary	Secure file handling with cloud storage
Environment Variables	dotenv configuration	Secure credential management
OAuth Security	Passport strategies	Secure third-party authentication
Error Handling	Custom error middleware	Prevent information leakage

9. Deployment Guide

9.1 Production Setup

HomyHive is designed for production deployment with the following considerations:

1. Server Requirements: • Node.js 22.18.0 or higher • MongoDB Atlas cluster • SSL certificate for HTTPS • Domain name configuration 2. Environment Setup: • Configure all environment variables • Set NODE_ENV=production • Enable MongoDB connection pooling • Configure reverse proxy (Nginx recommended) 3. Security Hardening: • Enable HTTPS/SSL • Configure Content Security Policy • Set secure session cookies • Enable rate limiting • Configure CORS appropriately 4. Performance Optimization: • Enable gzip compression • Configure caching headers • Optimize database queries • Enable CDN for static assets • Monitor application performance

9.2 Cloud Deployment Options

Platform	Pros	Configuration Notes
Heroku	Easy deployment, automatic scaling	Use Heroku Postgres addon, configure buildpacks
AWS EC2	Full control, scalable	Configure Load Balancer, Auto Scaling Groups
DigitalOcean	Cost-effective, simple	Use App Platform or Droplets
Google Cloud	GCP integration, reliable	Use Cloud Run or Compute Engine
Vercel	Easy Node.js deployment	Configure environment variables, database
Railway	Simple deployment process	Connect GitHub repo, set environment

9.3 Monitoring & Maintenance

```
// Application Health Check Endpoint app.get('/health', (req, res) => { const health = { status: 'OK', timestamp: new Date().toISOString(), uptime: process.uptime(), environment: process.env.NODE_ENV, database: mongoose.connection.readyState === 1 ? 'Connected' : 'Disconnected' }; res.status(200).json(health); }); // Error Logging and Monitoring const winston = require('winston'); const logger = winston.createLogger({ level: 'info', format: winston.format.json(), transports: [ new winston.transports.File({ filename: 'error.log', level: 'error' }), new winston.transports.File({ filename: 'combined.log' }) ], });
```

10. Code Structure & Best Practices

10.1 Project Organization

HomyHive follows industry best practices for Node.js application structure: • Separation of Concerns: Clear separation between routes, controllers, models, and views • Modular Architecture: Each feature is organized in its own module • Configuration Management: Environment-based configuration • Error Handling: Centralized error handling with custom error classes • Validation: Input validation using Joi schemas • Security: Multiple layers of security implementation

10.2 Code Quality Standards

Aspect	Implementation	Benefits
Code Organization	MVC pattern with modular structure	Maintainable and scalable codebase
Error Handling	Try-catch blocks with custom error classes	Robust error management and debugging
Input Validation	Joi schemas for all user inputs	Data integrity and security
Database Queries	Mongoose ODM with population	Type safety and relationship management
Async Handling	Async/await with proper error catching	Clean asynchronous code
Security Practices	Multiple authentication strategies	Comprehensive security coverage
Code Reusability	Middleware functions and utilities	DRY principle implementation
Documentation	Inline comments and README files	Easy onboarding and maintenance

10.3 Development Workflow

```
// Development Commands npm install # Install dependencies npm start # Start production server node app.js # Start development server // Environment Setup cp .env.example .env # Copy environment template # Configure all required environment variables // Database Initialization cd init node index.js # Seed database with sample data // Development Best Practices 1. Always validate user inputs 2. Use middleware for common functionality 3. Implement proper error handling 4. Test all routes and features 5. Keep environment variables secure 6. Use meaningful commit messages 7. Document API endpoints 8. Implement logging for debugging
```

10.4 Future Enhancements

Potential areas for enhancement and expansion: • Real-time Messaging: WebSocket implementation for instant communication • Payment Integration: Stripe/PayPal integration for secure payments • Advanced Search: Elasticsearch for complex search queries • Mobile App: React Native or Flutter mobile application • Analytics Dashboard: Real-time analytics for hosts and admins • AI Features: Recommendation engine and price optimization • Multi-language Support: Internationalization (i18n) implementation • Progressive Web App: PWA features for offline functionality • Advanced Security: Two-factor authentication and advanced fraud detection • API Documentation: Swagger/OpenAPI documentation for external developers

End of Documentation

This documentation provides comprehensive information about the HomyHive project for RAG system implementation. For technical support or questions, contact the development team.