- **Prisma Schema**

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

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

model User {
    id           Int            @id @default(autoincrement())
    email        String         @unique
    name         String
    password     String
    refreshToken RefreshToken[]
    properties   Property[]
    bookings     Bookings[]
}

model RefreshToken {
    id     Int    @id @default(autoincrement())
    token  String @unique
    userId Int
    user   User   @relation(fields: [userId], references: [id])
}

model Property {
    id          Int        @id @default(autoincrement())
    title       String
    address     String
    description String
    extraInfo   String
    checkIn     String
    checkOut    String
    price       String
    Image       Image[]
    User        User?      @relation(fields: [userId], references: [id])
    userId      Int?
    Bookings    Bookings[]
}

model Image {
    id         Int      @id @default(autoincrement())
    url        String
    property   Property @relation(fields: [propertyId], references: [id])
    propertyId Int
}

model Bookings {
    id         Int       @id @default(autoincrement())
    propertyId Int
    userId     Int
    checkIn    DateTime
    checkOut   DateTime
    totalPrice Int
    property   Property  @relation(fields: [propertyId], references: [id])
    user       User      @relation(fields: [userId], references: [id])
    Payment    Payment[]
}

model Payment {
    id        Int      @id @default(autoincrement())
    bookingId Int
    status    String
    booking   Bookings @relation(fields: [bookingId], references: [id])
}

model Admin {
    id       Int     @id @default(autoincrement())
    email    String  @unique
    password String
    token    String?
}

```

- **index.js – backend**

```
const express = require("express");
const cors = require("cors");
const morgan = require("morgan");
const cookieParser = require("cookie-parser");
require("dotenv").config();
const { PrismaClient } = require("@prisma/client");
const prisma = new PrismaClient();
const port = process.env.PORT;

const app = express();
app.use(
  cors({
    origin: ["http://localhost:5173", "http://localhost:5174"],
    credentials: true,
  })
);
app.use(morgan("dev"));
app.use(express.json());
app.use(cookieParser());
app.use("/uploads", express.static("uploads"));

const userRouter = require("./routes/userRoute");
const profileRouter = require("./routes/profileRoute");
const propertyRouter = require("./routes/propertyRoute");
const logoutRouter = require("./routes/logoutRoute");
const bookingRouter = require("./routes/bookingRoute");
const { verifyJWT } = require("./middleware/verifyJWT");
const { verifyAdminToken } = require("./middleware/verifyAdminToken");
const adminUsersRouter = require("./routes/admin/usersRoute");
const adminBookingsRouter = require("./routes/admin/bookingsRoute");
const adminPropertyRouter = require("./routes/admin/propertyRoute");
const adminDashboardRouter = require("./routes/admin/dashboardRoute");
const adminLoginRouter = require("./routes/admin/loginRoute");
const adminLogoutRouter = require("./routes/admin/logoutRoute");

app.use("/api/user", userRouter);

const axios = require("axios");

app.get("/api/properties", async (req, res) => {
  try {
    const properties = await prisma.property.findMany({
      include: {
        Image: true,
      },
    });

    res.json(properties);
  } catch (error) {
    console.error(error);
    res
      .status(500)
      .json({ error: "An error occurred while fetching the properties." });
  }
});

app.use("/api/property", verifyJWT, propertyRouter);
app.use("/api/profile", verifyJWT, profileRouter);
app.use("/api/logout", verifyJWT, logoutRouter);
app.use("/api/booking", verifyJWT, bookingRouter);

app.post("/api/khalti", verifyJWT, async (req, res) => {
  const { purchase_order_id, purchase_order_name, amount } = req.body;
  const userId = req.user ? req.user.id : null;
  console.log(userId);

  const user = await prisma.user.findUnique({
    where: {
      id: userId,
    },
  });

  const data = {
    return_url: "http://localhost:5173/payment",
    website_url: "http://localhost:5173/",
    amount: amount,
    purchase_order_id,
    purchase_order_name,
    customer_info: {
      name: user.name,
      email: user.email,
    },
  };

  try {
    const response = await axios({
      method: "post",
      url: "https://a.khalti.com/api/v2/epayment/initiate/",
      data: data,
      headers: {
        Authorization: "key 1f321a829ba14e379b80dedb83327539",
        "Content-Type": "application/json",
      },
    });

    res.json({ data: response.data });
  } catch (error) {
    console.error("Error from Khalti API", error.response.data);
    res.status(500).json({ message: "Payment failed", error: error.message });
  }
});

app.post("/api/payment", verifyJWT, async (req, res) => {
  const { bookingId, status } = req.body;
  const bookingIdInt = parseInt(bookingId, 10); // Convert bookingId to integer

  try {
    await prisma.payment.upsert({
      where: { id: bookingIdInt },
      update: { status },
      create: {
        status,
        bookingId: bookingIdInt,
        },
      });

      res.status(200).json({ message: "Payment status updated successfully." });
    } catch (error) {
      console.log(error);
      res
        .status(500)
        .json({ error: "An error occurred while updating the payment status." });
    }
  });

  app.get("/api/property/:id", async (req, res) => {
    const { id } = req.params;
    try {
      const property = await prisma.property.findUnique({
        where: {
          id: parseInt(id),
        },
        include: {
          Image: true,
        },
      });

      res.json(property);
    } catch (error) {
      console.error(error);
      res
        .status(500)
        .json({ error: "An error occurred while fetching the property." });
    }
  });

  app.get;

  app.use("/api/admin", adminLoginRouter);
  app.use("/api/admin", verifyAdminToken, adminDashboardRouter);
  app.use("/api/admin", verifyAdminToken, adminUsersRouter);
  app.use("/api/admin", verifyAdminToken, adminPropertyRouter);
  app.use("/api/admin", verifyAdminToken, adminBookingsRouter);
  app.use("/api/admin", verifyAdminToken, adminLogoutRouter);

  app.listen(port, (error) => {
    if (error) throw error;
    console.log("My app is running on port", port);
  });

```

- **Booking Controller - Backend**

```javascript
const { PrismaClient } = require("@prisma/client");
const prisma = new PrismaClient();

const addBooking = async (req, res) => {
  const propertyId = req.params.id;
  const userId = req.user.id;
  const { checkIn, checkOut } = req.body;

  // Validate checkIn and checkOut
  if (!Date.parse(checkIn) || !Date.parse(checkOut)) {
    return res.status(400).json({ error: "Invalid checkIn or checkOut date." });
  }

  try {
    // Fetch the property data
    const property = await prisma.property.findUnique({
      where: { id: parseInt(propertyId) },
    });

    // Check if the property is already booked for the requested dates
    const existingBooking = await prisma.bookings.findFirst({
      where: {
        propertyId: parseInt(propertyId),
        OR: [
          {
            AND: [
              { checkIn: { lte: new Date(checkIn) } },
              { checkOut: { gte: new Date(checkIn) } },
            ],
          },
          {
            AND: [
              { checkIn: { lte: new Date(checkOut) } },
              { checkOut: { gte: new Date(checkOut) } },
            ],
          },
        ],
      },
    });

    if (existingBooking) {
      return res.status(400).json({
        error: "The property is already booked for the requested dates.",
      });
    }
    // Calculate the number of days between checkIn and checkOut
    const checkInDate = new Date(checkIn);
    const checkOutDate = new Date(checkOut);
    const diffTime = Math.abs(checkOutDate - checkInDate);
    const diffDays = Math.ceil(diffTime / (1000 * 60 * 60 * 24));

    // Calculate the total price
    const totalPrice = diffDays * property.price;

    const booking = await prisma.bookings.create({
      data: {
        checkIn,
        checkOut,
        propertyId: parseInt(propertyId),
        userId,
        totalPrice, // Include the total price in the booking data
        Payment: {
          // Create a new payment with status "pending"
          create: {
            status: "pending",
          },
        },
      },
      include: {
        Payment: true, // Include the payment in the returned booking data
      },
    });

    res.json(booking);
  } catch (error) {
    console.error(error);
    res
      .status(500)
      .json({ error: "An error occurred while adding the booking." });
  }
};

const getBookings = async (req, res) => {
  try {
    const userId = req.user.id;
    const bookings = await prisma.bookings.findMany({
      where: {
        userId: userId,
      },
      include: {
        property: {
          include: {
            Image: true,
            User: true,
          },
        },
        user: true,
        Payment: true, // Include the Payment model
      },
    });

    // Print the user who added the property and payment status for each booking
    bookings.forEach((booking) => {
      console.log(booking.property.User);
      console.log(booking.Payment); // Print the payment status
    });

    res.json(bookings);
  } catch (error) {
    console.error(error);
    res
      .status(500)
      .json({ error: "An error occurred while fetching the bookings." });
  }
};

const deleteBooking = async (req, res) => {
  const { bookingId } = req.body;
  try {
    await prisma.payment.deleteMany({
      where: {
        bookingId: parseInt(bookingId),
      },
    });

    await prisma.bookings.delete({
      where: {
        id: parseInt(bookingId),
      },
    });

    res.json({ message: "Booking deleted successfully" });
  } catch (error) {
    console.error(error);
    res
      .status(500)
      .json({ error: "An error occurred while deleting the booking." });
  }
};

module.exports = { addBooking, getBookings, deleteBooking };
```

- **Booking routes – backend**

```
const express = require("express");
const bookingRouter = express.Router();
const {
  addBooking,
  getBookings,
  deleteBooking,
} = require("../controllers/bookingController");

bookingRouter.post("/add/:id", addBooking);
bookingRouter.get("/", getBookings);
bookingRouter.delete("/delete", deleteBooking);

module.exports = bookingRouter;

```

- **Booking.jsx – Front end**

```
import React, { useState, useEffect, useContext } from "react";
import axios from "axios";
import { toast } from "react-hot-toast";
import { UserContext } from "../util/UserContext";
import { Link } from "react-router-dom";
import BookingComponent from "../components/BookingComponent";

export default function Bookings() {
  const [bookings, setBookings] = useState([]);
  const [isLoading, setIsLoading] = useState(true);
  const [popup, setPopup] = useState(false);
  const [refresh, setRefresh] = useState(false);

  const { user } = useContext(UserContext);

  const handleDelete = (bookingId) => {
    axios
      .delete(`booking/delete`, { data: { bookingId: bookingId } })
      .then(() => {
        toast.success("Booking deleted");
        setRefresh((prev) => !prev); // Toggle refresh state to trigger re-fetching of
bookings
      })
      .catch((error) => {
        toast.error("Failed to delete booking");
        console.error(error);
      });
  };

  const initiatePayment = async (
      purchase_order_id,
      purchase_order_name,
      amount
    ) => {
      try {
        const response = await axios.post("/khalti", {
          purchase_order_id,
          purchase_order_name,
          amount,
        });
        console.log(response.data);
        console.log(response.data.data.payment_url);
        window.location.href = response.data.data.payment_url;
      } catch (error) {
        console.error("Error initiating payment", error);
      }
    };

    const handlePopup = () => {
      if (popup) {
        setPopup(false);
      } else {
        setPopup(true);
      }
    };

    useEffect(() => {
      setIsLoading(true);
      axios
        .get("booking")
        .then((response) => {
          const bookings = response.data;
          return Promise.all(
            bookings.map((booking) =>
              axios.get(`property/${booking.propertyId}`).then((response) => ({
                ...booking,
                property: response.data,
              }))
            )
          );
        })
        .then((bookingsWithProperty) => {
          const futureBookings = bookingsWithProperty.filter((booking) => {
            const checkInDate = new Date(booking.checkIn);
            const today = new Date();
            return checkInDate >= today;
          });
          setBookings(futureBookings);
      })
      .catch((error) => {
        console.log(error);
      })
      .finally(() => {
        setIsLoading(false);
      });
  }, [user, refresh]);

  if (isLoading) {
    return <div className="">Loading...</div>;
  }

  if (!user) {
    return (
      <div className="container text-center mt-20 text-3xl font-bold w-fit flex mx-auto
border border-gray-300 py-20 px-12 rounded-xl shadow-lg flex-col text-white">
        <div className="">Please login to view your bookings 🔒</div>
        <Link to="/login" className="text-lg text-accent underline">
          Login
        </Link>
      </div>
    );
  }

  if (bookings.length === 0) {
    return (
      <div className="container text-center mt-20 text-3xl font-bold w-fit flex mx-auto
border border-gray-300 py-20 px-12 rounded-xl shadow-lg flex-col text-white">
        <div className="">You have no bookings</div>
        <Link to="/" className="text-lg text-accent underline">
          Book a property
        </Link>
      </div>
    );
  }

  return (
    <div className="w-full">
      <h1 className="mt-6 font-semibold text-3xl">Bookings</h1>
      <div className="mt-4 flex flex-col gap-10">
        {bookings.map((booking) => (
          <BookingComponent
            handlePopup={handlePopup}
            handleDelete={handleDelete}
            initiatePayment={initiatePayment}
            booking={booking}
            key={booking.id}
            />
          ))}
        </div>
      </div>
    );
  }

```