GuestEase

A full-stack web app for small hospitality businesses

Report

Student Name: Andrea Nardinocchi
Student Number: 20108884
Programme: Higher Diploma in Science in Computer Science 2024
Project Type: Independent Project
Project Category: Web Application

Declaration of Authenticity

I declare that the work which follows is my own, and that any quotations from any sources (e.g., books, journals, the internet) are clearly identified as such by the use of 'single quotation marks', for shorter excerpts and identified italics for longer quotations. All quotations and paraphrases are accompanied by (date, author) in the text and a fuller citation is the bibliography. I have not submitted the work represented in this report in any other course of study leading to an academic award.

Student: Andrea Nardinocchi Date: 26/03/2026

Acknowledgements

I would like to recognize the outstanding work all lecturers of the Higher Diploma in Computer Science course consistently delivered throughout.

They surely deserve a lot of credit for being able to push me through the finishing line of this course, which I truly believe has been the most difficult and heaviest one I have ever undertaken in my life. I make no bones about the fact I have been on the brink of dropping out several times and considered it an incredible achievement of being able to finally complete it.

Last but not least, the greatest thank you ever to my excellent supervisor Catherine Fitzpatrick who provided me with precious advice and, above all, plenty of encouragement.

Table of Contents

Declaration of Authenticity **2**
Acknowledgements **2**
1 Introduction **9**
1.1. Objectives 9
1.2. Motivation 9
2 Research & Analysis **11**
2.1. Technology 11
2.1.1 Frontend 11
Figure 2-1: Web frameworks and technologies by popularity amongst professional developers 11
Figure 2-2: Frontend frameworks popularity trend 12
2.1.1.1 React 12
2.1.1.2 Svelte 12
2.1.1.3 Angular 13
2.1.1.4 Vue 14
2.1.1.5 Conclusions 14
Table 2-1 : Frontend frameworks evaluation 15
2.1.2 Backend/Database 15
2.1.2.1 Node.js 15
2.1.2.2 Firebase 16
2.1.2.3 Supabase 16
2.1.2.4 Conclusions 17
Table 2-2 : Backend/Database frameworks evaluation 18
2.1.3 The repos research 18
2.1.3.1 The Fern Hotel Booking Site (2025) 18
2.1.3.2 Full-Stack Hotel Booking System by Suresh K. (2025) 18
2.1.3.3 Hotel Booking System by Othmane Nissoukin (2025) 19
2.1.3.4 Conclusions 20
Table 2-3 : Repos evaluation 21
2.2. Market analysis 21
2.2.1 Digital Transformation in Ireland's Small-Scale Accommodation Sector 21
2.2.2 Market Conditions & Guest Behaviour in Ireland 21
2.2.3 What are the challenges that small hospitality operators are facing? 22
2.2.4 The Case for GuestEase 22
2.3. Comparative analysis of hospitality booking systems 22
2.3.1 Marriott.com 23
Figure 2-1: Marriott.com sticky booking form 23
Figure 2-2: Marriott.com hotel details pop up 24

Figure 2-3: Marriott.com search results page with map 24
Figure 2-4: Marriott.com search results page with no map 24
Figure 2-5: Marriott.com user account overview page 25
Figure 2-6: Marriott.com user account 'My Trips' page 25
2.3.2 IHG Hotels & Resorts 25
Figure 2-7: IHG sticky header with 'Book Now' CTA 26
Figure 2-8: IHG search results page 26
Figure 2-9: IHG user account overview page 27
Figure 2-10: IHG user account overview page 27
2.3.3 Hilton 27
Figure 2-11: Hilton search results page with map 27
Figure 2-12: Hilton search results page with list 28
Figure 2-13: Hilton search results page with list only 28
Figure 2-14: Hilton user account overview page 29
Conclusions and GuestEase Implications 29
3 Modelling & Design 29
3.1. User types and user stories 29
3.1.1 The GuestEase Guest Persona 29
3.1.2 The GuestEase Administrator Persona 30
3.1.3 Use case diagram 31
Figure 3-1: Diagram of use case Figma 31
3.2. Data Modelling 31
3.2.1 Model: Entities and Relationships 31
3.2.2 Core Relationships 32
3.2.3 Logical/Physical Data Model (ERD) 34
Figure 3-2: Diagram of use case 34
3.3. Wireframes 36
3.3.1 Homepage 36
Figure 3-3: Homepage wireframe 36
3.3.2 Search Results page 36
Figure 3-4: Search results page wireframe 37
3.3.3 Room details 37
Figure 3-5: Room details wireframe 38
3.3.4 Login page 38
Figure 3-6: Log in page wireframe 38
3.3.5 Sign up page 38
Figure 3-7: Sign up page wireframe 39
3.3.6 Account - My Trips page 39
Figure 3-8: Account - My Trips page wireframe 39
3.3.7 Account - Profile page 39
Figure 3-9: Account - Profile page wireframe 40

3.3.8 Account - User profile drawer 40
Figure 3-10: Account - User profile drawer wireframe 41
3.3.9 Account - Booking Review page 41
Figure 3-11: Account - Write a review page wireframe 42
3.3.10 Account - Room Details page payment widget 42
Figure 3-12: Account - Payment page 42
3.3.11 Account - Reservation confirmation page 42
Figure 3-13: Account - Reservation confirmation page 43
3.3.12 Admin Dashboard 43
Figure 3-14: Admin Dashboard 44
3.3.13 Admin Dashboard Filters Drawer 44
Figure 3-15: Example of an admin dashboard filter 44
3.3.14 Update Password page 45
Figure 3-16: Update password page 45
3.3.15 About Us page 45
Figure 3-17: About Us page 46
3.3.16 Facilities page 46
Figure 3-18: Facilities page 46
3.3.17 Our Rooms page 46
Figure 3-19: Our Rooms page 47
3.4. Final Design 47
3.4.1 Homepage 47
Figure 3-20: Homepage 48
3.4.2 Search Results page 48
Figure 3-21: Search Results page 48
3.4.3 Room Details 48
Figure 3-22: Room Details page 49
3.4.4 Login page 49
Figure 3-23: Login page 49
3.4.5 Sign up page 49
Figure 3-24: Sign up page 50
3.4.6 Account - My Trips page 50
Figure 3-25: Account My Trips page 50
3.4.7 Account - Profile page 50
Figure 3-26: Profile page 51
3.4.8 Account - User Profile Drawer 51
Figure 3-27: User Profile Drawer 51
3.4.9 Account - Booking Review page 51
Figure 3-28: Booking Review page 52
3.4.10 Account - Stripe payment modal 52
Figure 3-29: Stripe Payment Modal 52

3.4.11 Account - Reservation confirmation page 52
Figure 3-30: Reservation confirmation page 53
3.4.12 Admin Dashboard 53
Figure 3-31: Overview of the Admin Dashboard 53
3.4.13 Admin Dashboard DrawerFilters 53
Figure 3-32: Filters drawer on the Admin Bookings page 54
3.4.14 Update password page 54
Figure 3-33: Update Password page 54
3.4.15 About us page 54
Figure 3-34: About us page 55
3.4.16 Facilities page 55
Figure 3-35: Facilities page 55
3.4.17 Our rooms page 55
Figure 3-36: Our Rooms page 56
3.4.18 Example of a GuestEase email notification via Resend 56
Figure 3-37: GuestEase email notification via Resend 56
4 System Overview 56
4.1. Technologies 56
4.1.1 Frontend technologies 56
React & TypeScript 56
MUI (Material UI) 57
Vite 57
4.1.2 Backend technologies 57
Node.js & Express 57
Supabase Server Client 57
dotenv 57
4.1.3 Database & Authentication Technologies 57
Supabase (PostgreSQL Database) 57
Supabase Auth 58
Supabase Storage 58
Supabase Edge Functions 58
Row-Level Security (RLS) 58
4.1.4 Development & Deployment Tools 58
Node.js & NPM 58
ESLint & Prettier 58
Vercel & Render 58
4.1.5 Third-Party Services 58
Stripe 58
Resend (Email Delivery Service) 58
Conclusion 59
5 Project Implementation 59

5.1. Methodology and Tools 59
5.2 Agile Sprints 60
5.2.1 Trello plan 60
Figure 5-1: Trello plan 61
https://trello.com/b/WBdyHT0S/guestease 61
5.2.1 Sprint 1 — January 1st to January 15th 61
5.2.3 Sprint 2 — January 16th to January 30th 62
5.2.4 Sprint 3 — January 31st to February 13th 62
5.2.5 Sprint 4 — February 14th to February 28th 63
5.2.6 Sprint 5 — March 1st to March 14th 63
5.2.7 Sprint 6 — March 15th to March 28th 63
5.3. GuestEase ReadMe documentation 64
6 Reflection 64
6.1. System evaluation 64
6.2. App Testing 65
Table 6-1: Test Cases 66
6.3. Security aspects 66
6.4. Problems encountered 67
Figure 6-1: Supabase - Users Authentication dashboard 67
Figure 6-2: Supabase - ‘public.profiles’ table 67
Figure 6-3: Room Details page console 68
Figure 6-4: Excerpt of the useAvailabaleRooms.ts hook 68
Figure 6-5: Room Details booking error 69
Figure 6-6: calculateTotalPriceUtil.js code 69
Figure 6-7: Supabase PostgreSQL ‘function check_email_exists’ 70
Figure 6-8: Edit Booking modal on the /account/mytrips page 71
Figures 6-9: Excerpts of the editBookingDialog.tsx code 72
Figure 6-10: RoomDetailsPage ‘Rendered more hooks..’ error 72
Figure 6-11: RoomDetailsPage ‘loading and error returns’ 73
Figure 6-12: Example of updated image path on the bookingConfirmationPage.tsx 74
Figure 6-13: Excerpt of the Supabase Edge Function ‘charge-upcoming-bookings’ 75
Figure 6-14: Excerpt of the Supabase ‘Bookings’ table Edge Function ‘charge-upcoming-bookings’ 75
Figure 6-15: Excerpt of the backend Render logs when the email functionality is triggered 76
Figure 6-16: Excerpt of the Resend Domain section on https://resend.com/domains/add 76
6.5. Learning 77
6.6. Achievements 77
6.7. Future developments 77

7 AI role **78**
Bibliography **80**
Appendix A — AI Prompts and Responses **82**
AI-1 — React 82
AI-2 — Svelte 83
AI-3 — Angular 84
AI-4 — Vue 85
AI-5 — Node.js 86
AI-6 — Firebase 87
AI-7 — Supabase 88
AI-8 — Digital Transformation in Ireland's Small-Scale Accommodation Sector 89
AI-9 — What are the challenges that small hospitality operators are facing? 90
AI-10 — The GuestEase Guest Persona 91
AI-11 — The GuestEase Administrator Persona 92

1 Introduction

1.1. Objectives

GuestEase is a full-stack web app that helps small hospitality businesses manage bookings efficiently. It replaces manual reservations and OTAs with an easy-to-use platform where guests book online and admins manage rooms, pricing, and reservations via a secure dashboard.

1.2. Motivation

With a background in Tourism Science and experience in Marriott Digital Operations, I developed an interest in how modern hospitality booking systems are built. Working daily with global Agile squads, stakeholders, project managers, UX designers, QA specialists, and developers sparked my curiosity beyond content management into system architecture and development. This project was a way for me to explore the technical side of a simplified online booking system for small accommodation providers.

1.3. Application Final Features

GuestEase is a prototype app for an 8-room B&B which incorporates the following features:

User Roles and Access Control

- Two user roles (Guest, Administrator)
- Secure authentication
- Admin ability to create users

User Booking System

- Room browsing, date-based and room capacity availability
- Full booking CRUD functionality for guests
- Real-time availability management
- Error-free digitalized booking workflow

Administration Dashboard

- Room management (CRUD)
- Booking management (CRUD)
- User management (CRUD)
- View and monitor guest reviews

Automated Email Notifications

Guest notifications:

- Booking confirmation
- Booking updates or cancellations
- Payment confirmation
- Review request on check-out day
- Welcome email when a guest account is created by admin

Administrator notifications:

- Alerts when a guest submits a review

Payment Processing

Secure online payment handling using Stripe:

- Secure card authorization
- Payment capture within 24 hours of check-in
- Refund handling based on cancellation policies

Guest Review System

- Guests can submit reviews after their stay
- The admin receive notifications when reviews are submitted
- Reviews can be viewed within the admin dashboard

Responsive User Interface

- Mobile-friendly design
- Optimized for desktop, tablet, and mobile devices

2 Research & Analysis

2.1. Technology

2.1.1 Frontend

The frontend framework choice is a critical one since the UI is the first encounter of the user with an app. I had experience with Svelte and React, but I decided to evaluate other options, although React.js seems to be the most widely adopted framework (Stack Overflow, 2025).

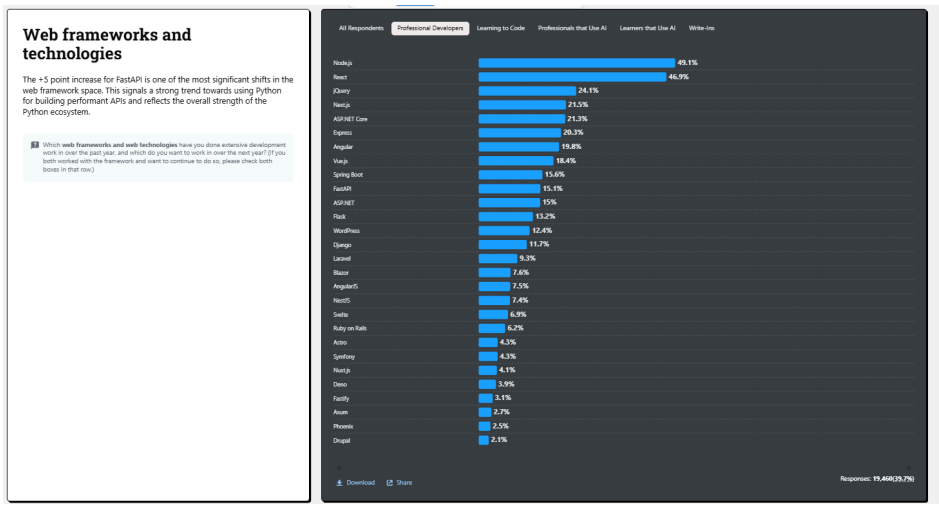


Figure 2-1: Web frameworks and technologies by popularity amongst professional developers

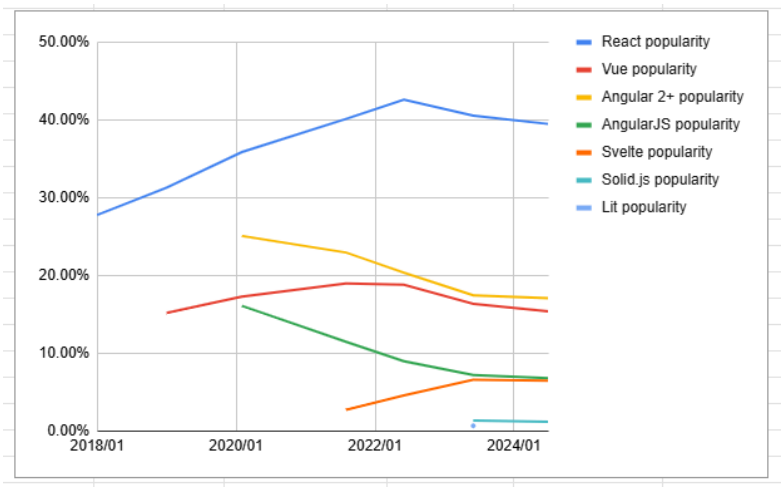


Figure 2-2: Frontend frameworks popularity trend

However, Svelte has been growing steeply, but Angular.js and Vue.js still maintain higher market share (Krotoff, n.d.).

2.1.1.1 React

React offers strong community support, efficient virtual DOM updates, and modular components suited for scalable, dynamic hotel booking interfaces (Dineuron, 2025; DevAceTech, 2025). AI was used in summarising React's pros and cons, but the output was edited by the author (Microsoft Copilot, 2026; see Appendix A, AI-1).

Pros:

- My familiarity with React speeds up development
- Marriott.com's ecosystem is built entirely on React
- Virtual DOM efficiently handles frequent data changes
- Modular components makes it easy to maintain and scale up
- Large ecosystem for forms, state, and API handling
- Enables responsive, interactive user interfaces

Cons:

- Complex state management may require extra tools
- Setup and routing need configuration
- Long-term maintenance requires periodic updates

2.1.1.2 Svelte

Svelte compiles to plain JavaScript, removing the virtual DOM. Its reactive syntax improves performance, readability, and maintainability, contributing to its growing adoption (Dineuron, 2025; DevAceTech, 2025). AI was used in summarising Svelte's pros and cons, but the output was edited by the author (Microsoft Copilot, 2026; see Appendix A, AI-2).

Pros:

- Fast performance and small bundles for quick room searches
- Reactive UI handles availability, pricing, and bookings smoothly
- Simple syntax speeds up development of forms and calendars

Cons:

- Svelte was too challenging when it comes to full-stack for me
- Limited third-party libraries and community support
- Smaller talent pool
- Perhaps, less scalable than React

2.1.1.3 Angular

Angular is designed for large-scale, enterprise, single-page applications (Dineuron, 2025; DevAceTech, 2025). AI was used in summarising Angular's pros and cons, but the output was edited by the author (Microsoft Copilot, 2026; see Appendix A, AI-3).

Pros:

- Comprehensive framework ideal for complex booking flows
- TypeScript and strong architecture support maintainable, scalable code
- Two-way data binding keeps availability and booking data in sync (Angular, n.d.)

Cons:

- I had no experience with this framework
- Steep learning curve
- Larger bundle sizes can slow initial load
- Verbose code with more boilerplate for components, etc.

2.1.1.4 Vue

Vue.js offers intuitive templates, reactive data binding, strong performance, and a gentle learning curve. AI was used in summarising Vue's pros and cons, but the output was edited by the author (Microsoft Copilot, 2026; see Appendix A, AI-4).

Pros:

- Easy to learn and build interactive booking interfaces
- Reactive system keeps availability and pricing updated automatically
- Composition API supports organized, scalable logic
- Lightweight for fast load times

Cons:

- I had no experience with this framework
- Smaller ecosystem may limit advanced features
- Less enterprise adoption than Angular or React
- Fewer experienced developers available

2.1.1.5 Conclusions

Based on my comparison of the major frontend frameworks and my experience level, React was the clear choice for this project.

Framework	Suitability	Learning Curve	Ecosystem	Performance	Confidence / Maintenance	Notes	Grade
React	High	Moderate	Very large	High	High – already used	Used by Marriott.com	**A** ✅

Svelte	Medium	Moderate	Small	Very High	Medium – less confident	Growing framework	B
Angular	Low	Steep	Large	Medium	Low – unfamiliar	Enterprise-focused	C-
Vue.js	Low	Gentle	Moderate	High	Low – no experience	Less enterprise adoption	C

Table 2-1 : Frontend frameworks evaluation

2.1.2 Backend/Database

The backend handles user data and booking logic. Node.js was my initial choice, but further research was required to assess market alternatives.

2.1.2.1 Node.js

Node.js runs JavaScript server-side, enabling dynamic, cloud-based web applications. It supports HTTP handling, database interaction (Khandelwal, 2019), and high concurrency (GeeksforGeeks, 2025). I also had experience using it with Express.js and Hapi.js. AI was used in summarising Node's pros and cons, but the output was edited by the author (Microsoft Copilot, 2026; see Appendix A, AI-5).

Pros:

- Non-blocking I/O and an event-driven model allow fast handling of many booking requests, searches, and availability checks without slowing down.
- Real-time updates support live room availability and instant confirmations.
- Scalable architecture supports growth in users or features.

Cons:

- I found it difficult to learn, though it offers strong flexibility.
- API changes may require maintenance or code adjustments.

- Callback/async complexity can get messy without good structure.

2.1.2.2 Firebase

Firebase offers a Backend-as-a-Service (BaaS) model that removes backend management. Although I was familiar with NoSQL, I still had limited confidence with it, but Google's authentication, tools, and integrations made Firebase a viable option (Adela, 2025; GeeksforGeeks, 2024; Justin, 2023). AI was used in summarising Firebase's pros and cons, but the output was edited by the author (Microsoft Copilot, 2026; see Appendix A, AI-6).

Pros:

- Fast development with minimal backend setup.
- Real-time updates for live availability and booking confirmations.
- Built-in hosting, authentication, storage, and serverless functions.
- Automatic scaling through the Google infrastructure.

Cons:

- NoSQL limitations make complex relational queries (e.g., multi-room bookings, reporting) more difficult.
- I had never felt confident with NoSQL, which could slow implementation.
- Limited server control makes custom backend logic harder (Ndungu, 2022).

2.1.2.3 Supabase

Supabase is an open-source BaaS offering authentication, storage, real-time features, and APIs on PostgreSQL (Ankolekar, 2025). I preferred it over Firebase due to prior experience and its SQL-based structure, which supports data consistency, relational queries, and transactions, important for preventing double-booking (Ulili, 2025; Justin,

2023). AI was used in summarising Supabase's pros and cons, but the output was edited by the author (Microsoft Copilot, 2026; see Appendix A, AI-7).

Pros:

- I was already familiar with it from coursework.
- Unified backend: database, auth, storage, and real-time features.
- PostgreSQL ensures relational integrity and transactions.
- Supports complex joins (bookings ↔ users ↔ rooms) and real-time updates.

Cons:

- Some features are less mature than larger platforms.
- Global scaling or heavy media workloads may be challenging.
- Relational modeling requires upfront design effort.

2.1.2.4 Conclusions

Supabase was chosen for its SQL-PostgreSQL base and real-time features, while **Node.js** with **Express.js** handled the secure API logic and Stripe integration that Supabase alone couldn't. This choice gave the project a flexible and scalable backend.

Framework	**Suitability**	**Learning Curve**	**Ecosystem**	**Performance**	**Confidence / Maintenance**	**Notes**	**Grade**
Node.js	High	Moderate	Very large	High	High – already used	Flexible, requires DB setup	**B+** ✅
Firebase	Medium	Gentle	Large	Medium	Medium – some NoSQL experience	Fast BaaS, limited relational support	C+

Supabase	High	Moderate	Medium	High	High – already used	PostgreSQL, real-time, relational queries	**A** ✅

Table 2-2 : Backend/Database frameworks evaluation

2.1.3 The repos research

In this chapter, I dug into the technologies being used for the hospitality booking system and investigated a few repos/projects created in 2025 in an effort to stay current.

2.1.3.1 The Fern Hotel Booking Site (2025)

This project (https://git.hubp.de/satishsurani/The-Fern-Hotel-Booking-Site) is a dynamic hotel booking system which uses the below technologies:

Technologies Used

- Front-End: HTML, CSS (Bootstrap), JavaScript / AJAX
- Back-End: PHP
- Database: MySQL
- Payments: Razorpay integration (online payment gateway)
- Additional: Admin panel support, email notifications for bookings

It supports real-time availability, Razorpay online payments (Razorpay, n.d.), an admin dashboard, and automated email notifications (Surani, 2025).

2.1.3.2 Full-Stack Hotel Booking System by Suresh K. (2025)

This hotel booking application (https://github.com/Binary-Shade/Full-stack-hotel-booking) is built using the MERN stack technology (MongoDB, Express, React, Node.js) (Blakely, 2024) and users can browse

hotels, view room details, check availability, and bookings are made with a secure authentication.

Technologies Used

- Frontend: React, CSS, Bootstrap (or other CSS frameworks as needed)
- Backend: Node.js, Express
- Database: MongoDB
- Authentication: JWT (JSON Web Tokens) for secure user sessions
- Deployment: Hosted on services like Heroku (backend) and Netlify/Vercel (frontend)

This repo features technologies closer to the ones I had researched above and seemed to be a strong fit for the combination of features it presents. It also has both a user and admin dashboards (K, 2024).

2.1.3.3 Hotel Booking System by Othmane Nissoukin (2025)

The hotel booking system developed by Othmane Nissoukin (Nissoukin, 2025) represents a modern full-stack approach and was probably the most relevant comparison for my project.

Technology used:

- Front-End: Next.js (React, Tailwind CSS)
- Back-End / Database: Supabase (PostgreSQL + Auth)
- Authentication: Supabase Auth (back-end), NextAuth.js (Front-end)
- Payments: Stripe

The system enables users to browse rooms, manage bookings, and complete secure online payments, while offering a dedicated back-office interface for managing rooms, reservations, and occupancy (https://github.com/OthmaneNissoukin/nextjs-hotel-booking-back-office) . The project

uses a separate repository for the administrative tasks for maintainability and separation of concerns.

This architecture reflects GuestEase's goals by using Supabase and PostgreSQL to maintain relational integrity and avert double bookings.

2.1.3.4 Conclusions

Othmane Nissoukin's hotel booking system impressed me with its modern full-stack architecture, particularly its use of Supabase.

Nevertheless, a combination of Node.js + Express, and Supabase remained my preferred approach for implementing backend functionalities.

Project	Purpose	Front-End	Back-End / Server	Database	Authentication / Admin / Notable Tech
Hotel Booking System (Othmane Nissoukin) ✅	Full hotel booking system with customer-facing and admin interfaces	Next.js (React, Tailwind CSS), React Leaflet, Date-fns, DatePicker, Zod	Next.js API routes	Supabase (PostgreSQL)	Supabase Auth (email, Google, Facebook), user dashboard, admin/back-office dashboards for managing rooms, bookings, and occupancy
Full-stack-hotel-booking (Suresh K)	Full-stack hotel booking with user management	React, Bootstrap, Axios	Node.js + Express	MongoDB	JWT authentication, admin dashboard, role-based access
The-Fern-Hotel-Booking-Sit	Customer-facing hotel booking	HTML, CSS (Bootstrap),	PHP	MySQL	Admin panel, email notifications,

e (Satish Surani)		JavaScript/AJAX			AJAX-based real-time availability

Table 2-3 : Repos evaluation

Reviewing these repos also reassured me that the direction taken for GuestEase was appropriate.

2.2. Market analysis

2.2.1 Digital Transformation in Ireland's Small-Scale Accommodation Sector

Many SMEs in Ireland's hospitality sector (guesthouses, B&Bs, boutique hotels) still rely on manual bookings and lack modern websites (McGarr, 2024; Gerry, 2025). With 80% of reservations now online, slow digital adoption creates inefficiencies and fails to meet guest expectations (john@secured, 2025; Escape2Project. (n.d.)).

AI was used in summarising the above paragraph, but the output was edited by the author (Microsoft Copilot, 2026; see Appendix A, AI-8).

2.2.2 Market Conditions & Guest Behaviour in Ireland

Recent data shows clear growth in digital and mobile booking behaviour:

- 59% of Irish hotel bookings were made on smartphones in April 2025, up from 52% the previous year (Haley, 2025).
- Short-term rental platforms saw a 15% year-on-year increase in 2024 (McCárthaigh, 2025).
- Domestic overnight stays rose by 8.2% in 2024 (CSO statistical release, 2025).
- Ireland is the only European country with hotel occupancy above pre-pandemic levels (102% of February 2019 figures) (Dooley, 2025).

These trends indicate rising demand for seamless mobile-friendly booking experience. Therefore, operators who fail to modernize risk losing guests to digitally enabled competitors.

2.2.3 What are the challenges that small hospitality operators are facing?

Small accommodation providers encounter several challenges:

- Manual bookings increase admin workload and risk double-bookings.
- Limited booking hours conflict with guest expectations for 24/7 self-service.
- Heavy reliance on OTAs reduces margins (15–30% commissions), limits branding, and restricts access to guest data.
- Weak digital presence, especially non-mobile-optimized sites, deters potential guests (Power-Cowley, n.d.).

AI was used in summarising the above paragraph, but the output was edited by the author (Microsoft Copilot, 2026; see Appendix A, AI-9).

2.2.4 The Case for GuestEase

GuestEase offers direct control over pricing, bookings, and guest data, avoiding OTA commissions (Bunnell, 2025). Its booking flow, automated emails, and mobile-friendly design reduce manual work, and enhance guest and admin experience.

Conclusion: Without modernization, many Irish accommodation providers risk falling behind. GuestEase enables cost savings, improved guest satisfaction, and stronger market positioning in a digital-first era.

2.3. Comparative analysis of hospitality booking systems

I investigated the web booking systems of Marriott, IHG, and Hilton, focusing on **language selection**, **navigation**, **booking forms**, **search results**, **filtering**, and **account pages** to identify best practices.

2.3.1 Marriott.com

Marriott (https://www.marriott.com/default.mi) offers a highly user-centered experience. The **language selector** opens a pop-up displaying all options, making selection quick and intuitive. A **sticky header** ensures the booking CTA remains visible and reappears on scroll-up, maintaining seamless navigation. The **booking form** shows destination, dates (specific or flexible), rooms, guests, rates, and loyalty points.

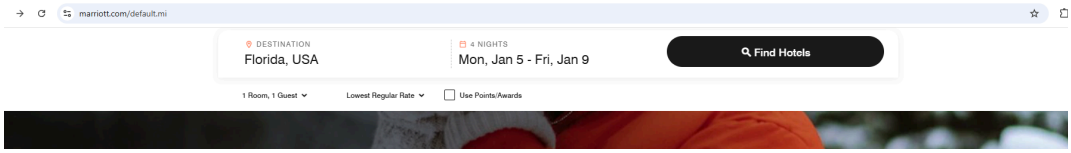


Figure 2-1: Marriott.com sticky booking form

Search results display hotels as banner cards with images, brand logos, price, distance, and ratings. Users can refine results with filters and view hotels on a map.

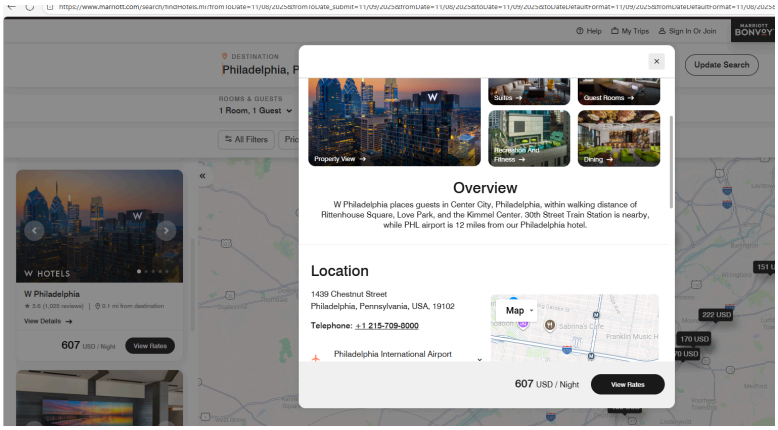


Figure 2-2: Marriott.com hotel details pop up

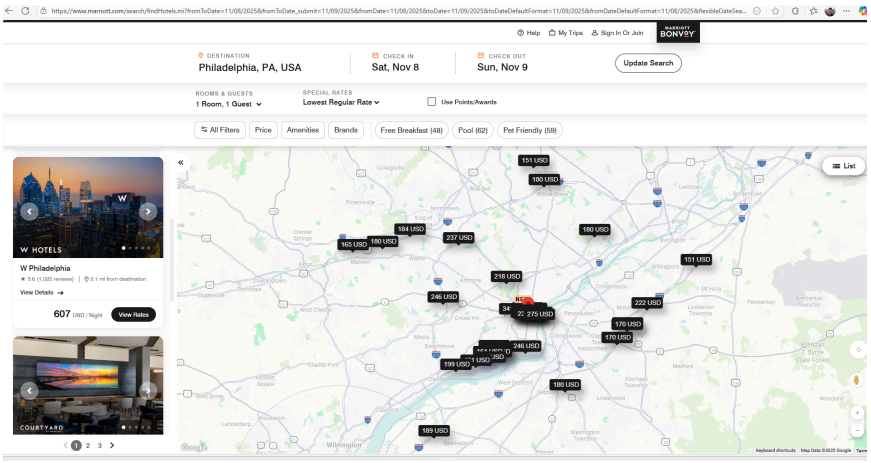


Figure 2-3: Marriott.com search results page with map

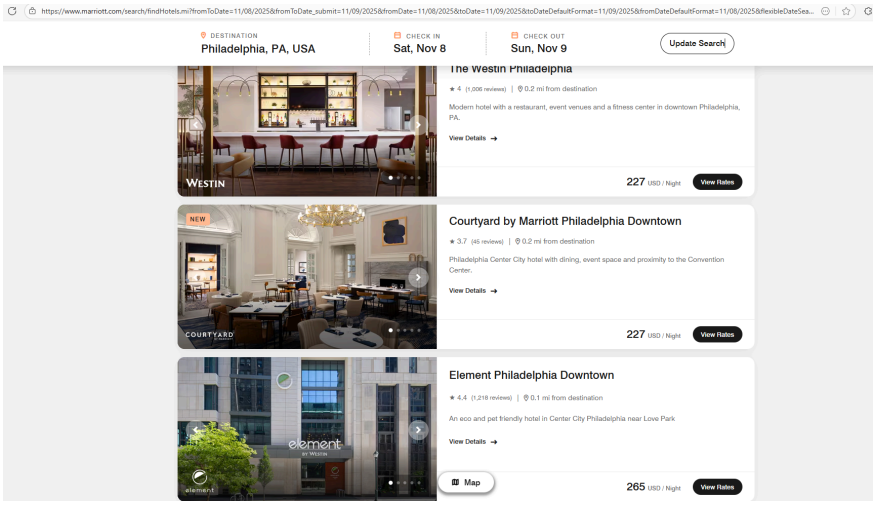


Figure 2-4: Marriott.com search results page with no map

The **account section** has a sub-navigation menu aiding orientation, and the 'My Trips' page provides an intuitive modal to search or browse bookings.

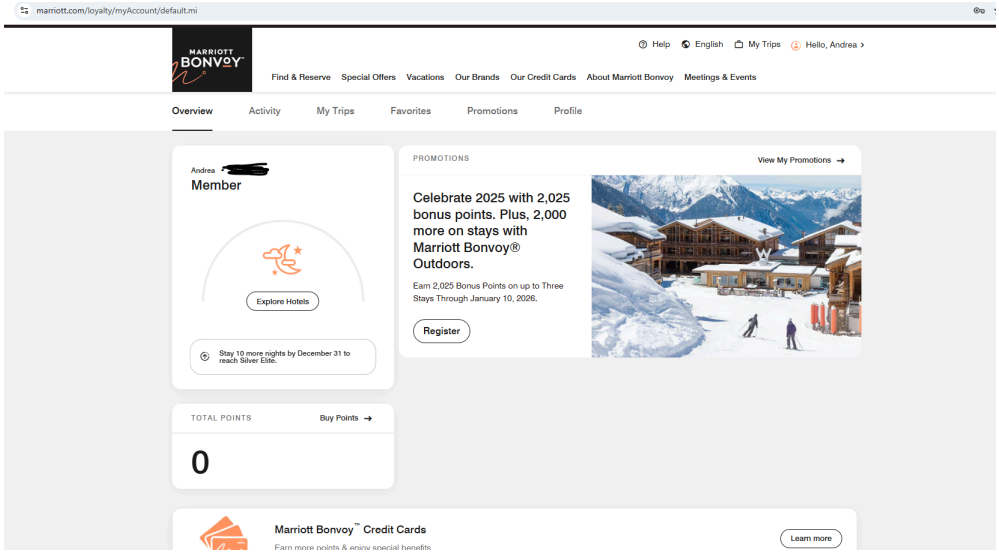


Figure 2-5: Marriott.com user account overview page

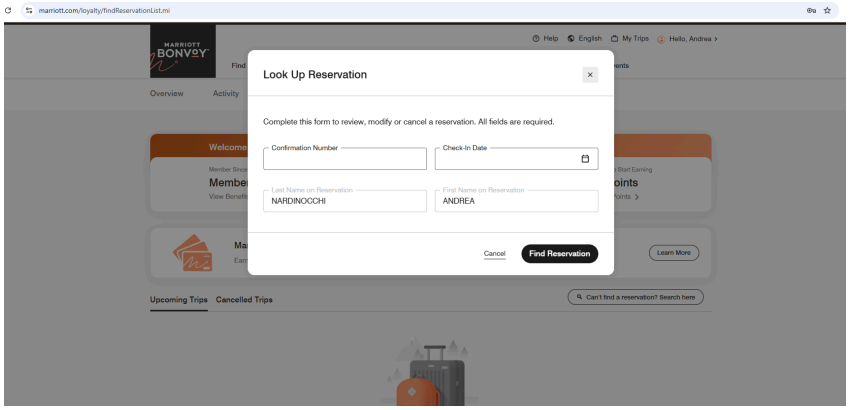


Figure 2-6: Marriott.com user account 'My Trips' page

2.3.2 IHG Hotels & Resorts

IHG (https://www.ihg.com) is functional but less fluid. **Language selection** requires first choosing a country, then a language. The **sticky header**'s 'Book Now' button jumps users to the top of the page, interrupting browsing.

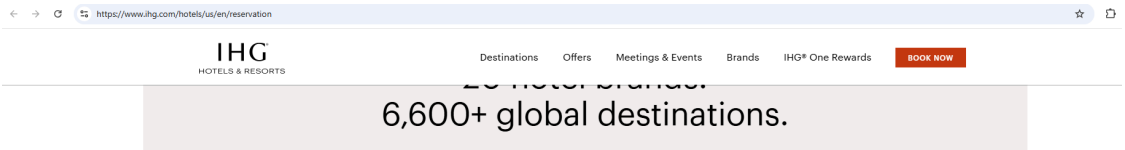


Figure 2-7: IHG sticky header with ‘Book Now’ CTA

The **booking form** lacks flexible dates. Hotel cards on the **search results** display amenities but no map view, and the header and form are not sticky.

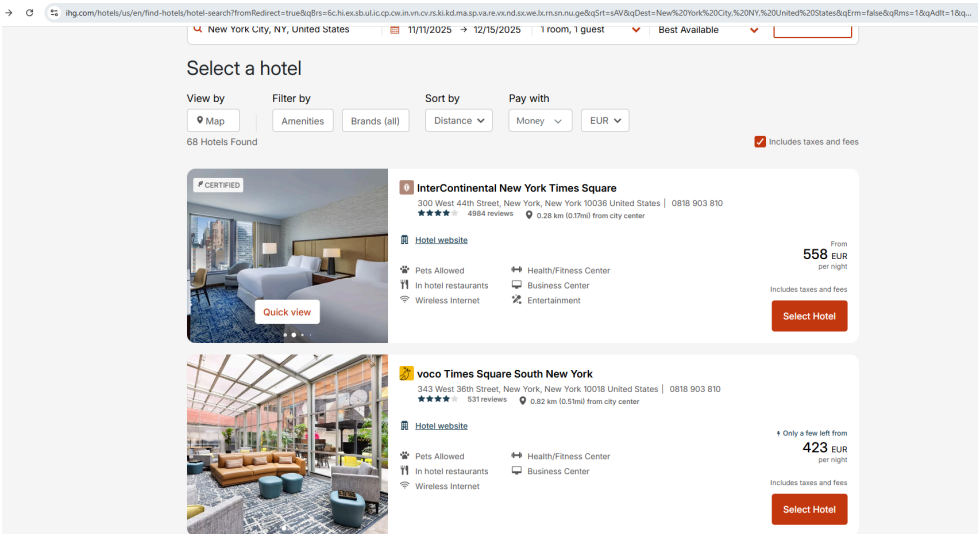


Figure 2-8: IHG search results page

The **account section** uses a subnav, but the design is less engaging than Marriott’s.

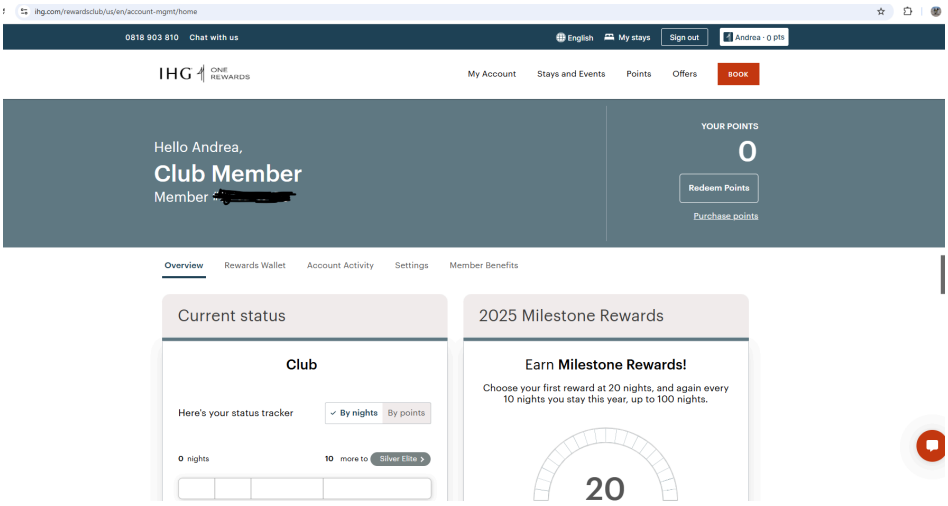


Figure 2-9: IHG user account overview page

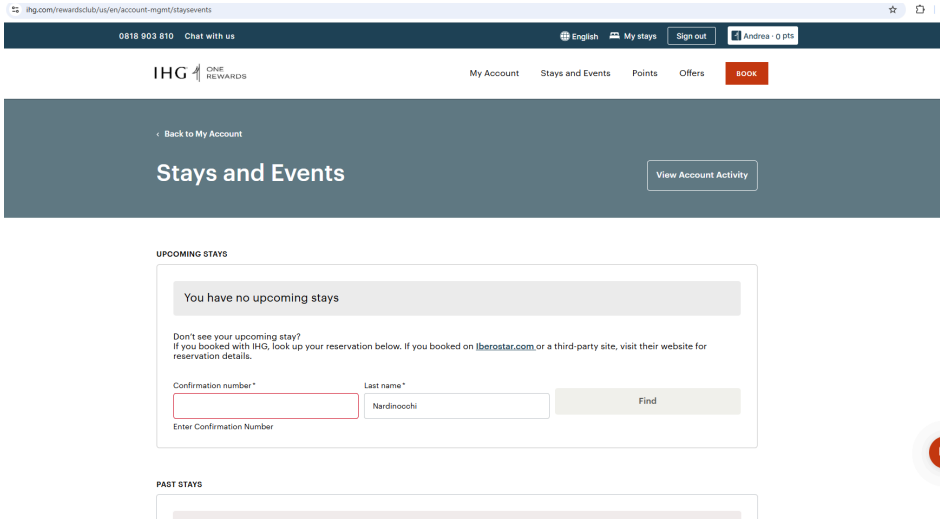


Figure 2-10: IHG user account overview page

2.3.3 Hilton

Hilton (https://www.hilton.com) uses a scrolling dropdown for **language**. **Sticky header** and **booking CTA** behave like Marriott's. **Booking forms** lack flexible dates.

Figure 2-11: Hilton search results page with map

Search results include hotel cards with a toggleable map, but the header and form are not sticky.

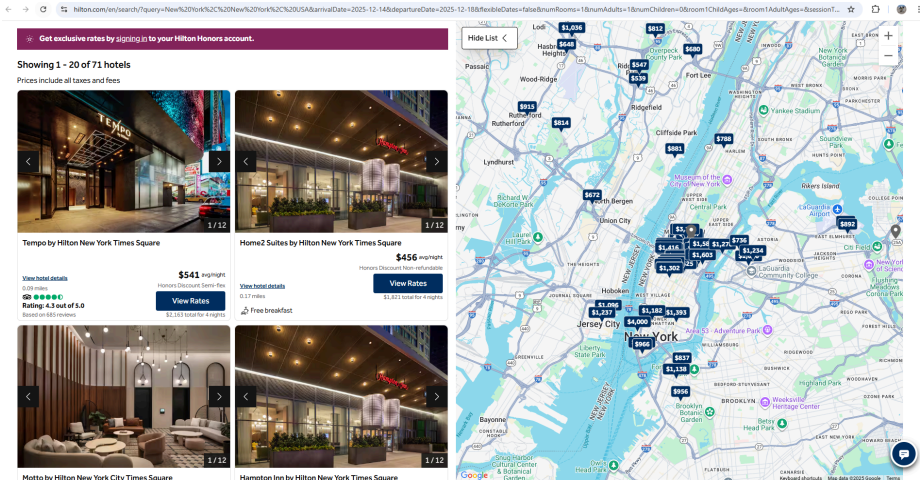


Figure 2-12: Hilton search results page with list

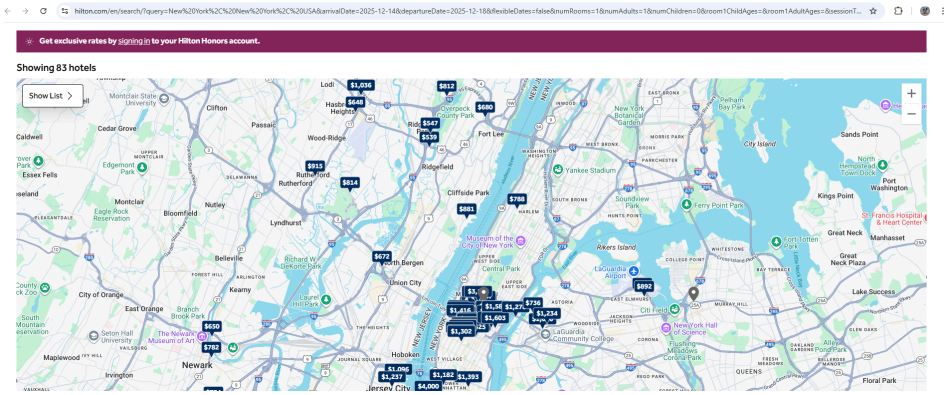


Figure 2-13: Hilton search results page with list only

The **account page** lacks sub-navigation, and empty accounts provide no guidance.

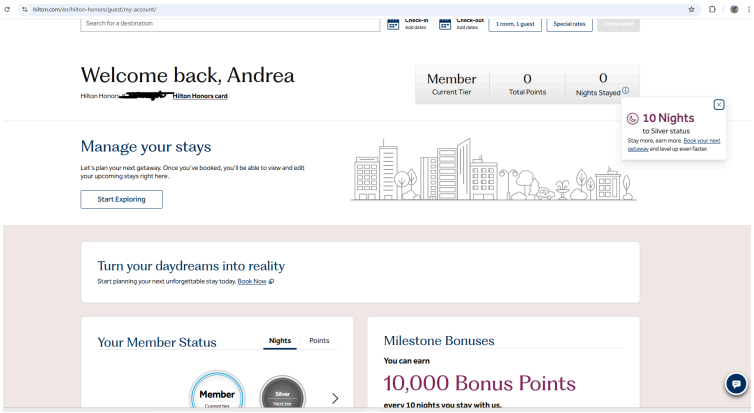


Figure 2-14: Hilton user account overview page

Conclusions and GuestEase Implications

Marriott excels with flexible booking, sticky navigation, and consistent design, while Hilton offers maps but lacks flexible dates. IHG falls behind with multi-step language selection.

My goal with GuestEase was to combine persistent navigation, clear booking CTAs, and visible amenities, prioritizing simplicity and mobile-friendly design.

3 Modelling & Design

3.1. User types and user stories

I came to the point in which personas and user stories for my small-scale hospitality web app needed to be created.

3.1.1 The GuestEase Guest Persona

Alex – Guest (35, Leisure/Family/Work Traveler)
Alex books stays for leisure, family holidays, or short-term remote work. He values easy booking, clear room info, and confirmation notifications.

User Stories:

- **Browse & Compare Rooms:** 'As a guest, I want to view room options with photos and amenities so I can choose the best stay.'
- **Quick Online Booking:** 'As a guest, I want to book quickly online to secure my stay without hassle.'
- **Manage Booking and receive notifications:** 'As a guest, I want to modify or cancel my booking if plans change.'

- **Write a Review:** 'As a guest, I want to leave feedback to share my experience with others.'

AI was used to create the above paragraph, but the output was edited by the author (Microsoft Copilot, 2026; see Appendix A, AI-10).

3.1.2 The GuestEase Administrator Persona

Lucas – GuestEase Administrator / Owner (57, Boutique Hotel Owner)
Lucas manages bookings, guest communications, and property details. He needs simple tools to update availability, rates, and monitor bookings.

User stories:

- **Update Availability & Pricing** - 'As the property owner, I want to quickly update room availability and pricing so the website always reflects reality.'
- **Manage Reservations & User Profiles** - 'As the property owner, I want to be able to easily manage reservations and do the same for user profiles, if a user is unable to.'
- **Receive Booking Notifications** - 'As the property owner, I want to receive notifications of new bookings so I can manage guest arrivals efficiently.'
- **Access Review Guest Feedback** - 'As the property owner, I want to access guest feedback easily so I can improve services.'

AI was used to create the above paragraph, but the output was edited by the author (Microsoft Copilot, 2026; see Appendix A, AI-11).

3.1.3 Use case diagram

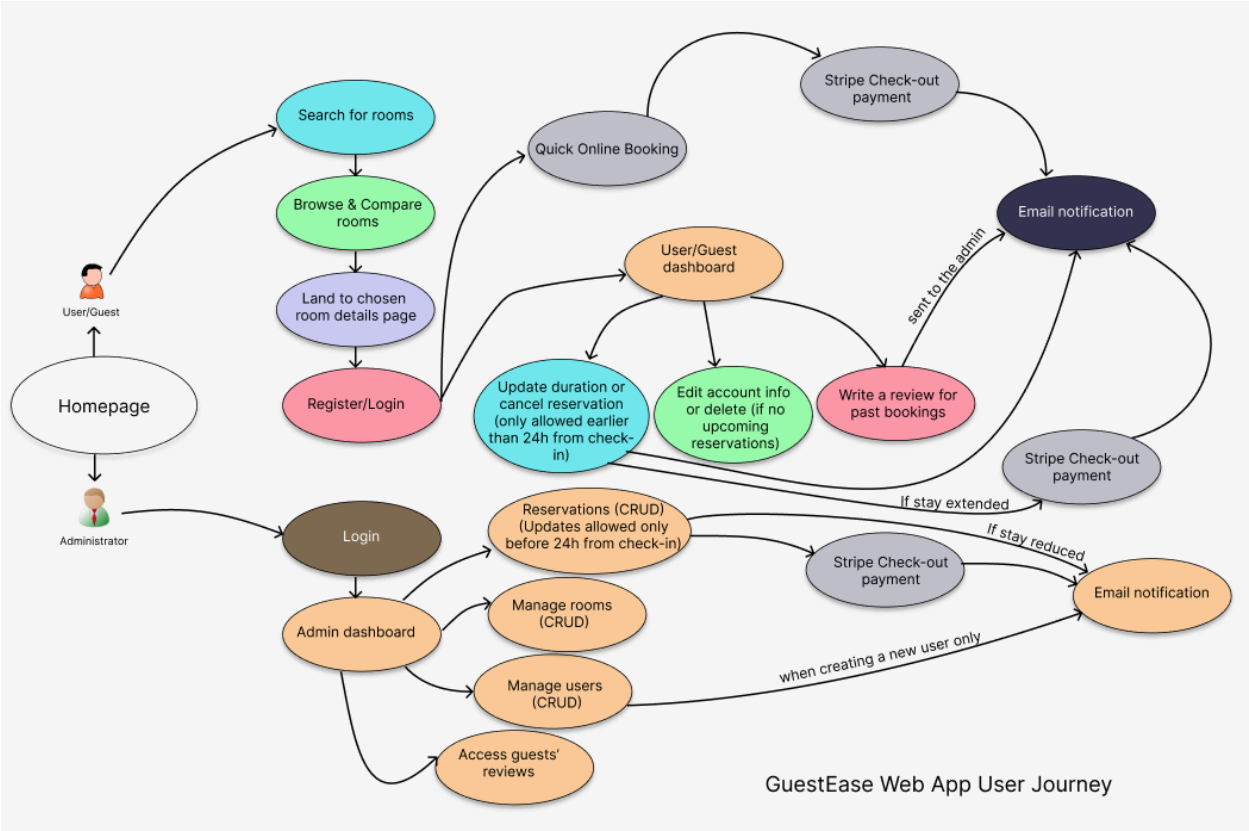


Figure 3-1: Diagram of use case [Figma]

3.2. Data Modelling

3.2.1 Model: Entities and Relationships

The **administrator** can create, update, or delete rooms.
The **administrator** can create, update, or delete users.
The **administrator** can create, query, update, delete bookings, and query reviews.
The **administrator** can receive email notifications about bookings, and reviews.

A **guest** can create, modify, and cancel bookings.
A **guest** cannot modify or cancel bookings within 24 hours before check-in.
A **guest** can initiate and complete Stripe payments for bookings.
A **guest** has their saved payment method automatically replaced whenever a new one is added.

A **guest** has their saved payment method automatically charged within 24 hours before check-in.
A **guest** can modify and delete their account.
A **guest** can receive email notifications about bookings.
A **guest** can write reviews following a complete stay.

3.2.2 Core Relationships

User(Guest/Admin) 1..1 creates 0..* Bookings

- A user can create multiple bookings. Each booking belongs to exactly one user.

User(Guest) 1..1 saves 0..1 PaymentMethod (user_payment_methods)

- A guest can only have a default payment method. If a new payment method is added, the previous one is replaced and removed.

Room 1..1 has 0..* Bookings

- A room can have many bookings over time, and each booking is for exactly one room.

Booking 1..1 generates 0..1 Review

- Each completed booking may result in one review, and each review is linked to a specific booking

Booking 1..1 has 1 Stripe PaymentIntent (external)

- Payments are handled directly by Stripe. The system stores only the payment_intent_id, the intended amount, and a charged flag indicating success.

User(Guest) 1..1 writes 0..* Reviews

- A guest can write multiple reviews (one per stay), and each review is written by exactly one guest.

Booking 1..1 triggers 0..* Email Notifications (Conceptual Only)

- A booking can generate one or more notifications (e.g., confirmation, update, cancellation), and each notification belongs to a specific booking.

User(Guest/Administrator) 1..1 receives 0..* Email Notifications (Conceptual Only)

- A user can receive multiple email notifications (e.g., booking confirmation, updates and cancellations), and each notification is linked to exactly one guest.

Room 1..1 receives 0..* Reviews

- A room can receive many reviews over time, and each review corresponds to one specific room.

Note:
Regarding the Admin actions such as creating rooms or receiving notifications, they are permissions handled at the application level, not stored as foreign-key relationships.

Email notifications are not stored in the database. They are generated by backend logic and sent via the external Resend API. Therefore, they appear in the conceptual model but not in the physical ERD.

3.2.3 Logical/Physical Data Model (ERD)

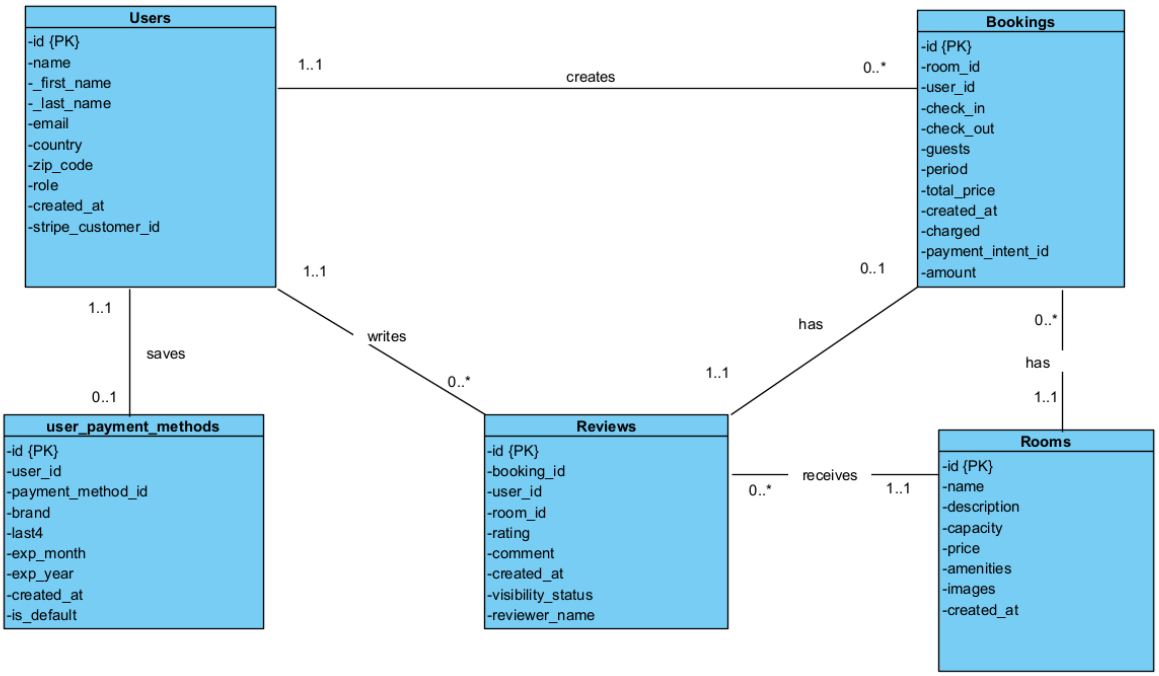


Figure 3-2: Diagram of use case

User(id, email, role, created_at, first_name, last_name, country, zip_code, stripe_customer_id)

- Primary key: id
- role ∈ {guest, admin}

Room(id, name, description, capacity, price, created_at, amenities, images)

- Primary key: id

Booking(id, user_id, room_id, check_in, check_out, guests, total_price, created_at, charged, payment_intent_id, amount)

- Primary key: id
- Foreign key: user_id references User(id)
- Foreign key: room_id references Room(id)

PaymentMethod(id, user_id, payment_method_id, brand, last4, exp_month, exp_year, is_default, created_at)

- Primary key: id
- Foreign key: user_id references User(id)

Review(id, booking_id, user_id, room_id, rating, comment, created_at, visibility_status, reviewer_name)

- Primary key: id
- Foreign key: booking_id references Booking(id)
- Foreign key: user_id references User(id)
- Foreign key: room_id references Room(id)

3.3. Wireframes

3.3.1 Homepage

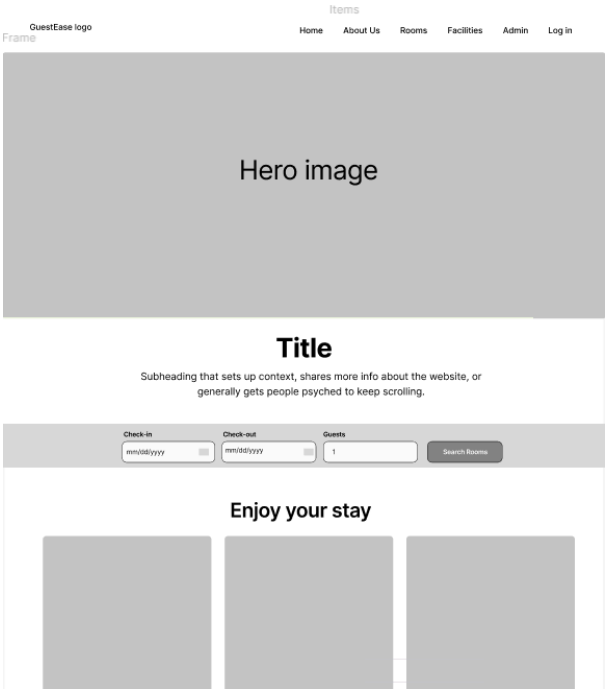


Figure 3-3: Homepage wireframe

3.3.2 Search Results page

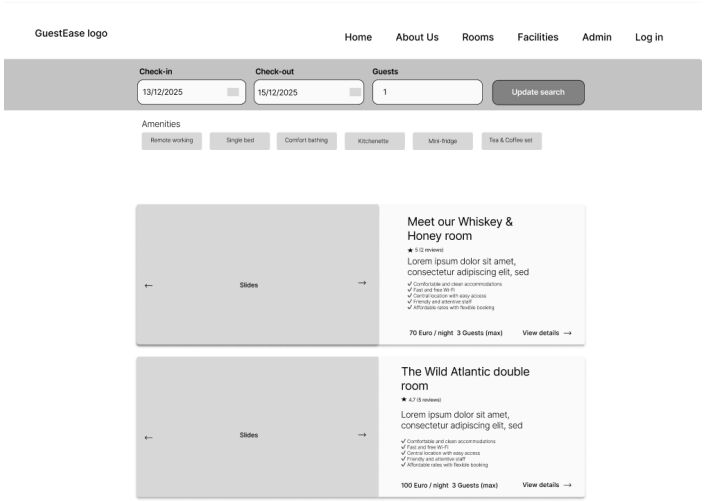


Figure 3-4: Search results page wireframe

3.3.3 Room details

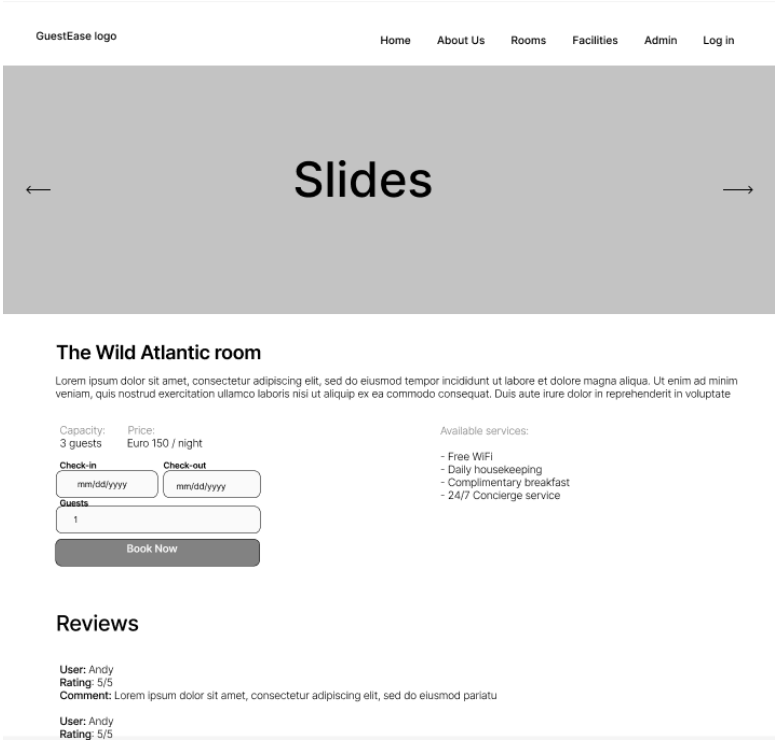


Figure 3-5: Room details wireframe

3.3.4 Login page

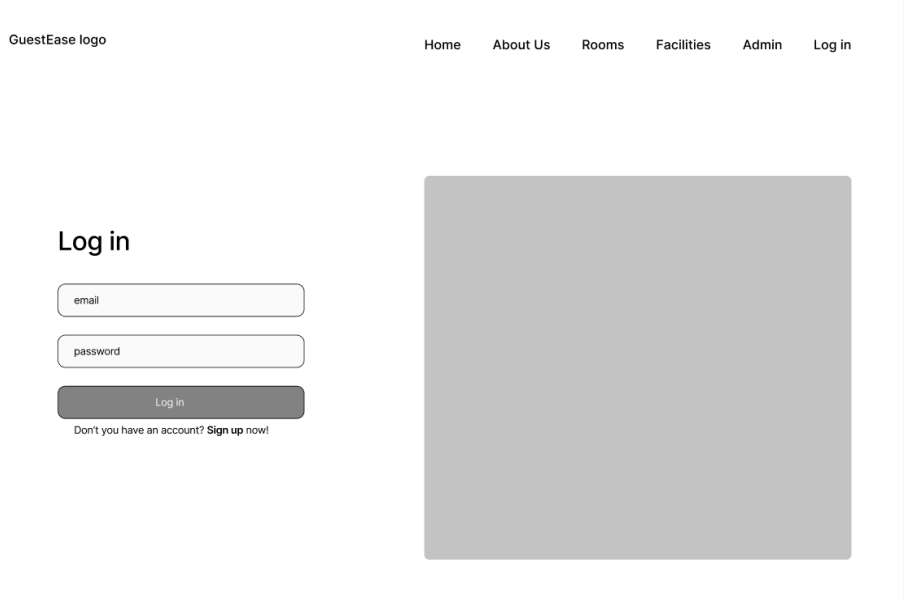


Figure 3-6: Log in page wireframe

3.3.5 Sign up page

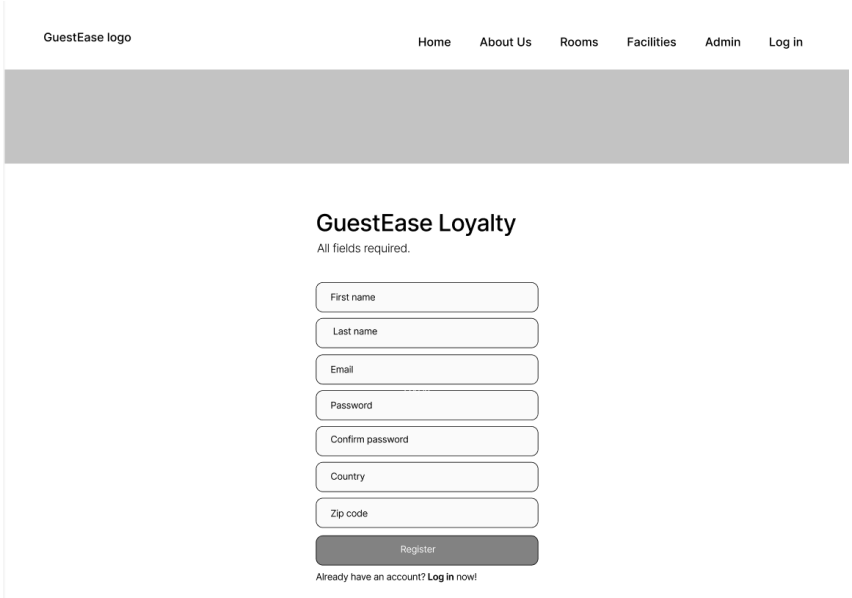


Figure 3-7: Sign up page wireframe

3.3.6 Account - My Trips page

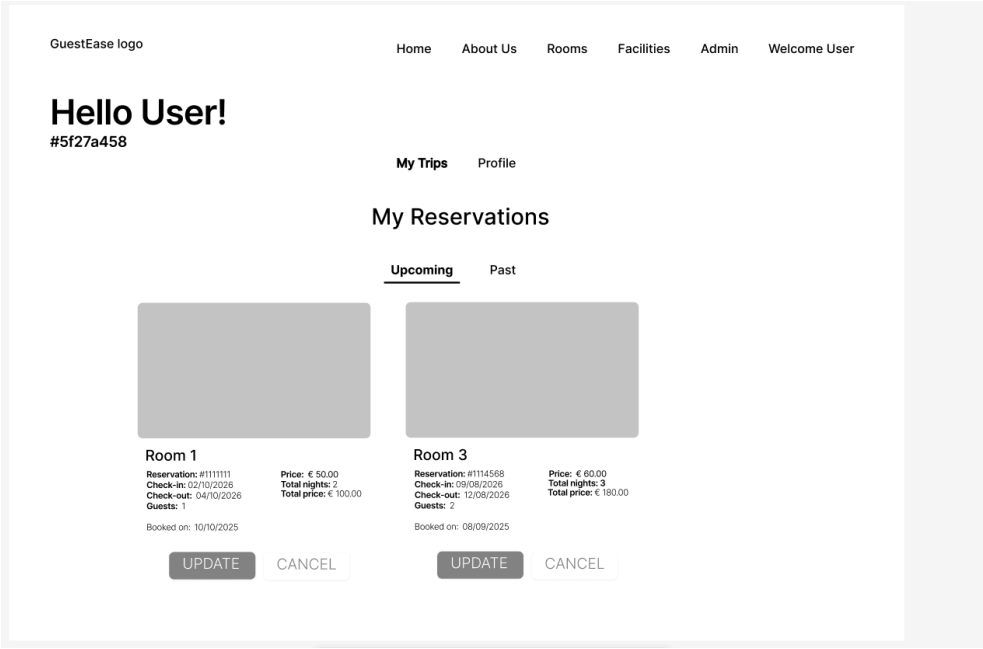


Figure 3-8: Account - My Trips page wireframe

3.3.7 Account - Profile page

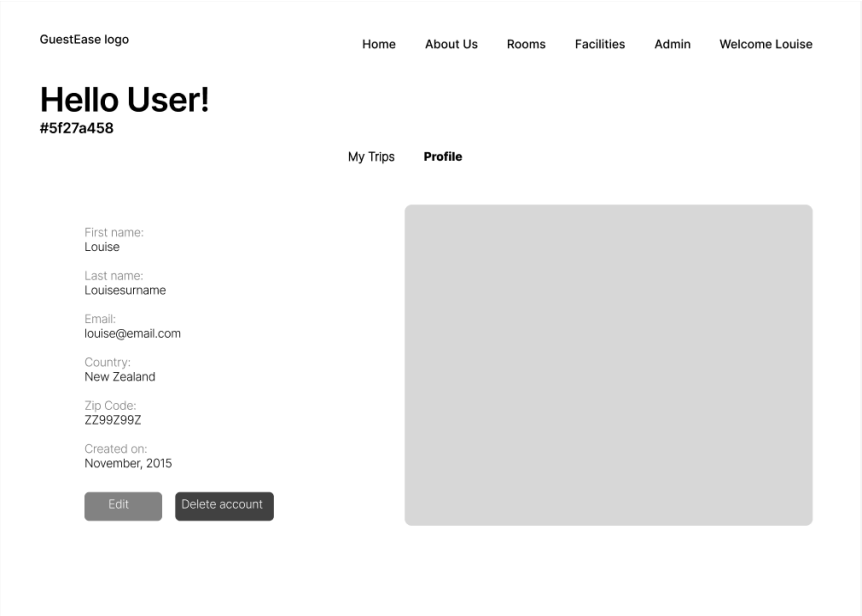


Figure 3-9: Account - Profile page wireframe

3.3.8 Account - User profile drawer

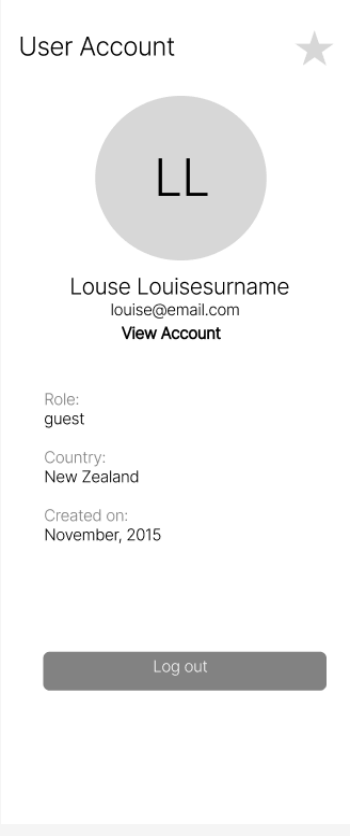


Figure 3-10: Account - User profile drawer wireframe

3.3.9 Account - Booking Review page

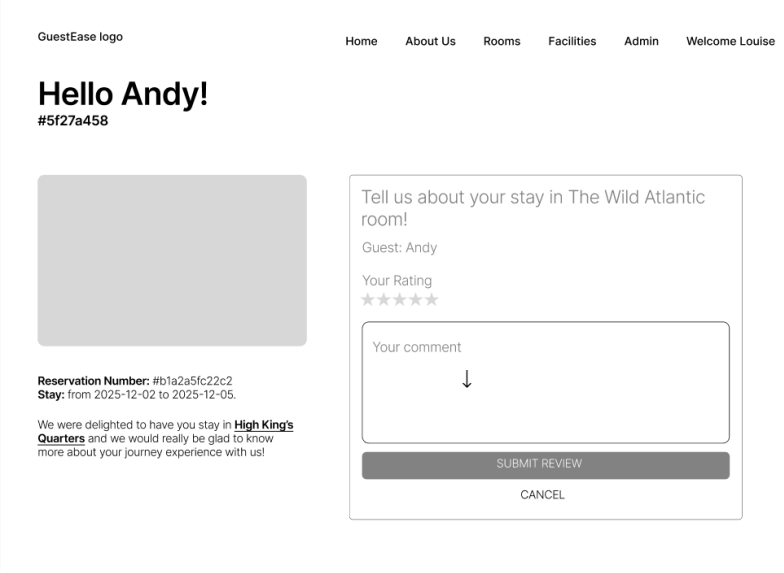


Figure 3-11: Account - Write a review page wireframe

3.3.10 Account - Room Details page payment widget

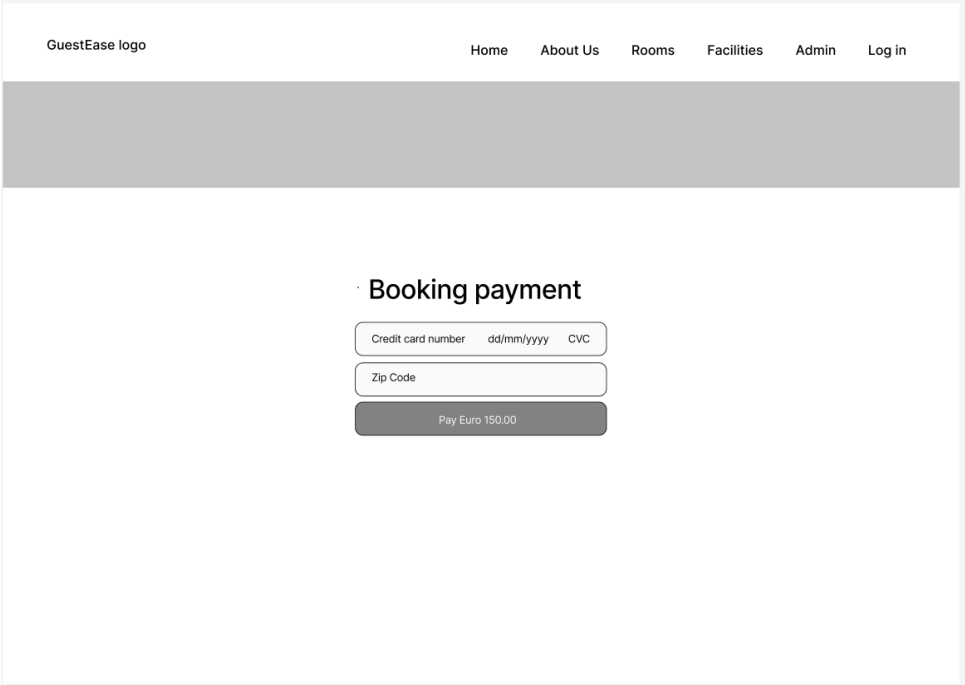


Figure 3-12: Account - Payment page

3.3.11 Account - Reservation confirmation page

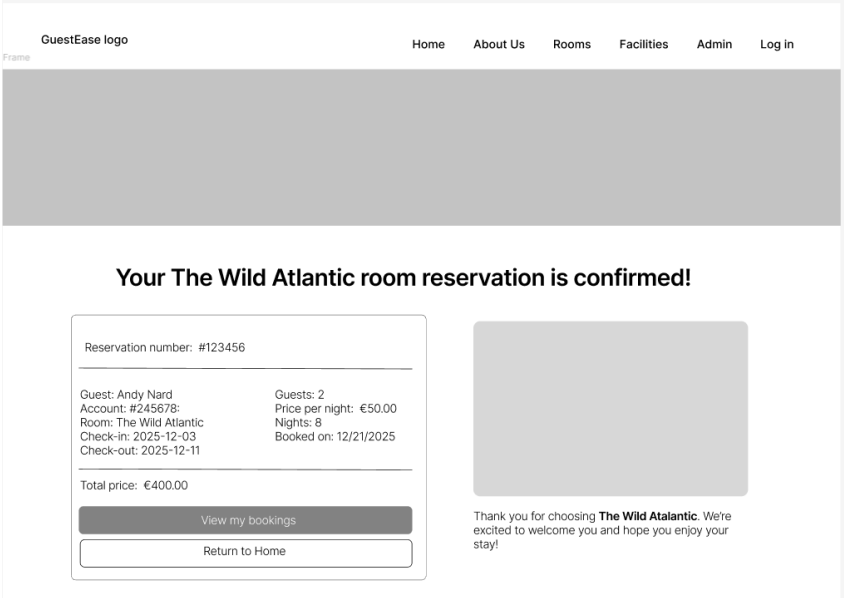


Figure 3-13: Account - Reservation confirmation page

3.3.12 Admin Dashboard

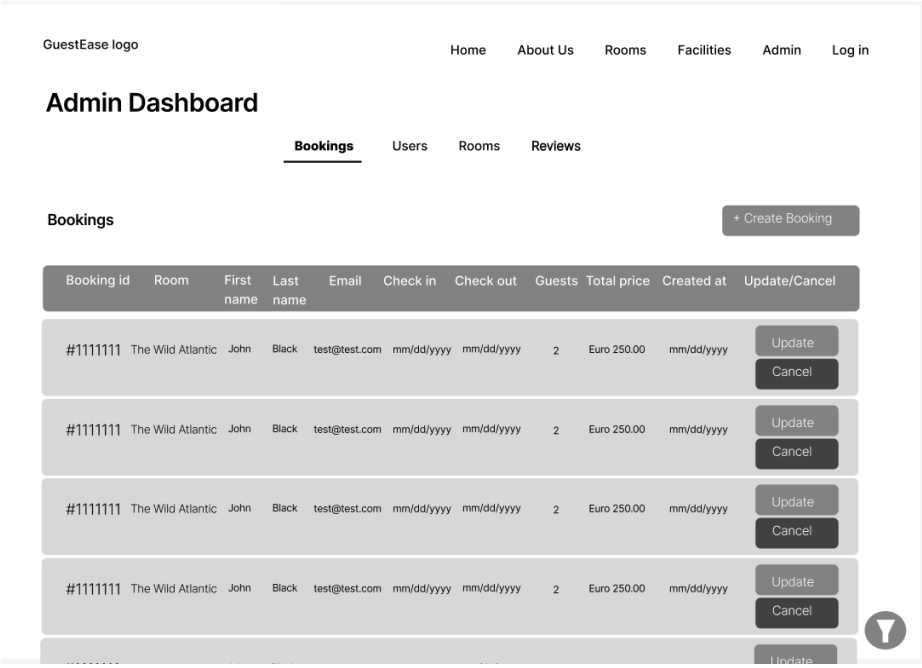


Figure 3-14: Admin Dashboard

3.3.13 Admin Dashboard Filters Drawer

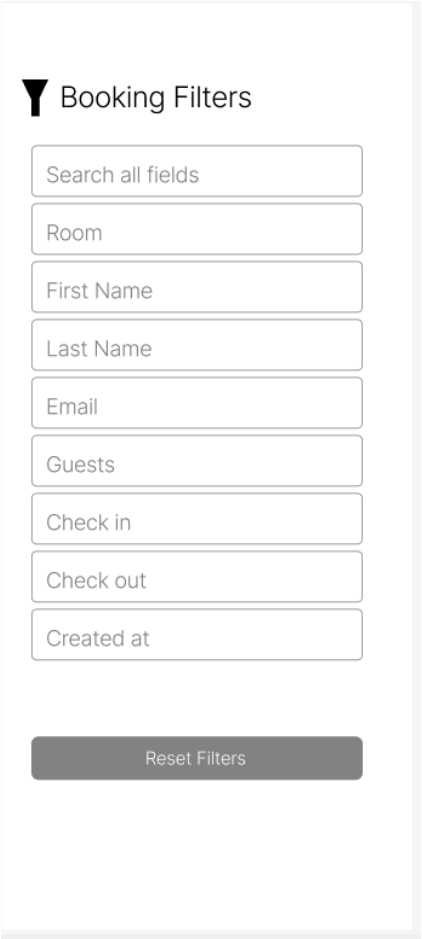


Figure 3-15: Example of an admin dashboard filter

3.3.14 Update Password page

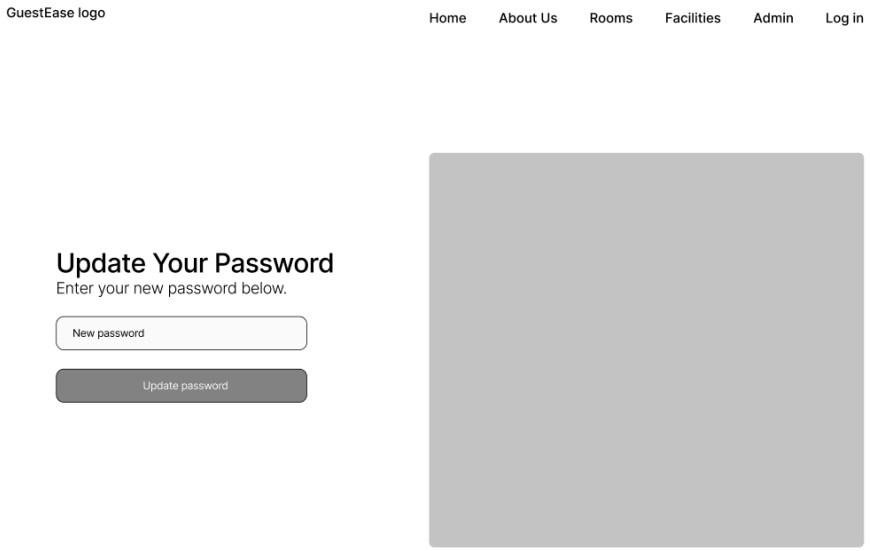


Figure 3-16: Update password page

3.3.15 About Us page

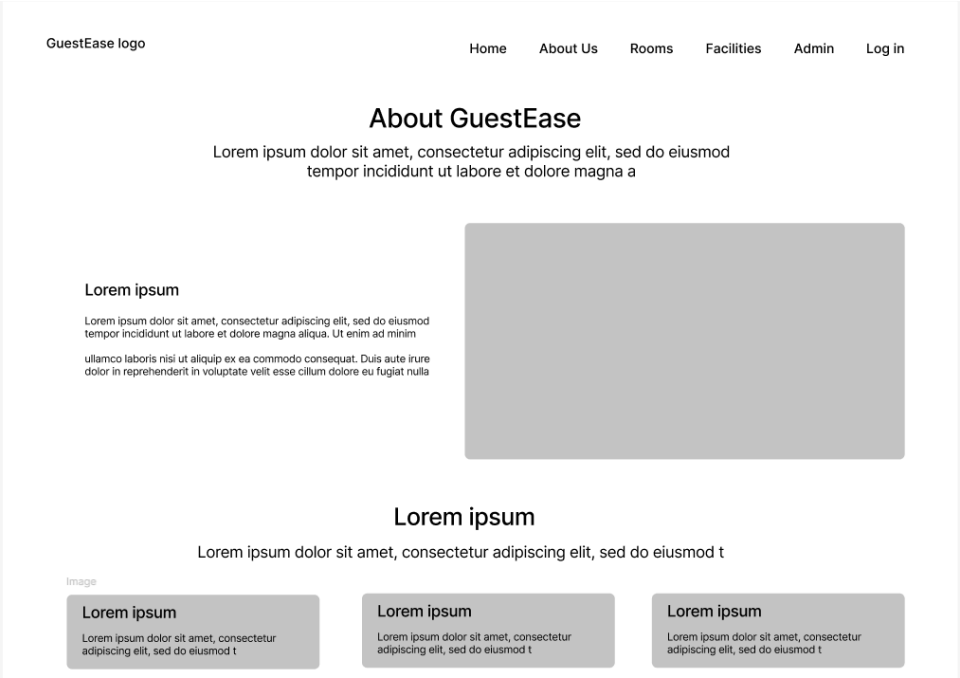


Figure 3-17: About Us page

3.3.16 Facilities page

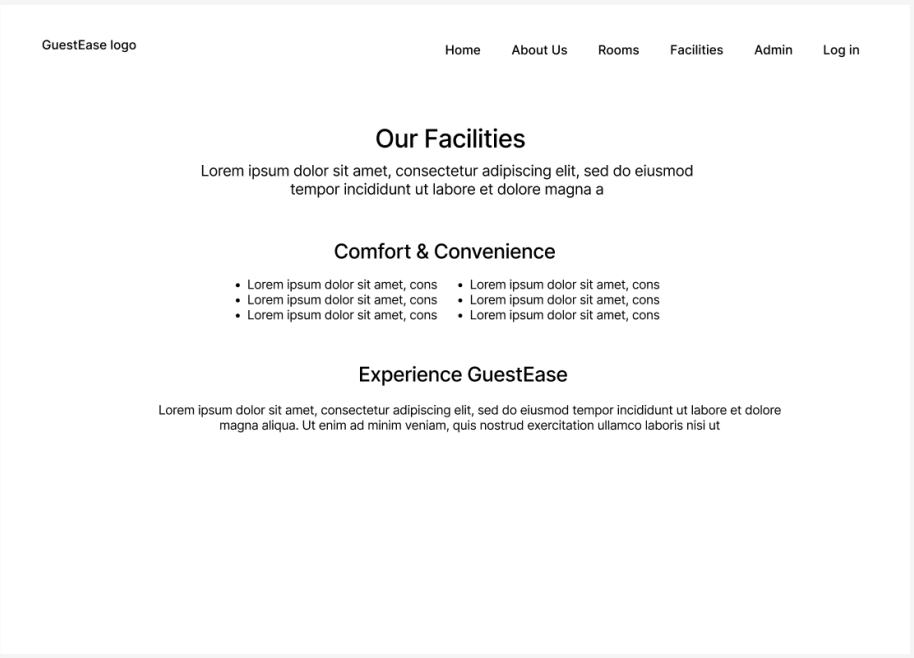


Figure 3-18: Facilities page

3.3.17 Our Rooms page

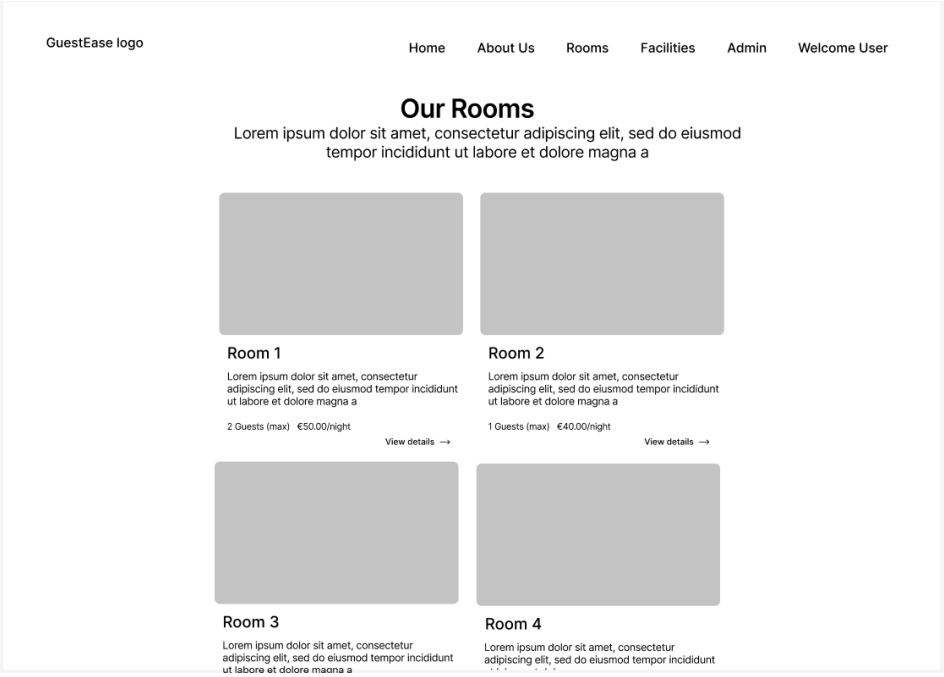


Figure 3-19: Our Rooms page

3.4. Final Design

3.4.1 Homepage

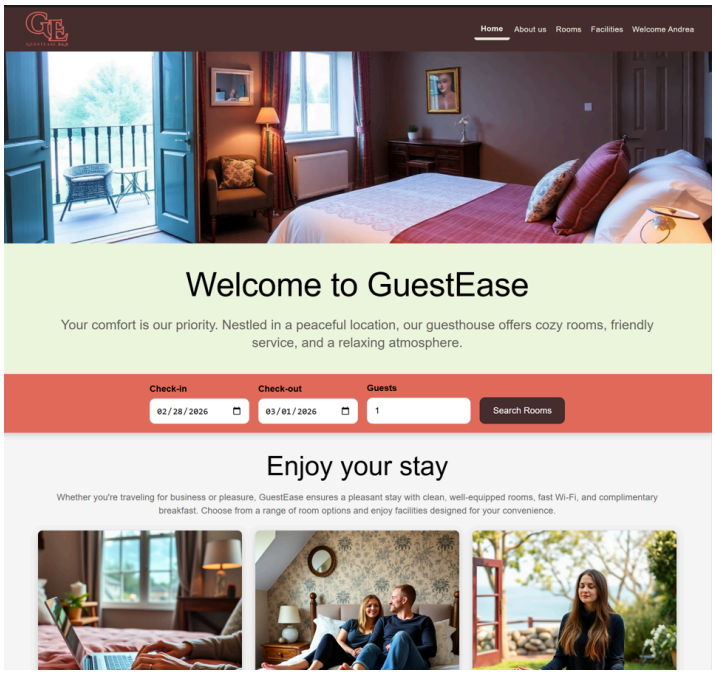


Figure 3-20: Homepage

3.4.2 Search Results page

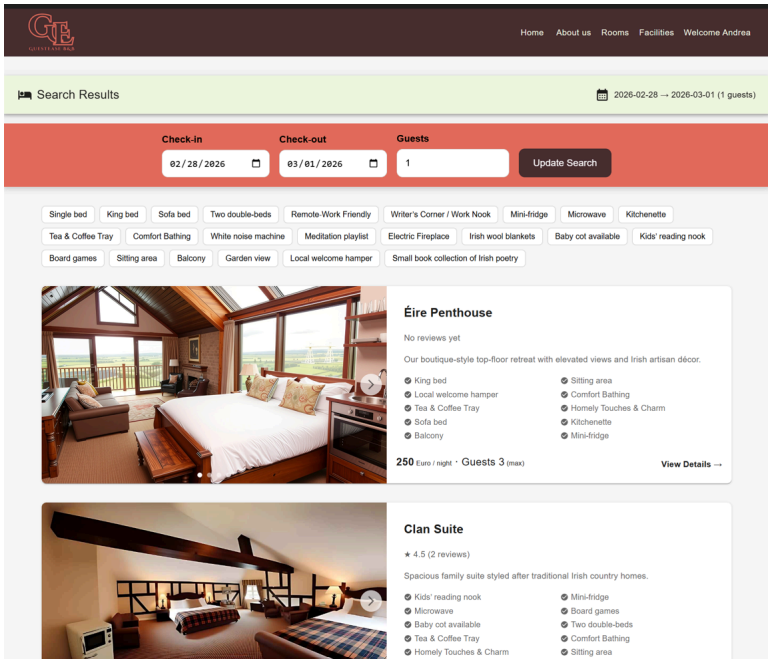


Figure 3-21: Search Results page

3.4.3 Room Details

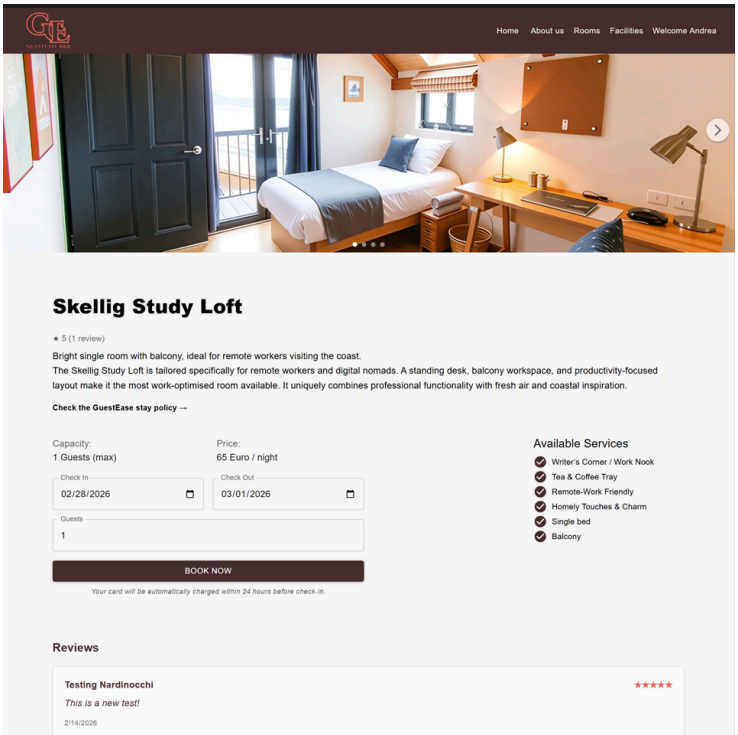


Figure 3-22: Room Details page

3.4.4 Login page

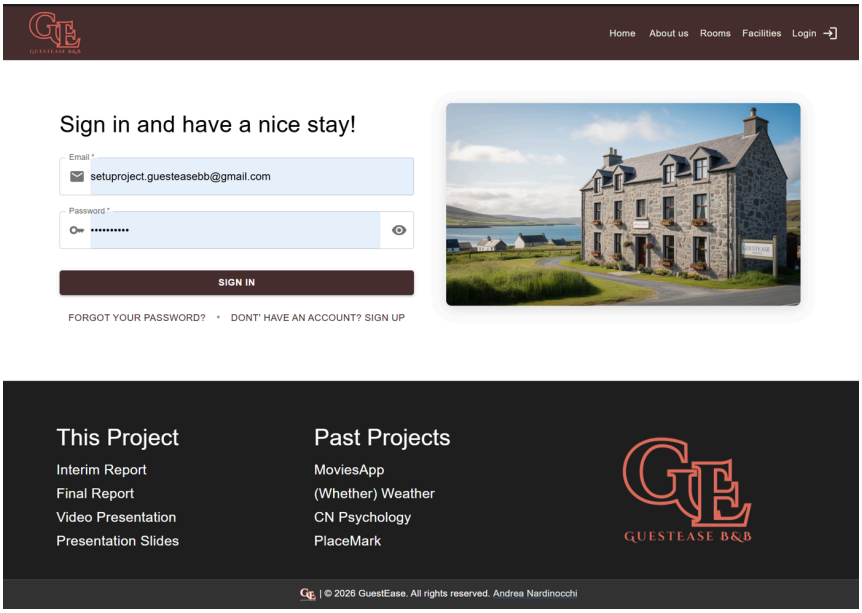


Figure 3-23: Login page

3.4.5 Sign up page

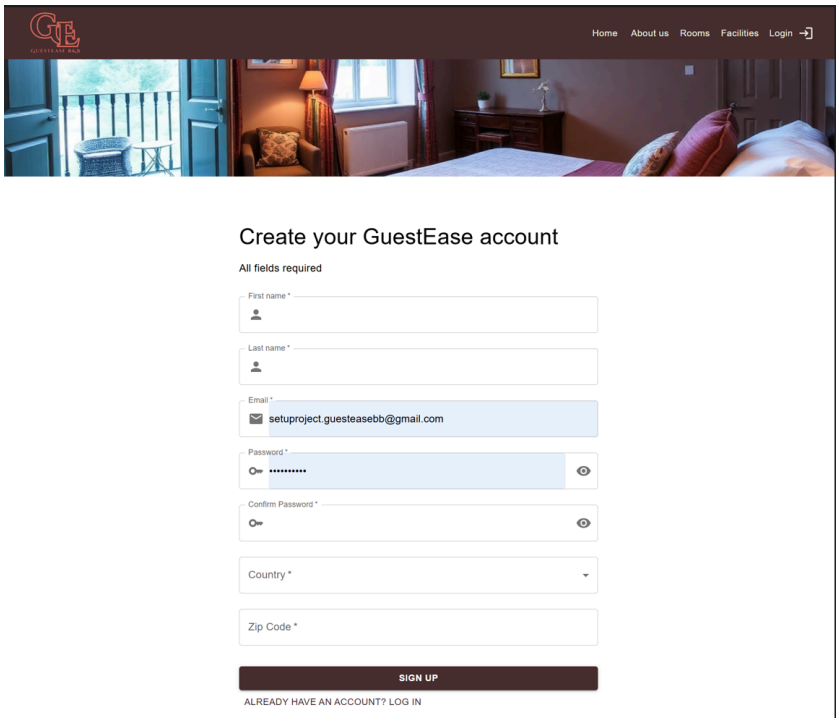


Figure 3-24: Sign up page

3.4.6 Account - My Trips page

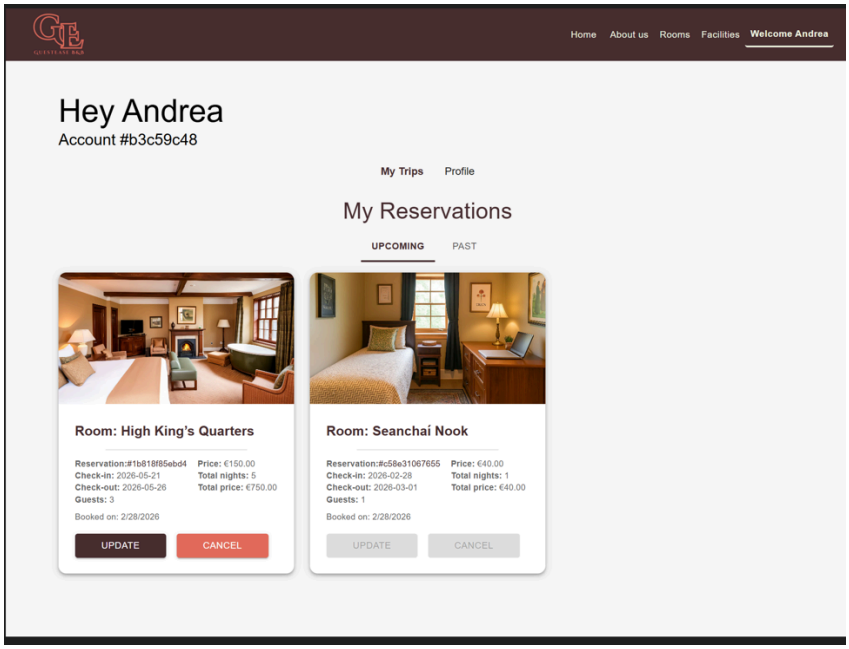


Figure 3-25: Account My Trips page

3.4.7 Account - Profile page

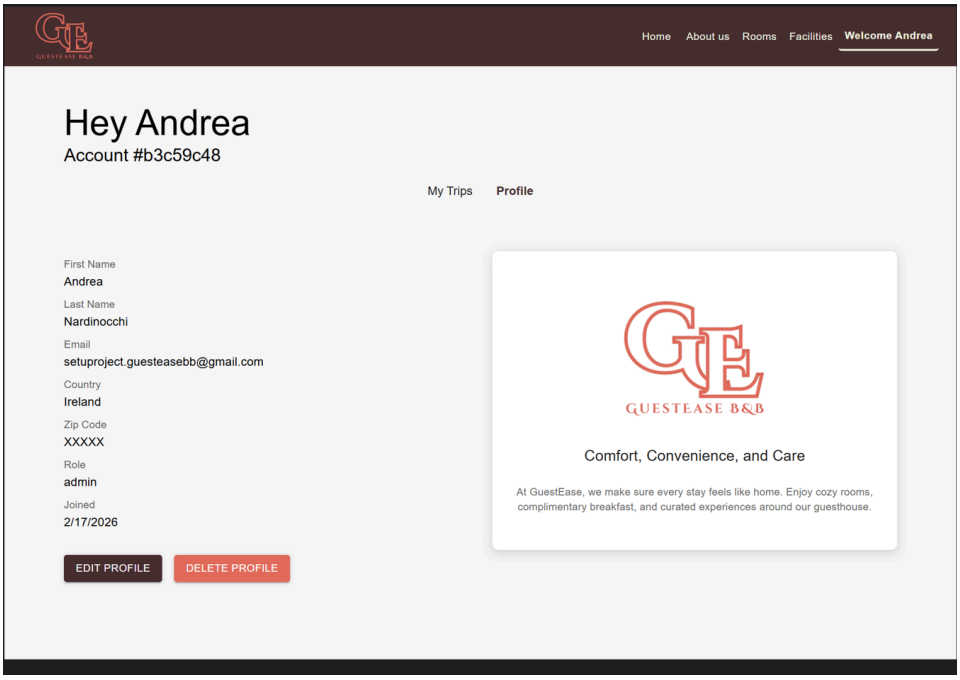


Figure 3-26: Profile page

3.4.8 Account - User Profile Drawer

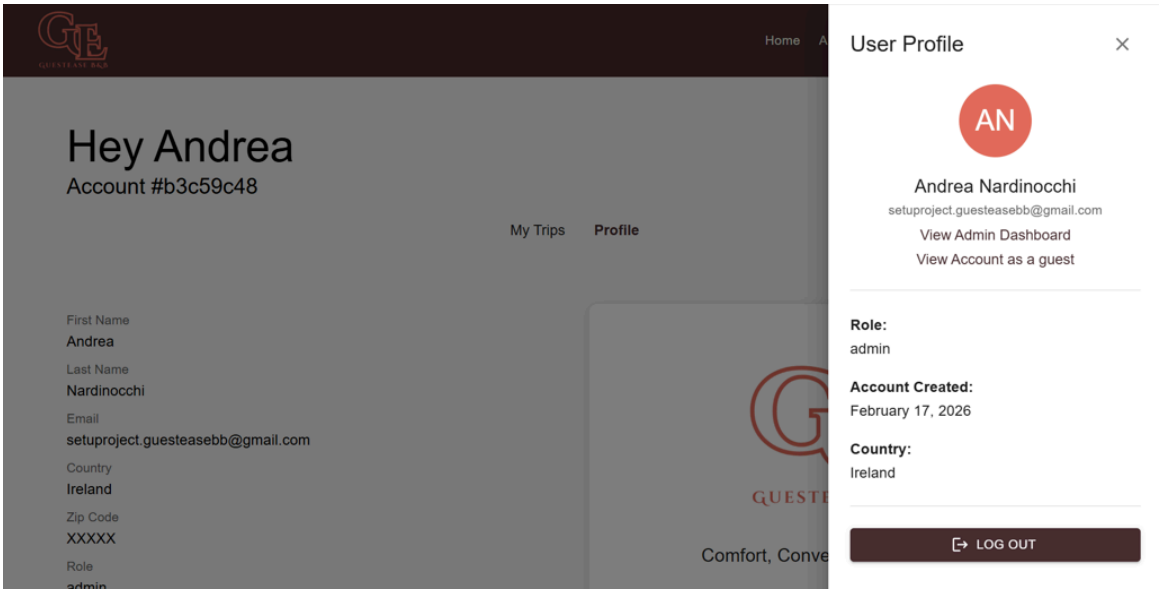


Figure 3-27: User Profile Drawer

3.4.9 Account - Booking Review page

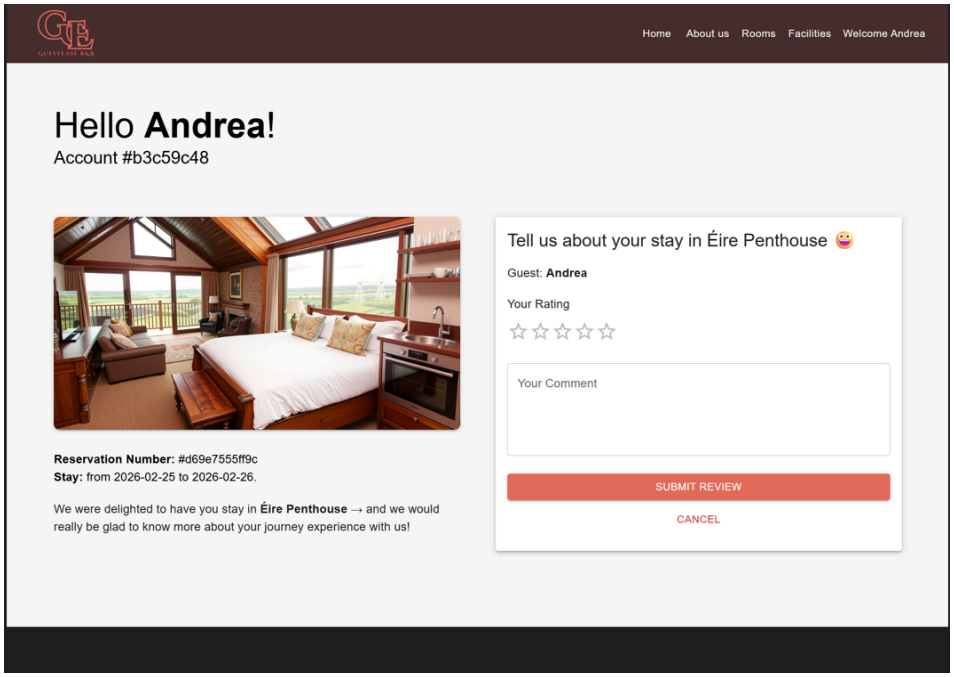


Figure 3-28: Booking Review page

3.4.10 Account - Stripe payment modal

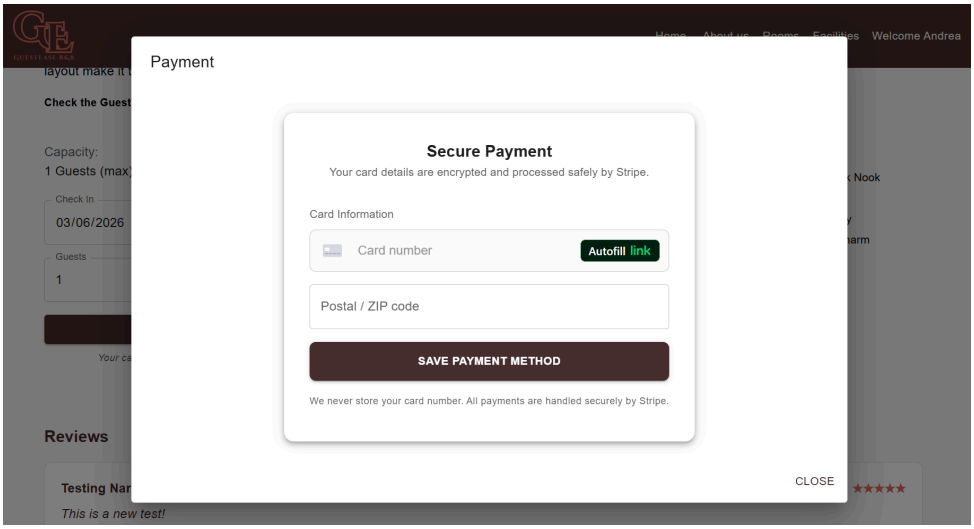


Figure 3-29: Stripe Payment Modal

3.4.11 Account - Reservation confirmation page

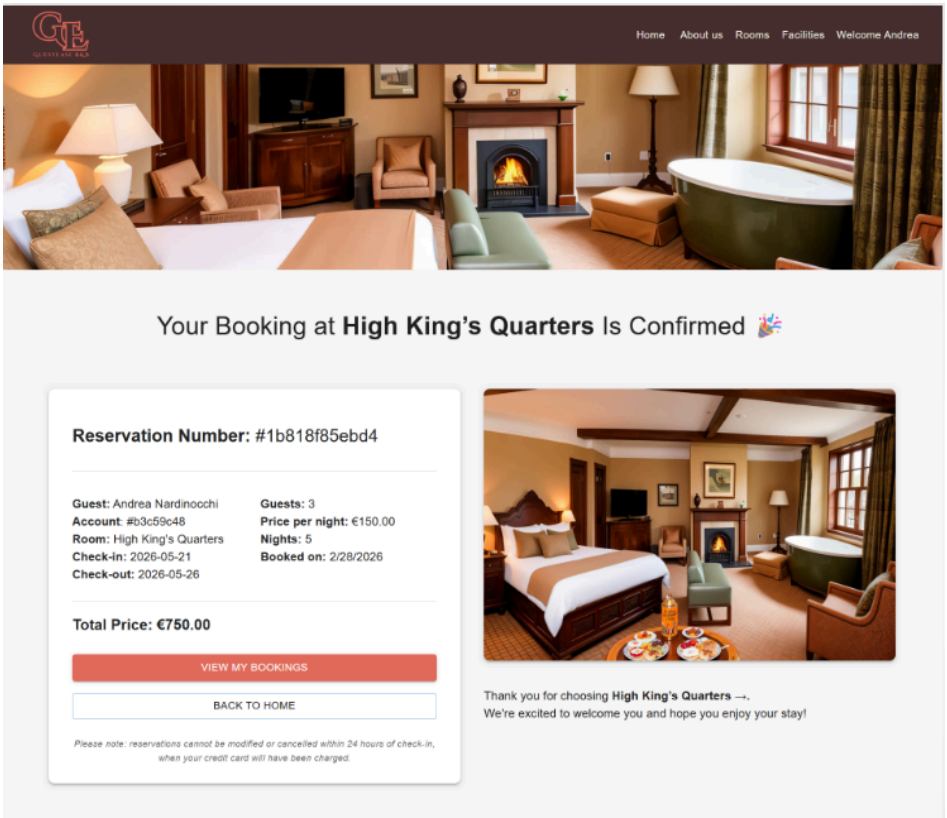


Figure 3-30: Reservation confirmation page

3.4.12 Admin Dashboard

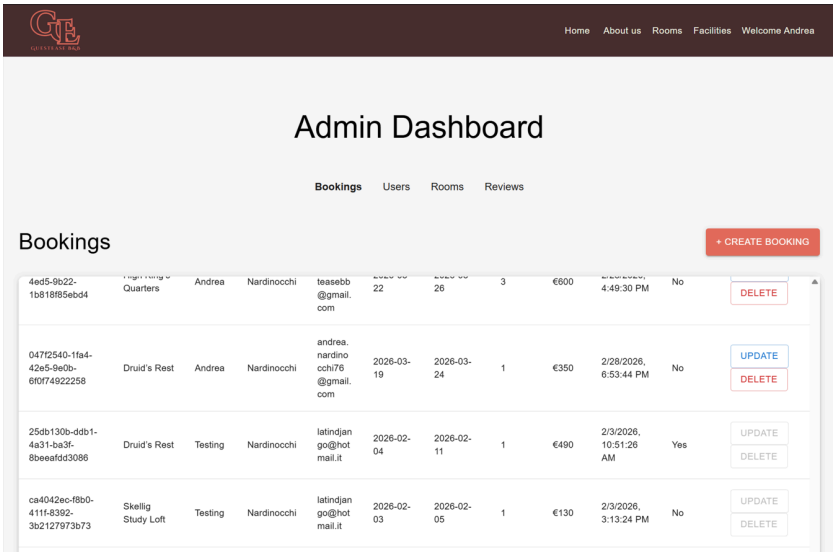


Figure 3-31: Overview of the Admin Dashboard

3.4.13 Admin Dashboard DrawerFilters

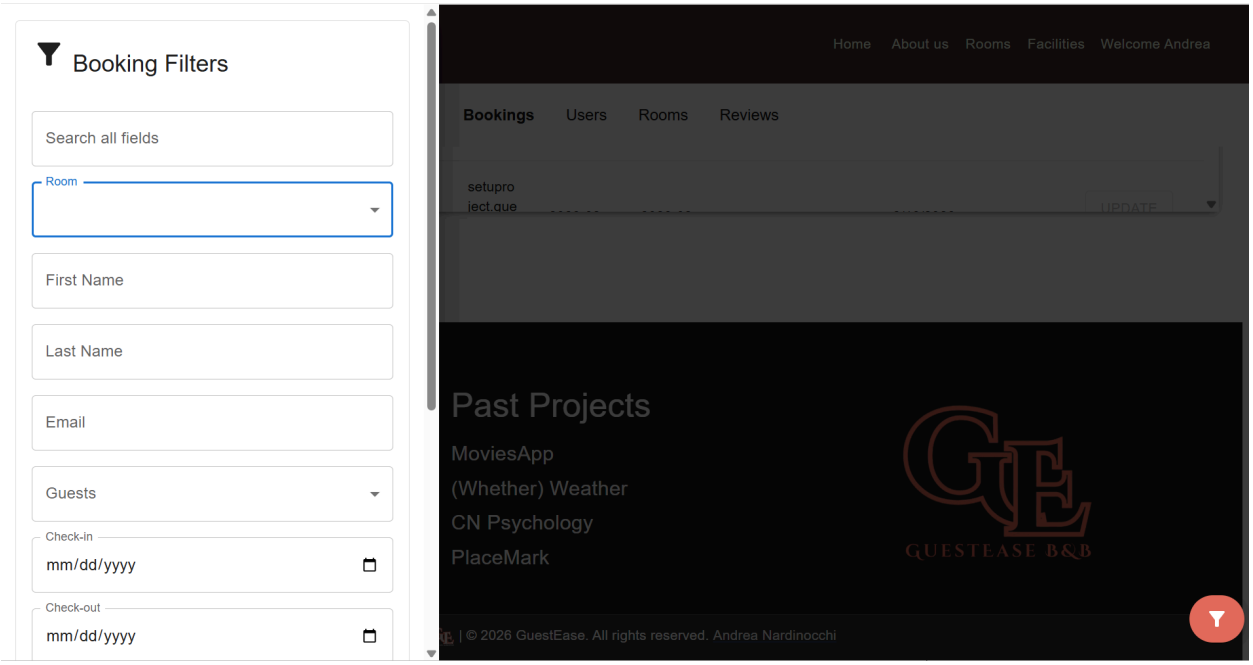


Figure 3-32: Filters drawer on the Admin Bookings page

3.4.14 Update password page

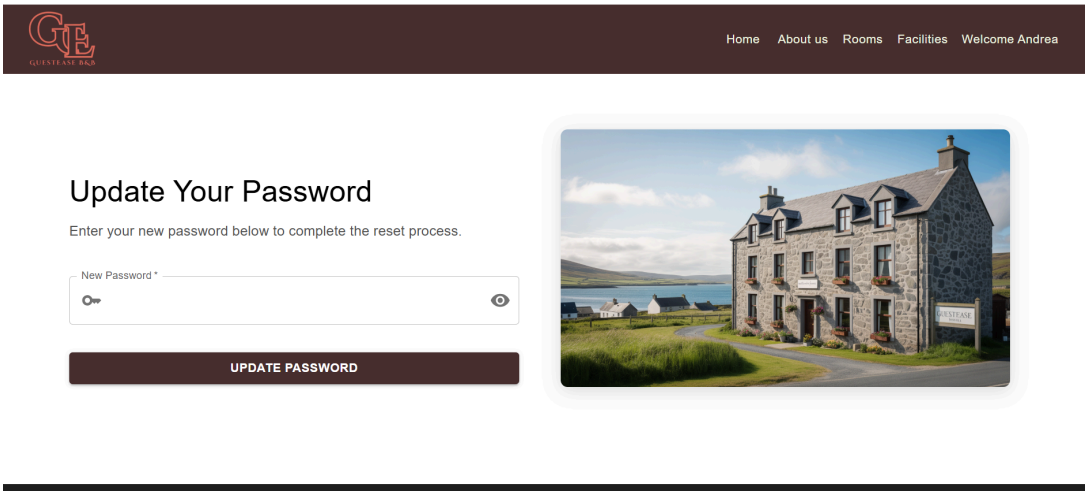


Figure 3-33: Update Password page

3.4.15 About us page

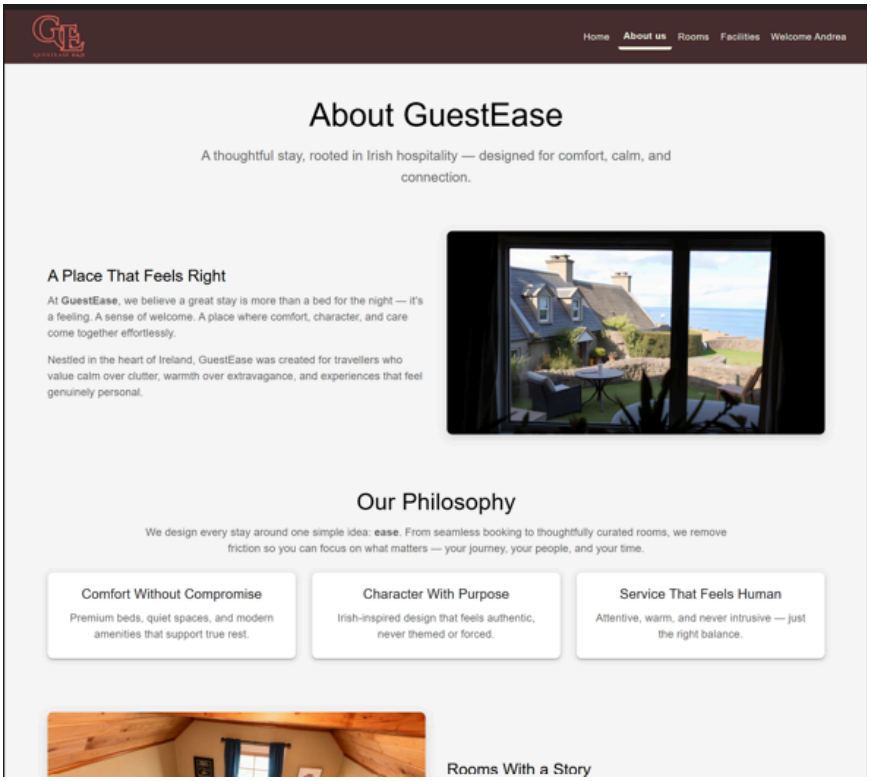


Figure 3-34: About us page

3.4.16 Facilities page

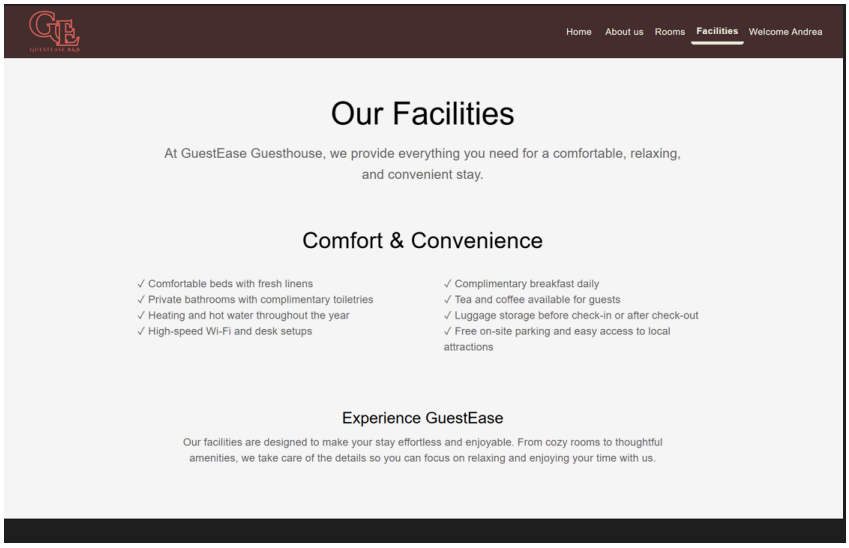


Figure 3-35: Facilities page

3.4.17 Our rooms page

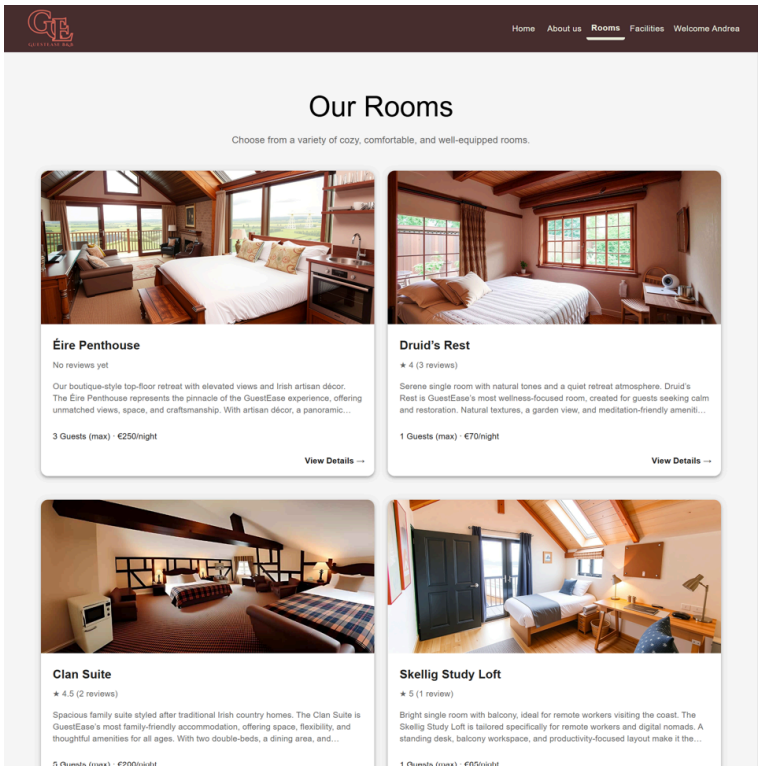


Figure 3-36: Our Rooms page

3.4.18 Example of a GuestEase email notification via Resend

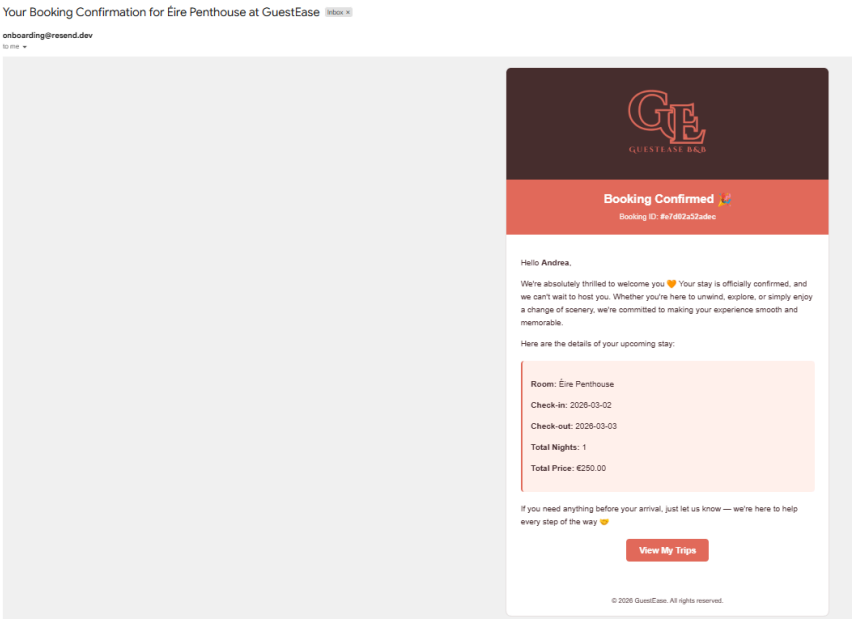


Figure 3-37: GuestEase email notification via Resend

4 System Overview

4.1. Technologies

GuestEase is built with a React and TypeScript frontend, a Node.js Express backend for secure server-side logic, and Supabase for authentication, assets storage, real-time data handling, and Edge functions for scheduled tasks..

4.1.1 Frontend technologies

React & TypeScript

The frontend uses React and TypeScript. React enables a component-based structure for dynamic, interactive interfaces, while TypeScript improves reliability through static typing. Together, they support reusable components and a scalable architecture.

React Query (TanStack Query)

React Query manages data fetching, caching, and synchronization from Supabase and the backend API. It supports background updates and automatic refreshes, removing manual fetch logic and ensuring consistent, reliable, and high-performance data handling.

MUI (Material UI)

MUI provides a modern, responsive component library that ensures consistent design and speed up UI development.

Vite

Vite provides fast development, instant updates, and optimized builds for smooth performance.

4.1.2 Backend technologies

Node.js & Express

The backend is built with Node.js and Express to manage secure server-side operations such as admin-level booking management, executing Supabase actions that require the service-role key, and access to sensitive configuration data that must remain server-side.

Supabase Server Client

It handles privileged service-role operations, including room management and booking validation, while keeping secret keys secure on the server.

dotenv

Manages environment variables via a .env file to protect service keys and backend secrets.

4.1.3 Database & Authentication Technologies

Supabase (PostgreSQL Database)

Supabase provides a managed PostgreSQL database to handle bookings, room availability, user accounts, and admin data with strong relational integrity and consistency.

Supabase Auth

Manages secure sign-in, sessions, and role-based access control for guests and administrators.

Supabase Storage

Securely stores room images using signed URLs.

Supabase Edge Functions

Runs secure server-side logic for scheduled tasks such as charging bookings and sending notification emails, triggered via cron-job scheduling.

Row-Level Security (RLS)

Implements database-level access control to ensure users only view or modify authorized data.

4.1.4 Development & Deployment Tools

Node.js & NPM

They are used for dependency management, running development environments, and executing production builds (Node.js, n.d.).

ESLint & Prettier

They ensure consistent code style, formatting, and maintainability across the project.

Vercel & Render

The frontend is deployed on Vercel.com, while the backend is hosted on Render.com.

4.1.5 Third-Party Services

Stripe

Stripe securely processes saving payments, booking charges, and refunds within the platform.

Resend (Email Delivery Service)

Resend delivers transactional emails such as booking confirmations, updates, cancellations, requests to leave a review and review notifications to maintain user and admin communication.

Conclusion

GuestEase uses a modern, scalable setup where React, TypeScript, Vite, React Query, Node.js + Express, and Supabase work together to deliver a secure backend and a fast, responsive booking experience.

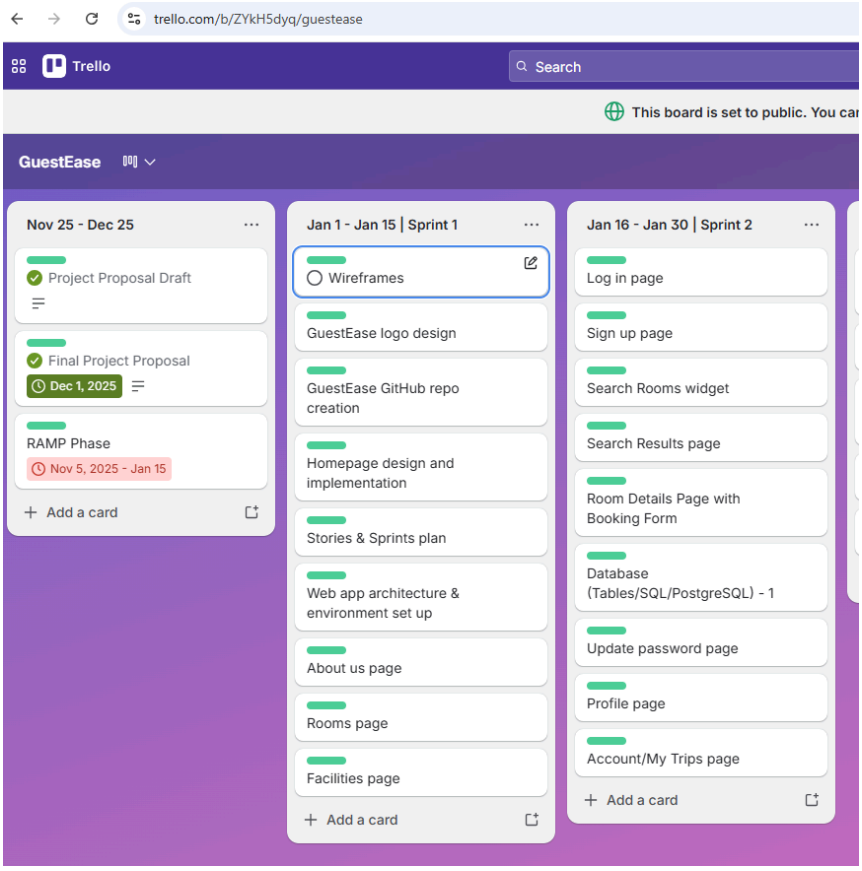
5 Project Implementation

5.1. Methodology and Tools

GuestEase was developed using an Agile-inspired, iterative approach across six sprints managed with Trello. Each cycle involved planning, building, testing, and refining features as requirements evolved.

5.2 Agile Sprints

5.2.1 Trello plan



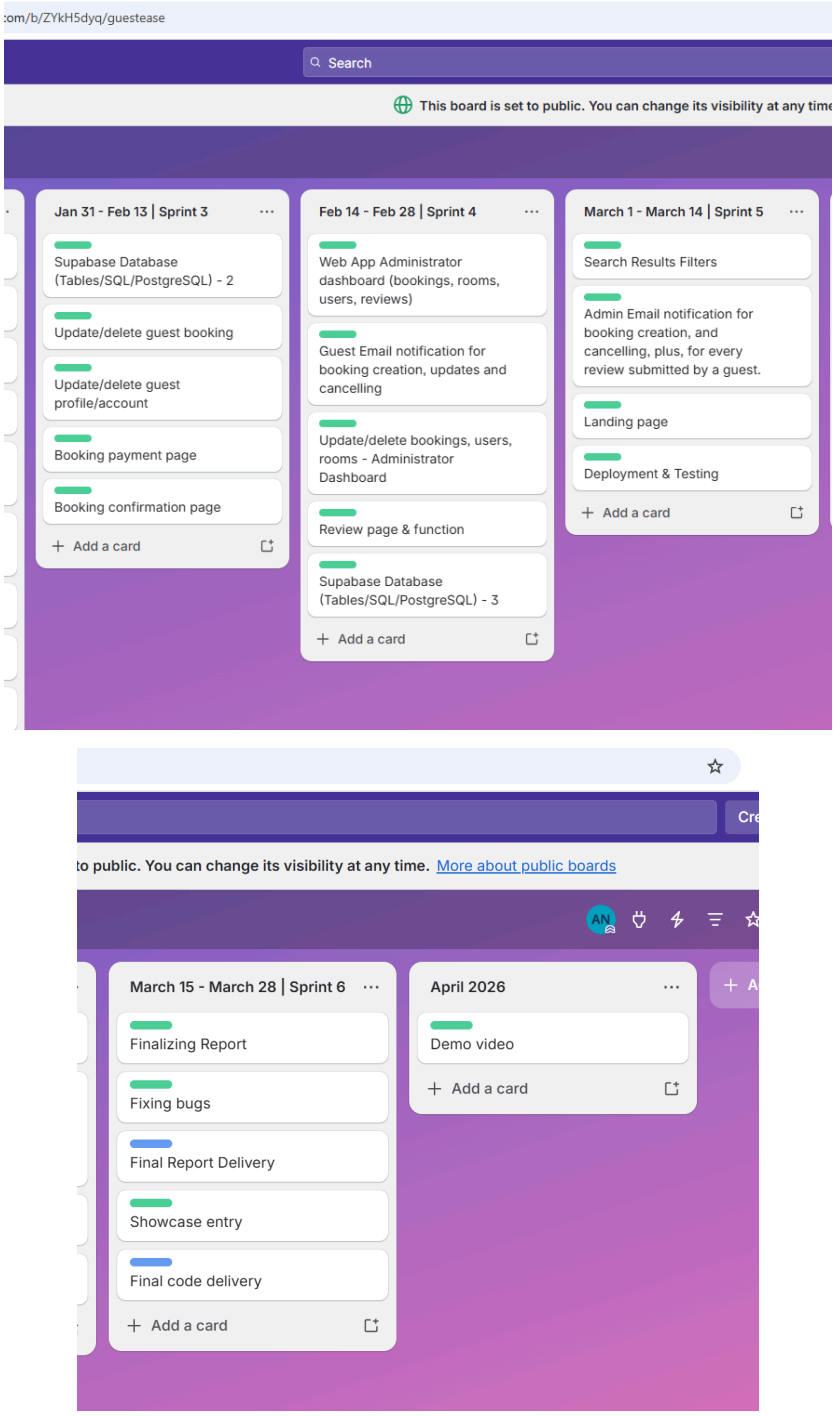


Figure 5-1: Trello plan

https://trello.com/b/WBdyHT0S/guestease

5.2.1 Sprint 1 — January 1st to January 15th

Focus: Establishing the platform foundation and core UI components.

Work Completed:

- Set up GitHub repo,
- development environment, and web architecture;
- designed logo and visual identity;
- planned user stories in Trello;
- completed wireframes;
- built static Homepage, About Us, Facilities pages;
- Rooms page connected to Supabase.

Outcome: Stable project setup with initial UI and first dynamic feature integrated.

5.2.3 Sprint 2 — January 16th to January 30th

Focus: User authentication, room search, and booking workflows implementation.

Work Completed:

- Login, Signup, Update Password pages
- Profile, and My Trips pages;
- Search Rooms form and Search Results page;
- Room Details page with Booking Form;
- Booking creation with Supabase and supporting PostgreSQL queries.

Outcome: The platform moved from static pages to functional, user-driven system.

5.2.4 Sprint 3 — January 31st to February 13th

Focus: Backend support for bookings and profile management.

Work Completed:

- Update/delete operations for bookings and profiles/users;
- Booking Payment and Confirmation pages integrated with Supabase.

Outcome: Functional backend supporting account changes, booking modifications, and core booking/payment flow.

5.2.5 Sprint 4 — February 14th to February 28th

Focus: Build admin dashboard, notifications, and CRUD features.

Work Completed:

- Admin dashboard for bookings, rooms, users, reviews;
- automated guest emails;
- update/delete admin operations;
- Review Page and workflow;
- expanded Supabase tables and SQL logic.

Outcome: Admin system set up with full management tools, automated notifications, and improved database.

5.2.6 Sprint 5 — March 1st to March 14th

Focus: Search results filters, admin notifications, landing page, and deployment/testing implemented.

Work Completed:

- Search results filters;
- admin email notifications;
- landing page https://guestease-landing.vercel.app/ published;
- deployment and testing completed.

Outcome: Enhanced search with filters, automated alerts, live landing page, and fully deployed system.

5.2.7 Sprint 6 — March 15th to March 28th

Focus:
Finalizing documentation, resolving remaining issues, and preparing the project for submission.

Work Completed:

- Finalized the project report
- Fixed remaining bugs
- Submitted the final report
- Submitted the showcase entry
- Delivered the final project code

Outcome:
Project completed and all final deliverables submitted.

5.3. GuestEase ReadMe documentation

A quite detailed and comprehensive README accompanies the project, documenting the overall development process, architectural decisions, and implementation details of GuestEase in https://github.com/AndreaNardinocchi/GuestEase/blob/main/README.md.

6 Reflection

6.1. System evaluation

This project was tough to pull off because of its size and all the interconnected parts. Overall, GuestEase is a modern, lightweight booking platform that can reliably support the full operational needs of a small hospitality business.

Functional Evaluation

- It consistently performs user and admin CRUD booking tasks, payments, automated email notifications, and other administration tasks such as managing rooms, and CRUD user operations.
- It is stable, secure, and maybe scalable for real-world use architecture.

Non-Functional Evaluation

- High accessibility
- The booking flow is simple and intuitive
- The UI walks the user through every step of the way
- Search queries, and CRUD tasks return data speedily because of React Query caching reusing cache data and invalidating it after mutations keeping data fresh.

Limitations

- One default credit card per user (see *6.7. Future developments* for more details)
- No mult-property support
- More unit testing required

6.2. App Testing

Types of Testing

- Manual testing was done throughout the development phase and after deployment live with both dummy and real-like data.
- Some Unit testing done for the booking flow
- Integration testing for the booking and payment flow

Test cases

Feature	Test	Result
Booking	Valid dates	Success
Booking	Empty dates	*Please select valid dates before booking.* (Error handled)
Booking	Overlapping dates	*This room is not available for the selected dates.* (Error handled)
Booking	check-in>check-out	Return ignoring invalid selection
Payment	Valid card	Success
Payment	Empty card fields	*Your card number is incomplete.* (Error handled)
Sign up	Already registered email	*This email is already registered. Please, use a different email, thank you!* (Error handled)
Admin Booking Create/Update	Overlapping dates	*This room is already booked for these dates* (Error handled)
Admin Booking	Empty fields	*Please fill in all required fields.* (Error

Create/Update		handled)
Admin Booking Create/Update	check-in>check-out	*Check-in must be before check-out* (Error handled)
Update User Booking	Empty fields	*Please fill in all required fields.* (Error handled)
Update User Booking	check-in>check-out	*Check-in must be before check-out*
Update User Profile	Empty fields	*Please fill in all required fields.* (Error handled)
Admin User Create/Update	Empty fields	*Please fill in all required fields.* (Error handled)
Admin User Create	Existing email	*User with this email already exists* (Error handled)
Admin Room Create/Update	Empty fields	*Please fill in all required fields.* (Error handled)
Review page	Empty fields	*Please provide both rating and comment.* (Error handled)

Table 6-1: Test Cases

6.3. Security aspects

Although the focus of the project was never data security, we implemented the below-listed features:

- Role-based authentication restricts access to guest and admin features.
- Supabase Auth secures sign-in, session handling, and identity verification through confirmation email link and password recovery email.
- All sensitive actions (payments, bookings, emails) run on the backend, keeping secrets (keys) protected (ex. *const stripe = new Stripe(process.env.STRIPE_SECRET_KEY)*);

6.4. Problems encountered

During the development phase of GuestEase, I hit many brickwalls, as I fully expected.

Supabase Authentication/Profile table synchronization
One of the many issues was how to synchronize users data in the Supabase authentication section

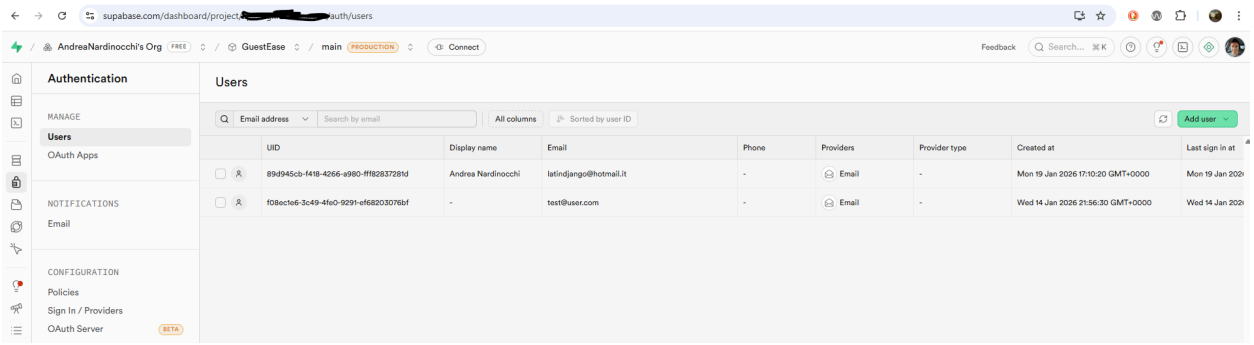


Figure 6-1: Supabase - Users Authentication dashboard

with the 'profiles' table data, created to be able to display users data not retrievable from the authentication section, such as user 'country', 'zip code' and so on.

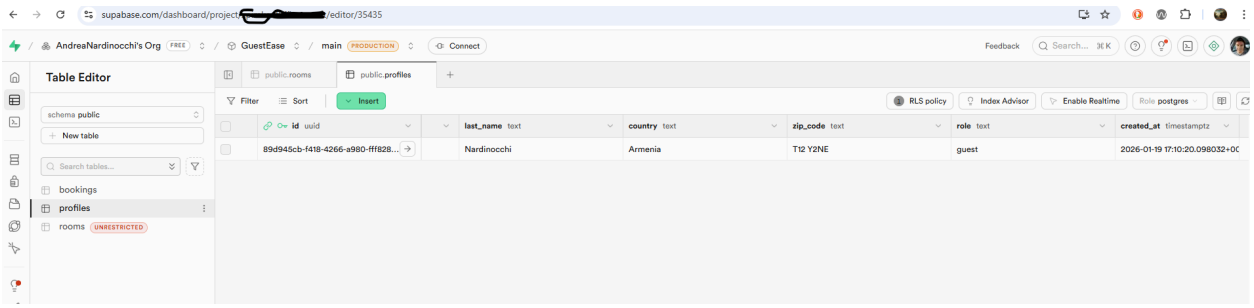


Figure 6-2: Supabase - 'public.profiles' table

This was resolved by adding a trigger in the PostgreSQL 'profiles' table, which automatically runs functions to insert, update, or delete user records whenever changes occur in the authentication system ('SupabasePostgreSQL/profilesTable' repo file for details).

Room Details page invalid input syntax

Another issue I came across was on the 'page/roomDetailsPage.tsx' page, which kept displaying the below error when trying to fetch room data:

```
"invalid input syntax for type date: \"\""
```

Figure 6-3: Room Details page console

A way to disentangle the error, was to add an 'enabled' option to the 'hooks/useAvailableRooms.ts' hook with a 'boolean' condition that, ultimately, 'enabled' the query to run only if the '(checkIn, checkOut, guests)' values were valid.

```
export function useAvailableRooms(
  return useQuery({
    queryKey: ["availableRooms", checkIn, checkOut, guests, excludeBookingId],
    queryFn: async () => {
      const result = await searchAvailableRooms(
        checkIn,
        checkOut,
        guests,
        excludeBookingId,
      );

      if (!result.success) {
        throw new Error(result.message || "Failed to fetch rooms");
      }
      return result.rooms;
    },
    refetchOnMount: "always",
    staleTime: 0,
    /**
    * enabled: boolean
    * 'Set this to false to disable this query from automatically running.
    * Can be used for Dependent Queries.'
    * Only run this query when all three values (checkIn, checkOut, guests) are valid.
    * https://tanstack.com/query/v3/docs/framework/react/reference/useQuery
    */
    enabled: Boolean(checkIn && checkOut && guests),
  });
}
```

Figure 6-4: Excerpt of the useAvailabaleRooms.ts hook

Booking backend error

I spent quite some time to find a way out of the below error when attempting to create the first 'user booking':

Figure 6-5: Room Details booking error

In the end, the error was caused by the 'backend/utils/calculateTotalPriceUtil.js' code where I was importing 'check_in', 'check_out' data (strings), but I was failing to convert the strings into 'date' format in order to be able to use them as factors in the 'total_price' calculation.

Once I used the 'new Date()' constructor, the issue got fixed immediately.

```
backend > utils > JS calculateTotalPriceUtil.js > calculateStay
/**
 * This is a helper to calculate the 'total_nights' parameter for the
 * 'booking' object
 */
export function calculateStay(check_in, check_out, roomPrice) {
  /**
   * check_in and check_out are strings and not dates, and this was causing
   * 'check_in.getTime' is not a function' error.
   * Hence, we are trasforming them into proper dates
   */
  const checkInDate = new Date(check_in);
  const checkOutDate = new Date(check_out);

  const nights =
    (checkOutDate.getTime() - checkInDate.getTime()) / (1000 * 60 * 60 * 24);

  if (nights < 1) {
    throw new Error("Stay must be at least 1 night");
  }

  const total_price = nights * roomPrice;

  return { nights, total_price };
}
```

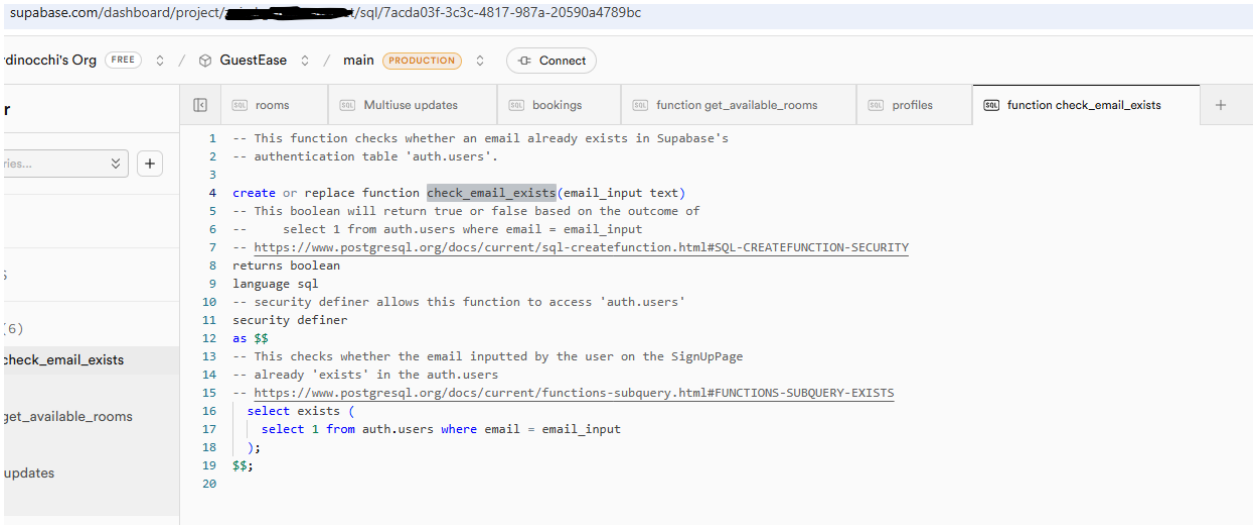
Figure 6-6: calculateTotalPriceUtil.js code

Existing user email

Testing a sign up with an already registered email in Supabase 'auth.users' was failing, meaning, a new user would get a 'success' confirmation message upon signing up with an

already registered email, even though Supabase would not add the new user in ‘auth.users’.

It took me to create a new PostgreSQL function ‘check_email_exists’ to fix the bug, and call it from the ‘page/signUpPage.tsx’.




```
try {
  /**
   * We introduced an existing email check through a supabase PostgreSQL function.
   */
  const { data: emailExists } = await supabase.rpc("check_email_exists", {
    email_input: email,
  });

  if (emailExists) {
    setRejectionMessage(
      "This email is already registered. Please, use a different email, thank you!",
    );
    return;
  }
```

Figure 6-7: Supabase PostgreSQL ‘function check_email_exists’

Booking overlap bug when editing booking dates

The update booking feature initially allowed guests to edit bookings but didn’t prevent overlapping reservations.

The issue was fixed by fetching available rooms via ‘hooks/useAvailableRooms.ts’ using Supabase’s ‘get_available_rooms’ RPC. Additionally, an inline overlapError message was

added, while the save button on the modal was updated to be disabled whenever overlaps occur.

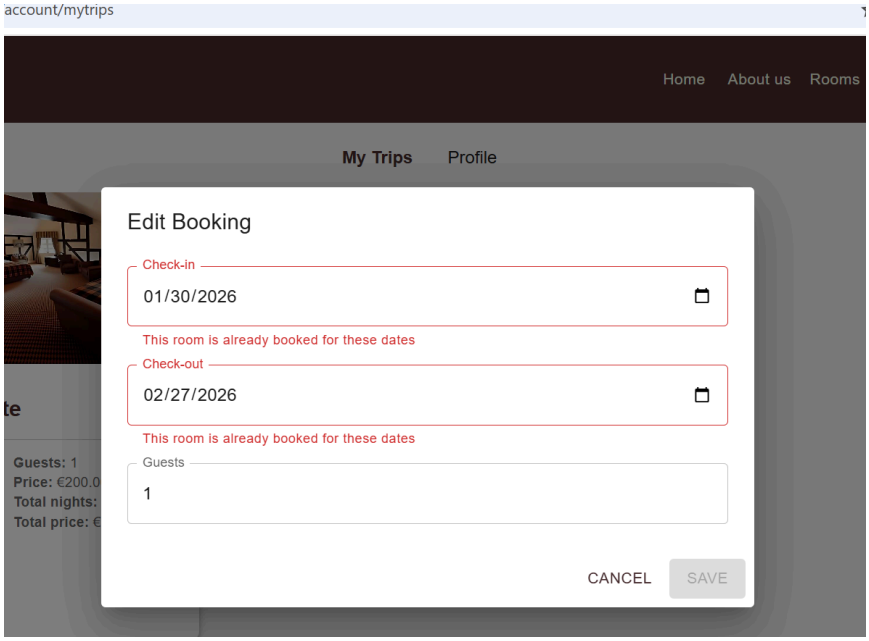


Figure 6-8: Edit Booking modal on the /account/mytrips page

```
/**
 * If the user selects a check-in date beyond the check-out one,
 * an error helper will display in the modal.
 */
const checkInDate = new Date(localBooking.check_in);
const checkOutDate = new Date(localBooking.check_out);
const dateError = checkInDate >= checkOutDate;

const { data: availableRooms } = useAvailableRooms(
  localBooking.check_in,
  localBooking.check_out,
  localBooking.guests,
  /**
   * Fetching the booking id to exclude when checking the room availability for updates.
   * If we do not exclude it, the modal will show the overlapError at any time.
   * We want it to only show when there is an actual overlap with other bookings.
   */
  localBooking.id,
);
```

```
README.md M   editBookingDialog.tsx ×   function get_available_rooms   searchRoomsContext.tsx   availableRooms.ts
frontend > src > components > editBookingDialog > editBookingDialog.tsx > ...
41  const EditBookingDialog: React.FC<EditBookingDialogProps>
234         </Button>
235
236         <Button
237           onClick={() =>
238             /**
239              * Pass the fully recalculated booking data (...local booking) to the parent.
240              * We explicitly include 'nights' and 'total_price' because the Stripe
241              * payment flow depends on these values being accurate and up-to-date.
242              * */
243             onSave({
244               ...localBooking,
245               // Required for Stripe logic
246               nights: localBooking.nights,
247               total_price: localBooking.total_price,
248             })
249           }
250           variant="contained"
251           /**
252            * By disabling the button in case of any errors, we prevent overlapping
253            * and invalid date selections.
254            */
255           disabled={overlapError || dateError}
256           sx={{
257             backgroundColor: "#472d30",
258             "&:hover": { bgcolor: "#e26d5c" },
259           }}
260         >
261           Save
262         </Button>
263       </DialogActions>
264
```

searchRoom Aa ab .* No results

Figures 6-9: Excerpts of the editBookingDialog.tsx code

Hook-Order Mismatch

While rendering the 'pages/roomDetailsPage.tsx', the component repeatedly crashed with 'Rendered more hooks than during the previous render.' error.

```
Current URL: /room/0c3b087-ea23-4242-8bf9-f94bd3450f9d?checkIn=2026-   roomDetailsPage.tsx:94
02-10&checkOut=2026-02-26&guests=2
▶ Uncaught Error: Rendered more hooks than during the   react-dom_client.js?v=d1377a35:5792
previous render.
    at RoomDetailsPage (roomDetailsPage.tsx:124:65)
  Show ignore-listed frames
▶ An error occurred in the <RoomDetailsPage> component.   App.tsx:31
```

Figure 6-10: RoomDetailsPage 'Rendered more hooks..' error

This happened because the loading and error returns were placed before several state and effect hooks. When those early returns were executed, React skipped the later hooks entirely, causing the hook sequence to differ on subsequent renders. The fix was to move all hooks above the conditional returns so the hook order never changes.

```
frontend > src > pages > roomDetailsPage.tsx > ...
24   const RoomDetailsPage: React.FC = () => {
181    const handlePaymentSuccessSoBookNow = async (_paymentMethodId: string) => {
198        setError(err.message);
199        setStripeCheckOutModalOpen(false);
200      }
201    };
202
203    /**
204     * These early returns are safe because all hooks in this component
205     * are declared above them. Previously, they appeared before several
206     * useState/useEffect calls, which caused React to run a different
207     * number of hooks between renders and triggered a hook-order error.
208     * Now the hook order is stable:
209     * 1. Hooks always run first
210     * 2. These returns only affect what UI is shown
211     * 3. React never sees a different hook sequence
212     */
213    if (roomLoading) {
214      return <CircularProgress sx={{ display: "block", mx: "auto", my: 10 }} />;
215    }
216
217    if (roomError || !room) {
218      return <Alert severity="error">Room not found.</Alert>;
219    }
220
221    if (availabilityError) {
222      return <Alert severity="error">Error loading availability.</Alert>;
223    }
224
225    return (
226      <>
227        {/* Main UI only when everything is valid */}
228
229        <>
230          <Box>
231            {/* This hero image will be replaced by a carousel to display all 4 room images */}
```

Figure 6-11: RoomDetailsPage 'loading and error returns'

This kept the component's render path consistent and resolved the error.

Image Upload Roadblock

We encountered an issue in **adminRoomsPage.tsx** where uploaded images failed to display due to broken file path logic. After multiple rewrites, the fix was about moving the upload process into a dedicated React Query mutation and custom hook ('hooks/useAdminCreateRoom.ts'), which stabilized image handling and ensured consistent UI rendering.

```
TS interfaces.ts   adminRoomsPage.tsx   adminBookingsPage.tsx   bookingConfirmationPage.tsx ×   TS uploadRoomImages.ts   TS useAdminCreateRoom.ts
rontend > src > pages > bookingConfirmationPage.tsx > BookingConfirmationPage
ookingConfirmationPage: React.FC = () => {
                                                                                                  > img   Aa ab
    <Box
      component="img"
      src={
        /**
         * The uploaded image path is like 'rooms/a77ddc44-0a5e-4585-b4e7-5b61cb2865d3/1770573915402-DruidsRest2.jpg',
         * as per 'const filePath = `rooms/${roomId}/${Date.now()}-${safeName}`;' in the adminRoomsPage.tsx file.
         * Hence, we are saying below, that if 'img' does include 'rooms/' in its path, that mean it has been uploaded by
         * the admin and will show the uploaded path. Otherwise, it will enable the old image path display, whose
         * image was originally manually uploaded straight into supabase
         */
        room.images[1].includes("rooms/")
          ? getPublicUrl(room.images[1]) // New uploaded images path
          : getPublicUrl(`/rooms/${room.id}/${room.images[1]}`) // old seeded images
      }
      alt={room.name}
```

Figure 6-12: Example of updated image path on the bookingConfirmationPage.tsx

I, then, also had to update the image path across several files to ensure that the 'uploaded images' would display consistently.

Cron unsuccessful

At some point, we noticed that the booking Stripe payments in the 'charged' column of the Supabase 'bookings' table were no longer being charged', even though Cron https://console.cron-job.org/jobs/7220668/history kept showing a successful history. We traced the issue to the Supabase Edge Function comparing timestamps against a DATE column. After reviewing the function code (https://supabase.com/dashboard/project/xxxxxxxxxxxxxxx/functions/charge-upcoming-bookings/code), we fixed it by switching to simple YYYY-MM-DD comparisons (we were mistakenly using timestamp formats), which immediately resolved the bug.

```
Deno.serve(async () => {
  /**
   * A fix that was needed:
   * check_in is a DATE column, not a TIMESTAMP.
   * We must compare date to date, otherwise bookings on the same day
   * (e.g. check_in = "2026-02-14") won't match when cron https://console.cron-job.org/jobs/7220668/history
   runs later in the day.
   https://stackoverflow.com/questions/47066555/remove-time-after-converting-date-toisostring#55231024
   */
  const today = now.toISOString().split("T")[0];
  const tomorrow = in24h.toISOString().split("T")[0];

   /**
   * 1. Fetch bookings with check-in within the next 24 hours
   *    AND not yet charged.
   * https://supabase.com/docs/reference/javascript/select
   * Compare DATE strings instead of timestamps
   * Limit to 5 booking to avoid Supabase EarlyDrop timeouts
   */
  const { data: bookings, error } = await supabase
    .from("bookings")
    .select("*")
    .lte("check_in", tomorrow)
    .gte("check_in", today)
    .eq("charged", false)
    .limit(5);
```

Figure 6-13: Excerpt of the Supabase Edge Function ‘charge-upcoming-bookings’

Refunds logic when cancelling a booking

We faced an issue where the system lacked reliable Stripe data for cancellations and subsequent refunds because the booking record didn’t store the payment method or exact charged amount. By adding ‘payment_method_id’ and ‘amount’ to the bookings table, each booking now securely retains the data needed for consistent refunds and admin cancellations.

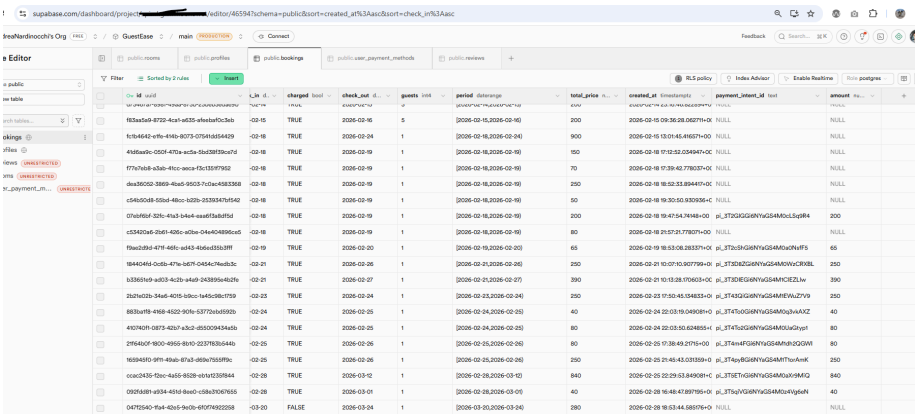


Figure 6-14: Excerpt of the Supabase ‘Bookings’ table Edge Function ‘charge-upcoming-bookings’

Resend.com emails can only be sent to one single email on the free tier

Unfortunately, we were limited by the free tier plan in Resend which would not allow us to

send testing emails for bookings, users, and reviews to recipients other than the email used to create the Resend account.

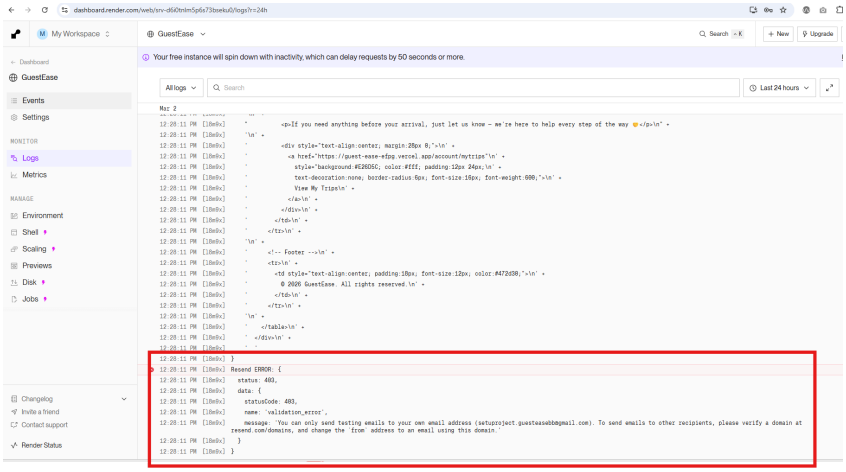


Figure 6-15: Excerpt of the backend Render logs when the email functionality is triggered

Resend.com would require us to register our own public domain to be able to send email notifications to guests.

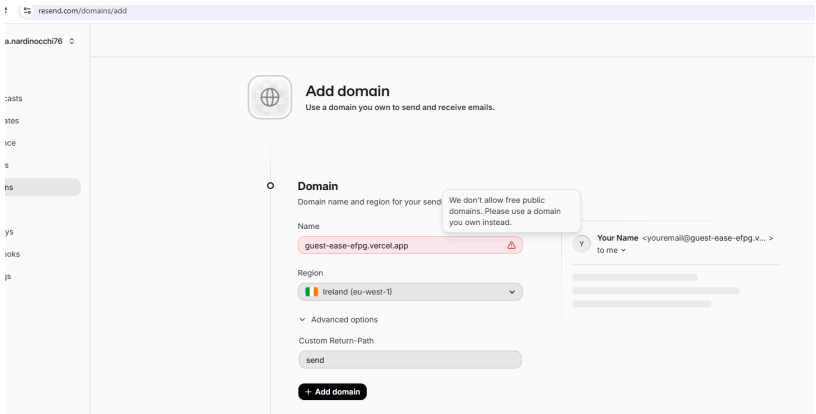


Figure 6-16: Excerpt of the Resend Domain section on https://resend.com/domains/add

That being said, GuestEase is an academic project, a prototype, and its goal is to solely show how the email notifications feature can be implemented through this app. Future developments and a public domain purchase would still be needed to make this feature ready-to-market.

6.5. Learning

Through this project, I worked with several new technologies, including PostgreSQL (and PostgREST), Supabase Storage and Edge Functions, Stripe, Resend, and cron-job.org, while strengthening my knowledge of React, Material UI, Node.js, Express, Supabase Auth, and SQL.

However, due to personal and professional commitments, I had to finish the project by mid-March, which required focused effort and careful time management.

6.6. Achievements

Overall, I delivered a working system that met the core CRUD requirements and brought together the main features I set out to build, especially the admin interface, Stripe payments, and the automated email workflows.

6.7. Future developments

One of the future developments could be that of creating a loyalty reward system for guests so that they can accumulate points with their stays and get access to special discounts or promos. The rewards system idea might resemble that of a big hospitality booking system, but GuestEase itself could be transformed into a multi-property booking platform that goes beyond the 8 rooms B&B single-property prototype. The current logic used would not stop future developments from injecting rooms from other small hospitality businesses into this platform and also from applying filters by property.

Also, some work could be done to introduce a map logic to let users see where the rooms are geo-located (based on where the business is located), etc.

Regarding the security side, while an admin route was implemented frontend (adminRoute.tsx) to only enable the 'admin' role to access the admin dashboard, the backend tables like bookings or rooms do not currently enforce RLS for admin actions,

and there is no admin role validation in the backend, which might be implemented in the future.

Regarding the Stripe payment flow, it keeps the system simple for a small app because it uses one default card per user, reducing complexity and making charges predictable. Future development should be focused on storing the default payment method directly on each booking, and the user should be able to set multiple payment methods:

User → Frontend (Stripe Elements from StripeCheckOut)
→ Stripe.js (create PaymentMethod)
→ Backend (/save-payment-method)
→ Stripe (attach PaymentMethod → Customer + set as default)
→ Supabase (user_payment_methods: delete old → insert new default)
→ Booking created
→ Cron Job (24h before check-in)
→ Supabase (fetch default from user_payment_methods)
→ Stripe (create PaymentIntent off_session)
→ Supabase (mark booking charged)
→ Email sent to user

Overall, the app should be refined under all aspects to make it a viable product to be introduced in the real world.

7 AI role

Throughout the project, I also used AI tools to support some of the creative aspects of the project, such as generating room descriptions, amenities (Microsoft Copilot, 2026; see Appendix A, AI-12), and visual assets.
For example, prompts like *"Create rooms with Irish-inspired names and tailored amenities"* or *"Show me the same room from a different angle"* helped me quickly produce consistent content and imagery for the interface.
Some of them were https://image-generator.com/ai-image-generator and https://deepai.org/ (AI Image Generator, n.d.; DeepAI, n.d., see Appendix A, AI-13).

On the development side, I used AI (Copilot) conversationally to overcome roadblocks and validate architectural choices.

For example, I discussed how to run Stripe logic server-side, how to structure the booking CRUD architecturally on the backend and why there rather than on the frontend, which Supabase would allow flawlessly. I also enquired about when React Query and mutations were more appropriate than alternative approaches, and the answer was that they solve three hard problems at once: server-state management, cache consistency, and UI synchronization (Microsoft Copilot, 2026; see Appendix A, AI-14).

In the end, the aim has always been to build a platform with modern 2025-2026 technologies.

Bibliography

1. Stack Overflow (2025). *2025 Developer Survey – Technology.* [online] Available at: https://survey.stackoverflow.co/2025/technology#most-popular-technologies-web frame-prof [Accessed 2 Nov. 2025].
2. Krotoff, T. (n.d.). *Front-end frameworks popularity (React, Vue, Angular and Svelte).* [online] Available at: https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190 [Accessed 2 Nov. 2026].
3. Dineuron. (2025). *Frontend Frameworks 2025: React vs Vue vs Angular vs Svelte - The Complete Developer's Guide.* [online] Available at: https://dineuron.com/frontend-frameworks-2025-react-vs-vue-vs-angular-vs-svelt e [Accessed 2 Nov. 2025].
4. DevAceTech, 2025. *What are the 8 Best Frontend Development Frameworks in 2025* [online]. Available at: https://www.devacetech.com/insights/best-frontend-development-frameworks [Accessed 2 November 2025].
5. Microsoft (2024) *Microsoft Copilot* [AI assistant]. Microsoft.Available at: https://copilot.microsoft.com [Accessed 7 Nov. 2025].
6. Angular. (n.d.). *Two-way binding.* [online] Available at: https://angular.dev/guide/templates/two-way-binding [Accessed 2 Nov. 2025].
7. Khandelwal, A. (2019). *Introduction to Hapi.js (Part 1)* [online] Medium. Available at: https://medium.com/the-resonant-web/an-introduction-to-hapi-js-part-1-5f2edcb 5eced [Accessed 3 Nov. 2025].
8. GeeksforGeeks. (2025). *The Pros and Cons of Node.JS in Web Development.* [online] Available at: https://www.geeksforgeeks.org/node-js/the-pros-and-cons-of-node-js-in-web-dev elopment/ [Accessed 3 Nov. 2025].
9. Justin (2023). *PostgreSQL vs NoSQL: Unveiling the Database Dilemma.* [online] Els@Soft. Available at: https://www.elsasoft.org/nosql-postgres/ [Accessed 3 Nov. 2025].
10. Adela (2025). *Supabase vs. Firebase: a Complete Comparison in 2025.* [online] Bytebase. Available at: https://www.bytebase.com/blog/supabase-vs-firebase/ [Accessed 3 Nov. 2025].
11. GeeksforGeeks. (2024). *What Is Firebase?* [online] Available at: https://www.geeksforgeeks.org/firebase/what-is-firebase/ [Accessed 3 Nov. 2025].

12. Ndungu, D. (2022). *Firebase Chat: Pros & Cons.* [online] cometchat. Available at: https://www.cometchat.com/blog/firebase-chat-pros-cons [Accessed 3 Nov. 2025].
13. Ankolekar, M. (2025). *Building a Node.js CRUD API with Supabase.* [online] Dev. Available at: https://dev.to/manthanank/building-a-nodejs-crud-api-with-supabase-3bkp [Accessed 3 Nov. 2025].
14. Ulili, S. (2025). *Supabase vs Firebase.* [online] Better Stack. Available at: https://betterstack.com/community/guides/scaling-nodejs/supabase-vs-firebase/ [Accessed 3 Nov. 2025].
15. Razorpay. (n.d.). Available at: https://razorpay.com/ [Accessed 4 Nov. 2025].
16. Surani, S. (2025). *The-Fern-Hotel-Booking-Site.* [online] GitHub. Available at: https://git.hubp.de/satishsurani/The-Fern-Hotel-Booking-Site [Accessed 4 Nov. 2025].
17. Blakely, C. (2024). *MERN Stack Roadmap – How to Learn MERN and Become a Full-Stack Developer.* [online] freeCodeCamp. Available at: https://www.freecodecamp.org/news/mern-stack-roadmap-what-you-need-to-know-to-build-full-stack-apps/ [Accessed 4 Nov. 2025].
18. K, Suresh (2024). *Full-stack-hotel-booking.* [online] GitHub. Available at: https://github.com/Binary-Shade/Full-stack-hotel-booking [Accessed 4 Nov. 2025].
19. Nissoukin, O. (2025). *nextjs-hotel-booking*. [online] GitHub. Available at: https://github.com/OthmaneNissoukin/nextjs-hotel-booking [Accessed 4 Nov. 2025].
20. McGarr, K. (2024). *Unlocking Direct Bookings – Sprint Digital's Audit Reveals Ireland's Hotels are not doing enough.* [online] Sprint Digital. Available at: https://www.sprintdigital.com/blog/unlocking-direct-bookings-sprint-digitals-audit-reveals-irelands-hotels-are-not-doing-enough [Accessed 7 Nov. 2025].
21. Gerry (2025). *Irish Small Business Website Challenges.* [online] Accent Webs. Available at: https://accentwebs.ie/irish-small-business-website-challenges/ [Accessed 7 Nov. 2025].
22. john@secured (2025). *Top 5 Challenges Facing the Irish Hospitality Industry in 2025.* [online] Hospitality Partners. Available at: https://hospitalitypartners.ie/top-5-challenges-facing-the-irish-hospitality-industry-in-2025/ [Accessed 7 Nov. 2025].
23. Escape2Project. (n.d.). *Supporting the Development of Transnational Thematic Tourism Strategy for Rural Regions in Europe.* [online] Available at: https://www.escape2project.org/uploads/1/2/3/6/123661942/io1_report_en.pdf [Accessed 7 Nov. 2025].
24. Haley, M. (2025). *Holidaymakers embrace last-minute planning, mobile bookings. [online] Travolution.* Available at:

https://www.travolution.com/news/holidaymakers-embrace-last-minute-planning-mobile-bookings/ [Accessed 7 Nov. 2025].
25. McCárthaigh, S. (2025). *An extra one MILLION nights booked in Airbnb-style accommodation last year compared to 2023.* [online] The Journal. Available at: https://www.thejournal.ie/almost-7-5-million-guest-nights-were-booked-in-airbnb-stye-accommodation-in-ireland-last-year-6755714-Jul2025/ [Accessed 7 Nov. 2025].
26. CSO statistical release (2025). *Household Travel Survey Quarter 4 and Year 2024.* [online] Central Statistics Office. Available at: https://www.cso.ie/en/releasesandpublications/ep/p-hts/householdtravelsurveyquarter4andyear2024/ [Accessed 7 Nov. 2025].
27. Dooley, H. (2025). Irish hotel industry fastest in EU to bounce back from pandemic, data shows. [online] The Irish Times. Available at: https://www.irishtimes.com/business/2025/04/03/irish-hotel-industry-fastest-in-eu-to-bounce-back-from-pandemic-data-shows/ [Accessed 7 Nov. 2025].
28. Power-Cowley, S. (n.d.). *Net Affinity: Insights from the 2024 Hotel Mobile Barometer.* [online] Irish Hotels Federation. Available at: https://www.ihf.ie/suppliers/resources/content/2024/netaffinity/hotel-mobile-barometer-24 [Accessed 7 Nov. 2025].
29. Bunnell, P. (2025). *The Hidden Cost of OTAs: What Hotels Really Pay.* [online] Linkedin. Available at: https://www.linkedin.com/pulse/hidden-cost-otas-what-hotels-really-pay-parrie-bunnell-bdzic [Accessed 8 Nov. 2025].
30. Node.js. (n.d.). *An introduction to the npm package manager.* [online] Available at: https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager [Accessed 14 Mar. 2026].
31. DeepAI. (n.d.). Available at: https://deepai.org/ [Accessed 3 Jan. 2026].
32. AI Image Generator. (n.d.). Available at: https://image-generator.com/ai-image-generator [Accessed 3 Jan. 2026].

Appendix A — AI Prompts and Responses

AI-1 — React

Prompt:

“Give me short, easy to understand pros and cons of using React for a booking system. Concise and bullet points!"

AI Response:

“Pros

- Huge ecosystem — tons of libraries for calendars, forms, payments, and routing.
- Strong community support — easy to find solutions, tutorials, and third-party tools.
- Reusable components — great for building consistent UI elements like date pickers or booking cards.
- Scales well — suitable for both small MVPs and large enterprise booking platforms.
- Backed by Meta — long-term stability and regular updates.

Cons

- More boilerplate — state management and data flow can feel heavy for simple features.
- Steeper learning curve — JSX, hooks, and state patterns take time to master.
- Performance overhead — virtual DOM adds complexity compared to lighter frameworks.
- Many choices — routing, state libraries, and architecture decisions can overwhelm teams.
- Requires extra libraries — React alone doesn’t handle routing, SSR, or data fetching.”

Used in:

Section 2.1.1.1

AI-2 — Svelte

Prompt:
“Give me short, easy to understand pros and cons of using Svelte for a booking system. Concise and bullet points!"

AI Response:

“Pros

- Fast performance thanks to compile-time optimization (no heavy runtime).
- Lightweight bundles, which help pages load quickly.
- Simple, readable syntax that’s easy to learn.
- Reactive state built-in, reducing boilerplate code.
- Great developer experience with minimal complexity.
- Smooth transitions/animations if the UI needs polish.

Cons

- Smaller ecosystem compared to React or Vue.
- Fewer ready-made components for complex booking features.
- Less community support for niche problems.
- SSR and routing require SvelteKit (extra learning curve).
- Fewer enterprise-level tools and integrations.”

Used in:

Section 2.1.1.2

AI-3 — Angular

Prompt:
“Give me short, easy to understand pros and cons of using Angular for a booking system. Concise and bullet points"

AI Response:

“Pros of Using Angular for a Booking System

- Full framework — batteries-included structure ideal for complex booking flows.
- Strong architecture — great for large teams and long-term scalability.
- TypeScript by default — reduces bugs in forms, calendars, and availability logic.
- Powerful tooling — CLI, built-in routing, and form handling streamline development.
- Excellent for enterprise — consistent patterns, strict conventions, and maintainability.

- Robust state management — NgRx and signals help with complex availability/booking states.

Cons of Using Angular for a Booking System

- Steeper learning curve — more complex than Vue or React.
- Heavier bundle size — can feel slower for simple customer-facing flows.
- Verbose code — more boilerplate for components, forms, and services.
- Slower initial development — setup and structure take more time upfront.
- Less flexible — strong conventions mean fewer lightweight or ad-hoc approaches."

Used in:

Section 2.1.1.3

AI-4 — Vue

Prompt:
"Give me short, easy to understand pros and cons of using Vue for a booking system. Concise and bullet points"

AI Response:

"Pros of Using Vue for a Booking System

- Easy to learn — gentle learning curve compared to React or Angular.
- Fast to develop — component system makes building forms, calendars, and dashboards quick.
- Reactive UI — great for live updates (availability, pricing changes, seat selection).
- Lightweight — loads fast, ideal for customer-facing booking flows.
- Flexible — works well with REST APIs, GraphQL, or existing backend systems.
- Good ecosystem — Vue Router, Pinia, and community UI libraries simplify development.
- Readable code — single-file components keep logic, template, and styles organized.

Cons of Using Vue for a Booking System

- Smaller ecosystem than React — fewer ready-made plugins for niche booking features.
- Fewer enterprise patterns — may require more custom architecture decisions.
- SSR complexity — server-side rendering (Nuxt) adds setup overhead if SEO is important.
- Talent availability varies — depending on region, fewer Vue developers than React.
- Long-term scaling — large teams sometimes prefer frameworks with stricter conventions."

Used in:

Section 2.1.1.4

AI-5 — Node.js

Prompt:
"Give me short, easy to understand pros and cons of using Node.js for a backend for a booking system. Concise and bullet points"

AI Response:

"Pros

- **High performance for real-time features** — great for live availability updates or notifications.
- **Non-blocking I/O** — handles many simultaneous requests efficiently (ideal for busy booking systems).
- **Large ecosystem (npm)** — plenty of libraries for authentication, payments, scheduling, etc.
- **Same language front and back** — JavaScript everywhere simplifies development.
- **Scales horizontally** — easy to distribute load across multiple servers.

Cons

- **Not ideal for heavy CPU tasks** — complex pricing calculations or large data processing can slow it down.

- **Callback/async complexity** — can get messy without good structure.
- **Requires careful architecture** — easy to create performance bottlenecks with poor code.
- **Less opinionated** — you must choose frameworks, patterns, and tools yourself.
- **Security needs attention** — npm ecosystem can introduce vulnerabilities if not managed well."

Used in:

Section 2.1.2.1

AI-6 — Firebase

Prompt:
"Give me short, easy to understand pros and cons of Firebase for backend/database for a booking system. Concise and bullet points"

AI Response:

"Pros

- **Fast to build** — no server setup; great for MVPs and small teams.
- **Real-time updates** — ideal for showing live availability changes.
- **Scales automatically** — Google handles infrastructure for you.
- **Built-in authentication** — easy login with email, Google, etc.
- **Strong security rules** — flexible, database-level access control.
- **Good for mobile apps** — excellent SDK support for iOS/Android.

Cons

- **Complex relational data** is harder — Firestore is NoSQL, so joins and multi-table logic require workarounds.
- **Pricing can spike** — pay-as-you-go costs grow with traffic and reads/writes.
- **Vendor lock-in** — migrating away from Firebase is difficult.
- **Limited backend logic** — Cloud Functions are powerful but can feel restrictive.
- **Offline/real-time behaviour** can complicate booking conflict handling."

Used in:

Section 2.1.2.2

AI-7 — Supabase

Prompt:
"Give me short, easy to understand pros and cons of Supabase for backend/database for a booking system. Concise and bullet points"

AI Response:

"Pros

- **SQL database (PostgreSQL)** — great for complex relational booking data like users, bookings, rooms, and availability.
- **Built-in auth** — easy login with email, magic links, OAuth, etc.
- **Real-time features** — useful for live availability updates.
- **Row-level security** — strong, flexible access control for sensitive booking data.
- **Fast development** — APIs, database, storage, and auth all included.
- **Open-source** — no vendor lock-in; easier to self-host if needed.

Cons

- **Can get expensive at scale** — especially with heavy real-time or storage usage.
- **Still maturing** — smaller ecosystem compared to Firebase or AWS.
- **Requires SQL knowledge** — relational modelling and queries need more planning.
- **Real-time features add complexity** — conflict handling for bookings must be carefully designed.
- **Self-hosting adds overhead** — if you don't use the managed cloud, you manage updates and scaling."

Used in:

Section 2.1.2.3

AI-8 — Digital Transformation in Ireland's Small-Scale Accommodation Sector

Prompt:
"Briefly explain why many small hospitality businesses in Ireland (guesthouses, B&Bs, boutique hotels) struggle with digital adoption. Mention that they still use manual bookings, don't have modern websites, and that about 80% of reservations are now online. Include references like McGarr (2024) in https://www.sprintdigital.com/blog/unlocking-direct-bookings-sprint-digitals-audit-reveals-irelands-hotels-are-not-doing-enough, Gerry (2025) in https://accentwebs.ie/irish-small-business-website-challenges/, john@secured (2025) in https://hospitalitypartners.ie/top-5-challenges-facing-the-irish-hospitality-industry-in-2025/, and Escape2Project (n.d.) in https://www.escape2project.org/uploads/1/2/3/6/123661942/io1_report_en.pdf ."

AI Response:

"Many small hospitality businesses in Ireland—such as guesthouses, B&Bs, and boutique hotels—struggle with digital adoption because they often rely on outdated, manual processes and lack the resources or expertise to modernize. Research shows that many still depend on phone-based or handwritten bookings and operate with outdated or non-existent websites, even though roughly **80% of reservations now occur online** (McGarr, 2024). This digital gap makes it harder for them to compete with larger, tech-enabled operators.

Several factors contribute to this challenge:

- **Limited digital skills and investment capacity:** Many small operators lack the technical knowledge or budget to upgrade systems or maintain modern websites (Gerry, 2025).
- **Operational pressures:** Owners often manage day-to-day tasks themselves, leaving little time to focus on digital transformation (john@secured, 2025).
- **Slow adoption of online tools:** Despite the growing dominance of online booking platforms, many businesses have been slow to adopt integrated booking engines or digital marketing practices (McGarr, 2024).
- **Broader structural barriers:** Small tourism enterprises across Europe face similar issues, including low digital readiness and limited access to training or support (Escape2Project, n.d.).

Together, these factors create a persistent digital lag that limits visibility, reduces direct bookings, and ultimately affects competitiveness in a market where online presence is now essential."

Used in:

Section 2.2.1

AI-9 — What are the challenges that small hospitality operators are facing?

Prompt:
"Write about the challenges small hospitality operators are facing, mentioning that many still use manual bookings which creates extra work for admin and risks double-bookings, that they have limited booking hours, that they rely heavily on OTAs with high commission fees, include that mobile-friendliness is essential for conversion rates according to Ihttps://www.ihf.ie/suppliers/resources/content/2024/netaffinity/hotel-mobile-barometer-24. Be concise and use bullet points."

AI Response:

"

- Many still rely on **manual booking processes**, creating extra admin work and increasing the risk of **double-bookings**.
- **Limited booking hours** mean missed opportunities, especially as guests increasingly expect 24/7 availability.
- Heavy dependence on **OTAs with high commission fees** reduces margins and makes it harder to grow sustainably.
- OTAs often dominate visibility, making it difficult for small operators to compete for direct bookings.
- Mobile performance is now **critical for conversions**, with industry data showing that mobile-friendly booking experiences are essential for capturing modern travellers (as highlighted by the IHF's Hotel Mobile Barometer 2024). "

Used in:

Section 2.2.3

AI-10 — The GuestEase Guest Persona

Prompt:
"Create a guest persona in their 30s for a small hospitality booking app called GuestEase, mention his name, age, booking reasons, what they value about the booking app. Write 3 user stories as 'As a user, I want…' style. The guest should care about being able to browse rooms, booking easily, getting email notifications, managing the booking online, and leaving reviews. Use bullet points for user stories"

AI Response:

"Alex Murphy (Age 34)

Alex is a frequent GuestEase guest who books stays for weekend getaways, short family trips, and the occasional remote-work break. He values a smooth, stress-free booking experience. What matters most to him is being able to browse rooms easily, compare options, book quickly, receive clear email updates, and manage everything online without needing to call the property. He also likes leaving reviews to help other travellers.

User Stories

- **As a user, I want** to browse and compare rooms with photos and details so I can choose the best option.
- **As a user, I want** to book my stay quickly and easily online so the process feels effortless.
- **As a user, I want** to receive email notifications and manage or update my booking online so I always know what's happening with my reservation."

Used in:

Section 3.1.1

AI-11 — The GuestEase Administrator Persona

Prompt:
"Create an administrator persona in his 50s for a small hospitality business booking app called GuestEase, mention his name, age, reasons to have this app to run their business, what they value about the booking app. Write a few user stories such as 'As a user, I

want…' style. The admin should value managing room availability, reservations, getting email notifications, managing users, and checking guests' feedback"

AI Response:

"Lucas O'Donnell (Age 57)

Lucas owns a small family-run boutique hotel and has been managing it for over two decades. As bookings increasingly move online, he relies on GuestEase to simplify the day-to-day work he used to handle manually. He values having one place where he can update room availability, adjust pricing, manage reservations, and keep track of guest profiles. Clear email notifications help him stay on top of new bookings, cancellations, and changes. He also appreciates being able to read guest feedback easily so he can improve the experience for future visitors.

User Stories

- **As a user, I want** to update room availability and pricing quickly so the information guests see is always accurate.
- **As a user, I want** to manage reservations and user accounts so I can assist guests when needed.
- **As a user, I want** to receive email notifications for new bookings or changes so I can stay organised.
- **As a user, I want** to access guest feedback easily so I can understand what's working and what needs improvement."

Used in:

Section 3.1.2

AI-12 — AI Role (AI copy)

Prompts

"Create the rooms with fancy names rooted in the irish culture and add a field with specific amenities for each room, a couple of them should be for families, a couple for

remote workers, a couple should be luxurious, and so on...pay attention to capacity, update as per the targeted guests....

This is about my b&B GuestEase web app, there is an image of a remote worker. I need a catchy headline and a couple of body lines...
Be more Irish informal like and do the same for a relaxing weekend, there is a couple smiling and sitting on a bed with meditation like stuff

Section 7

AI-13 — AI Role (AI Images)

Prompts
“
Close up of towel in bathroom with shower of Boutique-style suite inspired by Irish...
I want a picture of the same room from a different angle....
Walk-in shower for the above room...
Close-up of the sofa bed...
Bathroom with shower of Boutique-style suite inspired by Irish heritage..
Add a towel with GE initials for GuestEase..
Bathroom with rain shower, meditation inspired...
Bathroom from an higher angle so that the shower fits better into the image, but let me see the sink too.
Ensure it be more rustic...
Show me a cozy small balcony only outside, put 1 small table and 2 chairs, sea view..
Sunset showing the sun being partially covered by the sea...
Close up of Dining table and Kids’ reading nook...
Close-up of the board games...
Add a mini-fridge and microwave retro style...
Different angle closing up to the pour-over coffee kit...
Add the balcony view...
Add a compact rustic bathroom...

Remote worker in a B&B, ultra ultra close up to their laptop and hands on it with a cup of coffee…
Irish relaxing weekend in a B&B, couple close-up sitting on a bed. Focus on the bed and woman and do not show details of the room…
Irish hiking trails experience on cliffs…
Kayaking experience ireland atlantic close-up of kayakers smiling…
Rich and abundant irish complimentary breakfast in the GuestEase guesthouse bed& breakfast

Section 7

AI-14 — AI Role (AI on app architecture)

1. Server-Side Stripe Logic
Prompt:
“Can you walk me through the correct way to run Stripe payment logic on the server side in a Next.js/Supabase setup? I want to avoid exposing secrets on the client. How should I organize the architecture, and which parts belong in API routes versus client components?”

Response:
“To maintain security, all Stripe operations that rely on secret keys must execute on the server. The recommended architecture is to expose server-side endpoints responsible for creating payment intents, validating sessions, and handling webhooks. Client components should only request these values and never interact with Stripe directly. This separation prevents key exposure, ensures compliance with Stripe’s security model, and centralises payment logic in a controlled environment.”

2. Backend Architecture for Booking CRUD
Prompt:

“I’m designing a booking system and would like to figure out where the CRUD logic should live. Supabase allows client-side operations too, but would like to use some backend. Should I implement the booking creation, update, and deletion logic on the backend instead, like Node Express?”

Response:
“Although Supabase supports client-side CRUD, implementing booking logic on the backend provides stronger guarantees around validation, access control, and business-rule enforcement. A backend-first structure allows you to define predictable workflows and prevents clients from bypassing constraints. A clean approach is to expose dedicated endpoints for create, update, and delete operations, each performing validation and permission checks before interacting with the database.”

3. Validating the booking logic on the backend
Prompt:
“Can you help me confirm whether moving booking logic to the backend is the right choice? Data integrity should be the priority and avoid exposing too much logic to the client is too. What are the risks of doing this on the frontend even if Supabase makes it possible?”

Response:
“Centralising booking logic on the backend is the more robust option. Frontend-side CRUD increases the risk of inconsistent validation, unauthorised modifications, and exposure of internal rules. Even with row-level security, distributing logic across clients makes long-term maintenance more difficult. A backend-first approach ensures that all operations pass through a single, controlled layer where rules can be enforced reliably.”

4. Why React Query for fetching data?
Prompt:

“I’m trying to decide whether to use React Query for my fetching data. Is React Query with mutations better than alternatives like manual fetch calls or global state libraries? My goal is to achieve cache consistency and UI synchronization.”

Response:
“React Query is well-suited for applications that depend on frequently changing server data. It abstracts away the complexity of caching, background refetching, and synchronising UI state with remote data sources. Compared to manual fetch calls or global state libraries, React Query provides built-in mechanisms for handling stale data, retry logic, and automatic updates, significantly reducing the amount of custom code required.”

5. React Query Mutations vs. Other Approaches
Prompt:
“When performing CRUD operations, should I use React Query mutations or handle these differently? Can you compare the approaches and explain how React Query helps with cache invalidation and UI synchronization with server state?”

Response:
“React Query mutations integrate directly with the query cache, enabling automatic invalidation or targeted updates after a successful operation. This ensures that the UI reflects the latest server state without manual intervention. Handling mutations manually requires explicit refetching or state updates, which increases the likelihood of stale data. React Query’s mutation system standardises this workflow and reduces the risk of inconsistencies.”

6. General Architectural Validation
Prompt:

“My app has Stripe integration, Supabase Auth and database with PostgreSQL, booking CRUD run through Node Express backend, and React Query and mutations data-fetching on the frontend with Material UI. How effective and performant is my architectural choice?”

Response:
“The overall architecture aligns with backend-first principles. Centralising payment logic, booking operations, and business rules on the server improves security, consistency, and maintainability. Pairing this with React Query on the frontend ensures reliable synchronisation between client and server. Potential pitfalls include over-fetching, insufficient error handling, and mixing business logic into client components. Maintaining clear boundaries between layers mitigates these issues.”

Section 7