CSE471 : System Analysis and Design

Project Report

Project Title : Peer to Peer Accommodation Booking Platform (HomieStay)

Introduction

The world of travel and accommodation has evolved with the advent of technology, giving rise to platforms that enable individuals to connect directly for renting accommodations. In response to this trend, we have developed a **Peer to Peer Accommodation Booking Platform (HomieStay)**. This platform offers a seamless experience for both hosts looking to rent out their properties and travelers seeking unique and personalized stays.

This project is a culmination of our efforts in system analysis, design, frontend and backend development, and the application of modern technologies. The Peer to Peer Accommodation Booking Platform (HomieStay) caters to the growing demand for alternative accommodations beyond traditional hotels. Hosts can list their properties, set availability, and manage bookings, while travelers can browse and book accommodations directly from hosts.

The Peer to Peer Accommodation Booking Platform covers the following aspects:

- User-friendly interfaces for hosts to list properties and travelers to search and book accommodations.
- A secure authentication system that ensures the safety of user data.
- Efficient algorithms for matching travelers with suitable accommodations based on preferences.
- Booking management and communication features that streamline the interaction between hosts and travelers.
- Interactive maps to provide visual information about accommodation locations.

However, certain aspects are out of scope for this project:

- Advanced property management features beyond listing and availability management.
- Payment integration to facilitate secure and hassle-free transactions.
- Legal and financial considerations related to property rentals and tax implications.
- Localization for different languages and regional preferences.
- Integration with external travel services beyond basic accommodation booking.

In this report, we present an overview of our Peer to Peer Accommodation Booking Platform, detailing its functional requirements, user manual, frontend and backend development, technology stack, and individual contributions.

Functional Requirements

Requirement 1: User Authentication

Feature 1: Sign Up

- Implement a user registration process that collects user details.

Feature 2: Log In

- Develop a secure login system that verifies user credentials.

- Prevent unauthorized access through authentication mechanisms.

Feature 3: Third-Party OAuth Logins
- Allow users to log in using their Google/Gmail or GitHub accounts through OAuth authentication.
- Integrate OAuth authentication flow to securely authenticate users without storing their sensitive credentials.
- Link the authenticated third-party account with the existing user profile in the application.

Feature 4: Enhanced Security Authentication
- The platform uses JWT for secure transmission of information. This ensures that the data passed between the server and client is authenticated and can be trusted.
- User passwords are never stored in plain text. Instead, they are hashed using advanced cryptographic algorithms before storage, adding an extra layer of security against potential breaches.

Requirement 2: Property Management (Property Listings)

Feature 1: Create Listings
- Develop a form for users to create new property listings.
- Capture essential details such as property type, location, price, and availability dates.

Feature 2: Image Upload
- Enable hosts to upload images of their properties to showcase them.
- Enhance the visual representation of listings with images.

Feature 3: Delete Listings
- Provide hosts with the ability to remove their property listings.
- Ensure deleted listings are promptly removed from the platform.

Feature 4: Search Listings
- Implement a search functionality that allows users to search for properties.

- Include filters based on location, availability dates, and other attributes.

Feature 5: Favorite Listings
- Enhance the Search Listings feature by allowing users to mark properties as favorites.
- Provide users with a way to keep track of properties they are interested in.
- Implement a "Favorites" section where users can easily access and manage their saved listings.

Requirement 3: Property Reservation

Feature 1: Reserve Property
- Develop a process for guests to reserve available properties for specific dates.
- Capture reservation details, including property, dates, and user information.

Feature 2: Cancel Booking
- Allow both hosts and guests to cancel bookings under certain conditions.
- Notify both parties when a booking is canceled.

Feature 3: Calendar Management
- Provide hosts with a calendar view to manage property availability.
- Update availability dates based on reservations.

Feature 4: Availability Calendar
- Extend the Calendar Management feature with an availability calendar for each property.
- Display a visual representation of property availability, showing dates as colored blocks.
- Provide an intuitive way for hosts to quickly see which dates are booked and which are available.

Feature 5: Individual Property Display

- Each property listing has a unique URL that can be accessed directly to view the specific property in detail.
- Users have the ability to reserve the property directly from this page.

Requirement 4: Property Location Display

Feature 1: Property Location
- Display property locations accurately or approximately in property listings.
- Enhance user understanding of property proximity.

Feature 2: Search by Location
- Enable users to search for properties based on location through a search function.
- Match user queries with property locations.

Feature 3: Map View
- Incorporate map views to visually showcase property locations.
- Enhance user experience with an interactive map interface.

Requirement 5: User Experience and Support

Feature 1: Easy Navigation
- The platform is designed with an intuitive UI that guides users seamlessly through different sections, ensuring that even first-time visitors navigate effortlessly.
- Essential sections, such as property listings, user profiles, and booking pages, are clearly marked with noticeable icons and descriptive labels, reducing the need for users to search extensively.

Feature 2: Adaptability
- The platform seamlessly functions on various devices, from desktops to tablets to mobile phones, providing users with a consistent experience.
- The platform adjusts the display of content based on user preferences, device specifications, and network conditions to optimize performance and user experience.

Feature 3: Customer Support Integration
- Integrate a live chat functionality for users to directly communicate with customer support.
- Allow users to ask location-related questions and receive instant assistance.

User Manual

The user manual is divided into two main sections: User-Side and Admin-Side, each explaining the functionalities available to users and administrators.

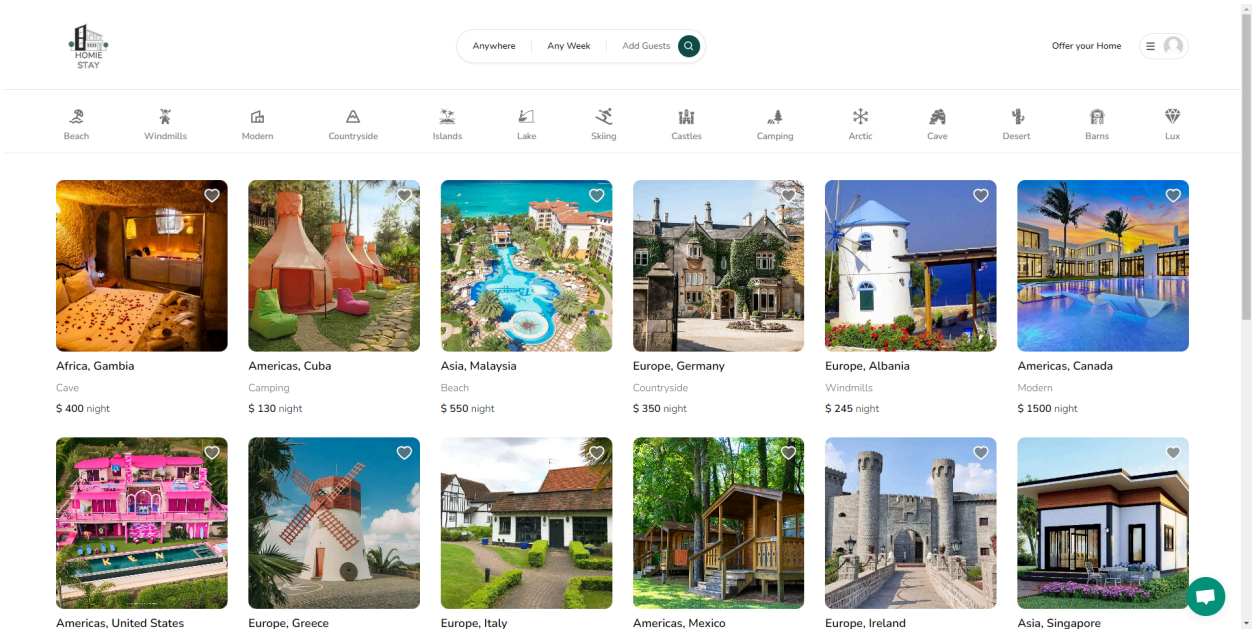
4.1 User-Side

The User-Side section focuses on the functionalities accessible to registered users and visitors.

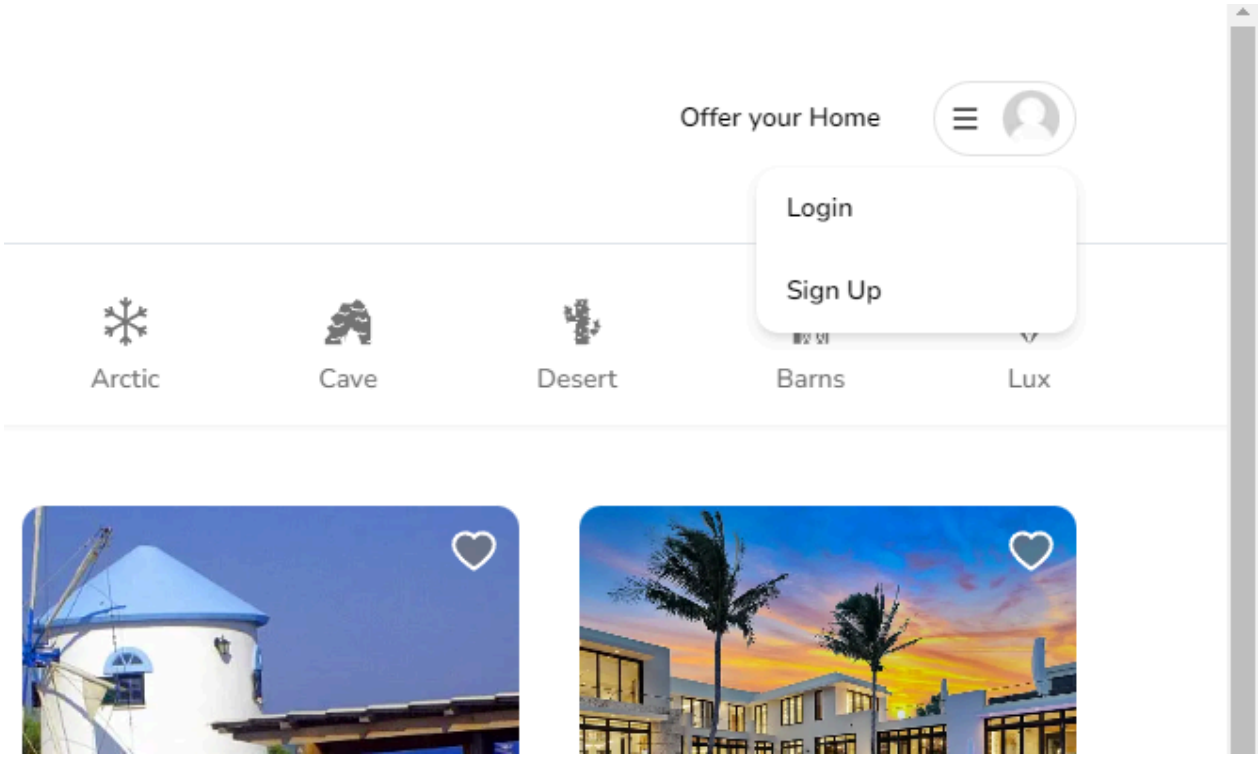
4.1.1 Homepage

Upon accessing the Peer to Peer Accommodation Booking Platform, both members and non-members will land on the homepage. The homepage provides the following options:

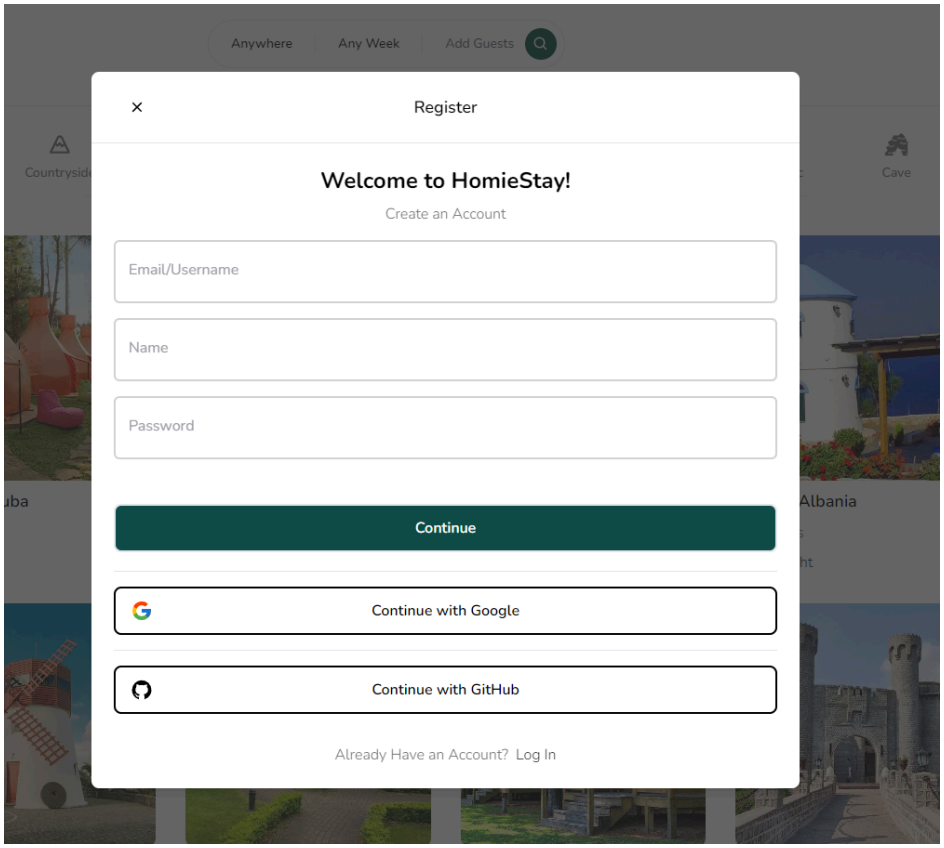
Home: Navigate to the main page or homepage of the platform.



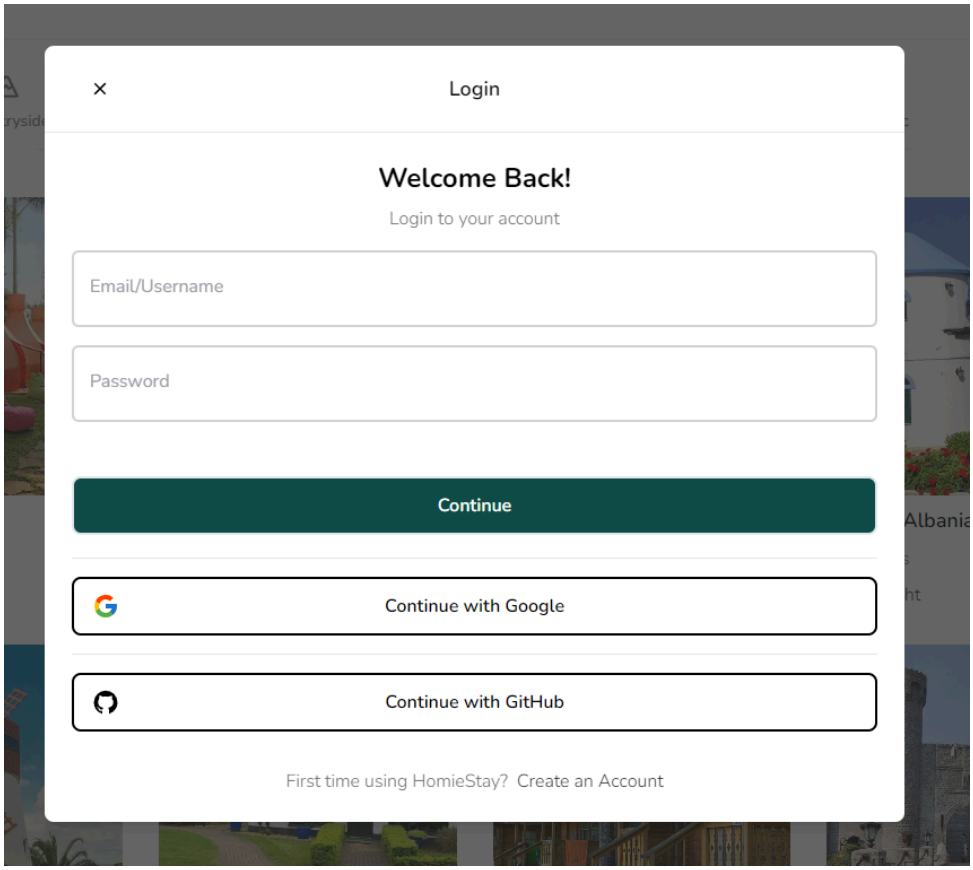
Sign Up / Log In: Register as a member using an email address and password, or log in if you're an existing member.



- **Sign Up:**
  If you're a new user and want to experience the benefits of the platform, click on the "Sign Up" button. This will open the Sign Up modal:
- **Enter your email address:** Provide a valid email address that you want to associate with your account.
- **Enter your name:** Provide your name, which will be associated with your account.
- **Enter a password:** Choose a secure password to protect your account.
- **Sign Up with Google/Gmail or GitHub**: For added convenience, you can also sign up using your Google/Gmail or GitHub account via OAuth authentication.
- **Complete Sign Up:** Click the "Continue" button to create your account.

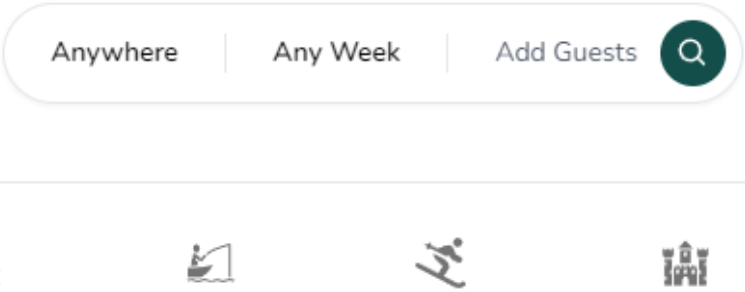


- **Log In:** If you're already a registered user, click on the "LogIn" button. This will open the Log In modal:
- **Enter your email address:** Provide the email associated with your account.
- **Enter your password:** Input the password you set during registration.
- **Log In with Google/Gmail or GitHub:** For added convenience, you can also log in using your Google/Gmail or GitHub account via OAuth authentication.
- **Log In:** Click the "Continue" button to access your member account.
- **First time user:** Click "Create an Account" to start your journey with us.



4.1.2 Property Search and Booking

The platform allows users to seamlessly search for available properties and manage their bookings:

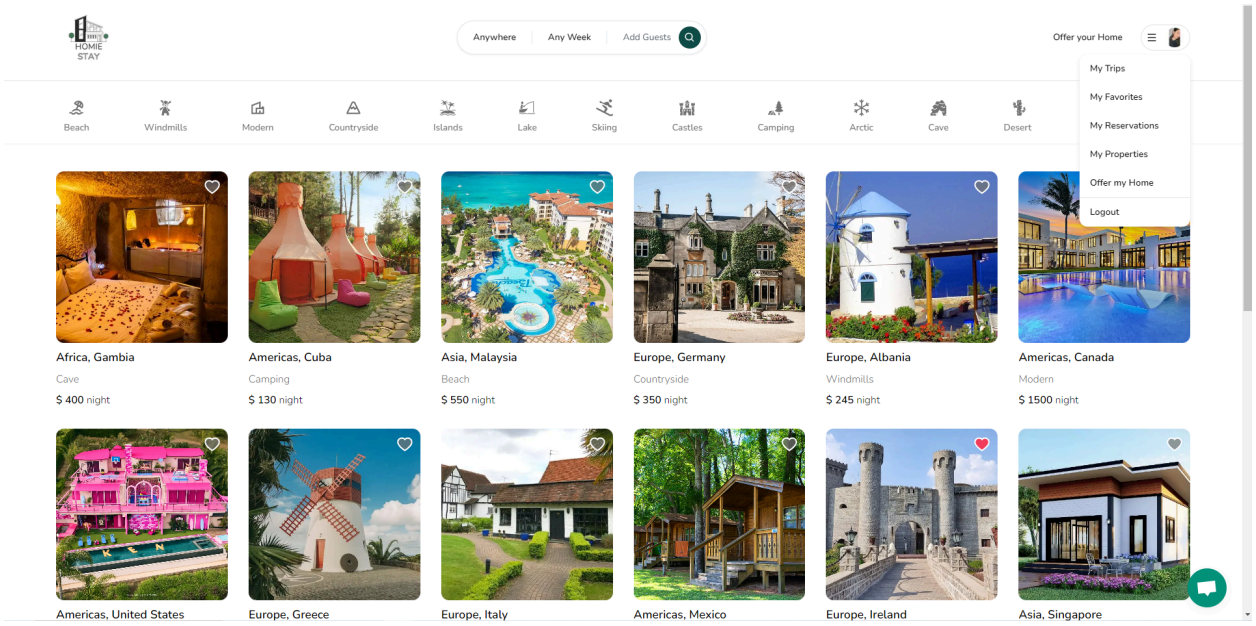


Search Listings: Browse and search for properties based on location, dates, and preferences.
View Listings: Explore detailed listings with property information, images, and pricing.

Book Property: Reserve available properties for specific dates.
View Booking Details: Access comprehensive details of booked properties.

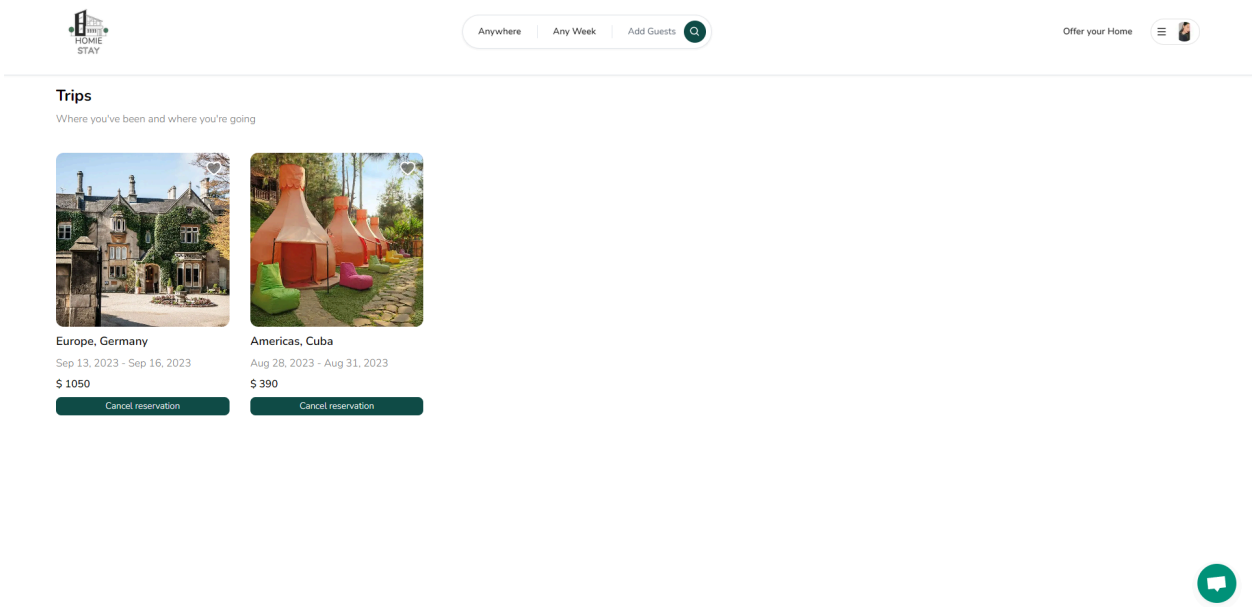
4.1.3 Member Account

When logged in, members can access their personal accounts and utilize a variety of functionalities:
Homepage: Return to the homepage to explore properties and offerings.



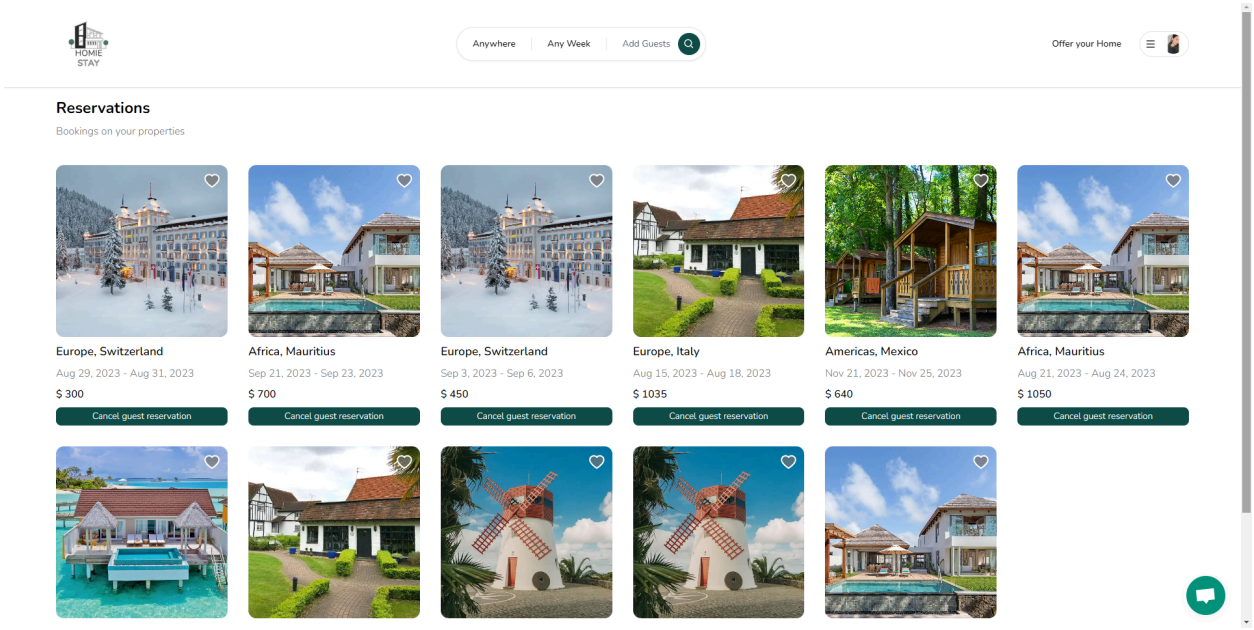
Search: Utilize the search module to find properties that match your preferences.
My Trips:

- **My Trips:** View and manage your property bookings conveniently from one place, allowing you to keep track of your travel plans.
- **Cancel My Trips:** Cancel any upcoming trips you have booked, following the platform's cancellation guidelines.

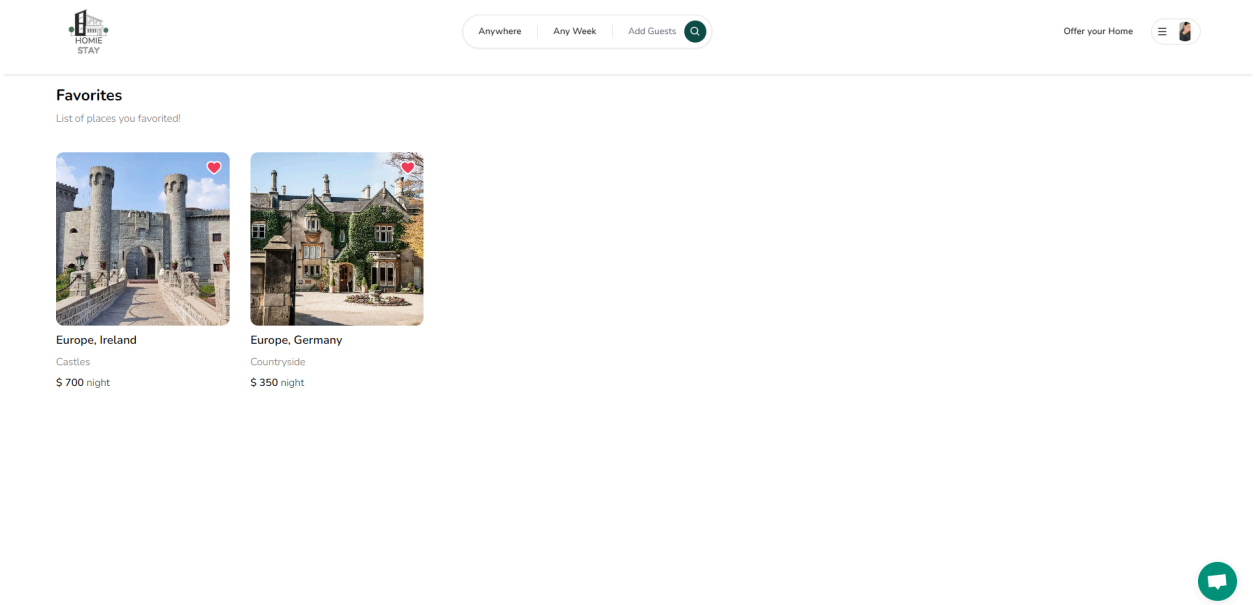


My Reservations:

- **My Reservations:** Get an overview of your upcoming property reservations, helping you stay organized and prepared for your travels.
- **Cancel Reservations:** Cancel a booking if needed.
- **Cancel Guest Reservation:** As a host, cancel a guest's reservation if necessary.

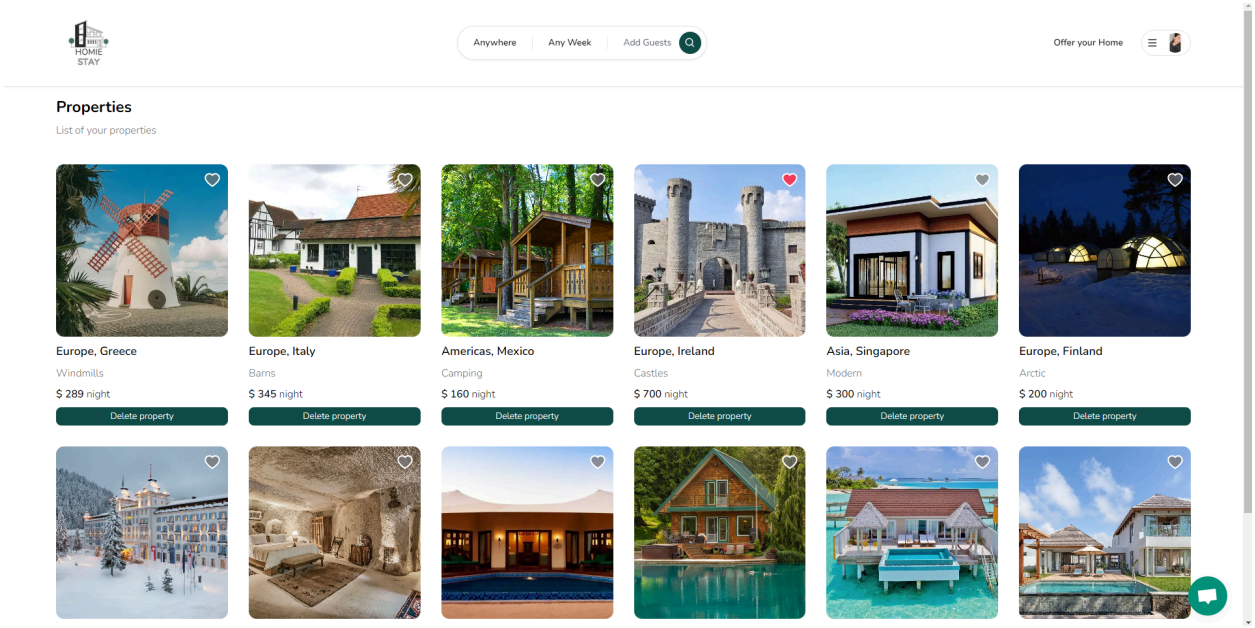


My Favorites: Access a list of your saved favorite properties. This makes it convenient to review and choose from your preferred options when planning your trips.

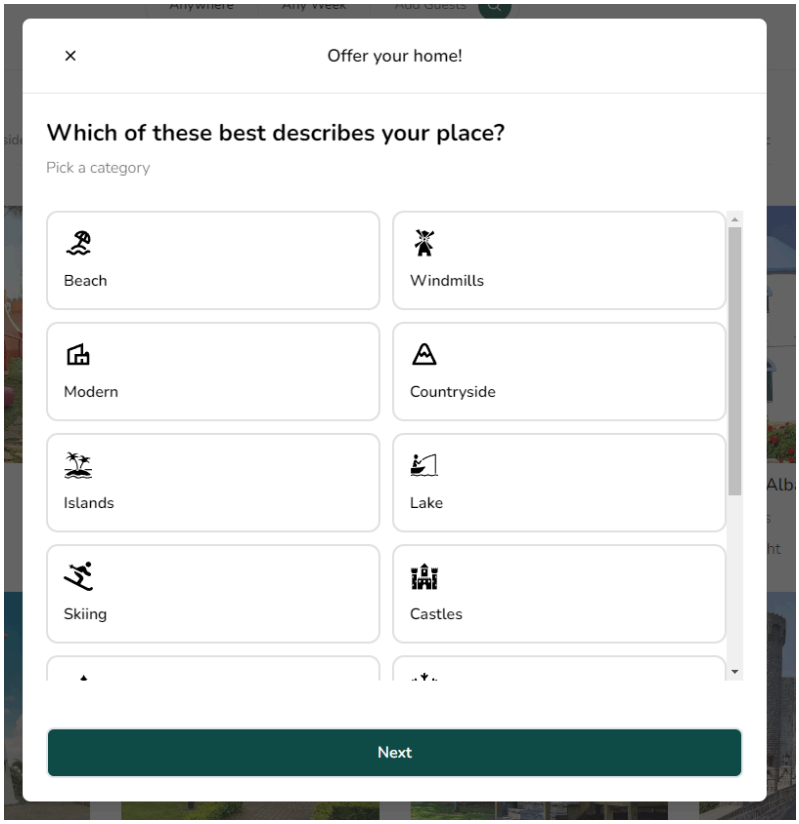


My Properties:

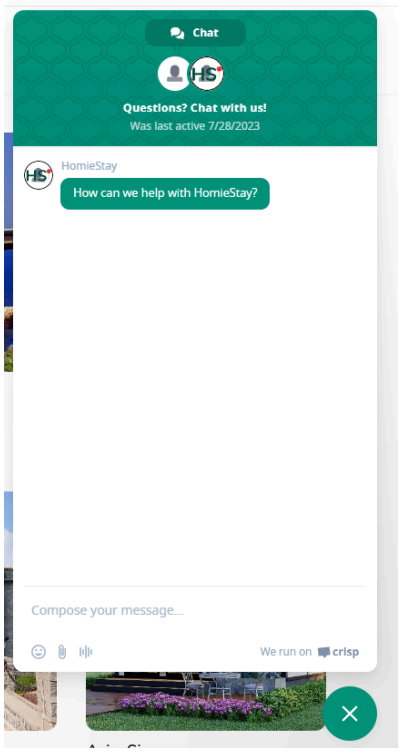
- **My Properties:** If you're a host, manage and update your listed properties from one central location.
- **Delete My Property:** Remove a property listing from your portfolio when it's no longer available for booking.



Offer My Home: Host your property and start welcoming travelers. List your property with ease and manage bookings efficiently.



Customer Support Chat: Engage in real-time conversations with customer support.



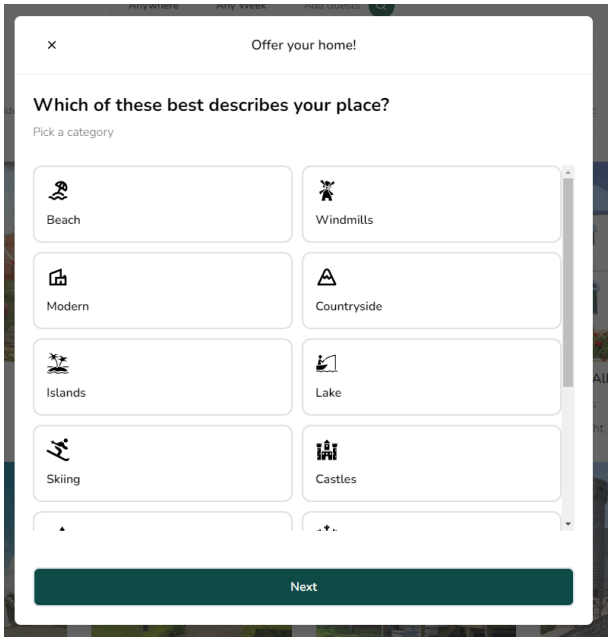
Logout: Safely log out of your account when you're done exploring or managing your bookings.

4.1.4 Property Listings Management

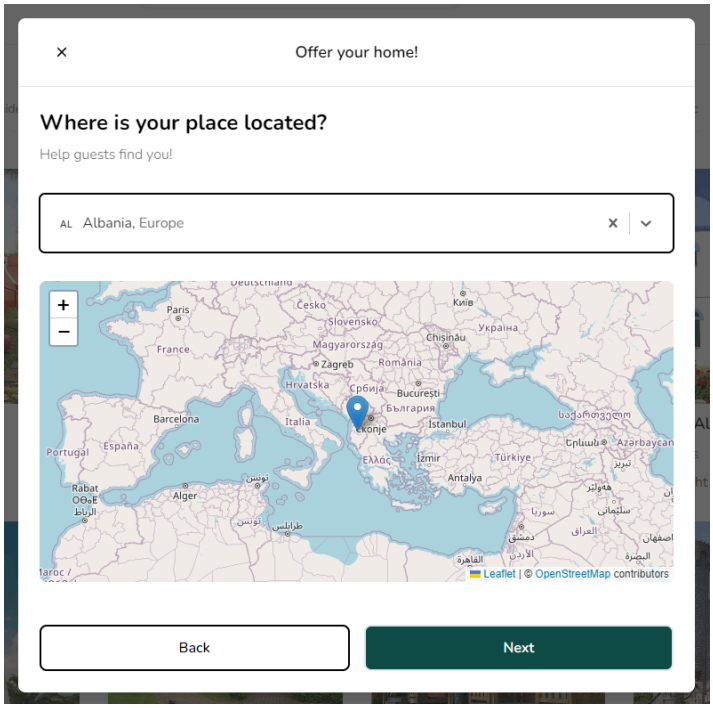
For users interested in listing their properties:

List Property: Create detailed listings with property information to attract potential guests. This module allows you to customize the listing as follows:

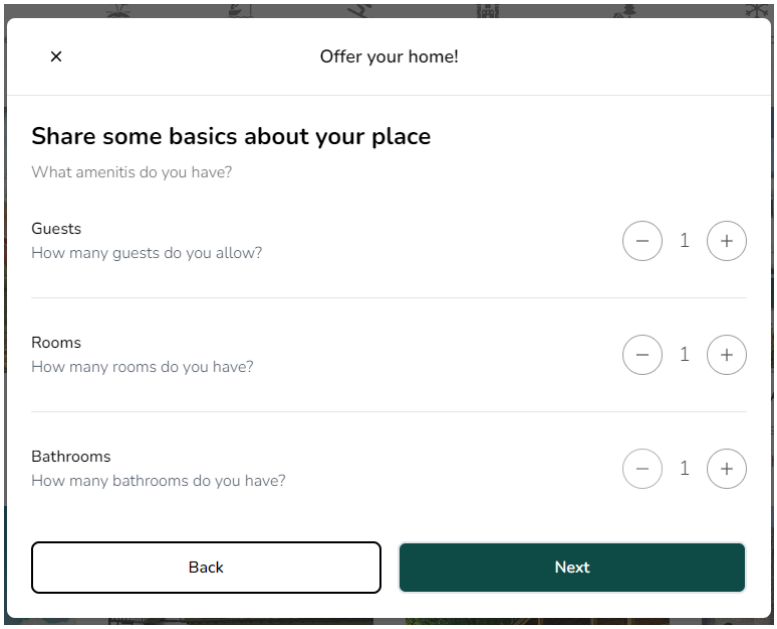
- **Property Category:** Choose an appropriate category that best describes your property (e.g., apartment, villa, cottage).



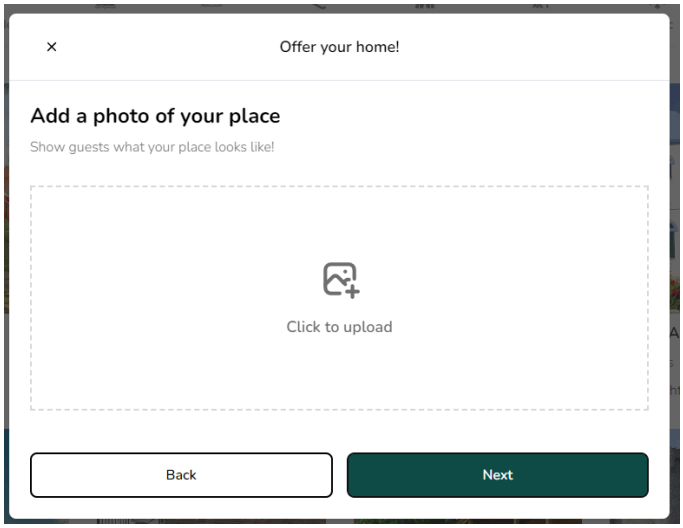
- **Country and Location:** Specify the country and location where your property is situated.

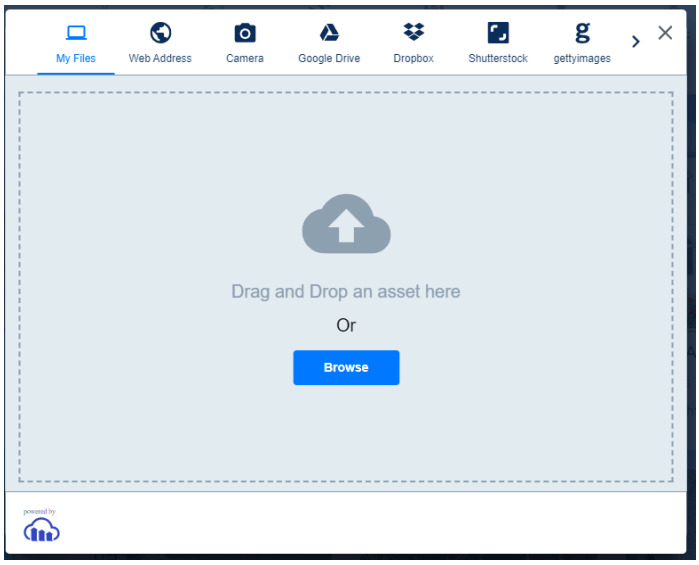


- **Add basics about the place:** Specify the maximum number of guests, rooms and bathrooms your property can accommodate.

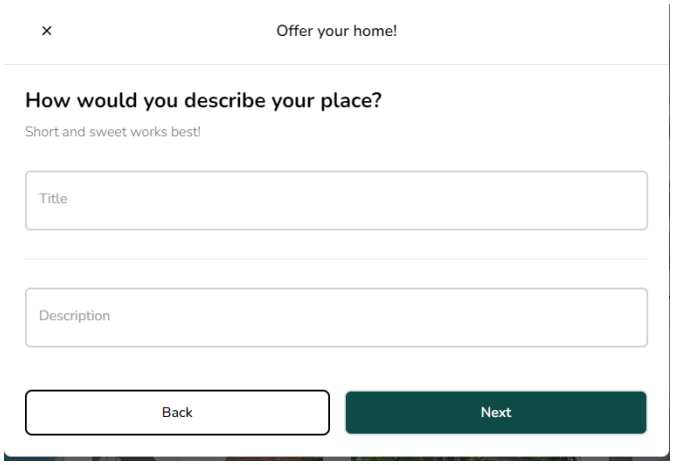


- **Property Image:** Upload captivating images that showcase your property's features and surroundings.

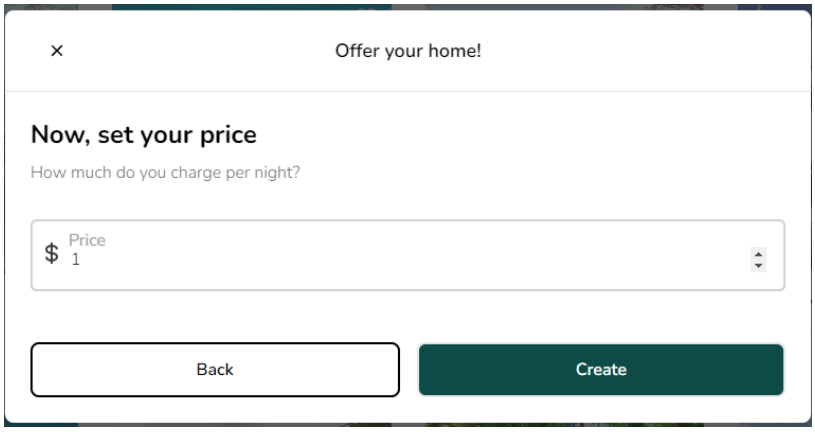




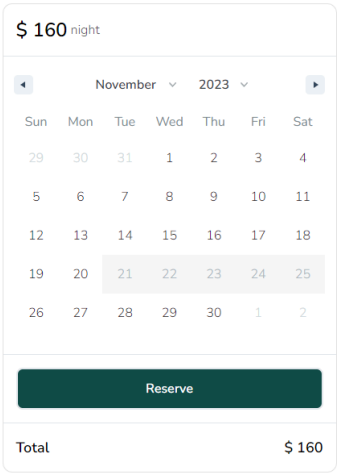
- **Description and Title:** Add a compelling title and provide a comprehensive description highlighting the unique features and amenities your property offers.



- **Pricing per Night:** Set the pricing for guests to stay at your property per night.



Delete Listing: Remove the listing from the platform.
Availability Calendar: Manage property availability using an interactive calendar. Update and maintain availability dates based on reservations, ensuring accurate booking information.



4.1.5 Maps Integration

The platform integrates with maps to enhance property searching:

Property Location: Display property locations on the map within property listings.
Search by Location: Search for properties by specifying a location or area.
Map View: Visualize property locations through an interactive map.

4.2 Admin-Side

The Admin-Side section is intended for administrators of the platform, who have direct access to manage and maintain the platform's data and operations.

4.2.1 Authentication

Administrators can access the database with their authorized credentials. This direct access enables administrators to perform essential actions within the database.

4.2.2 Admin Account

Administrators are granted comprehensive control over the platform's data and content:

- **Data Management:** Admins can directly manage and maintain the database content, including properties, user accounts, trips, and other related data.
- **Data Removal:** Administrators have the ability to remove properties, user accounts, trips, and other data directly from the database as needed.

Frontend Development

4.2 Frontend Development

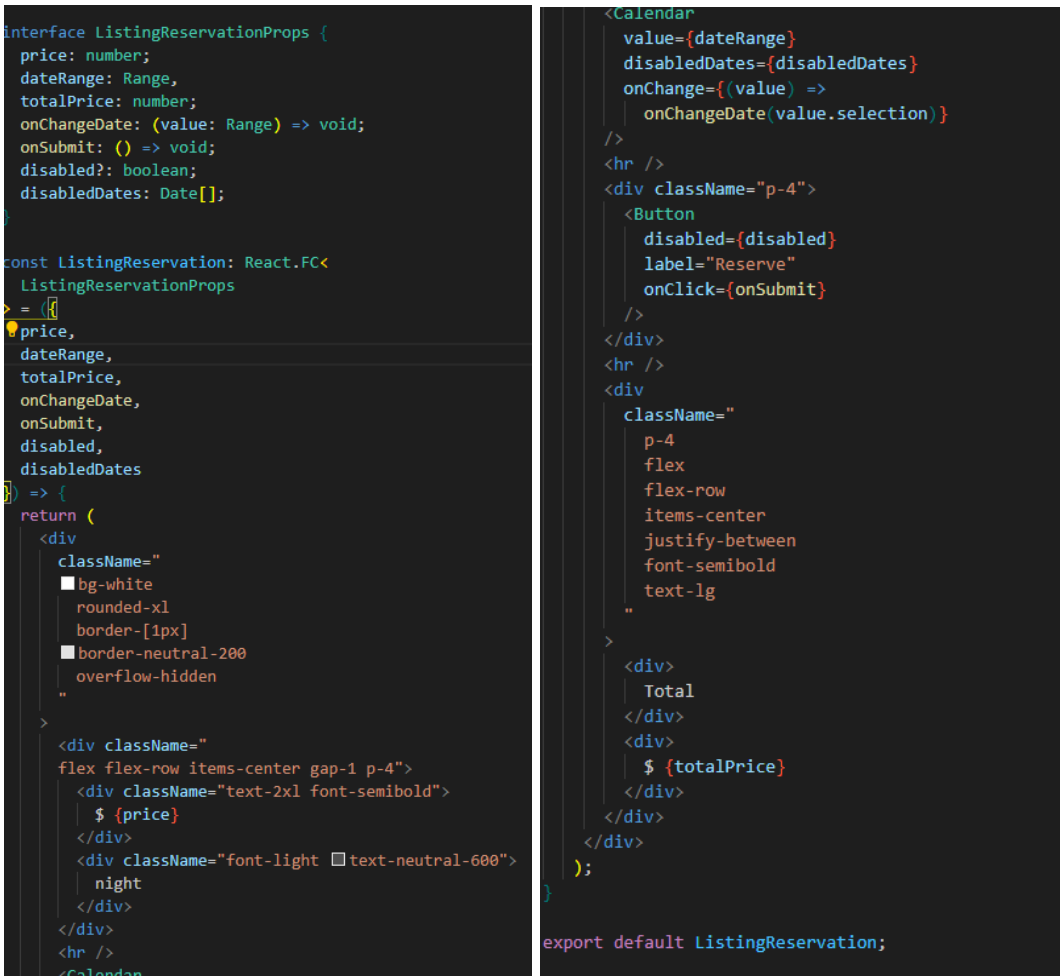
The frontend of the Peer to Peer Accommodation Booking Platform has been thoughtfully crafted using a powerful stack of technologies to ensure a seamless and visually captivating user experience. Leveraging React, Next.js, Tailwind CSS, and TypeScript, the frontend development has been meticulously tailored to deliver an intuitive interface for effortless property searching, efficient booking management, and engaging user account interactions.

User-Friendly Interface:
The frontend interface is designed to be user-friendly and easy to navigate. It incorporates a clean and modern design that allows users to quickly find the information they need.

```
const font = Nunito({

  subsets: ['latin']

})

export const metadata = {
  title: "HomieStay",
  description: "This is a University Project",
};

export default async function RootLayout({
  children,
}: {
  children: React.ReactNode
}) {

  const currentUser = await getCurrentUser();
  return (
    <html lang="en">
      <CrispProvider />
      <body className={font.className}>
        <ClientOnly>
          <ToasterProvider />
          <RentModal />
          <SearchModal />
          <RegisterModal />
          <LoginModal />
          <Navbar currentUser={currentUser} />
        </ClientOnly>
        <div className="pb-20 pt-28">{children}</div>
      </body>
    </html>
  );
}
```

Property Listings:

Property listings are displayed in a structured format, showcasing essential details such as property type, location, pricing, and availability dates. Each listing includes high-quality images to give users a visual representation of the property.



```
interface ListingReservationProps {
  price: number;
  dateRange: Range,
  totalPrice: number;
  onChangeDate: (value: Range) => void;
  onSubmit: () => void;
  disabled?: boolean;
  disabledDates: Date[];
}

const ListingReservation: React.FC<
  ListingReservationProps
> = ({
  price,
  dateRange,
  totalPrice,
  onChangeDate,
  onSubmit,
  disabled,
  disabledDates
}) => {
  return (
    <div
      className="
        bg-white
        rounded-xl
        border-[1px]
        border-neutral-200
        overflow-hidden
      "
    >
      <div className="
      flex flex-row items-center gap-1 p-4">
        <div className="text-2xl font-semibold">
          $ {price}
        </div>
        <div className="font-light text-neutral-600">
          night
        </div>
      </div>
      <hr />
      <Calendar
        value={dateRange}
        disabledDates={disabledDates}
        onChange={(value) =>
          onChangeDate(value.selection)}
      />
      <hr />
      <div className="p-4">
        <Button
          disabled={disabled}
          label="Reserve"
          onClick={onSubmit}
        />
      </div>
      <hr />
      <div
        className="
          p-4
          flex
          flex-row
          items-center
          justify-between
          font-semibold
          text-lg
        "
      >
        <div>
          Total
        </div>
        <div>
          $ {totalPrice}
        </div>
      </div>
    </div>
  );
}

export default ListingReservation;
```

Interactive Search:

The search functionality is implemented with interactive filters that enable users to narrow down their property search based on location, price range, dates, and other preferences.

```
"use client";

import qs from "query-string";
import dynamic from "next/dynamic";
import { useCallback, useMemo, useState } from "react";
import { Range } from "react-date-range";
import { formatISO } from "date-fns";
import { useRouter, useSearchParams } from "next/navigation";

import useSearchModal from "@/app/hooks/useSearchModal";

import Modal from "./Modal";
import Calendar from "../inputs/Calendar";
import Counter from "../inputs/Counter";
import CountrySelect, { CountrySelectValue } from "../inputs/CountrySelect";
import Heading from "../Heading";

enum STEPS {
  LOCATION = 0,
  DATE = 1,
  INFO = 2,
}

const SearchModal = () => {
  const router = useRouter();
  const searchModal = useSearchModal();
  const params = useSearchParams();

  const [step, setStep] = useState(STEPS.LOCATION);

  const [location, setLocation] = useState<CountrySelectValue>();
  const [guestCount, setGuestCount] = useState(1);
  const [roomCount, setRoomCount] = useState(1);
  const [bathroomCount, setBathroomCount] = useState(1);
  const [dateRange, setDateRange] = useState<Range>({
    startDate: new Date(),
    endDate: new Date(),
    key: "selection",
  });

  const Map = useMemo(
    () =>
      dynamic(() => import("../Map"), {
        ssr: false,
      }),
    [location]
  );
```

```
const onBack = useCallback(() => {
  setStep((value) => value - 1);
}, []);

const onNext = useCallback(() => {
  setStep((value) => value + 1);
}, []);

const onSubmit = useCallback(async () => {
  if (step !== STEPS.INFO) {
    return onNext();
  }

  let currentQuery = {};

  if (params) {
    currentQuery = qs.parse(params.toString());
  }

  const updatedQuery: any = {
    ...currentQuery,
    locationValue: location?.value,
    guestCount,
    roomCount,
    bathroomCount,
  };

  if (dateRange.startDate) {
    updatedQuery.startDate = formatISO(dateRange.startDate);
  }

  if (dateRange.endDate) {
    updatedQuery.endDate = formatISO(dateRange.endDate);
  }

  const url = qs.stringifyUrl(
    {
      url: "/",
      query: updatedQuery,
    },
    { skipNull: true }
  );

  setStep(STEPS.LOCATION);
  searchModal.onClose();
  router.push(url);
}, [
  step,
```

Reservation Management:

Logged-in users can manage their property bookings through a user-friendly dashboard. The dashboard provides an overview of upcoming bookings, past bookings, and reservation details.

```
import EmptyState from "@/app/components/EmptyState";
import ClientOnly from "@/app/components/ClientOnly";

import getCurrentUser from "@/app/actions/getCurrentUser";
import getReservations from "@/app/actions/getReservations";

import TripsClient from "./ReservationsClient";

const ReservationsPage = async () => {
  const currentUser = await getCurrentUser();

  if (!currentUser) {
    return (
      <ClientOnly>
        <EmptyState
          title="Unauthorized"
          subtitle="Please login"
        />
      </ClientOnly>
    )
  }

  const reservations = await getReservations({ authorId: currentUser.id });

  if (reservations.length === 0) {
    return (
      <ClientOnly>
        <EmptyState
          title="No reservations found"
          subtitle="Looks like you have no reservations on your properties."
        />
      </ClientOnly>
    );
  }

  return (
    <ClientOnly>
      <TripsClient
        reservations={reservations}
        currentUser={currentUser}
      />
    </ClientOnly>
  );
}

export default ReservationsPage;
```

```
'use client';

import { toast } from "react-hot-toast";
import axios from "axios";
import { useCallback, useState } from "react";
import { useRouter } from "next/navigation";

import { SafeReservation, SafeUser } from "@/app/types"
;
import Heading from "@/app/components/Heading";
import Container from "@/app/components/Container";
import ListingCard from "@/app/components/listings/ListingCard";

interface ReservationsClientProps {
  reservations: SafeReservation[],
  currentUser?: SafeUser | null,
}

const ReservationsClient: React.FC<ReservationsClientProps> = ({
  reservations,
  currentUser
}) => {
  const router = useRouter();
  const [deletingId, setDeletingId] = useState('');

  const onCancel = useCallback((id: string) => {
    setDeletingId(id);

    axios.delete(`/api/reservations/${id}`)
    .then(() => {
      toast.success('Reservation cancelled');
      router.refresh();
    })
    .catch(() => {
      toast.error('Something went wrong.')
    })
    .finally(() => {
      setDeletingId('');
    })
  }, [router]);

  return (
    <Container>
      <div className="mt-9"></div>
      <Heading
        title="Reservations"
        subtitle="Bookings on your properties"
      />
      <div
```

```
  }, [router]);

  return (
    <Container>
      <div className="mt-9"></div>
      <Heading
        title="Reservations"
        subtitle="Bookings on your properties"
      />
      <div
        className="
          mt-10
          grid
          grid-cols-1
          sm:grid-cols-2
          md:grid-cols-3
          lg:grid-cols-4
          xl:grid-cols-5
          2xl:grid-cols-6
          gap-8
        "
      >
        {reservations.map((reservation: any) => (
          <ListingCard
            key={reservation.id}
            data={reservation.listing}
            reservation={reservation}
            actionId={reservation.id}
            onAction={onCancel}
            disabled={deletingId === reservation.id}
            actionLabel="Cancel guest reservation"
            currentUser={currentUser}
          />
        ))}
      </div>
    </Container>
  );
}

export default ReservationsClient;
```

Responsive Design:

The frontend is designed to be responsive, ensuring that the platform is accessible and

```tsx
const Map: React.FC<MapProps> = ({
    center
}) => {

  return (
    <MapContainer
      center={(center as L.LatLngExpression) || [51, -0.09]}
      zoom={center ? 4 : 2}
      scrollWheelZoom={false}
      className="h-[35vh] rounded-lg"

    >
      <TileLayer
        attribution='&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"

      />

      {center&& (
        <Marker
        position={center as L.LatLngExpression}
        />
      )}

    </MapContainer>
  );

};

export default Map;
```

```tsx
interface HeartButtonProps {
  listingId: string;
  currentUser?: SafeUser | null;
}

const HeartButton: React.FC<HeartButtonProps> = ({
  listingId,
  currentUser,
}) => {
  const { hasFavorited, toggleFavorite } = useFavorite({
    listingId,
    currentUser,
  });

  return (
    <div
      onClick={toggleFavorite}
      className="
        relative
        hover:opacity-80
        transition
        cursor-pointer
      "
    >
      <AiOutlineHeart
        size={28}
        className="
          fill-white
          absolute
          -top-[2px]
          -right-[2px]
        "
      />
      <AiFillHeart
        size={24}
        className={hasFavorited ? "fill-rose-500" : "fill-neutral-500/70"}
      />
    </div>
  );
};

export default HeartButton;
```

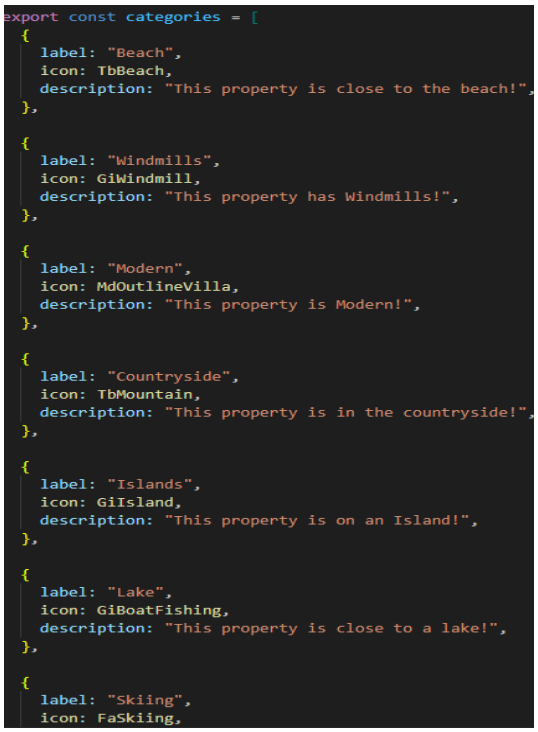
```tsx
"use client";

import { useEffect } from "react";
import { Crisp } from "crisp-sdk-web";

export const CrispChat = () => {
  useEffect(() => {
    Crisp.configure[redacted];
  }, []);

  return null;
};
```

```tsx
"use client";

import Image from "next/image";

interface AvatarProps {
  src?: string | null | undefined;
};


const Avatar: React.FC<AvatarProps> = ({
  src
}) => {
  return (
    <Image
      className="rounded-full"
      height="30"
      width="30"
      alt="Avatar"
      src={src || "/images/placeholder.jpg"}
    />
  );
};

export default Avatar;
```

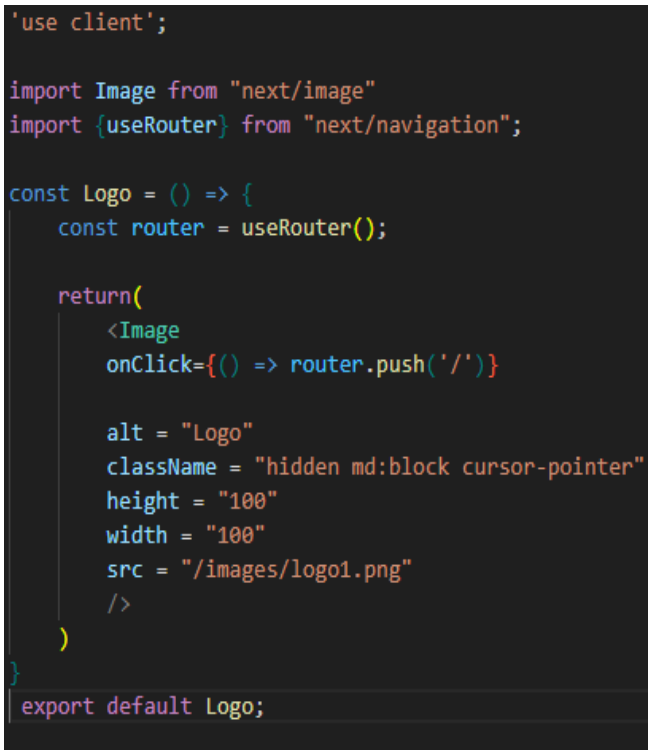
Intuitive Navigation:

The navigation bar is strategically positioned, allowing users to access different sections of the platform seamlessly. Links to the homepage, property listings, user account, and more are readily available.

```
interface NavbarProps{
  currentUser?: SafeUser | null
}

const Navbar: React.FC<NavbarProps> = ({
  currentUser
}) => {

    return (
      <div className="fixed w-full bg-white z-10 shadow-sm">
        <div
          className="
            py-4
            border-b-[1px]"
        >
          <Container>
            <div
              className="
                    flex
                    flex-row
                    items-center
                    justify-between
                    gap-3
                    md:gap-0
                    "
            >
              <Logo />
              <Search />
              <UserMenu currentUser = {currentUser}/>
            </div>
          </Container>
        </div>
        <Categories />
      </div>
    );
}
export default Navbar;
```



```
export const categories = [
  {
    label: "Beach",
    icon: TbBeach,
    description: "This property is close to the beach!",
  },

  {
    label: "Windmills",
    icon: GiWindmill,
    description: "This property has Windmills!",
  },

  {
    label: "Modern",
    icon: MdOutlineVilla,
    description: "This property is Modern!",
  },

  {
    label: "Countryside",
    icon: TbMountain,
    description: "This property is in the countryside!",
  },

  {
    label: "Islands",
    icon: GiIsland,
    description: "This property is on an Island!",
  },

  {
    label: "Lake",
    icon: GiBoatFishing,
    description: "This property is close to a lake!",
  },

  {
    label: "Skiing",
    icon: FaSkiing,
```



```
'use client';

import Image from "next/image"
import {useRouter} from "next/navigation";

const Logo = () => {
    const router = useRouter();

    return(
        <Image
        onClick={() => router.push('/')}

        alt = "Logo"
        className = "hidden md:block cursor-pointer"
        height = "100"
        width = "100"
        src = "/images/logo1.png"
        />
    )
}
export default Logo;
```

```
interface MenuItemProps {
    onClick: () => void;
    label: string;
}

const MenuItem: React.FC<MenuItemProps> = ({
    onClick,
    label
}) => {
  return (
    <div
      onClick={onClick}
      className="
        px-4
        py-3
        hover:bg-neutral-100
        transition
        font-semibold
    "
    >
      {label}
    </div>
  );
};

export default MenuItem;
```

```tsx
const UserMenu: React.FC <UserMenuProps> = ({
  currentUser
}) => {
    const router = useRouter();
    const registerModal = useRegisterModal();
    const loginModal = useLoginModal();
    const rentModal = useRentModal();

    const [isOpen, setIsOpen] = useState(false)

    const toggleOpen = useCallback(() => {
        setIsOpen((value) => !value);
    }, []);


    const onRent = useCallback(() => {
      if (!currentUser) {
        return loginModal.onOpen();
      }

      rentModal.onOpen();

    },[currentUser, loginModal, rentModal]);
```

```tsx
'use client';

interface ContainerProps {
    children: React.ReactNode;
}

const Container: React.FC<ContainerProps> = ({
    children
}) => {
    return (
        <div
        className="
        max-w-[2520px]
        mx-auto
        xl:px-20
        md:px-10
        sm:px-2
        px-4
        "
        >{children}</div>
    );
}

export default Container;
```

Property Details:

Clicking on a property listing opens a detailed view that provides comprehensive information about the property. Users can view images, read property descriptions, check availability, and initiate the booking process.

```tsx
const PropertiesPage = async () => {
  const currentUser = await getCurrentUser();

  if (!currentUser) {
    return <EmptyState title="Unauthorized" subtitle="Please login" />;
  }

  const listings = await getListings({ userId: currentUser.id });

  if (listings.length === 0) {
    return (
      <ClientOnly>
        <EmptyState
          title="No properties found"
          subtitle="Looks like you have no properties."
        />
      </ClientOnly>
    );
  }

  return (
    <ClientOnly>
      <PropertiesClient listings={listings} currentUser={currentUser} />
    </ClientOnly>
  );
};

export default PropertiesPage;
```

```tsx
const PropertiesClient: React.FC<PropertiesClientProps> = ({
  listings,
  currentUser,
}) => {
  const router = useRouter();
  const [deletingId, setDeletingId] = useState("");

  const onDelete = useCallback(
    (id: string) => {
      setDeletingId(id);

      axios
        .delete(`/api/listings/${id}`)
        .then(() => {
          toast.success("Listing deleted");
          router.refresh();
        })
        .catch((error) => {
          toast.error(error?.response?.data?.error);
        })
        .finally(() => {
          setDeletingId("");
        });
    },
    [router]
  );
```

Personalized User Account:

The user account section offers personalized features such as viewing reservations, managing favorite properties and updating personal information.

```tsx
interface FavoritesClientProps {
  listings: SafeListing[],
  currentUser?: SafeUser | null,
}

const FavoritesClient: React.FC<FavoritesClientProps> = ({
  listings,
  currentUser
}) => {
  return (
    <Container>
      <div className="mt-9"></div>
      <Heading
        title="Favorites"
        subtitle="List of places you favorited!"
      />
      <div
        className="
          mt-10
          grid
          grid-cols-1
          sm:grid-cols-2
          md:grid-cols-3
          lg:grid-cols-4
          xl:grid-cols-5
          2xl:grid-cols-6
          gap-8
        "
      >
        {listings.map((listing: any) => (
          <ListingCard
            currentUser={currentUser}
            key={listing.id}
            data={listing}
          />
        ))}
      </div>
    </Container>
  );
}

export default FavoritesClient;
```

```tsx
const ListingPage = async () => {
  const listings = await getFavoriteListings();
  const currentUser = await getCurrentUser();

  if (listings.length === 0) {
    return (
      <ClientOnly>
        <EmptyState
          title="No favorites found"
          subtitle="Looks like you have no favorite listings."
        />
      </ClientOnly>
    );
  }

  return (
    <ClientOnly>
      <FavoritesClient
        listings={listings}
        currentUser={currentUser}
      />
    </ClientOnly>
  );
}

export default ListingPage;
```

```tsx
const ReservationsPage = async () => {
  const currentUser = await getCurrentUser();

  if (!currentUser) {
    return (
      <ClientOnly>
        <EmptyState
          title="Unauthorized"
          subtitle="Please login"
        />
      </ClientOnly>
    )
  }

  const reservations = await getReservations({ authorId: currentUser.id });

  if (reservations.length === 0) {
    return (
      <ClientOnly>
        <EmptyState
          title="No reservations found"
          subtitle="Looks like you have no reservations on your properties."
        />
      </ClientOnly>
    );
  }

  return (
    <ClientOnly>
      <TripsClient
        reservations={reservations}
        currentUser={currentUser}
      />
    </ClientOnly>
  );
}
```

```tsx
interface ReservationsClientProps {
  reservations: SafeReservation[],
  currentUser?: SafeUser | null,
}

const ReservationsClient: React.FC<ReservationsClientProps> = ({
  reservations,
  currentUser
}) => {
  const router = useRouter();
  const [deletingId, setDeletingId] = useState('');

  const onCancel = useCallback((id: string) => {
    setDeletingId(id);

    axios.delete(`/api/reservations/${id}`)
    .then(() => {
      toast.success('Reservation cancelled');
      router.refresh();
    })
    .catch(() => {
      toast.error('Something went wrong.')
    })
    .finally(() => {
      setDeletingId('');
    })
  }, [router]);
```

Backend Development

4.3 Backend Development

The backend development of the Peer to Peer Accommodation Booking Platform is the backbone that supports the functionality and interactions of the platform. It is responsible for managing user data, property listings, bookings, and various other core features.

Database Management:
The backend utilizes a robust database management system to store and organize essential data. This includes user information, property details, booking records, and more.

```
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[]
}

model Account {
  id String @id @default(auto()) @map("_id") @db.ObjectId
  userId String @db.ObjectId
  type String
  provider String
  providerAccountId String
  refresh_token String? @db.String
  access_token String? @db.String
  expires_at Int?
  token_type String?
  scope String?
  id_token String? @db.String
  session_state String?

  user User@relation(fields: [userId], references: [id], onDelete: Cascade)

  @@unique([provider, providerAccountId])
}
```

```
model Listing {
  id String @id @default(auto()) @map("_id") @db.ObjectId
  title String
  description String
  imageSrc String
  createdAt DateTime @default(now())
  category String
  roomCount Int
  bathroomCount Int
  guestCount Int
  locationValue String
  userId String @db.ObjectId
  price Int

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)

  reservations Reservation[]
}

model Reservation {
  id String @id @default(auto()) @map("_id") @db.ObjectId
  userId String @db.ObjectId
  listingId String @db.ObjectId
  startDate DateTime
  endDate DateTime
  totalPrice Int
  createdAt DateTime @default(now())

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)
  listing Listing @relation(fields: [listingId], references: [id], onDelete: Cascade)
}
```

User Authentication:

A secure user authentication system is implemented to ensure that user accounts are protected. Users can create accounts, log in securely, and manage their personal information.

```
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;
      },
    }),
  ],
  pages: {
```

Property Listings Management:

The backend handles property listings, allowing hosts to add, update, and remove their properties. Property details such as location, pricing, availability, and images are stored and retrieved as needed.

```
export async function POST(request: Request) {
  const currentUser = await getCurrentUser();

  if (!currentUser) {
    return NextResponse.error();
  }

  const body = await request.json();
  const {
    title,
    description,
    imageSrc,
    category,
    roomCount,
    bathroomCount,
    guestCount,
    location,
    price,
  } = body;
```

```
Object.keys(body).forEach((value: any) => {
  if (!body[value]) {
    NextResponse.error();
  }
});

const listing = await prisma.listing.create({
  data: {
    title,
    description,
    imageSrc,
    category,
    roomCount,
    bathroomCount,
    guestCount,
    locationValue: location.value,
    price: parseInt(price, 10),
    userId: currentUser.id,
  },
});

return NextResponse.json(listing);
}
```

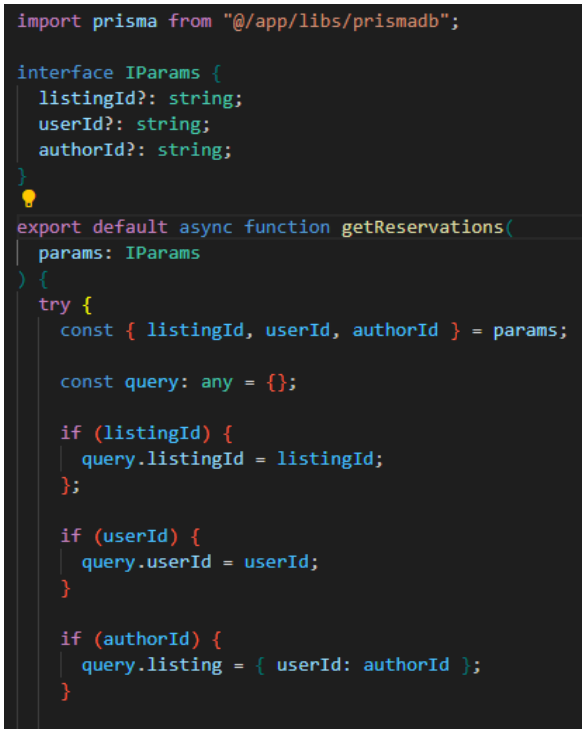
const location: any

Reservation Processing:

The booking system is managed through the backend, facilitating the reservation process for users. Bookings are tracked, confirmed, and updated within the system.

```
interface IParams {
  reservationId?: string;
}

export async function DELETE(
  request: Request,
  { params }: { params: IParams }
) {
  const currentUser = await getCurrentUser();

  if (!currentUser) {
    return NextResponse.error();
  }

  const { reservationId } = params;

  if (!reservationId || typeof reservationId !== "string") {
    throw new Error("Invalid ID");
  }

  const reservation = await prisma.reservation.deleteMany({
    where: {
      id: reservationId,
      OR: [{ userId: currentUser.id }, { listing: { userId: currentUser.id } }],
    },
  });

  return NextResponse.json(reservation);
}
```

```
export async function POST(request: Request) {
  const currentUser = await getCurrentUser();

  if (!currentUser) {
    return NextResponse.error();
  }

  const body = await request.json();
  const { listingId, startDate, endDate, totalPrice } = body;

  if (!listingId || !startDate || !endDate || !totalPrice) {
    return NextResponse.error();
  }

  const listingAndReservation = await prisma.listing.update({
    where: {
      id: listingId,
    },
    data: {
      reservations: {
        create: {
          userId: currentUser.id,
          startDate,
          endDate,
          totalPrice,
        },
      },
    },
  });

  return NextResponse.json(listingAndReservation);
}
```



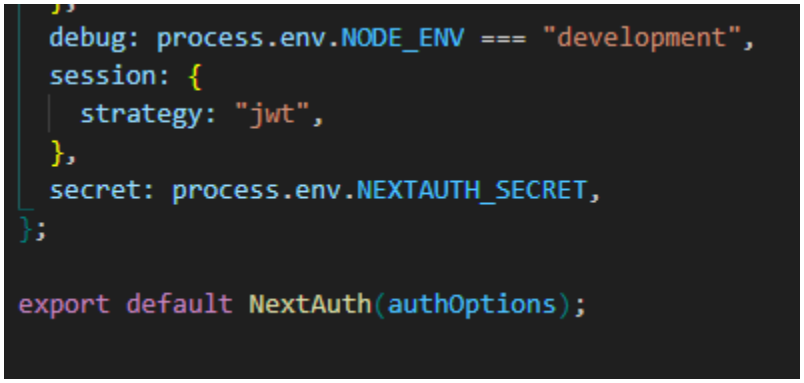
```
import prisma from "@/app/libs/prismadb";

interface IParams {
  listingId?: string;
  userId?: string;
  authorId?: string;
}

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);
  }
}
```

Data Security and Privacy:

Data security is a top priority in backend development. Sensitive user information is securely stored and encrypted, protecting user privacy.

```
import bcrypt from "bcrypt";
import prisma from "@/app/libs/prismadb";
import {NextResponse} from "next/server";

export async function POST(
    request: Request
) {
    const body = await request.json();
    const {
        email,
        name,
        password
    } = body;

    const hashedPassword = await bcrypt.hash(password, 12);

    const user = await prisma.user.create({
        data: {
            email,
            name,
            hashedPassword
        }
    });

    return NextResponse.json(user);
}
```



```
  debug: process.env.NODE_ENV === "development",
  session: {
    strategy: "jwt",
  },
  secret: process.env.NEXTAUTH_SECRET,
};

export default NextAuth(authOptions);
```

Server-Side Logic:

The backend implements server-side logic to handle various actions, such as processing bookings and generating dynamic content for users.

Performance Optimization:
The backend is optimized for performance to ensure fast response times and smooth user interactions. Caching mechanisms and efficient data retrieval strategies are employed.

Scalability:
The backend architecture is designed to be scalable, allowing the platform to handle increased user traffic and data growth without compromising performance.

Technology (Framework, Languages)

6.1 Framework and Languages

The Peer to Peer Accommodation Booking Platform is built using a combination of modern technologies, frameworks, and programming languages that contribute to its functionality, performance, and user experience.

Frontend Technologies:

- React: The platform's frontend is developed using the React JavaScript library. React enables the creation of dynamic and interactive user interfaces, facilitating smooth user interactions and efficient UI updates.

- Next.js: Next.js, a widely used React framework, is employed to construct server-rendered React applications. It brings advantages such as enhanced performance, SEO optimization, and advanced routing capabilities.

- TailwindCSS: TailwindCSS is chosen as the styling framework. Alongside, TypeScript, a statically typed superset of JavaScript, is incorporated into the workflow. TypeScript's strong typing enhances code quality and analysis, while TailwindCSS streamlines styling through its utility-first approach.

- TypeScript: TypeScript, integrated into the development stack, is used for its statically typed nature. It not only raises code quality and analysis but also contributes to developer productivity. TypeScript frontend works with .tsx files, providing enhanced type checking for React components.

Backend Technologies:

- Next.js: Next.js, a popular React framework, is utilized to build server-rendered React applications. It provides benefits such as improved performance, SEO optimization, and routing capabilities.

- Node.js: The backend of the platform is powered by Node.js, a JavaScript runtime environment. Node.js enables server-side JavaScript execution, allowing for efficient and scalable backend development.

- MongoDB: MongoDB, a NoSQL database, is chosen for its scalability and flexibility. It stores structured and unstructured data related to users, properties, bookings, and more.

- TypeScript: To elevate code quality and provide comprehensive code analysis, TypeScript, a statically typed superset of JavaScript, is employed. It contributes to enhanced developer productivity through early error detection and improved code structure.

Authentication and Security:

- JWT (JSON Web Tokens): JWT is used for secure user authentication and authorization. It ensures that users can securely access their accounts and perform actions within the platform.

- bcrypt: The bcrypt hashing algorithm is employed to securely store user passwords. This adds an extra layer of security to user data.