Airbnb Clone

A PROJECT REPORT

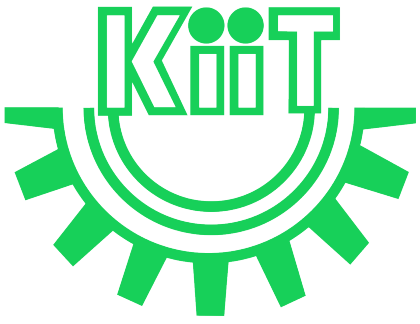
submitted by

VASU DUTT 20051608
SUVANGI PAUL 20051298
ABHINAV YADAV 20051559
RONIT KAMILLA 20051606

to
KIIT Deemed to be University

in partial fulfilment of the requirements for the award of the Degree
of
Bachelor of Technology
in
Computer Science and Engineering

UNDER THE GUIDANCE OF
DR. JAYANTA MONDAL



KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

BHUBANESWAE, ODISHA - 751024

April 2023

Airbnb Clone

A PROJECT REPORT

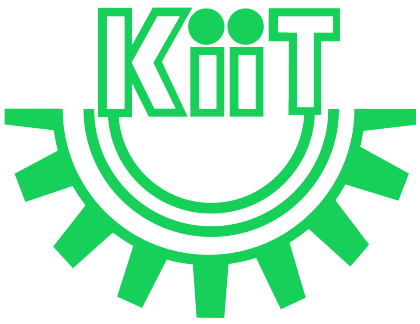
submitted by

VASU DUTT 20051608
SUVANGI PAUL 20051298
ABHINAV YADAV 20051559
RONIT KAMILLA 20051606

to
KIIT Deemed to be University

in partial fulfilment of the requirements for the award of the Degree
of
Bachelor of Technology
in
Computer Science and Engineering

UNDER THE GUIDANCE OF
DR. JAYANTA MONDAL



SCHOOL OF COMPUTER ENGINEERING

CONTENTS

ACKNOWLEDGEMENT

We extend our profound gratitude to **DR. JAYANTA MONDAL** of **KIIT School of Computer Engineering** for his expert guidance and continuous encouragement throughout the project. His invaluable insights and unwavering support have been instrumental in ensuring the project's success. We appreciate his dedication, mentorship, and prompt responsiveness to our queries. Dr. Jayanta Mondal's expertise and guidance have significantly contributed to our growth and learning experience. We are sincerely grateful for his invaluable contributions.

VASU DUTT
SUVANGI PAUL
ABHINAV YADAV
RONIT KAMILLA

ABSTRACT

This report presents the development and implementation of a full stack application that serves as a functional clone of the popular website Airbnb. The application allows users to list their properties for rent, make reservations for accommodations, and provides various filtering options based on location, room count, bathroom count, and guest count. The project utilizes modern web development technologies, including NextJS, Typescript, Prisma, TailwindCSS, MongoDB, and Cloudinary. The application employs both server-side and client-side rendering for optimal performance. Additionally, NextAuth is integrated to provide users with authentication options, including traditional email and password credentials as well as social login using Google or GitHub accounts.

In this project, we have successfully developed a full stack application that closely replicates the core functionality and user experience of Airbnb. The application offers users the ability to list their properties for rent and make reservations for accommodations, all while providing a user-friendly interface that matches the design of the original Airbnb website. By leveraging NextJS, Typescript, Prisma, TailwindCSS, MongoDB, and Cloudinary, we have ensured a robust and scalable application that meets the demands of a real-world marketplace. The use of server-side and client-side rendering techniques enhances performance, while NextAuth enables secure authentication with multiple login options. This project demonstrates proficiency in web development technologies, database design, and UI/UX implementation.

Keywords: full stack application, Airbnb clone, NextJS, Typescript, Prisma, TailwindCSS, MongoDB, Cloudinary, server-side rendering, client-side rendering, NextAuth, authentication, UI/UX design, database design.

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In today's digital age, the online marketplace for short-term accommodations has experienced significant growth. Platforms like Airbnb have revolutionized the way people find and rent properties, providing a convenient and personalized experience. The purpose of this project is to develop a full stack application that replicates the core functionality and user experience of Airbnb, catering to the increasing demand for such platforms.

The current available solutions for property rentals often lack the flexibility, customization, and seamless user experience that users desire. While there are existing platforms like Airbnb that offer a wide range of properties for rent, they may not fully cater to specific user preferences or provide a user interface that closely matches the original website. This project aims to bridge these gaps by offering a functional clone of Airbnb that closely mimics its features, design, and user experience.

The significance of this project lies in its ability to provide a comprehensive solution for property rentals, offering users a platform that matches the familiarity and functionality of Airbnb. By developing a full stack application, we can address the limitations of current solutions and provide users with a more tailored experience. The project's focus on responsive design, advanced filtering options, and integration of popular web development technologies ensures an enhanced user experience and improved efficiency for both property owners and renters.

This report will provide an overview of the project's objectives, the technologies employed, and the database design. We will discuss the implementation details, including the utilization of NextJS, Typescript, Prisma, TailwindCSS, MongoDB, and Cloudinary. Additionally, we will delve into the integration of NextAuth for secure user authentication and social login options. The report will also highlight

the significance of this project in meeting the current market demands and address the gaps present in existing solutions.

By the end of this report, readers will have a clear understanding of the development process, the technologies utilized, and the overall contribution of this project to the field of online property rentals.

CHAPTER 2

BASIC CONCEPTS

In this section, we will introduce the basic concepts related to the tools and techniques used in this project. These concepts are essential for understanding the implementation and functioning of the full stack application. Each subsection provides a description of a specific concept.

2.1 NextJS

NextJS is a popular React framework that allows for efficient server-side rendering (SSR) and client-side rendering (CSR). SSR generates HTML on the server before sending it to the client, resulting in faster initial page loads and improved search engine optimization. CSR, on the other hand, renders pages on the client side, enabling dynamic and interactive user experiences. NextJS also provides features like automatic code splitting, route prefetching, and built-in API routes, making it an ideal choice for building modern web applications.

```
import getCurrentUser from '@/actions/getCurrentUser';
import getListings, { IListingsParams } from
↪  '@/actions/getListings';
import Container from '@/components/Container';
import EmptyState from '@/components/EmptyState';
import ListingCard from '@/components/listings/ListingCard';

interface HomeProps {
  searchParams?: IListingsParams;
}

const Home = async ({ searchParams }: HomeProps) => {
  const listings = await getListings(searchParams);
  const currentUser = await getCurrentUser();

  if (listings.length === 0) {
    return <EmptyState showReset />;
  }

  return (
    <Container>
      <div>
        <div className="grid grid-cols-1 gap-8 pt-24 sm:grid-cols-2
        ↪  md:grid-cols-3 lg:grid-cols-4 xl:grid-cols-5
        ↪  2xl:grid-cols-6">
          {listings.map((listing: any) => (
            <ListingCard
              currentUser={currentUser}
              key={listing.id}
              data={listing}
            />
          ))}
        </div>
      </div>
    </Container>
  );
};

export default Home;
```

Figure 2.1: Home page built with Next.js

2.2 Typescript

Typescript is a strongly typed superset of JavaScript that adds static typing to the language. By enforcing strict type-checking, Typescript helps catch poten-

tial errors during development and improves code maintainability. It provides a rich set of features such as type annotations, interfaces, generics, and modules, enabling developers to write more robust and scalable applications. Typescript also provides excellent editor support and helps enhance productivity by providing accurate code hints and refactoring tools.

```
import { Listing, Reservation, User } from '@prisma/client';

export type SafeListing = Omit<Listing, 'createdAt'> & {
  createdAt: string;
};

export type SafeReservation = Omit<
  Reservation,
  'createdAt' | 'startDate' | 'endDate' | 'listing'
> & {
  createdAt: string;
  startDate: string;
  endDate: string;
  listing: SafeListing;
};

export type SafeUser = Omit<
  User,
  'createdAt' | 'updatedAt' | 'emailVerified'
> & {
  createdAt: string;
  updatedAt: string;
  emailVerified: string | null;
};
```

Figure 2.2: Creating safe type definitions for User, Listing and Reservation entities

2.3 Prisma

Prisma is an open-source ORM (Object-Relational Mapping) tool that simplifies database connectivity and data management. It provides a type-safe and auto-generated query builder that enables seamless interaction with the database. Prisma supports multiple databases, including PostgreSQL, MySQL, and SQLite, and offers features like data modeling, schema migrations, and advanced filtering capabilities. By abstracting away the complexities of database interactions, Prisma improves developer productivity and ensures secure and optimized database operations.

```
generator client {
  provider = "prisma-client-js"
  }

  datasource db {
  provider = "mongodb"
  url      = env("DATABASE_URL")
  }

  model User {
  id String @id @default(auto()) @map("_id") @db.ObjectId
  name String?
  email String? @unique
  emailVerified DateTime?
  image String?
  hashedPassword String?
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  favoriteIds String[] @db.ObjectId

  accounts Account[]
  listings Listing[]
  reservations Reservation[]
  }
```

Figure 2.3: Prisma schema defining the User model

2.4 TailwindCSS

TailwindCSS is a utility-first CSS framework that allows developers to rapidly build custom user interfaces. It provides a set of pre-defined utility classes that can be combined to create complex layouts and designs without writing custom CSS. TailwindCSS emphasizes a responsive-first approach, enabling the creation of responsive designs that adapt to different screen sizes. It also supports customization through configuration files, making it highly flexible and suitable for designing the user interface of the full stack application.

```tsx
'use client';

import { IconType } from 'react-icons';

interface ButtonProps {
  label: string;
  onClick: (e: React.MouseEvent<HTMLButtonElement>) => void;
  disabled?: boolean;
  outline?: boolean;
  small?: boolean;
  icon?: IconType;
}

const Button: React.FC<ButtonProps> = ({
  label,
  onClick,
  disabled,
  outline,
  small,
  icon: Icon,
}) => {
  return (
    <button
      disabled={disabled}
      onClick={onClick}
      className={` relative disabled:opacity-70
      ↪  disabled:cursor-not-allowed rounded-lg hover:opacity-80
      ↪  transition w-full
        ${outline ? 'bg-white' : 'bg-rose-500'}
        ${outline ? 'border-black' : 'border-rose-500'}
        ${outline ? 'text-black' : 'text-white'}
        ${small ? 'text-sm' : 'text-md'}
        ${small ? 'py-1' : 'py-3'}
        ${small ? 'font-light' : 'font-semibold'}
        ${small ? 'border-[1px]' : 'border-2'}
      `}
    >
      {Icon && <Icon size={24} className="absolute left-4 top-3"
      ↪  />}
      {label}
    </button>
  );
};

export default Button;
```

Figure 2.4: Button component applying default and conditional styles with TaiwlwindCSS

2.5 MongoDB

MongoDB is a popular NoSQL database that offers high performance, scalability, and flexibility. It stores data in JSON-like documents, providing a schema-less structure that allows for easy handling of unstructured and semi-structured data. MongoDB's document model and flexible querying capabilities make it well-suited for applications with evolving data requirements. It also supports horizontal scaling, replication, and sharding, ensuring reliable and efficient data storage and retrieval.

```
export default async function getReservations(params: IParams) {
  try {
    const { listingId, userId, authorId } = params;

    const query: any = {};

    if (listingId) {
      query.listingId = listingId;
    }

    if (userId) {
      query.userId = userId;
    }

    if (authorId) {
      query.listing = { userId: authorId };
    }

    const reservations = await prisma.reservation.findMany({
      where: query,
      include: {
        listing: true,
      },
      orderBy: {
        createdAt: 'desc',
      },
    });

    const safeReservations = reservations.map((reservation) => ({
      ...reservation,
      createdAt: reservation.createdAt.toISOString(),
      startDate: reservation.startDate.toISOString(),
      endDate: reservation.endDate.toISOString(),
      listing: {
        ...reservation.listing,
        createdAt: reservation.listing.createdAt.toISOString(),
      },
    }));

    return safeReservations;
  } catch (error: any) {
    throw new Error(error);
  }
}
```

Figure 2.5: Fetching reservations from MongoDB based on listingId or userId or authorId

2.6 Cloudinary

Cloudinary is a cloud-based media management platform that simplifies the storage, transformation, and delivery of media files. It offers a scalable and reliable infrastructure for hosting images, videos, and other media assets. Cloudinary provides features like automatic image optimization, resizing, cropping, and format conversion, enabling efficient delivery of media files across different devices and network conditions. Integrating Cloudinary into the full stack application ensures optimal performance and a seamless media viewing experience for users.

```
<CldUploadWidget
  onUpload={handleUpload}
  uploadPreset={uploadPreset}
  options={{
    maxFiles: 1,
  }}
>
  {({ open }) => {
    return (
      <div
        onClick={() => open?.()}
        className="relative flex flex-col items-center
        ↪  justify-center gap-4 p-20 transition border-2
        ↪  border-dashed cursor-pointer hover:opacity-70
        ↪  border-neutral-300 text-neutral-600"
      >
        <TbPhotoPlus size={50} />
        <div className="text-lg font-semibold">
          Click to upload
        </div>
        {value && (
          <div className="absolute inset-0 w-full h-full ">
            <Image
              fill
              style={{ objectFit: 'cover' }}
              src={value}
              alt="House"
            />
          </div>
        )}
      </div>
    );
  }}
</CldUploadWidget>
```

Figure 2.6: Cloudinary react component being used to upload media to Cloudinary servers

2.7 NextAuth

NextAuth is an authentication library for NextJS applications that provides a simple and extensible solution for user authentication. It supports various authentication providers, including traditional email and password login, as well as social login options through platforms like Google and GitHub. NextAuth handles the authentication flow, session management, and secure storage of user credentials, ensuring a robust and secure authentication mechanism for the full stack

application.

By understanding these basic concepts, readers will gain a solid foundation for comprehending the implementation and functionality of the full stack application. These concepts play a crucial role in the successful development of the project and enable users to appreciate the technical aspects behind its design and functionality.

Throughout the development of the full stack application, NextJS serves as the core framework, providing the necessary tools and features for efficient server-side and client-side rendering. Typescript enhances the development process by enforcing strict type-checking, improving code quality, and reducing potential runtime errors. Prisma simplifies database connectivity and management, allowing for seamless interaction with the database and efficient data operations.

TailwindCSS enables rapid UI development by providing a utility-first approach, where pre-defined utility classes can be combined to create customized and responsive user interfaces. MongoDB, a flexible and scalable NoSQL database, efficiently stores and retrieves data, making it ideal for handling the dynamic nature of property listings and reservations. Cloudinary, as a cloud-based media management platform, ensures the efficient storage and delivery of media files, enhancing the overall user experience.

NextAuth handles user authentication, offering multiple login options such as email and password credentials, as well as social login through platforms like Google and GitHub. This secure authentication mechanism ensures that users can access and interact with the full stack application safely.

Understanding these basic concepts is essential for readers to grasp the technical foundation of the project. They provide insights into the utilization of the chosen technologies and highlight their importance in addressing the challenges faced by existing solutions in the market.

In the subsequent sections of this report, we will delve into the implementation details of the full stack application, including the architecture, database design, user interface, and functionality. By examining these aspects, readers will gain a comprehensive understanding of the project's development process and its contribution to the field of online property rentals.

```ts
export const authOptions: AuthOptions = {
  adapter: PrismaAdapter(prisma),
  providers: [
    GithubProvider({
      clientId: process.env.GITHUB_ID as string,
      clientSecret: process.env.GITHUB_SECRET as string,
    }),
    GoogleProvider({
      clientId: process.env.GOOGLE_CLIENT_ID as string,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET as string,
    }),
    CredentialsProvider({
      name: 'credentials',
      credentials: {
        email: { label: 'email', type: 'text' },
        password: { label: 'password', type: 'password' },
      },
      async authorize(credentials) {
        if (!credentials?.email || !credentials?.password) {
          throw new Error('Invalid credentials');
        }

        const user = await prisma.user.findUnique({
          where: {
            email: credentials.email,
          },
        });

        if (!user || !user?.hashedPassword) {
          throw new Error('Invalid credentials');
        }

        const isCorrectPassword = await bcrypt.compare(
          credentials.password,
          user.hashedPassword
        );

        if (!isCorrectPassword) {
          throw new Error('Invalid credentials');
        }

        return user;
      },
    }),
  ]
};
```

Figure 2.7: Next-auth snippet defining social and credential authentication options

CHAPTER 3

PROBLEM STATEMENT

The problem statement for this project revolves around the limitations and gaps present in current available solutions for online property rentals. While platforms like Airbnb have successfully disrupted the market and provided users with a convenient way to find and rent accommodations, there are still areas that can be improved upon. The goal of this project is to develop a full stack application that addresses these limitations and provides a functional clone of Airbnb with enhanced features and user experience.

The following subsections present the Software Requirements Specification (SRS) according to the IEEE format, outlining the project planning, analysis, and system design.

3.1 Project Planning

The planning phase of the project involves defining the steps to be followed for successful execution of the development process. The following list of requirements and features outline the key aspects to be developed in the full stack application:

1. **User Registration and Authentication:** Users should be able to register and authenticate using email and password credentials or through their existing Google or GitHub accounts.

2. **Property Listing:** Users should be able to list their properties for rent, providing details such as title, description, images, location, room count, bathroom count, and guest count.

3. **Property Search and Filtering:** Users should be able to search for properties based on location, room count, bathroom count, guest count, and other

criteria. Advanced filtering options should be provided to refine search results.

4. **Property Reservation:** Users should be able to make reservations for properties, specifying the start and end dates of their stay.

5. **Favorite Properties:** Users should be able to add properties to their list of favorites.

6. **User Profile Management:** Users should have the ability to manage their profiles, including updating their own listings, modifying the favorites list and viewing their past reservations.

7. **Responsive UI/UX:** The user interface should be designed to be responsive, ensuring optimal viewing and interaction across different devices and screen sizes.

3.2 Project Analysis

After collecting the requirements and conceptualizing the problem statement, a thorough analysis needs to be performed to identify any ambiguities, mistakes, or inconsistencies. This analysis phase ensures a clear understanding of the project scope and requirements, minimizing potential risks and challenges during the development process.

3.3 System Design

3.3.1 Design Constraints

The design constraints for the full stack application involve the utilization of specific technologies and frameworks to ensure compatibility and optimal performance. The following constraints are considered:

1. **NextJS:** The application should be built using NextJS, leveraging its server-side rendering and client-side rendering capabilities.

2. **Typescript:** Typescript should be used to enforce static typing and improve code quality.

3. **Prisma:** The application should utilize Prisma as the ORM for database connectivity and management.

4. **TailwindCSS:** The UI should be designed using TailwindCSS, taking advantage of its utility-first approach for rapid development.

5. **MongoDB:** MongoDB should be used as the database for efficient storage and retrieval of property and user-related data.

6. **Cloudinary:** Cloudinary should be integrated to handle media file storage and delivery.

3.3.2 System Architecture

The system architecture for the full stack application involves the integration of various components and technologies. The architecture should follow a modular and scalable approach, allowing for easy maintenance and future enhancements. Key components of the system architecture include:

1. **NextJS Framework:** The core framework for server-side rendering, client-side rendering, and routing.

2. **Prisma ORM:** Facilitates database connectivity, query building, and data modeling.

3. **MongoDB Database:** Stores property and user-related data in a flexible and scalable manner.

4. **TailwindCSS:** Enables rapid UI development through utility classes and responsive design.

5. **Cloudinary:** Manages storage and delivery of media files, optimizing performance.

The deployment of the full stack application was done on Vercel, a cloud platform that specializes in hosting static sites and serverless functions. Vercel simplifies the deployment process and ensures scalability, reliability, and efficient updates through its robust continuous integration and continuous deployment (CI/CD) pipelines.

By adopting this system architecture, the full stack application is designed to be modular, scalable, and optimized for performance. The integration of NextJS, Prisma, MongoDB, TailwindCSS, and Cloudinary provides a comprehensive and efficient solution for property listing, search, reservation, and user profile management.

Through proper planning, analysis, and system design, the development process of the full stack application is well-structured and lays a solid foundation for successful implementation. The subsequent sections of this report will delve into the implementation details, showcasing how the project's objectives and requirements are translated into a fully functional and user-friendly application.

CHAPTER 4

IMPLEMENTATION

This section presents the implementation details of the full stack application developed during the project. It includes the methodology or proposal followed, the testing or verification plan, screenshots showcasing the output, and any relevant quality assurance measures taken.

4.1 Methodology

For the development of this project, a structured approach was adopted, which involved the following steps:

1. **Requirement Gathering:** The initial phase involved gathering and analyzing the requirements for the full stack application. This included identifying the essential features, user interface design, and database structure.

2. **System Design:** Based on the gathered requirements, a system design was formulated. This included the architectural design, database schema design, and UI/UX wireframing. The design phase focused on ensuring modularity, scalability, and a seamless user experience.

3. **Implementation:** The development phase involved coding the application using NextJS, Typescript, Prisma, TailwindCSS, MongoDB, and Cloudinary. Throughout the implementation, ESLint was integrated into the codebase to enforce coding standards and maintain consistency. ESLint rules and plugins were configured to check for errors, stylistic inconsistencies, and adherence to best practices.

4. **Testing:** Although there was no dedicated QA team, testing was performed

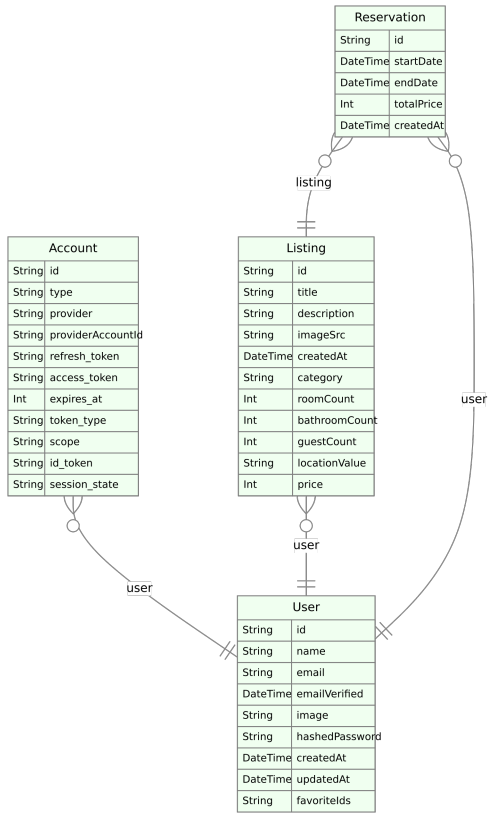


Figure 4.1: ER Diagram

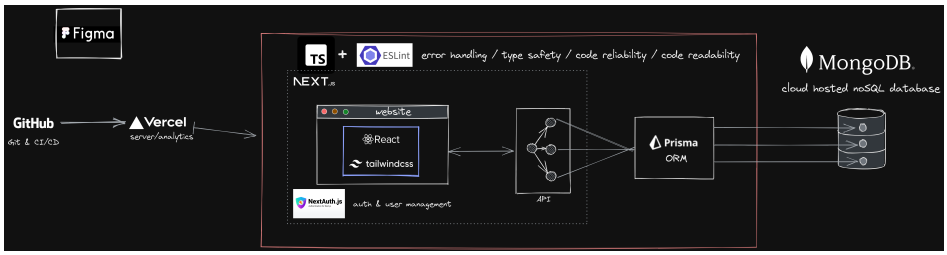


Figure 4.2: System Architecture diagram

during the implementation phase. While unit testing and integration testing were not explicitly conducted, the application's functionality was manually tested to ensure its correctness and robustness.

5. **Deployment:** Once the application was deemed stable and fully functional, it was deployed to a production environment. Continuous integration and continuous deployment (CI/CD) pipelines were set up to automate the deployment process, ensuring smooth and efficient updates.

4.2 Testing Plan

Manual testing was performed to verify the outcome of the project. The testing plan included the following aspects:

1. **Functionality Testing:** The application's features and functionalities were tested to ensure they worked as intended. This involved manually executing test cases that covered different user scenarios and workflows.

2. **User Interface Testing:** The user interface was tested for responsiveness and visual consistency across different devices and screen sizes.

3. **Usability Testing:** The application's usability and user experience were evaluated by soliciting feedback from users and incorporating their suggestions for improvements.

4. **Error Handling Testing:** The application's error handling capabilities were tested by intentionally triggering various error scenarios and ensuring that appropriate error messages or fallbacks were displayed to the users.

5. **Performace Testing:** The application's performance was evaluated by simulating high user loads and stress testing. This ensured that the application could handle a significant number of concurrent users without performance degradation.

While unit testing and integration testing were not explicitly carried out, the manual testing approach aimed to identify and address any issues or bugs during the implementation phase.

4.3 Screenshots

The following screenshots provide a visual representation of the output and user interface of the implemented full stack application:

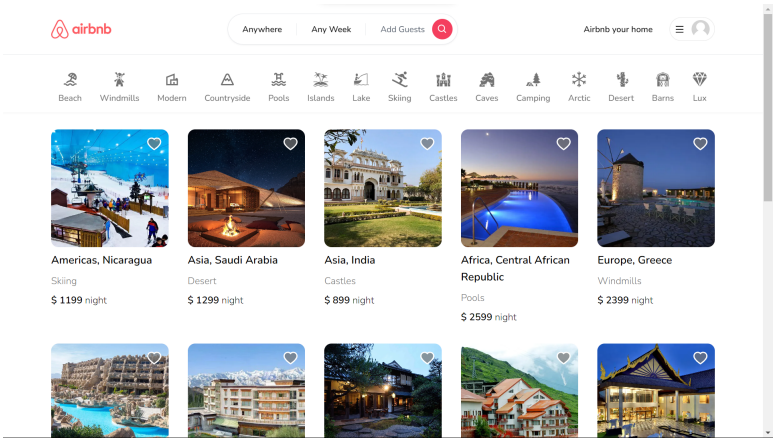


Figure 4.3: Home page of the website

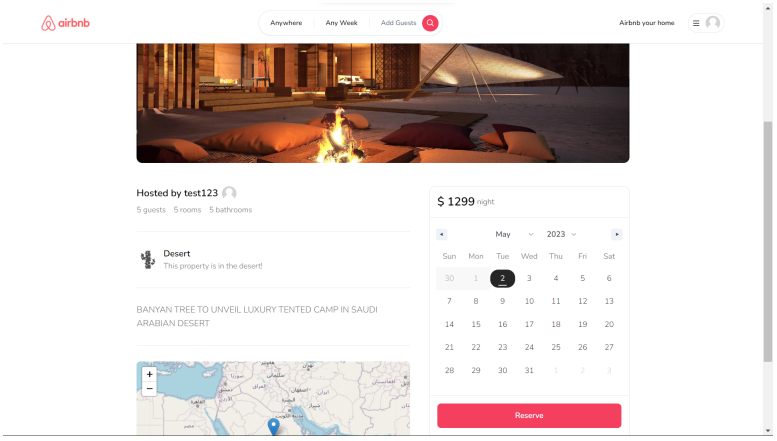


Figure 4.4: Page displaying information about a single listing

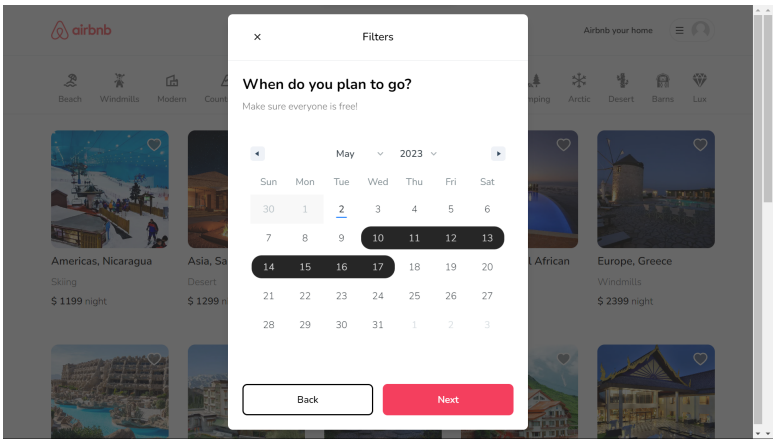


Figure 4.5: Modal to apply filters on properties list

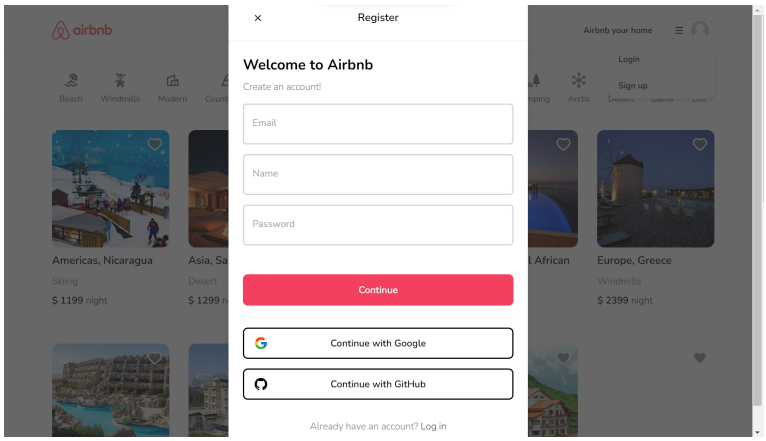


Figure 4.6: Modal to register as a new user

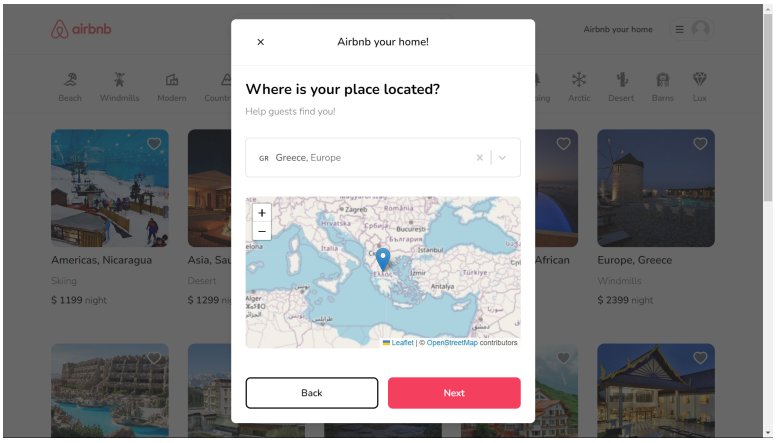


Figure 4.7: Listing creation flow

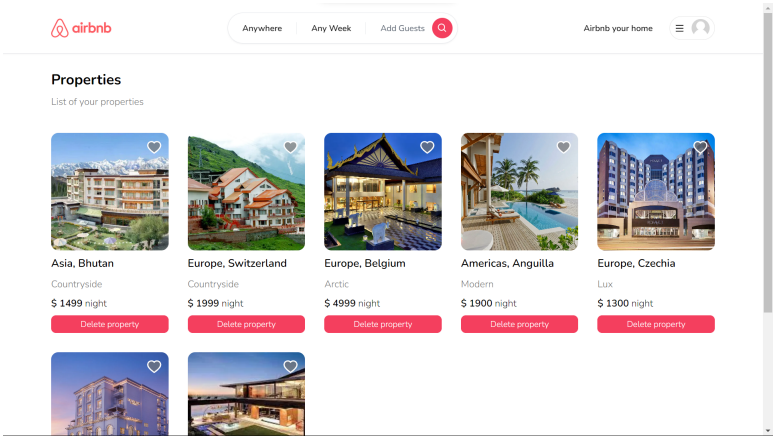


Figure 4.8: Property management page

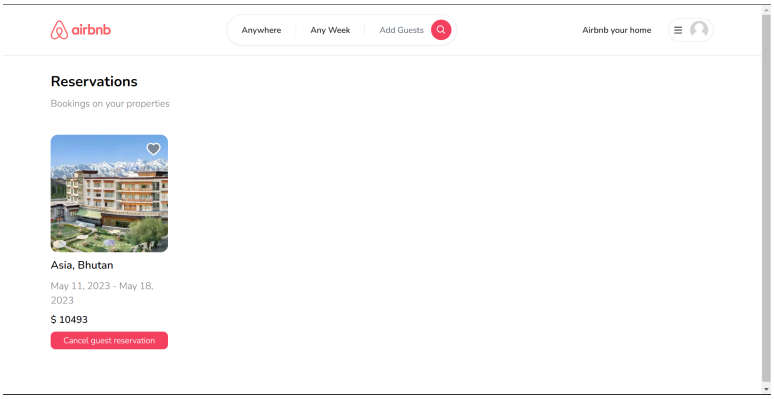


Figure 4.9: Reservations management page

4.4 Quality Assurance

Although there was no dedicated QA team involved, quality assurance measures were taken during the implementation phase. These measures included the integration of ESLint into the codebase. ESLint was used to enforce coding standards, maintain code consistency, and identify potential errors or stylistic issues. By configuring ESLint with appropriate rules and plugins, the codebase was thoroughly checked for readability, maintainability, and correctness.

The implementation phase encompassed a structured approach, manual testing for functionality and user experience, visual representation through screenshots, and the integration of ESLint for maintaining code quality and consistency. These steps contributed to the successful development of the full stack application, resulting in a functional and user-friendly solution.

The deployed full stack application can be accessed at:
rent-application-example.vercel.app

CHAPTER 5

STANDARDS ADOPTED

This chapter discusses the standards adopted during the development of the full stack application, including design standards, coding standards, and testing standards.

5.1 Design Standards

In the field of software engineering, adhering to design standards is crucial for creating robust and maintainable applications. While there are various design standards available, the following recommended practices were followed for the project:

- **Use of UML Diagrams:** Unified Modeling Language (UML) diagrams were utilized to visually represent the system architecture, data models, and relationships between different components.
- **Consistent User Interface Design:** The user interface design followed industry best practices and aimed to provide a seamless and intuitive user experience. Consistency in terms of layout, color schemes, and typography was maintained to ensure familiarity and usability for users.
- **Database Design Standards:** The database schema design followed standard practices to ensure data integrity, normalization, and efficient querying. The structure of the database entities and relationships between them were carefully defined based on the requirements of the application.

5.2 Coding Standards

Coding standards play a vital role in producing clean, readable, and maintainable code. During the development of the full stack application, the following coding standards and best practices were adhered to:

1. **ESLint Integration:** ESLint, a popular static code analysis tool, was integrated into the development workflow. ESLint helped enforce coding standards, detect potential errors, and ensure consistency throughout the codebase. Custom ESLint rules were configured to align with industry best practices.

2. **TypeScript Usage:** TypeScript, a statically typed superset of JavaScript, was utilized in the project. TypeScript's strict type-checking capabilities helped catch errors early and improve code reliability. It enforced type annotations, making the codebase more robust and maintainable.

3. **Naming Conventions:** Appropriate naming conventions were followed to ensure clarity and readability of the code. Descriptive and meaningful names were used for variables, functions, and components, following standard naming conventions such as camel case or Pascal case.

4. **Code Organization:** The codebase was structured in a modular and organized manner. Functions and classes were kept concise and focused on a single task, promoting reusability and maintainability. Indentation and formatting were applied consistently to improve code readability.

5.3 Testing Standards

While there was no explicit testing carried out in this project, it is important to acknowledge that there are established standards for quality assurance and testing in software development. Common standards followed for testing and verification include ISO/IEC/IEEE 29119, which provides guidelines for test processes, test documentation, and test techniques. While these specific standards were not implemented in this project, it is recognized that testing and verification are integral components of software development projects.

By adhering to design standards, utilizing tools like ESLint and TypeScript to enforce coding standards, and recognizing the importance of testing standards, the project aimed to maintain high-quality software development practices. These

standards contribute to the overall quality, reliability, and maintainability of the full stack application.

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

In conclusion, the development of the Airbnb clone full stack application has been successfully completed. The project aimed to replicate the core functionalities and user experience of the original Airbnb website. By utilizing modern technologies such as NextJS, TypeScript, Prisma, TailwindCSS, MongoDB, and Cloudinary, a responsive and feature-rich application was implemented.

Throughout the project, various key objectives were achieved. The user interface and user experience closely match that of the original Airbnb website, providing a familiar and intuitive platform for users to list and rent properties. The use of NextJS and intelligent rendering techniques ensured optimal performance and a seamless browsing experience. Integration with NextAuth enabled users to sign up and authenticate using email credentials or their existing Google or Github accounts.

The project also followed industry-standard coding practices, incorporating ES-Lint and TypeScript to enforce coding standards and maintain code quality. The database schema design adhered to best practices, ensuring efficient data storage and retrieval.

Overall, the Airbnb clone project has demonstrated the ability to develop a functional and responsive full stack application that replicates the core features of the original Airbnb website.

6.2 Future Scope

While the project has met the defined objectives, there are several areas that can be further enhanced and expanded in the future:

1. **User Reviews and Ratings:** Implementing a user review and rating system would enhance the trust and credibility of the platform. Users can leave reviews and ratings for properties they have rented, providing valuable feedback to both hosts and potential renters.

2. **Messaging System:** Introducing a messaging system between hosts and guests would facilitate communication and streamline the booking process.

3. **Payment Integration:** Integrating a secure and reliable payment gateway would allow users to make online payments for property reservations. This would provide a seamless booking experience and ensure secure transactions.

4. **Mobile Application:** Developing a mobile application for the Airbnb clone would extend the reach and accessibility of the platform, allowing users to access and interact with the application on their mobile devices.

5. **Localization and Internationalization:** Implementing localization and internationalization features would enable the application to support multiple languages and currencies, catering to a global user base.

In conclusion, the Airbnb clone project has laid a solid foundation for a fully functional and user-friendly platform. With the potential future enhancements mentioned above, the application can be further improved to meet the evolving needs of users and provide a comprehensive solution for property rental services.