Final Project Report

Project Name: Hotel Booking Website

Course: Internet Programming
University: Institute of Technology of Cambodia
Lecturer: Mr. CHUN Thavorac

Team Members

No.	Student Name	Student ID
1	Sam Sokleap	P20230031
2	Kheang Ann	P20230027

Table of Contents

1. Introduction

1.1 Problem Statement

In Cambodia's growing tourism industry, many hotels face significant challenges in managing their online presence and booking processes. Our research identified four critical pain points:

1.1.1 Discovery Problem (Source)

Many Cambodian travelers still rely on **phone calls and social media** (Facebook, Telegram) to find and book hotels:

- **Limited Exposure:** Hotels without websites miss potential customers
- **Inconsistent Information:** Hotel details scattered across multiple social media posts
- **Time-Consuming Process:** Customers must contact each hotel individually

- **Language Barriers:** International tourists struggle with Khmer-only pages

1.1.2 Cost Problem (Price)

Most small to medium-sized hotels **cannot afford to build and maintain their own booking website**:

- **Development Costs:** Custom website development costs $3,000-$15,000+
- **Hosting Expenses:** Monthly server and domain costs
- **Technical Expertise:** Hotels lack IT staff to manage systems
- **Payment Integration:** Secure payment gateways require technical knowledge

Our Solution: Provide a **free platform** where hotels can list their properties without development or hosting costs.

1.1.3 Operations Problem (Management)

Handling bookings, payments, and communications creates operational overhead:

- **Manual Tracking:** Bookings tracked in notebooks or spreadsheets
- **Payment Collection:** Cash-only or bank transfers are difficult to track
- **Communication Gaps:** Missed calls and delayed responses lose customers
- **No Analytics:** Hotels lack data to optimize pricing and occupancy

Our Solution: Centralized dashboard with **payment processing, booking management, and analytics**.

1.1.4 Confidence Problem (Trust)

Both customers and hotels need assurance when transacting online:

- **Customer Concerns:** "Will the hotel honor my booking?" "Is my payment secure?"
- **Hotel Concerns:** "Will the customer actually show up?" "How do I verify payment?"

Our Solution:

- Secure Stripe payments with verified transactions
- Rating and review system for accountability
- Booking confirmation workflow with status tracking

1.2 Objectives

The primary objectives of the Hotel Booking Website project are:

1. **Develop a User-Friendly Booking Platform:** Create an intuitive web interface where guests can easily search for hotels, view room availability, and complete bookings with minimal friction.

2. **Implement Real-Time Room Availability Management:** Build a sophisticated availability algorithm that prevents double-bookings and accurately tracks room inventory across overlapping date ranges.

3. **Integrate Secure Payment Processing:** Implement Stripe payment gateway to handle secure credit card transactions with proper webhook handling and payment status tracking.

4. **Create a Comprehensive Admin Dashboard:** Develop a full-featured administrative interface for managing hotels, rooms, users, bookings, amenities, and viewing business analytics.
5. **Ensure System Security:** Implement robust authentication (including OAuth 2.0), authorization, and data protection mechanisms to safeguard user information.
6. **Build a Scalable Architecture:** Design a modular, maintainable system architecture that can scale to accommodate future growth.
7. **Support Multi-Hotel Operations:** Enable the system to manage multiple hotels with individual configurations, room types, amenities, and pricing structures.
8. **Implement Rating and Review System:** Allow guests to provide feedback on their stays, helping future customers make informed decisions.

1.3 Scope

In Scope

User-Facing Features:

- User registration with email/password or Google OAuth 2.0
- User authentication and profile management
- Hotel search with filters (destination, dates, guest count)
- Room browsing with availability checking
- Multi-room booking capability
- Secure payment processing via Stripe
- Booking management (view, cancel)
- Hotel rating and review submission
- Password reset functionality

Admin Features:

- Admin authentication with role-based access control
- Dashboard with analytics and charts
- Hotel CRUD operations with image management
- Room CRUD operations with bed and amenity configuration
- User management (view, activate/deactivate)
- Booking management (approve, reject, view)
- Amenity and bed type management
- Payment/bill monitoring

System Features:

- Automated booking expiration handling
- Real-time room availability calculation
- Image storage (local development, Cloudinary for production)
- Tax calculation (10% tax rate)
- Booking status lifecycle management

Out of Scope

The following features are not included in the current version:

- Mobile applications (iOS/Android)
- Multi-language support
- Multi-currency support
- KHQR payment method
- Staff and hotel owner roles
- Room service management
- Loyalty/rewards program
- Chat/messaging system between guests and hotels

---

2. System Design

2.1 Project Structure

The project follows a monorepo structure with clearly separated frontend and backend codebases:

```
Hotel-Booking-System/
├── docker-compose.yml
├── .env                        # Env for Backend
├── README.md
│
├── backend/                    # NestJS Backend API
│   ├── package.json
│   ├── src/
│   │   ├── main.ts
│   │   ├── app.module.ts
│   │   ├── config/             # Database, Cloudinary, Upload configs
│   │   ├── auth/               # Authentication & User management
│   │   ├── hotels/             # Hotels CRUD
│   │   ├── rooms/              # Rooms CRUD
│   │   ├── booking/            # Booking management
│   │   ├── payment/            # Stripe payment integration
│   │   ├── ratings/            # Reviews & Ratings
│   │   ├── amenities/          # Amenities management
│   │   └── bed-types/          # Bed types configuration
│   └── uploads/                # Local file storage (dev)
│
└── frontend/
    ├── user/                   # User-facing Vue.js App
    │   ├── src/
    │   │   ├── views/          # Page components
    │   │   ├── components/     # Reusable components
    │   │   ├── stores/         # Pinia state management
    │   │   └── router/         # Vue Router
    │   └── .env                # env for user frontend
    │
    └── admin/                  # Admin Vue.js App
```

```
├── src/
│   ├── views/          # Admin pages
│   ├── components/     # Admin components
│   ├── stores/         # State management
│   └── composables/    # Reusable logic
└── .env                # env for admin frontend
```

2.2 System Architecture

The Hotel Booking System follows a **three-tier architecture** with clear separation of concerns:

Layer	Components	Technologies
Presentation	User Frontend, Admin Frontend	Vue.js 3, Pinia, Vue Router, Axios, Chart.js
Application	Backend API	NestJS, Passport, JWT, Helmet, Validation Pipe
Data	Database, Payments, Storage	PostgreSQL (Neon DB), Stripe, Cloudinary
External	Social Auth	Google OAuth 2.0

Data Flow:

- User Frontend → (HTTPS/REST) → Backend API
- Admin Frontend → (HTTPS/REST) → Backend API
- Backend API → PostgreSQL, Stripe, Cloudinary, Google OAuth

2.3 Architecture Patterns

2.3.1 Backend Architecture Pattern: Modular MVC

The NestJS backend follows a **Modular MVC (Model-View-Controller)** pattern:

Layer	Responsibility	Implementation
Controllers	Handle HTTP requests, route to services	`*.controller.ts` files
Services	Business logic, data processing	`*.service.ts` files
Entities	Data models, database schema	`entities/*.entity.ts`
DTOs	Request/Response validation	`dto/*.dto.ts`
Guards	Authorization middleware	`guards/*.guard.ts`
Strategies	Authentication strategies	`strategies/*.strategy.ts`

2.3.2 Frontend Architecture Pattern: Component-Based with Composition API

The Vue.js frontends utilize:

Pattern	Purpose	Implementation

Pattern	Purpose	Implementation
Composition API	Logic reuse, better TypeScript support	`<script setup>` syntax
Pinia State Management	Centralized state, reactive stores	`stores/*.ts`
Vue Router	Client-side routing, route guards	`router/index.ts`
Composables	Extractable, reusable logic	`composables/*.ts`

2.3.3 Database Design Pattern: Relational with TypeORM

- **Entity-Relationship Model** with normalized tables
- **TypeORM Repository Pattern** for data access
- **Query Builder** for complex queries
- **Auto-synchronization** for development (disabled in production)

2.4 Entity Relationship Diagram (ERD)

Entity Relationships:

Parent Entity	Relationship	Child Entity
USER	has many	USER_ROLE
ROLE	has many	USER_ROLE
USER	creates many	BOOKING
USER	makes many	PAYMENT
USER	writes many	RATING
BOOKING	contains many	BOOKING_ITEM
BOOKING	has one	PAYMENT
BOOKING	receives one	RATING
HOTEL	has many	ROOM
HOTEL	has many	HOTEL_AMENITY
HOTEL	receives many	RATING
ROOM	booked in many	BOOKING_ITEM
ROOM	has many	ROOM_AMENITY
ROOM	has many	ROOM_BED
AMENITY	assigned to many	HOTEL_AMENITY, ROOM_AMENITY
BED_TYPE	assigned to many	ROOM_BED

Key Entity Attributes:

Entity	Key Attributes
USER	id, email, password, firstName, lastName, provider, profileImage, isActive
BOOKING	id, userId, totalPrice, status, guestPhone, checkIn, checkOut, paymentExpiresAt
BOOKING_ITEM	id, bookingId, roomId, roomName, hotelName, priceAtBooking, quantity
PAYMENT	id, bookingId, userId, amount, status, stripePaymentId, stripeSessionId
HOTEL	id, name, description, destination, location, email, phone, avgRating, images
ROOM	id, hotelId, name, price, available, maxOccupancy, discountPercentage, images
RATING	id, hotelId, userId, bookingId, overallScore, service, facilities, comfort, value, location, comment

2.5 API Architecture

Module	Base URL	Description
Auth	`/auth`	Authentication, registration, OAuth
Admin	`/admin`	Admin-specific operations
Hotels	`/hotels`	Hotel CRUD and search
Rooms	`/rooms`	Room management and availability
Booking	`/bookings`	Booking lifecycle management
Payment	`/payments`	Stripe payment processing
Ratings	`/ratings`	Hotel reviews and ratings
Amenities	`/amenities`	Amenity management
Bed Types	`/bed-types`	Bed configuration

3. Technology Stack

3.1 Backend Technologies

Technology	Version	Purpose
NestJS	11.x	Backend framework
TypeScript	5.7.x	Type-safe development
TypeORM	0.3.x	ORM for database
PostgreSQL	16.x	Primary database
Passport/JWT	11.x	Authentication
Stripe SDK	20.x	Payment processing

Technology	Version	Purpose
Helmet	8.x	Security headers
Cloudinary SDK	1.41.x	Cloud image storage

3.2 Frontend Technologies

Technology	Version	Purpose
Vue.js	3.5.x	Frontend framework
TypeScript	5.9.x	Type-safe development
Pinia	3.x	State management
Axios	1.13.x	HTTP client
Vite	7.x	Build tool
Chart.js	4.5.x	Analytics charts

3.3 DevOps & Deployment

Technology	Purpose
Render	Backend & Frontend hosting
Neon	PostgreSQL database hosting
Cloudinary	Image CDN and storage
GitHub Actions	CI/CD auto-deploy
Docker/Docker Compose	Local development

4. Features

4.1 User Features

4.1.1 Authentication & Authorization

Feature	Description
User Registration	Create account with email/password validation (min 8 characters, uppercase, lowercase, number)
Email/Password Login	Secure login with bcrypt password verification
Google OAuth 2.0	One-click sign-in with Google account
JWT Session Management	Stateless authentication with 1-day token expiration
Password Reset	Email-based password recovery with secure tokens

Feature	Description
Profile Management	Update personal information and profile picture
Account Deactivation Handling	Graceful handling when admin deactivates user account

4.1.2 Hotel Discovery

Feature	Description
Destination Search	Filter hotels by destination/location
Date-Based Search	Search with check-in and check-out dates
Guest Count Filter	Filter rooms by occupancy capacity
Hotel Listing	Browse all available hotels with images and ratings
Hotel Details	View hotel information, amenities, location, and contact
Room Browsing	View available room types with prices and amenities
Real-Time Availability	See actual room availability for selected dates

4.1.3 Booking System

Feature	Description
Multi-Room Selection	Add multiple rooms to a single booking
Price Calculation	Real-time price calculation with discount and tax (10%)
Booking Summary	Review booking details before confirmation
Booking Creation	Submit booking request with guest information
Booking History	View all past and current bookings
Booking Details	View individual booking information and status
Booking Cancellation	Cancel pending or confirmed bookings
Status Tracking	Monitor booking status (Pending → Confirmed → Completed)

4.1.4 Payment Processing

Feature	Description
Stripe Checkout	Secure payment via Stripe Checkout Session
Payment Status	Track payment status (Pending, Completed, Failed)
Payment Expiration	1-hour window to complete payment after admin approval
Automatic Failure Handling	Booking marked as failed if payment not completed in time

4.1.5 Rating & Review System

Feature	Description
Post-Stay Rating	Rate hotels after completed stays
Category Scores	Rate service, facilities, comfort, value, location (1-10)
Overall Score	Automatic calculation of overall rating (1-5 stars)
Written Review	Leave detailed comments about the stay
Review Visibility	View other guests' reviews on hotel pages

4.2 Admin Features

4.2.1 Dashboard & Analytics

Feature	Description
Overview Dashboard	Central view of system statistics
Booking Statistics	Charts showing booking trends and status distribution
Revenue Analytics	Payment and revenue visualization
User Statistics	User registration and activity metrics
Chart.js Visualizations	Interactive charts for data analysis

4.2.2 Hotel Management

Feature	Description
Hotel List	View all hotels with search and filter
Create Hotel	Add new hotels with details and images
Edit Hotel	Modify hotel information and amenities
Delete Hotel	Remove hotels (with booking validation)
Image Management	Upload, reorder, and remove hotel images
Status Toggle	Activate/deactivate hotels

4.2.3 Room Management

Feature	Description
Room List	View all rooms grouped by hotel
Create Room	Add rooms with pricing, beds, and amenities
Edit Room	Modify room details and configuration

Feature	Description
Delete Room	Remove rooms (with active booking check)
Bed Configuration	Configure bed types and quantities
Discount Management	Set percentage discounts on rooms
Inventory Control	Set available room count

4.2.4 User Management

Feature	Description
User List	View all registered users
User Details	View user profile and booking history
Activate User	Re-enable deactivated user accounts
Deactivate User	Disable user access (prevents login)
Role Assignment	Manage user roles (admin/user)

4.2.5 Booking Management

Feature	Description
Booking List	View all bookings with filters
Booking Details	View full booking information
Approve Booking	Confirm pending bookings (starts 1-hour payment timer)
Reject Booking	Reject bookings with reason
Cancel Booking	Cancel confirmed bookings
Status Filtering	Filter by booking status

4.2.6 Amenity & Bed Type Management

Feature	Description
Hotel Amenities	Manage hotel-level amenities (WiFi, Pool, Parking)
Room Amenities	Manage room-level amenities (AC, TV, Minibar)
Icon Selection	Choose icons for amenities
Bed Types	Manage bed types (Single, Double, King, etc.)

4.2.7 Bill/Payment Monitoring

Feature	Description

Feature	Description
Payment History	View all payment transactions
Payment Status	Monitor payment statuses
Transaction Details	View Stripe transaction IDs
Revenue Tracking	Track completed payments

5. Third-Party Integration

5.1 Stripe Payment Gateway

Purpose: Secure online payment processing for booking payments

Integration Type: Stripe Checkout Sessions with Webhooks

Payment Flow

1. User clicks "Pay Now" on Frontend
2. Frontend sends POST /payments/stripe/checkout to Backend
3. Backend creates Checkout Session with Stripe
4. Stripe returns session URL to Backend
5. Backend returns checkout URL to Frontend
6. Frontend redirects User to Stripe Checkout Page
7. User completes payment on Stripe
8. Stripe sends Webhook (payment_intent.succeeded) to Backend
9. Backend updates Payment Status = COMPLETED
10. Backend updates Booking Status = COMPLETED
11. Stripe redirects User to Success Page

Key Features

Feature	Implementation
Checkout Sessions	Stripe-hosted payment page for security
Webhook Handling	Real-time payment status updates
Signature Verification	Validate webhook authenticity
Payment Intents	Stripe Payment Intents API for SCA compliance
Metadata	Store booking ID in payment metadata

5.2 Google OAuth 2.0

Purpose: Social login allowing users to sign in with their Google accounts

OAuth Flow

1. User clicks "Sign in with Google" on Frontend
2. Frontend redirects User to /auth/google
3. User sends GET /auth/google to Backend
4. Backend redirects User to Google Consent Screen
5. User authorizes the App on Google
6. Google sends Callback with auth code to Backend
7. Backend exchanges code for user info with Google
8. Google returns profile data to Backend
9. Backend creates or finds User in database
10. Backend generates JWT Token
11. Backend redirects User with JWT token
12. User is now logged in!

5.3 Cloudinary (Cloud Image Storage)

Purpose: Cloud-based image storage for hotel, room, and profile images in production

Features

Feature	Description
Automatic Upload	Images uploaded directly to Cloudinary
Image Transformation	Automatic resizing (1200x800 max)
Format Support	JPG, JPEG, PNG, WEBP formats
Folder Organization	Separate folders for hotels, rooms, profiles
CDN Delivery	Global CDN for fast image loading
Cleanup	Automatic deletion of replaced images

6. Security Implementation

6.1 Authentication Security

Measure	Implementation
Password Hashing	bcrypt with 10 salt rounds
Password Requirements	Min 8 chars, uppercase, lowercase, number
JWT Token	HMAC-SHA256, 1-day expiration
Stateless Auth	No server-side sessions

6.2 Authorization

Guard	Purpose
JwtAuthGuard	Validates JWT on protected routes

Guard	Purpose
RolesGuard	Enforces role-based access control
@Public()	Marks routes as public

6.3 Additional Security

Security Layer	Implementation
HTTP Headers	Helmet.js (X-Frame-Options, XSS Protection, HSTS, etc.)
CORS	Configurable origins, credentials enabled
Input Validation	ValidationPipe with whitelist, auto-transform
SQL Injection	TypeORM parameterized queries
Webhook Security	Stripe signature verification
File Upload	Allowed formats validation (jpg, jpeg, png, webp)

7. Business Logic

7.1 Booking Status Lifecycle

The booking system implements a comprehensive status lifecycle:

Current State	Action/Trigger	Next State	Notes
(Start)	User Creates Booking	PENDING	Rooms reserved
PENDING	Admin Approves	CONFIRMED	1hr payment window starts
PENDING	Admin Rejects	CANCELLED	Rooms released
PENDING	User Cancels	CANCELLED	Rooms released
CONFIRMED	Payment Success	COMPLETED	Booking active
CONFIRMED	Payment Timeout (1hr)	FAILED	Rooms released (cron job)
CONFIRMED	User Cancels	CANCELLED	Rooms released

Status Details:

Status	Description	Room Impact	Next Actions
PENDING	Booking created, awaiting admin approval	Reserved (blocked)	Admin: Approve/Reject; User: Cancel
CONFIRMED	Admin approved, awaiting payment	Still reserved	User: Pay/Cancel; System: Auto-fail
COMPLETED	Payment successful, booking active	Booked	User: Rate after stay

Status	Description	Room Impact	Next Actions
CANCELLED	Cancelled by user or admin	Released	None
FAILED	Payment timeout (auto)	Released	User: Create new booking

7.2 Room Availability Algorithm

The system implements a sophisticated overlap detection algorithm that queries the database for all bookings that might conflict with a requested date range. The algorithm counts bookings with active statuses (pending, confirmed) as well as completed bookings that haven't ended yet.

Date Overlap Formula

Two date ranges overlap when: **existingCheckIn < newCheckOut AND existingCheckOut > newCheckIn**

This formula ensures that:

- New bookings starting before existing ones end are detected
- New bookings ending after existing ones start are detected
- Edge cases where dates align exactly are handled correctly

7.3 Price Calculation

The price calculation follows a multi-step process:

1. **Apply Discount:** Base price × (1 - discount percentage / 100)
2. **Calculate Room Total:** Discounted price × number of nights
3. **Sum All Rooms:** Subtotal = sum of all room totals
4. **Apply Tax:** Tax = subtotal × 10%
5. **Final Total:** Total = subtotal + tax

Example: A Deluxe Suite at $200/night with 15% discount for 3 nights would be: $200 × 0.85 × 3 = $510 subtotal + $51 tax = **$561 total**

7.4 Payment Expiration System

The system uses an automated task scheduler (cron job) that runs every minute to monitor payment deadlines:

1. **Query:** Find all CONFIRMED bookings where `paymentExpiresAt` has passed
2. **Check:** For each expired booking, verify if a completed payment exists
3. **Update:** If no payment found, mark booking as FAILED with reason "Payment not completed within 1 hour. Rooms released."

This ensures rooms are automatically released back to inventory when users don't complete payment within the 1-hour window.

7.5 Rating Calculation

Overall rating is calculated from 5 category scores (service, facilities, comfort, value, location):

1. **Average Categories:** Sum all 5 category scores (each 1-10) and divide by 5
2. **Convert Scale:** Divide average by 2 to convert from 1-10 scale to 1-5 scale
3. **Round:** Round to one decimal place for display

This allows users to rate specific aspects while the system generates a comparable overall score.

7.6 Booking Validation Rules

Validation	Rule	Error Message
Age Check	Guest must be 16+ years old	"Guest must be at least 16 years old"
Date Validation	Check-out must be after check-in	"Check-out date must be after check-in"
Availability	Requested quantity ≤ available count	"Room only has X available for selected dates"
Status Transition	Only valid status changes allowed	"Only pending bookings can be confirmed"
Duplicate Rating	One rating per booking	"You have already rated this booking"

8. Deployment

8.1 Deployment Architecture

Component	Platform	Type
User Frontend	Render	Static Site
Admin Frontend	Render	Static Site
Backend API	Render	Web Service
PostgreSQL	Neon DB	Database
Images	Cloudinary	CDN

CI/CD: GitHub push → Render auto-build → Deploy

8.2 Deployment Configuration

Backend (Render Web Service):

Setting	Value
Root Directory	`backend`
Build Command	`npm install && npm run build`
Start Command	`npm run start:prod`
Auto-Deploy	Yes (production branch)

Frontend (Render Static Sites):

Setting	Value
Build Command	`npm install && npm run build`
Publish Directory	`dist`
Rewrite Rule	`/* → /index.html`

8.3 URLs

Service	URL
User Frontend	`https://cambook.onrender.com`
Admin Frontend	`https://cambook-admin.onrender.com`
Backend API	`https://hotel-booking-system-qjxx.onrender.com`

---

9. Challenges & Solutions

9.1 Validation Handling

Challenge: Implementing comprehensive input validation across the entire application to prevent invalid data from entering the system while providing meaningful error messages.

Why Both Backend AND Frontend Validation?

Layer	Purpose	Benefits
Frontend	User experience, instant feedback	Prevents unnecessary API calls, shows errors before submission
Backend	Security, data integrity	Protects against malicious requests, API consumers, bypassed frontend

Solution: Dual-layer validation:

- **Frontend (Vue.js):** Provides instant feedback with regex validation for emails, password length checks, and form completeness validation before submission
- **Backend (NestJS DTO):** Uses class-validator decorators for security-critical validation including email format, password requirements (8+ characters, uppercase, lowercase, number), and sanitization

9.2 Duplicate Prevention

Challenge: Preventing duplicate entries for critical data like hotel names, room names within hotels, user emails, and preventing double-booking of rooms.

Solutions:

- **Case-Insensitive Duplicate Checking:** Uses SQL LOWER() function to compare names regardless of capitalization, throwing ConflictException when duplicates found

- **Scoped Uniqueness:** Room names must be unique within their hotel (same room name allowed across different hotels)
- **Availability Validation:** Before creating a booking, the system checks if the requested quantity is available for the specified dates and throws an error if insufficient rooms exist

9.3 Google OAuth Integration

Challenge: Implementing Google OAuth 2.0 authentication while seamlessly integrating with the existing JWT-based authentication system.

Issues Faced:

- Configuring Google Cloud Console credentials
- Handling callback URL differences between development and production
- Managing users who sign up via Google vs email/password
- Cross-origin popup handling with security headers

Solutions:

- **Passport Google Strategy:** Extracts email and name from Google profile, then calls unified OAuth login handler
- **Unified User Creation:** If user doesn't exist, creates new account with Google profile data and a random password (so OAuth users can't login with password)
- **Security Headers:** Uses Helmet middleware with `crossOriginOpenerPolicy: 'same-origin-allow-popups'` to enable OAuth popup flow while maintaining security

9.4 Hotel Business Logic

Challenge: Understanding and implementing complex hotel booking business logic, including room inventory management, booking workflows, payment timelines, and status transitions.

Solutions:

1. **Room Inventory Model:** Each room type has an `available` count representing physical rooms
2. **Booking Lifecycle:** Clear status transitions with defined rules
3. **Availability Algorithm:** Sophisticated overlap detection for date ranges
4. **Payment Window:** 1-hour expiration with automated cron job monitoring
5. **Tax Calculation:** Standard 10% tax rate applied to all bookings

---

10. Future Plans

10.1 Microservice Architecture

Service	Responsibilities
Auth Service	User Auth, JWT/OAuth, Profiles
Booking Service	Bookings, Availability, Scheduling
Payment Service	Stripe, KHQR, Refunds

Service	Responsibilities
Hotel Service	Hotels, Rooms, Amenities
Notification Service	Email, SMS, Push

10.2 KHQR Payment Integration

Cambodian QR payment standard for local payment support:

1. User selects KHQR Payment
2. Backend requests QR generation from Bank
3. Frontend displays QR Code
4. User scans and pays via Banking App
5. Bank sends webhook to Backend
6. Booking confirmed

10.3 More Roles

Role	Key Permissions
SUPER ADMIN	All access, System config, All hotels
ADMIN	All hotels, User mgmt, Analytics
HOTEL OWNER	Own hotels, Room mgmt, Staff mgmt
OWNER STAFF	View hotel, Manage rooms, View bookings
STAFF	View bookings, Check-in/out
USER	Book rooms, View/cancel, Rate hotels

10.4 Additional Future Enhancements

Enhancement	Target	Description
Reporting	Phase 1	Advanced analytics with PDF/Excel exports
Seasonal Pricing	Phase 1	Date-based dynamic pricing
Mobile App	Phase 2	React Native or Flutter applications
Multi-language	Phase 2	i18n support (English, Khmer, Chinese)
Multi-currency	Phase 2	USD, KHR, and other currencies
Loyalty Program	Phase 3	Points system with rewards
Chat System	Phase 3	Real-time messaging
Channel Manager	Phase 4	Integration with Booking.com, Agoda, Airbnb

11. Conclusion

The Hotel Booking Website project successfully delivers a comprehensive, modern hotel booking solution that addresses the core challenges faced by the hospitality industry. Through careful system design and implementation, the team has created a platform that serves both guests seeking accommodation and administrators managing hotel operations.

Key Achievements

1. **Full-Stack Implementation:** Complete end-to-end solution with Vue.js frontend applications and NestJS backend API, demonstrating proficiency in modern web development technologies.
2. **Robust Business Logic:** Implementation of complex hotel booking workflows including real-time availability checking, booking status lifecycle management, and automated payment expiration handling.
3. **Security-First Approach:** Multiple layers of security including JWT authentication, bcrypt password hashing, Google OAuth 2.0, role-based access control, and comprehensive input validation.
4. **Third-Party Integration:** Successful integration with Stripe for payment processing, Google for social authentication, and Cloudinary for cloud image storage.
5. **Scalable Architecture:** Modular design with clear separation of concerns, preparing the system for future microservice migration and feature expansion.
6. **Production-Ready Deployment:** Full deployment pipeline with Docker containerization for local development and cloud hosting on Render for backend and frontends, with PostgreSQL database on Neon.

Technical Accomplishments

Metric	Achievement
Database Entities	10+ entities with complex relationships
Backend Modules	9 modules with complete CRUD operations
Frontend Applications	2 apps (User and Admin)
Authentication Methods	3 methods (Local, JWT, Google OAuth)
API Endpoints	50+ RESTful endpoints
Automated Jobs	Payment expiration cron scheduling
Real-Time Features	Availability calculation algorithm
Cloud Integration	Image CDN with transformations

Learning Outcomes

Through this project, the team gained valuable experience in:

- Enterprise-level application architecture
- TypeScript and modern JavaScript frameworks
- Database design and ORM implementation

- RESTful API design and documentation
- Authentication and authorization patterns
- Payment gateway integration
- Cloud deployment and DevOps practices
- Agile development and teamwork

Future Direction

The project lays a solid foundation for future enhancements including microservice architecture migration, KHQR payment integration for the Cambodian market, expanded role management for hotel owners and staff, and mobile application development.

The Hotel Booking Website demonstrates the team's ability to deliver a production-quality full-stack application that meets real-world business requirements while maintaining code quality, security standards, and user experience excellence.

Appendix

A. Default Admin Credentials

The system automatically seeds a default administrator account on first startup:

Field	Value
Email	`admin@cambook.kh`
Password	`Hello123!`
Role	Admin

B. Status Code Reference

Booking Status

Status	Code	Description
Pending	`pending`	Awaiting admin approval
Confirmed	`confirmed`	Approved, awaiting payment
Completed	`completed`	Payment received, booking active
Cancelled	`cancelled`	Cancelled by user or admin
Failed	`failed`	Payment timeout or failure

Payment Status

Status	Code	Description
Pending	`pending`	Payment initiated

Status	Code	Description
Completed	`completed`	Payment successful
Failed	`failed`	Payment failed
Refunded	`refunded`	Payment refunded

C. Full API Endpoints Summary

Authentication (`/auth`)

Method	Endpoint	Auth	Description
POST	`/auth/register`	Public	User registration with email/password
POST	`/auth/login`	Public	User login (LocalStrategy)
GET	`/auth/google`	Public	Initiate Google OAuth 2.0
GET	`/auth/google/redirect`	Public	Google OAuth callback
GET	`/auth/profile`	User	Get current user profile
PATCH	`/auth/profile`	User	Update profile (with image upload)
PATCH	`/auth/change-password`	User	Change password

Admin Management (`/admin`)

Method	Endpoint	Auth	Description
GET	`/admin/users`	Admin	Get all users with filters
POST	`/admin/users`	Admin	Create new user
GET	`/admin/users/stats`	Admin	Get user statistics
GET	`/admin/users/:id`	Admin	Get user by ID
PATCH	`/admin/users/:id`	Admin	Update user
PATCH	`/admin/users/:id/roles`	Admin	Update user roles
PATCH	`/admin/users/:id/status`	Admin	Update user status (active/inactive)

Hotels (`/hotels`)

Method	Endpoint	Auth	Description
GET	`/hotels`	Public	Get all active hotels
GET	`/hotels/:id`	Public	Get hotel by ID
POST	`/hotels`	Admin	Create hotel (with images)

Method	Endpoint	Auth	Description
PATCH	`/hotels/:id`	Admin	Update hotel
DELETE	`/hotels/:id`	Admin	Delete hotel
PATCH	`/hotels/:id/status`	Admin	Toggle hotel active status
GET	`/hotels/admin/all`	Admin	Get all hotels (including inactive)
GET	`/hotels/search/availability`	Public	Search hotels with availability
GET	`/hotels/filter/lowest-price`	Public	Hotels sorted by lowest price
GET	`/hotels/filter/highest-rating`	Public	Hotels sorted by rating

Rooms (`/rooms`)

Method	Endpoint	Auth	Description
GET	`/rooms/available`	Public	Get all available rooms
GET	`/rooms/:id`	Public	Get room by ID
GET	`/rooms/hotel/:hotelId`	Public	Get rooms by hotel ID
GET	`/rooms/hotel/:hotelId/availability`	Public	Get rooms with availability check
GET	`/rooms/:id/availability`	Public	Check room availability for dates
POST	`/rooms`	Admin	Create room (with images)
PATCH	`/rooms/:id`	Admin	Update room
DELETE	`/rooms/:id`	Admin	Delete room

Bookings (`/bookings`)

Method	Endpoint	Auth	Description
POST	`/bookings`	User	Create new booking
GET	`/bookings`	User	Get user's bookings
GET	`/bookings/:id`	User	Get booking by ID
PATCH	`/bookings/:id/cancel`	User	Cancel booking
POST	`/bookings/calculate-price`	User	Calculate booking price
GET	`/bookings/admin/all`	Admin	Get all bookings
PATCH	`/bookings/admin/:id/approve`	Admin	Approve pending booking
PATCH	`/bookings/admin/:id/reject`	Admin	Reject pending booking

Payments (`/payments`)

Method	Endpoint	Auth	Description
GET	`/payments/:id/status`	User	Get payment status
GET	`/payments/booking/:bookingId`	User	Get payment by booking ID
GET	`/payments/my-payments`	User	Get user's payments
POST	`/payments/stripe/checkout`	User	Create Stripe checkout session
GET	`/payments/stripe/verify`	User	Verify payment session
POST	`/payments/stripe/webhook`	Public	Stripe webhook handler
GET	`/payments/admin/all`	Admin	Get all payments

Ratings (`/ratings`)

Method	Endpoint	Auth	Description
POST	`/ratings`	User	Create rating
GET	`/ratings/:id`	User	Get rating by ID
PATCH	`/ratings/:id`	User	Update rating
DELETE	`/ratings/:id`	User	Delete own rating
GET	`/ratings/hotel/:hotelId`	Public	Get ratings by hotel
GET	`/ratings/booking/:bookingId`	User	Get rating by booking
GET	`/ratings/my-ratings`	User	Get user's ratings

Amenities (`/amenities`)

Method	Endpoint	Auth	Description
GET	`/amenities`	Public	Get all amenities
GET	`/amenities/:id`	Public	Get amenity by ID
GET	`/amenities/category/:category`	Public	Get amenities by category
POST	`/amenities`	Admin	Create amenity
PATCH	`/amenities/:id`	Admin	Update amenity
DELETE	`/amenities/:id`	Admin	Delete amenity

Bed Types (`/bed-types`)

Method	Endpoint	Auth	Description
GET	`/bed-types`	Public	Get all bed types

Method	Endpoint	Auth	Description
GET	`/bed-types/:id`	Public	Get bed type by ID
POST	`/bed-types`	Admin	Create bed type
PATCH	`/bed-types/:id`	Admin	Update bed type
DELETE	`/bed-types/:id`	Admin	Delete bed type

D. Environment Variables Reference

The system requires the following environment variable categories:

Category	Variables
Database	`DB_HOST`, `DB_PORT`, `DB_USER`, `DB_PASSWORD`, `DB_NAME`, `DATABASE_URL`
JWT	`JWT_SECRET`
Stripe	`STRIPE_SECRET_KEY`, `STRIPE_WEBHOOK_SECRET`, `STRIPE_SUCCESS_URL`, `STRIPE_CANCEL_URL`
Google OAuth	`GOOGLE_CLIENT_ID`, `GOOGLE_CLIENT_SECRET`, `GOOGLE_CALLBACK_URL`
Cloudinary	`CLOUDINARY_CLOUD_NAME`, `CLOUDINARY_API_KEY`, `CLOUDINARY_API_SECRET`
CORS	`CORS_ORIGIN`
General	`NODE_ENV`, `PORT`, `FRONTEND_URL`

E. Important Links

Resource	URL
GitHub Repository	`https://github.com/Sokleap-SAM/Hotel-Booking-System`
User Frontend	`https://cambook.onrender.com`
Admin Frontend	`https://cambook-admin.onrender.com`
Backend API	`https://hotel-booking-system-qjxx.onrender.com`
API Documentation	`https://hotel-booking-system-qjxx.onrender.com/api`

End of Final Project Report