FULL STACK WEB APPLICATION

HOPE HARBOR

A project report submitted in the partial fulfillment of

Requirements for the award of the Degree of

BACHELOR OF TECHNOLOGY IN

COMPUTER SCIENCE AND ENGINEERING

Team Members: **K.Sowjanya, K.Tejasree, K.Varshitha, K.Manikanta**

Registration Numbers: **21501A05A1,21501A0572,21501A0598,21501A0577**

Name of the College: **Prasad V.Potluri Siddhartha Institute of Technology**

Under the Esteemed Guidance of
Mr.B.Vishnu Vardhan M.Tech,
Assistant Professor,
Department of CSE

Department of Computer Science and Engineering

PRASAD V POTLURI SIDDHARTHA INSTITUTE OF TECHNOLOGY

(Permanently affiliated to JNTU: Kakinada, Approved by AICTE)

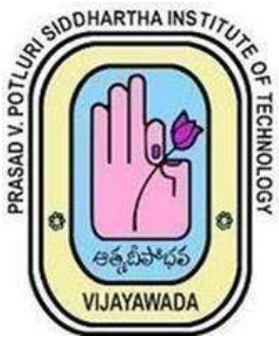
(An NBA &NAAC A+ accredited and ISO 9001:2015 certified Institution)

Kanuru, Vijayawada -520007 20232024

PRASAD V POTLURI SIDDHARTHA INSTITUTE OF TECHNOLOGY

Autonomous & Permanent Affiliation to JNTUK-Kakinada, AICTE approvedAn NBA & NAAC accredited and ISO 9001:2015 Certified Institution **KANURU, VIJAYAWADA – 520007**

DEPARTMENTOF COMPUTERSCIENCEAND ENGINEERING



CERTIFICATE

This is to certify that the project entitled "**HOPE HARBOR**" is submitted by **K.Sowjanya(21501A05A1),K.Tejasree(21501A0572),K.Varshitha(21501A0598) , K.Manikanta(21501A0577)** III B.Tech II Semester in partial fulfillment of the requirement for the award of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING** in the academic year 2022-2023.

Signature of the Guide
Mr.B.Vishnu Vardhan M.Tech,
Assistant Professor,
Dept. of CSE, PVPSIT.

Signature of the HOD
Dr. A. Jayalakshmi,
Professor & HOD,
Dept. of CSE, PVPSIT.

(Signature and Date)

CONTENTS

CHAPTER 1 : OVERVIEW OF THE PROJECT

The aim of the project "Hope Harbor" is to create a platform that connects generous individuals with orphanages in need to streamline the donation process and ensure that orphanages receive the support they require.

To build this website,we used a combination of technologies including a database management system MongoDB and a open-source JavaScript library,react for frontend development. This website includes two types of logins, user and admin. Users can access features related to donating items or money, while admins can manage orphanages and their requirements.

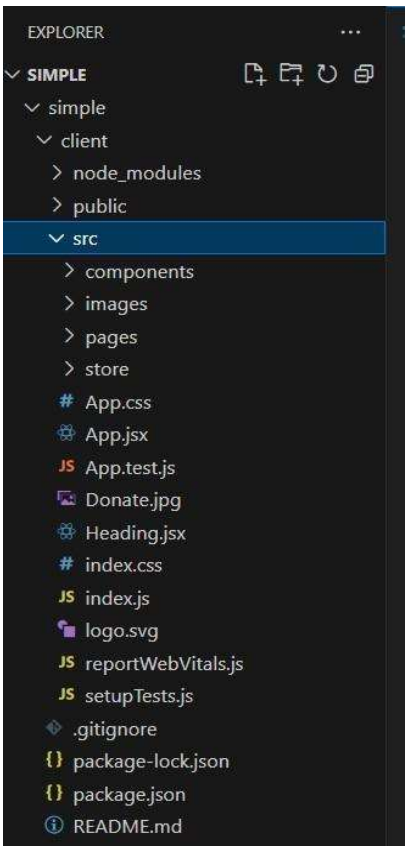
Users can donate food, money, or other items to orphanages. This feature likely includes a user interface where users can select the orphanage they want to donate to and the type of donation they wish to make. Admins can add new orphanages to the platform or delete existing ones. They can also manage the requirements of each orphanage, which likely includes specifying the items or funds needed by each orphanage.

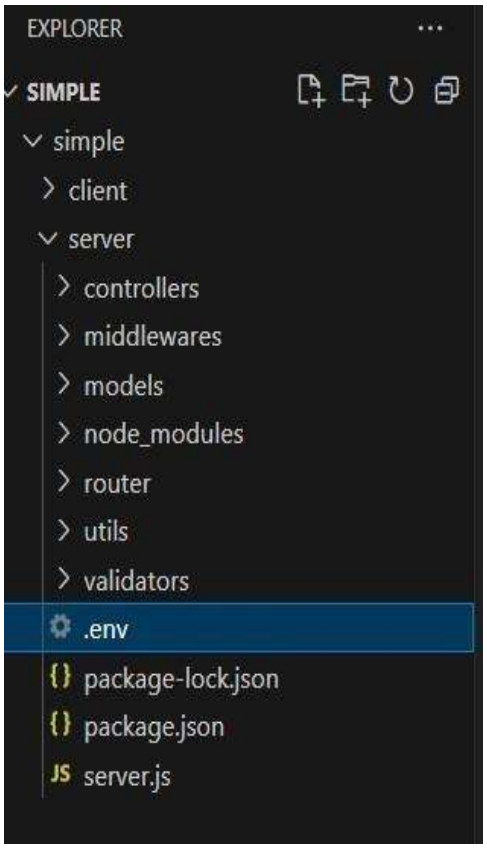
The project also includes a crowdfunding feature based on the requirements of orphanages. This could involve displaying the current requirements of each orphanage and allowing users to donate money towards fulfilling those requirements.

MongoDB is a popular NoSQL document-oriented database that provides a flexible and scalable way to store and retrieve data. MongoDB stores data in JSON-like documents, which allows for greater flexibility in data modeling compared to traditional relational databases. It also provides features such as automatic sharding and high availability, making it a popular choice for building scalable web applications.

React is a popular JavaScript library for building user interfaces (UIs) in web applications. It was developed by Facebook and is widely used by developers to create dynamic and interactive UI components. React follows a component-based architecture, which means that the user interface is divided into reusable and self-contained building blocks called components.

CHAPTER 2: DESCRIPTION OF THE CODE





FILENAME:admin-controller.js

1. **getAllUsers**: Retrieves all users from the database, excluding the password field, and returns them as a JSON response. If no users are found, it returns a 404 status with a message.

2. **updateUserById:** Updates a user's data based on their ID. It takes the ID from the request parameters and the updated data from the request body. It uses User.updateOne to update the user and returns the updated data as a JSON response.

3. **postServices**: Creates a new service entry in the database for food. It takes the photo, description, place, and need from the request body, creates a new Food document, and returns the created document as a JSON response.

4. **deleteUserById**: Deletes a user based on their ID. It takes the ID from the request parameters, uses User.deleteOne to delete the user, and returns a success message as a JSON response.

5. **getUserById**: Retrieves a user's data based on their ID, excluding the password field, and returns it as a JSON response.

6. **getAllContacts**: Retrieves all contacts from the database and returns them as a JSON response. If no contacts are found, it returns a 404 status with a message.

7. **deleteContactById**: Deletes a contact based on their ID. It takes the ID from the request parameters, uses Contact.deleteOne to delete the contact, and returns a success message as a JSON response.

FILENAME:auth-controller.js

1. **home:** Handles requests to the root URL ("/") and simply returns a message indicating that the server is running using the router.

2. **register:** Handles user registration requests. It checks if the provided email already exists in the database. If not, it creates a new user with the provided username, email, and password, and returns a success message along with a token generated for the user.

3. **login:** Handles user login requests. It checks if the provided email exists in the database and if the password matches the stored hash. If successful, it returns a success message along with a token generated for the user.

4. **user:** Handles requests to retrieve user data. It assumes that the request is authenticated and extracts the user data from the request object (presumably set by a middleware). It then returns the user data as a JSON response.

FILENAME:contact-controller.js

The code defines a route handler for handling contact form submissions. It expects the request body to contain the necessary contact form data, which it then saves to the database using the **Contact.create** method.

contactForm: Handles requests to submit a contact form. It expects the request body to contain the necessary data for the contact form submission. It creates a new Contact document in the database with the provided data and returns a success message if the operation is successful. If there's an error, it returns a 500 status with a message indicating

that the submission was not successful.

FILENAME:food-controller.js

foods: Handles requests to fetch all food services. It uses to retrieve all food services from the database. If no services are found, it returns a 404 status with a message indicating that no services were found. If services are found, it returns a 200 status with the list of services as a JSON response

.

FILENAME:admin-middleware.js

adminMiddleware: Middleware function that checks if the user making the request is an admin. It accesses the isAdmin flag from the req.user object, which is assumed to be set by a previous authentication middleware. If the user is not an admin, it returns a 403 status with a message indicating access denied. Otherwise, it calls next() to pass control to the next middleware.

FILENAME:auth-middleware.js

The code defines an authentication middleware that verifies a JWT token provided in the Authorization header. It uses the jsonwebtoken library to verify the token against the JWT_SECRET_KEY stored in your environment variables. If the token is valid, it retrieves the user data from the database based on the email address extracted from the token and attaches it to the req.user object for use in subsequent middleware or route handlers.

1. **authMiddleware:** Middleware function that verifies the JWT token provided in the Authorization header. If the token is not provided or is invalid, it returns a 401 status with a message indicating unauthorized access. If the token is valid, it extracts the user data from the database based on the email address in the token and attaches it to the req.user object.

FILENAME:error-middleware.js

errorMiddleware: Middleware function that handles errors in the application. It expects errors to be passed with properties status (defaulting to 500 if not provided), message (defaulting to

"backend error" if not provided), and extraDetails (defaulting to "error from backend" if not provided). It then returns a JSON response with the provided status code and error message.

FILENAME:validate-middleware.js

validate: Middleware function that validates the request body using a schema. It expects the schema to have a **parseAsync** method. If the validation fails, it constructs an error object with a 422 status code, a message indicating that the input should be filled properly, and the specific validation error message from the schema. It then passes this error object to the next middleware using **next(error)**.

FILENAME:contact-model.js

The code defines a Mongoose schema for the contact model with three fields: username, email, and message. Each field has a type of String and is required. The schema is then used to create a Mongoose model called Contact, which is exported for use in other parts of your application.

Here's a summary of the contact schema:

• username: A required string field representing the username of the contact.

• email: A required string field representing the email address of the contact.

• message: A required string field representing the message sent by the contact.

FILENAME:food-model.js

1. **Imports:** The code imports the Schema and model functions from the mongoose package.

2. **Schema Definition:** The foodSchema is defined using the Schema constructor. It has four fields:

• **photo:** A required string field representing the URL of the food photo.

• **description:** A required string field representing the description of the food.

• **place:** A required string field representing the place where the food is available.

• **need:** A required string field representing the need for the food (e.g., urgent,

regular).

3. **Model Creation:** The Food model is created using the model function, which takes two arguments:

• The name of the collection in the database ("Food").

• The schema (foodSchema) that defines the structure of documents in the collection.

4. **Export:** The Food model is exported so that it can be used in other parts of the application to interact with the "Food" collection in the database.

FILENAME:user-model.js

1. **Imports:** The code imports mongoose for database operations, bcryptjs for hashing passwords, and jsonwebtoken for generating JWT tokens.

2. **Schema Definition:** The userSchema is defined using the Schema constructor. It has four fields:

• **username:** A string field representing the username of the user.

• **email:** A string field representing the email address of the user.

• **password:** A string field representing the hashed password of the user.

• **isAdmin**: A boolean field representing whether the user is an admin (default is false).

3. **Middleware (pre-save):** The userSchema defines a pre-save middleware function that hashes the password before saving it to the database. It uses bcrypt.hash to hash the password with a salt round of 10.

4. **Method (generateToken):** The userSchema defines a method called generateToken that generates a JWT token for the user. It signs a payload containing the user's ID, email, and admin status using the jsonwebtoken.sign method and returns the token. The token expires in 30 days.

5. **Model Creation:** The User model is created using the mongoose.model method, which takes two arguments:

• The name of the collection in the database ("User").

• The schema (userSchema) that defines the structure of documents in the collection.

6. **Export:** The User model is exported so that it can be used in other parts of the application to interact with the "User" collection in the database.

FILENAME:admin-router.js

This code sets up Express routes for handling admin-related functionalities. It defines routes for managing users, contacts, and services, and uses middleware to ensure that only authenticated users with admin privileges can access these routes.

Here's a summary of the routes defined in the code:

1. /users:

• GET: Fetch all users.

2. /users/delete/:id:

• DELETE: Delete a user by ID.

3. /users/:id:

• GET: Get a user by ID.

4. /users/update/:id:

• PUT: Update a user by ID.

5. /contacts:

• GET: Fetch all contacts.

6. /contacts/delete/:id:

• DELETE: Delete a contact by ID.

7. /services:

• POST: Create a new service.

For each route, the code specifies that both the **authMiddleware** and **adminMiddleware** should be applied, ensuring that the user is authenticated and has admin privileges before accessing the routes. The actual request handling logic is implemented in the **adminController** module.

FILENAME:auth-router.js

This code sets up Express routes for authentication-related functionalities, such as user registration, login, and fetching user data. It uses middleware for request validation (validate) and authentication (authMiddleware), and it defines routes for handling these operations using the corresponding controller methods (authcontrollers).

Here's a summary of the routes defined in the code:

1**. /:**

• **GET**: Home route, returns a message indicating that the server is running.

2. **/register**:

• **POST:** Register a new user. It uses the validate middleware with the signupSchema to validate the request body before calling the register method in the authcontrollers module.

3. **/login**:

• POST: User login. It calls the login method in the authcontrollers module.

4. **/user:**

• GET: Get user data. It uses the authMiddleware to ensure the user is authenticated before calling the user method in the authcontrollers module.

FILENAME:db.js

1. **URI**: Specifies the connection URI for the MongoDB Atlas cluster. It includes the username, password, cluster address, database name, and options.

2. **connectDb:** An async function that attempts to connect to the MongoDB database using mongoose.connect. If the connection is successful, it logs "connection successful" to the console. If there's an error, it logs "connection failed" and exits the Node.js process.

3. **Export**: The connectDb function is exported so that it can be used to connect to the MongoDB database in other parts of the application.

FILENAME:auth-validator.js

1. **Imports:** The code imports the z object from the zod package, which is used to define schemas for data validation.

2. **signupSchema:** The schema is defined using the z.object method, which creates an object schema with specific validation rules for each field:

• **username**: A string field that is required, trimmed, and must be between 3 and 255 characters long.

address between 3 and 255 characters long.

• **email:** A string field that is required, trimmed, and must be a valid email

• **password:** A string field that is required and must be between 6 and 1024 characters long.

Each field specifies custom error messages for required fields and length constraints.

3. **Export:** The signupSchema object is exported so that it can be used to validate user signup data in other parts of the application.

FILENAME:server.js

This code sets up an Express server with routes for user authentication, contact form submissions, food data management, and admin functionalities. It uses middleware for handling CORS, parsing JSON requests, and error handling. Here's a breakdown of the code:

1. **dotenv Configuration:** Loads environment variables from a .env file into process.env.

2. **Imports:**

• express: Creates an Express application.

• cors: Handles Cross-Origin Resource Sharing.

• authRoute, contactRoute, foodRoute, adminRoute: Routes for different parts of the application.

• connectDb: Function to connect to the MongoDB database.

• errorMiddleware: Middleware for handling errors.

3. **Express Setup:**

• app.use(cors()): Enables CORS for all routes.

• app.use(express.json()): Parses incoming JSON requests.

• app.use("/api/auth",authRoute), app.use("/api/form",contactRoute), app.use("/api/data",foodRoute), app.use("/api/admin",adminRoute): Routes for different parts of the application.

• app.use(errorMiddleware): Error handling middleware.

4. **Server Start:**

• connectDb().then(...): Connects to the MongoDB database.

• app.listen(PORT, ...): Starts the server on port 5000 and logs a message to the console.

FILENAME:Admin-Layout.jsx

This code defines a React component called AdminLayout that serves as the layout for the admin section of the application. It uses React Router's NavLink, Navigate, and Outlet components for navigation and rendering child routes.

1. **Imports:**

• NavLink, Navigate, Outlet from "react-router-dom": Components for navigation and rendering nested routes.

• useAuth from "../../store/auth": Custom hook for accessing authentication state.

2. **Component Logic:**

• Uses the useAuth hook to access the authentication state (user and isLoading).

• If isLoading is true, it displays a loading message.

• If the user is not an admin (user.isAdmin is false), it redirects to the home page (<Navigate to="/" />).

• Otherwise, it renders the admin layout with a header containing navigation links (<NavLink /> components) for managing users, contacts, services, and home.

3. **Rendering:**

• Renders a header with a navigation bar (<nav> element) containing the

navigation links (<NavLink /> components).

• Uses the Outlet component to render child routes defined in the parent route.

FILENAME:Admin-Contacts.jsx

1. **Imports:**

• useEffect, useState from "react": Hooks for managing side effects and state in functional components.

• useAuth from "../store/auth": Custom hook for accessing authentication state.

2. **Component Logic:**

• Defines a state contactData to store the contact data fetched from the server.

• Uses the useAuth hook to access the authorization token for making authenticated requests.

using the authorization token.

• Defines a function getContactsData to fetch contact data from the server authorization token.

• Defines a function deleteContactById to delete a contact by its ID using the

• Uses the useEffect hook to fetch contact data when the component mounts.

3. **Rendering:**

• Renders a table with contact data fetched from the server.

• Provides a delete button for each contact row, which calls the deleteContactById function with the contact's ID as an argument.

FILENAME:App.jsx

1. **Imports:**

• logo and ./App.css are imported for logo and CSS styling, respectively.

• BrowserRouter, Routes, and Route are imported from "react-router-dom" for setting up routing in the application.

• Various components and pages are imported, including Home, About, Login,

Logout, Food, Signup, Contact, Navbar, List, AdminLayout, AdminUsers, AdminContacts, AdminUpdate, AdminServices, Healthchecks, and Clothing.

2. **Component Functionality:**

• The App component is a functional component that renders the main content of the application.

• Inside the App component, the BrowserRouter component wraps the application, providing the routing functionality.

• The Routes component contains all the route definitions for the application.

• Each Route component defines a path and the component to render when that path is matched.

• Nested routes are used for the admin section (/admin), where the AdminLayout component acts as the layout for all admin-related pages.

• Nested Route components inside the AdminLayout component define the paths for different admin pages (users, contacts, services, etc.).

3. **Rendering:**

• The Navbar component is rendered outside the Routes component to provide navigation links at the top of the page.

component to render when that path is matched.

• Inside the Routes component, each Route component defines a path and the

• The element prop of each Route component is an array containing the components to render for that route. This is used for rendering multiple components for a single route.

4. **Route Configuration:**

• The / route renders the Home, About, and Contact components.

CHAPTER 3 : CODE OF THE PROJECT:

FILENAME:admin-controller.js

```
const User=require("../models/user-model")
const Contact=require("../models/contact-model")
const Food=require("../models/food-model")
const getAllUsers=async(req,res)=>{ try{
  const users=await User.find({},{password:0});
  console.log(users)
  if(!users || users.length==0)
  {
       return res.status(404).json({message:"no users found"})
  }
  return res.status(200).json(users);
}catch(error){
console.log(error)
}
}
const updateUserById=async(req,res)=>{ try{
         const id=req.params.id;
         const updateUserData=req.body;
         const updatedData=await User.updateOne({_id:id},{
           $set:updateUserData,
         })
     return res.status(200).json(updatedData);
     }catch(error){ next(error)
     }
}
const postServices=async(req,res)=>{ try{
         const {photo,description,place,need}=req.body;
         const updatedData=await Food.create({photo,description,place,need});
         console.log(req.body);
     return res.status(200).json(updatedData);

     }catch(error){
   console.log(error);
     }
}
const deleteUserById=async(req,res)=>{ try{
     const id=req.params.id;
     await User.deleteOne({_id:id});
     return res.status(200).json({message:"user deleted successfully"})
   }catch(error){ next(error)
```

```
    }
}
const getUserById=async(req,res)=>{ try{
        const id=req.params.id;
        const data=await User.findOne({_id:id},{password:0}) return
        res.status(200).json(data)
      }catch(error){ next(error)
      }
  }
const getAllContacts=async(req,res)=>{ try{
              const contacts=await Contact.find();
              console.log(contacts)
              if(!contacts || contacts.length==0)
              {
                  return res.status(404).json({message:"no contacts found"})
              }
              return res.status(200).json(contacts);
            }catch(error){ console.log(error)
            }
}
const deleteContactById=async(req,res)=>{ try{
        const id=req.params.id;
        await Contact.deleteOne({_id:id});
        return res.status(200).json({message:"contact deleted successfully"})
      }catch(error){ next(error)
      }
  }
module.exports={getAllUsers,getAllContacts,deleteUserById,getUserById,updateUs
erById,deleteContactById,postServices};
```

FILENAME:auth-controller.js

```
const User=require("../models/user-model");
const bcrypt=require("bcryptjs");
const home=async(req,res)=>{ try{
        res.status(200).send("server is running using router");
    }catch(error)
    {
        console.log(error);
    }
}
const register=async(req,res)=>{ try{
              const {username,email,password}=req.body; const
              userExist=await User.findOne({email});
              if(userExist){
                  return res.status(400).json({msg:"email already exist"});
```

```
            }
            const usercreated=await User.create({username,email,password});
            console.log(req.body); res.status(201).json({msg:usercreated,token:await
usercreated.generateToken(),userId:usercreated._id.toString(),});
        }
        catch(error){
            next(error);
        }

}
const login=async(req,res)=>{ try{
        const {email, password}=req.body;
        const userExist=await User.findOne({email});
        if(!userExist){
            return res.status(400).json({message:"Invalid credentials"});
        }
      const user=await bcrypt.compare(password,userExist.password); if(user)
      {
        res.status(200).json({msg:"login successful",token:await
userExist.generateToken(),userId:userExist._id.toString(),});
      }
      else
      {
        res.status(401).json({message:"invalid email or password"});
      }
   }
   catch(error){
      res.send(500).json({msg:"page not found"});
}
};
const user=async(req,res)=>{ try{
        const userData=req.user;
        console.log(userData);
        return res.status(200).json({userData});

   }
   catch(error)
   {
      console.log(`error from the user route ${error}`);
   }
}
module.exports={home,register,login,user};
```

FILENAME:contact-controller.js

```
const Contact=require("../models/contact-model"); const
contactForm=async(req,res)=>{
      try{
            const response=req.body;
            await Contact.create(response);
            return res.status(200).json({message:"send successfully"});
      }catch(error){
            return res.status(500).json({message:"not send successfully"});
      }
};
module.exports=contactForm;
```

FILENAME:food-controller.js

```
const Food=require('../models/food-model'); const

foods=async(req,res)=>{
      try{
         const response=await Food.find(); if(!response){
            res.status(404).json({msg:"no service found"}); return;
         }
         res.status(200).json({msg:response});
      }catch(error)
      {
            console.log(`error from food page ${error}`);
      }
};

module.exports=foods;
```

FILENAME:admin-middleware.js

```
const adminMiddleware=async(req,res,next)=>{ try{
            console.log(req.user);
            const adminRole=req.user.isAdmin; if(!adminRole){
                  return res.status(403).json({message:"access denied.User is not an
admin"})
            }
            next()
      }catch(error)
      {
            next(error)
      }
```

```
}

module.exports=adminMiddleware
```

FILENAME:contact-model.js

```
const {Schema,model}=require("mongoose");
const contactSchema=new Schema({
    username:{
        type:String,
        required:true
    },
    email:{
        type:String,
        required:true
    },
    message:{
        type:String,
        required:true
    },
});
const Contact=new model('Contact',contactSchema);
module.exports=Contact;
```

FILENAME:user-model.js

```
const mongoose=require('mongoose');
 const bcrypt=require("bcryptjs");
const jwt=require("jsonwebtoken");
const userSchema=new mongoose.Schema({
    username:{
        type:String,
    },
    email:{
        type:String,
    },
    password:{
        type:String,
    },
    isAdmin:{
        type:Boolean,
        default:false,
    }
});
userSchema.pre('save',async function(next){
const user=this; if(!user.isModified('password'))
```

```
        next();
try{
        const saltRound=await bcrypt.genSalt(10);
        const hash_password=await bcrypt.hash(user.password,saltRound);
        user.password=hash_password;
}
catch(error)
{
        next(error);
}
});
userSchema.methods.generateToken=async function(){ try{
         return jwt.sign({
              userId:this._id.toString(),
              email:this.email,
              isAdmin:this.isAdmin,
           },process.env.JWT_SECRET_KEY,{
              expiresIn:"30d",
         });
    }catch(error)
    {
        console.error(error);
    }
}
const User=new mongoose.model("User",userSchema);
module.exports=User;
```

FILENAME:admin-router.js

```
const express=require('express');
const adminController = require('../controllers/admin-controller'); const
authMiddleware=require("../middlewares/auth-middleware"); const
adminMiddleware = require('../middlewares/admin-middleware'); const
router=express.Router();
router.route('/users').get(authMiddleware,adminMiddleware,adminController.getA llUsers);
router.route('/users/delete/:id').delete(authMiddleware,adminMiddleware,adminC
ontroller.deleteUserById)
router.route('/users/:id').get(authMiddleware,adminMiddleware,adminController. getUserById)
router.route('/users/update/:id').put(authMiddleware,adminMiddleware,adminCont
roller.updateUserById)
router.route("/contacts").get(authMiddleware,adminMiddleware,adminController.g
etAllContacts) router.route('/contacts/delete/:id').delete(authMiddleware,adminMiddleware,adm
inController.deleteContactById)
router.route('/services').post(authMiddleware,adminMiddleware,adminController. postServices)
module.exports=router;
```

FILENAME:db.js

```
const mongoose=require('mongoose'); const
URI="mongodb+srv://21501a05a1:sowjanya@cluster0.eqobpmu.mongodb.net/msd?retryW
rites=true&w=majority"
const connectDb=async()=>{
    try{
        await mongoose.connect(URI);
        console.log("connection successful");
    }
    catch(error)
    {
        console.error("connection failed"); process.exit(0);
    }
};
module.exports=connectDb;
```

FILENAME:server.js

```
require("dotenv").config();   const
express=require('express'); const
cors=require('cors');
const app=express();
const  authRoute=require("./router/auth-router");
const contactRoute=require("./router/contact-router"); const
foodRoute=require("./router/food-router")
const connectDb=require('./utils/db');
const errorMiddleware = require("./middlewares/error-middleware"); const
adminRoute=require('./router/admin-router')
app.use(cors()); app.use(express.json());
app.use("/api/auth",authRoute);
app.use("/api/form",contactRoute);
app.use("/api/data",foodRoute);
app.use("/api/admin",adminRoute)
app.use(errorMiddleware);
const PORT=5000; connectDb().then(()=>{
    app.listen(PORT,()=>{
        console.log(`server is running at port: ${PORT}`);
    });
});
```

FILENAME:Admin-Layout.jsx

```
import { NavLink, Navigate, Outlet } from "react-router-dom"; import {
useAuth } from "../../store/auth";

const AdminLayout = () => {
   const {user,isLoading}=useAuth()
   console.log(user)
   if(isLoading)
   {
      return <h1>
         loading...
      </h1>
   }
   if(!user.isAdmin){
      return <Navigate to="/"/>
   }
   return (
      <>
         <header>
            <div className="container">
               <nav className="navbar navbar-expand-lg navbar-light bg-light">
                  <div className="collapse navbar-collapse" id="navbarSupportedContent">
                     <ul className="navbar-nav mr-auto">
                        <li className="nav-item">
                           <NavLink className="nav-link" to="/admin/users"> Users
                           </NavLink>
                        </li>
                        <li className="nav-item">
                           <NavLink className="nav-link" to="/admin/contacts">
                              Contacts
                           </NavLink>
                        </li>
                        <li className="nav-item">
                           <NavLink className="nav-link" to="/admin/services">
                              Services
                           </NavLink>
                        </li>
                        <li className="nav-item">
                           <NavLink className="nav-link" to="/admin/home">
                              Home
                           </NavLink>
                        </li>
                     </ul>
                  </div>
```

```
            </nav>
          </div>
        </header>
        <Outlet />
      </>
    );
};

export default AdminLayout;
```

FILENAME:Admin-Contact.jsx

```
import { useEffect, useState } from "react"; import {
useAuth } from "../store/auth";
export const AdminContacts=()=>{
    const [contactData,setContactData]=useState([]) const {
    authorizationToken } = useAuth(); const
    getContactsData=async()=>{
        try{
          const response=await fetch('http://localhost:5000/api/admin/contacts',{
            method:"GET",
            headers:{
                Authorization:authorizationToken
            }
          })
          const data=await response.json()
          console.log("contact data",data); if(response.ok)
          {
            setContactData(data)
          }
        }catch(error){ console.log(error)
        }
    }
    const deleteContactById=async(id)=>{ try {
            const response = await
fetch(`http://localhost:5000/api/admin/contacts/delete/${id}`, {
                method: "DELETE",
                headers: {
```

```
                    Authorization: authorizationToken,
                  },
                });

                if(response.ok)
              {
                  getContactsData();
              }
            } catch (error) { console.log(error);
            }

        }

    useEffect(()=>{getContactsData()},[]) return(
        <>
        <section  className="admin-users-section">
        <div className="container">
          <h1>Admin Contact Data</h1>
        </div>
        <div className="container admin-users">
          <table className="table">
            <tbody>
              {contactData.map((curContactData, index) => { const
                {_id}=curContactData
                return (
                  <tr key={index}>
                    <td>{curContactData.username}</td>
                    <td>{curContactData.email}</td>
                    <td>{curContactData.message}</td>

                    <td><button
onClick={()=>deleteContactById(_id)}>delete</button></td>
                  </tr>
                );
              })}
            </tbody>
          </table>
        </div>
      </section>
        </>
    )
}
```

FILENAME:Home.jsx

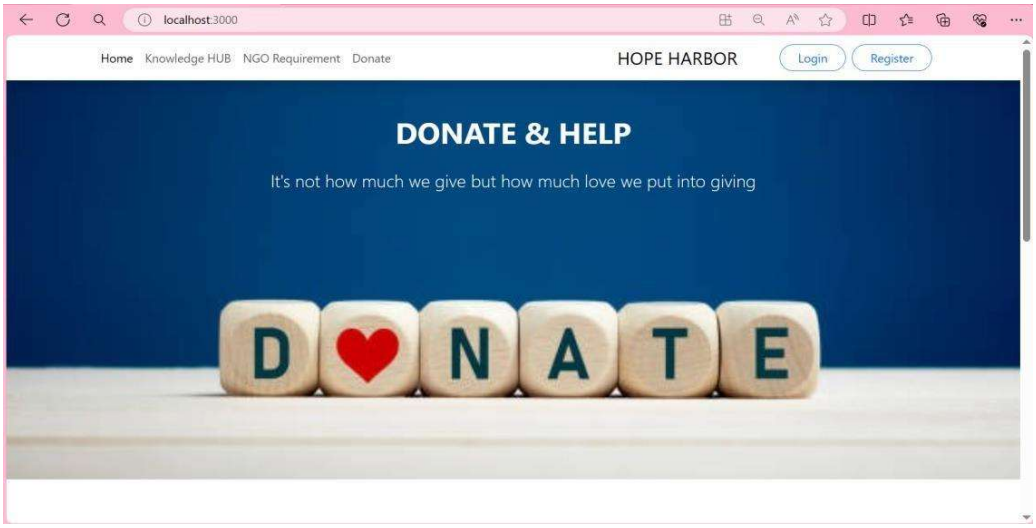
```
import React from "react" import
About from './About';
import Container from 'react-bootstrap/Container'; import Row
from 'react-bootstrap/Row';
import Col from 'react-bootstrap/Col'; import Image
from 'react-bootstrap/Image'; import photo1 from
'../images/photo1.jpeg'; const Home = () =>{
      return(
            <div>
                  <section id="home">
                        <div className="container">
                              <div className="row justify-content-center">
                                    <div className="col-md-8 mt-5">
                                          <h1 className="diaplay-4 fw-bolder mb-4 text-
center text-white">
                                                DONATE & HELP
                                          </h1>
                                          <p className="lead text-center fs-4 mb-5 text-
white">It's not how much we give but how much love we put into giving</p>

                                    </div>
                              </div>
                        </div>
                        </section>
            </div>

      );
}

export default Home;
```

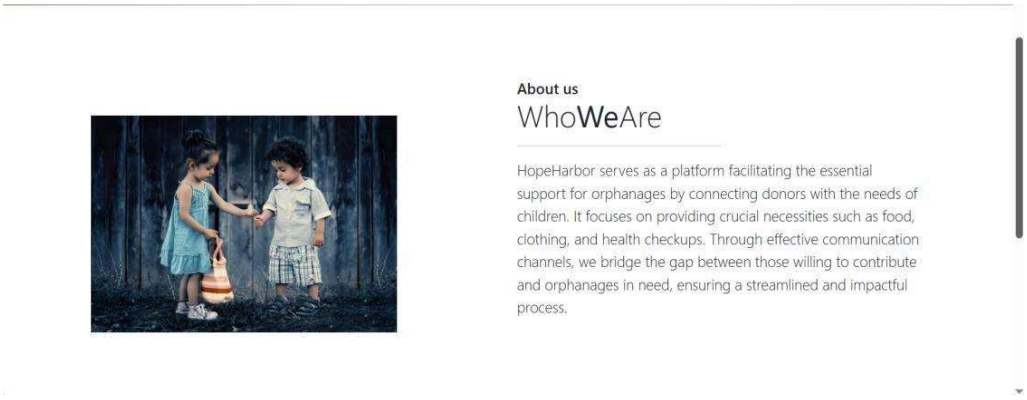
FILENAME:App.jsx

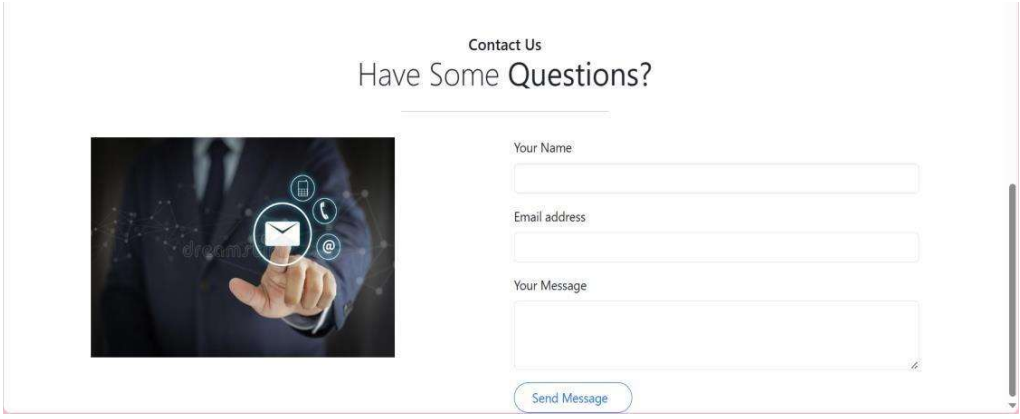
```
import logo from './logo.svg'; import
'./App.css';
import {BrowserRouter,Routes,Route}from "react-router-dom"; import
Home from './pages/Home';
import About from './pages/About'; import
Login from './pages/Login'; import {Logout}
from './pages/Logout';
```

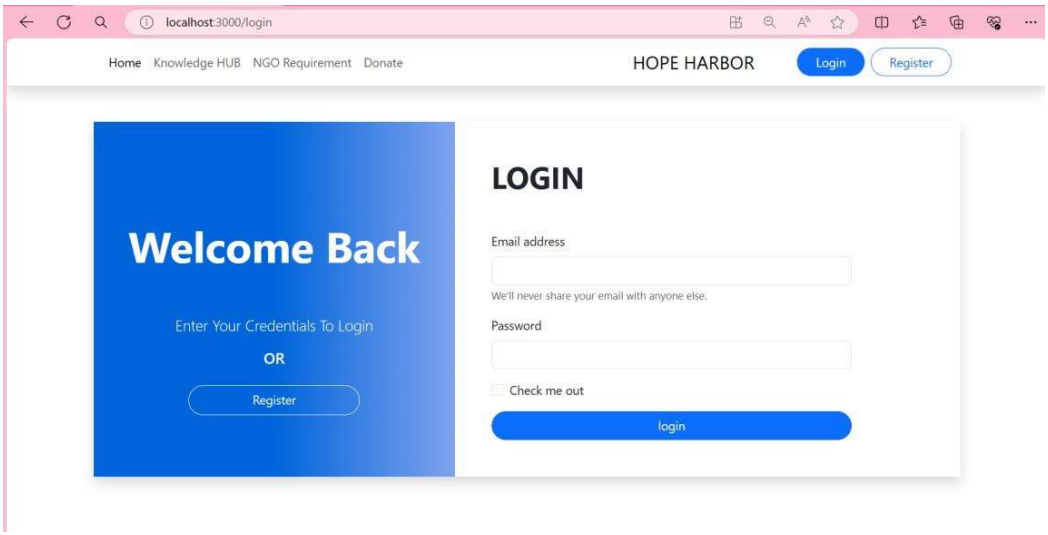
```
import {Food} from './pages/Food'; import
Signup from './pages/Signup'; import Contact
from './pages/Contact'; import Navbar from
'./pages/Navbar'; import List from
'./pages/List';
import 'bootstrap/dist/css/bootstrap.min.css';
import AdminLayout from './components/layouts/Admin-Layout'; import {
AdminUsers } from './pages/Admin-Users';
import { AdminContacts } from './pages/Admin-Contacts'; import {
AdminUpdate } from './pages/Admin-Update'; import
AdminServices from './pages/Admin-Services'; import {
Healthchecks } from './pages/Healthchecks'; import { Clothing }
from './pages/Clothing';
function App() { return (
      <>
       <BrowserRouter>
       <Navbar/>
         <Routes>
         <Route path="/" element={[<Home/>,<About/>,<Contact/>]} />
         <Route path="/about" element={<About />}/>
         <Route path="/contact" element={<Contact />}/>
         <Route path="/login" element={<Login />}/>
         <Route path="/logout" element={<Logout />}/>
         <Route path="/signup" element={<Signup />}/>
         <Route path="/donate" element={[<List />,<Food/>]}/>
         <Route path="/donate/healthcheckup" element={[<List
/>,<Healthchecks/>]}/>
         <Route path="/donate/clothing" element={[<List />,<Clothing/>]}/>
         <Route path="/admin" element={<AdminLayout/>}>
           <Route path="users" element={<AdminUsers/>}/>
           <Route path="contacts" element={<AdminContacts/>}/>
           <Route path='users/:id/edit' element={<AdminUpdate/>}/>
           <Route path='services' element={<AdminServices/>}/>
          </Route>
         </Routes>
         </BrowserRouter>
      </>
   );
}

export default App;
```

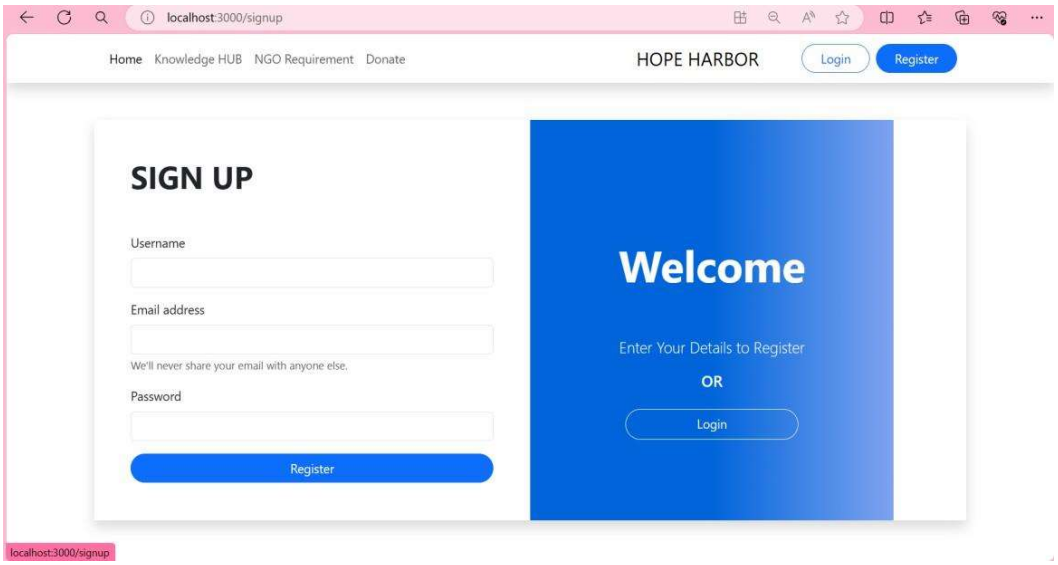
CHAPTER 4 : OUTPUT OF THE PROJECT

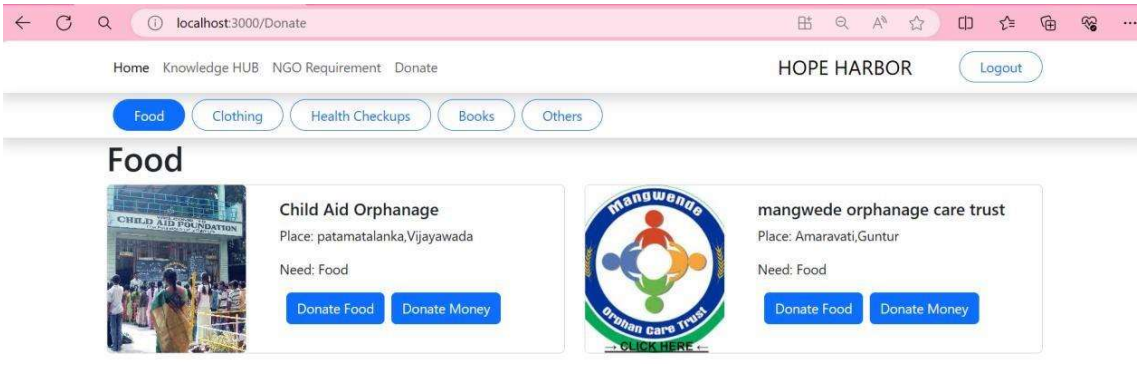


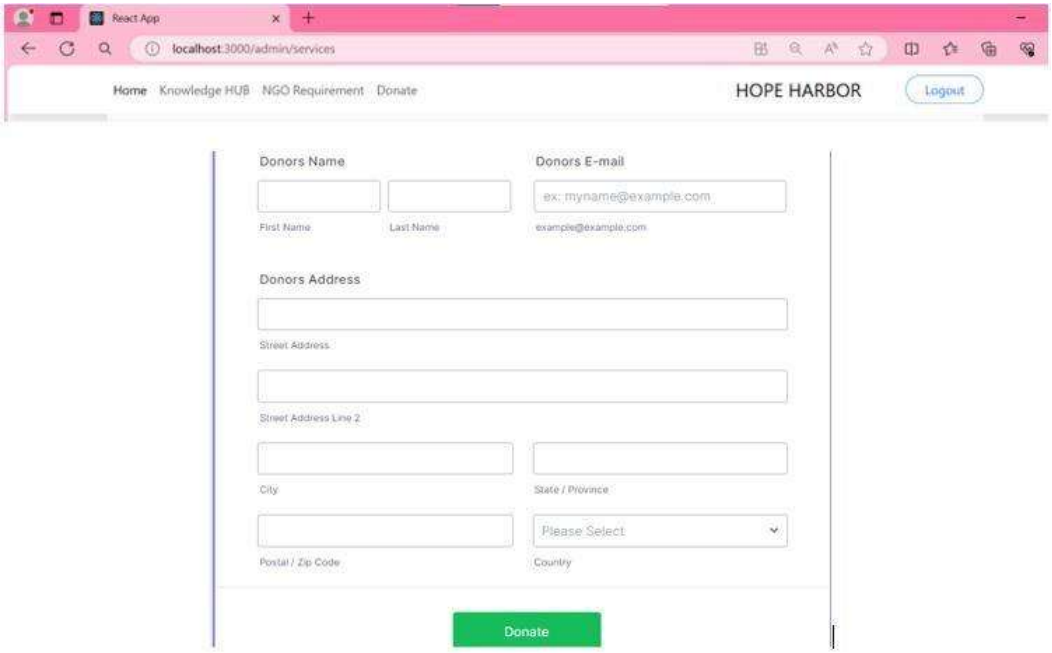


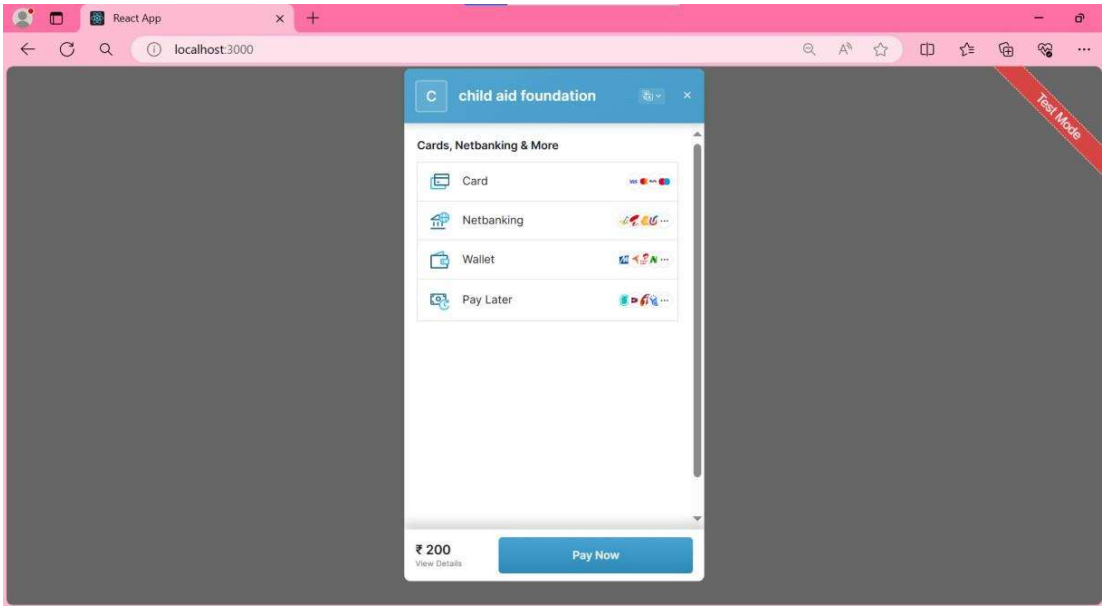


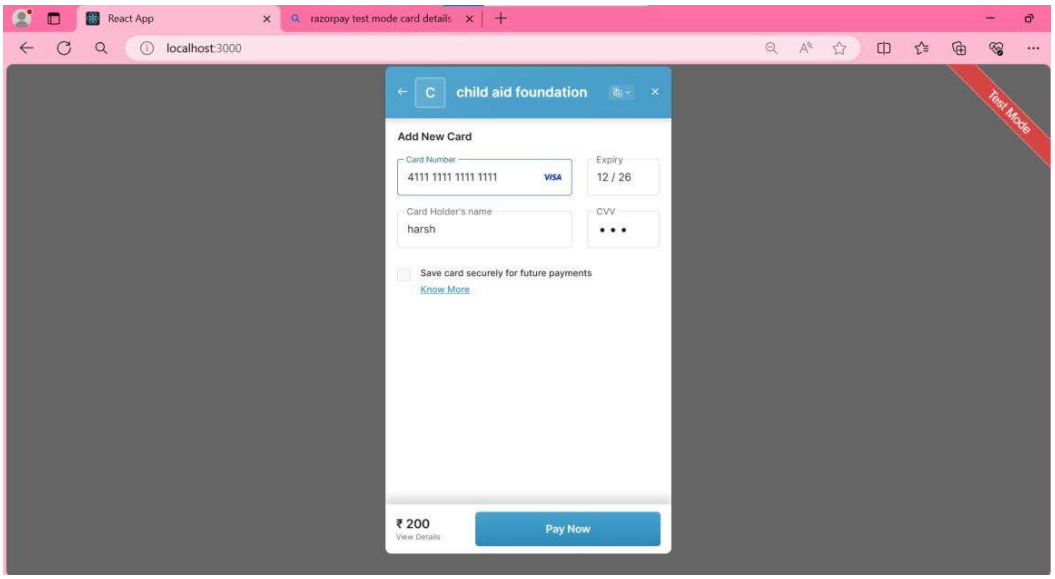

localhost:3000/login
Home
Knowledge HUB
NGO Requirement
Donate
HOPE HARBOR
Login
Register
Welcome Back
Enter Your Credentials To Login
OR
Register
LOGIN
Email address
We'll never share your email with anyone else.
Password
Check me out
login

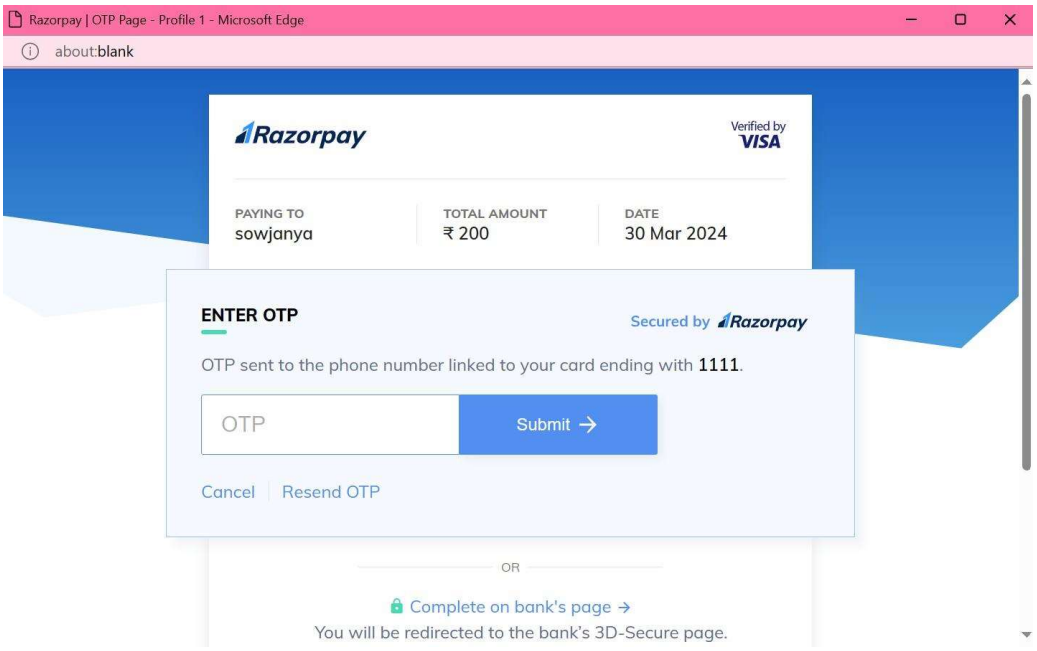

localhost:3000/signup
Home
Knowledge HUB
NGO Requirement
Donate
HOPE HARBOR
Login
Register
SIGN UP
Username
Email address
We'll never share your email with anyone else.
Password
Register
Welcome
Enter Your Details to Register
OR
Login
localhost:3000/signup

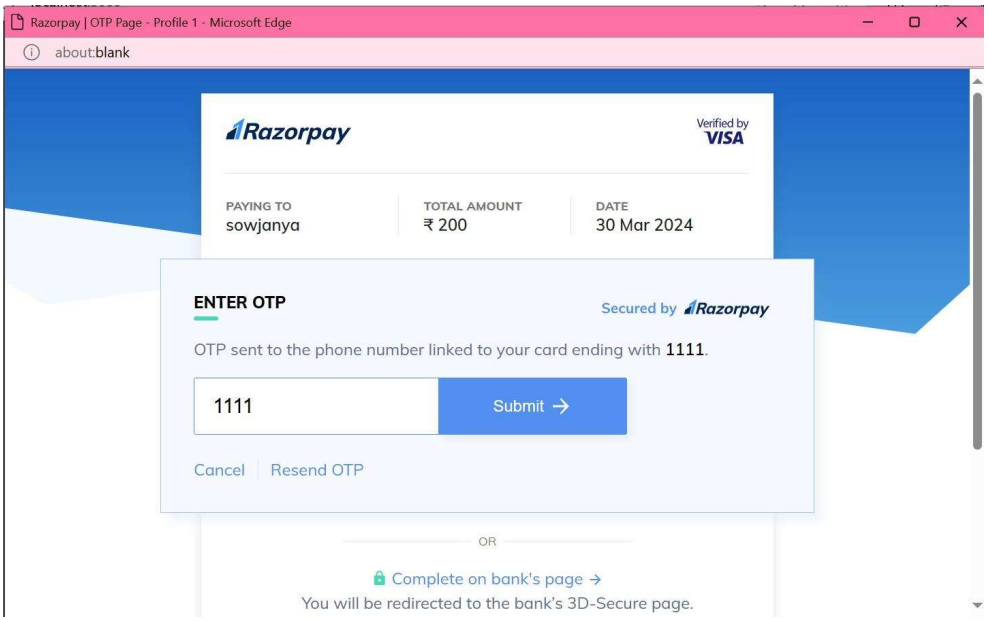

localhost:3000/Donate
Home
Knowledge HUB
NGO Requirement
Donate
HOPE HARBOR
Logout
Food
Clothing
Health Checkups
Books
Others
Food
Child Aid Orphanage
Place: patamatalanka,Vijayawada
Need: Food
Donate Food
Donate Money
mangwede orphanage care trust
Place: Amaravati,Guntur
Need: Food
Donate Food
Donate Money

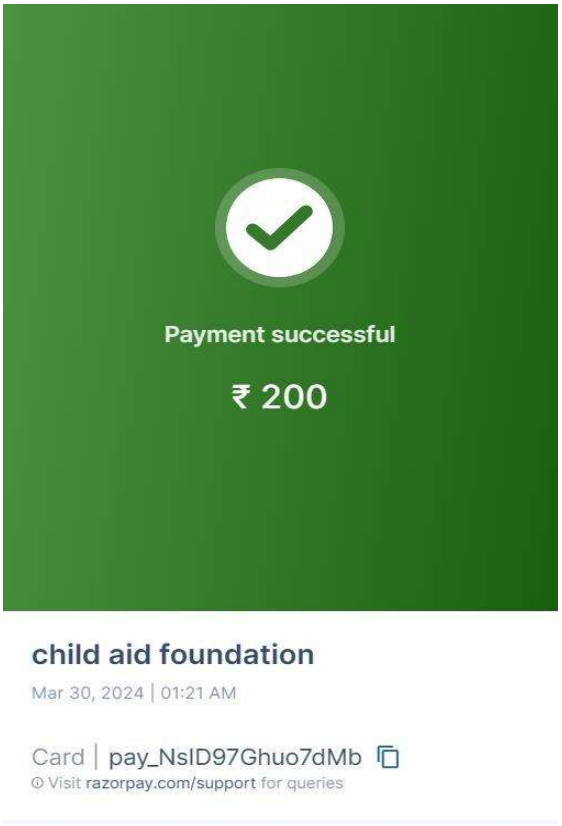

React App
localhost:3000/admin/services
Home
Knowledge HUB
NGO Requirement
Donate
HOPE HARBOR
Logout
Donors Name
First Name
Last Name
Donors E-mail
ex: myname@example.com
example@example.com
Donors Address
Street Address
Street Address Line 2
City
State / Province
Please Select
Postal / Zip Code
Country
Donate

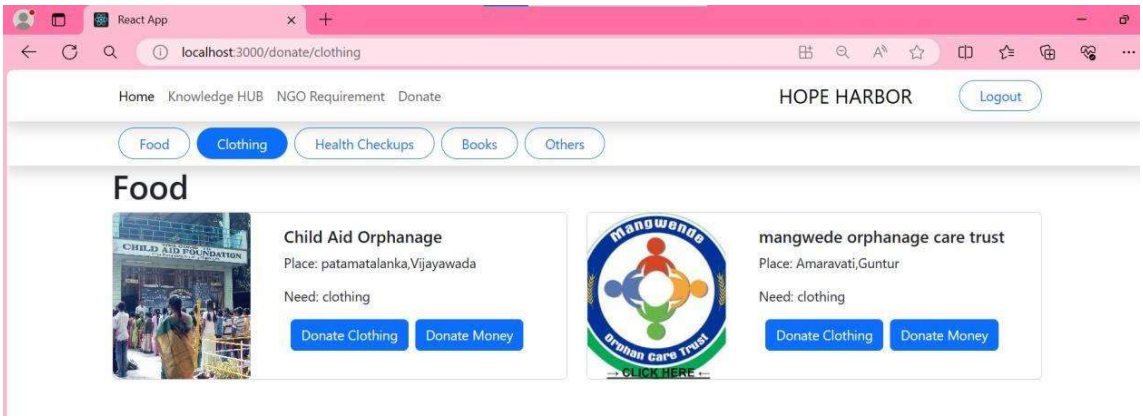

React App
localhost:3000
child aid foundation
Cards, Netbanking & More
Card
Netbanking
Wallet
Pay Later
₹ 200
View Details
Pay Now
Test Mode

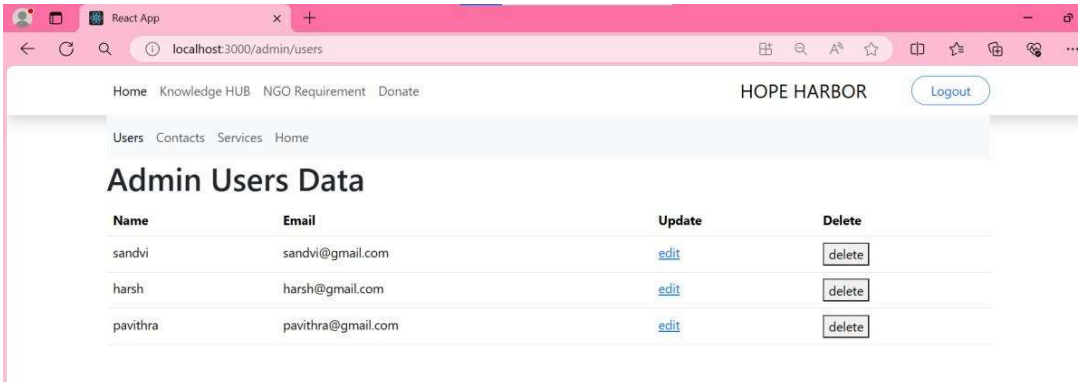

React App
razorpay test mode card details
localhost:3000
child aid foundation
Test Mode
Add New Card
Card Number
4111 1111 1111 1111
VISA
Expiry
12 / 26
Card Holder's name
harsh
CVV
• • •
Save card securely for future payments
Know More
₹ 200
View Details
Pay Now

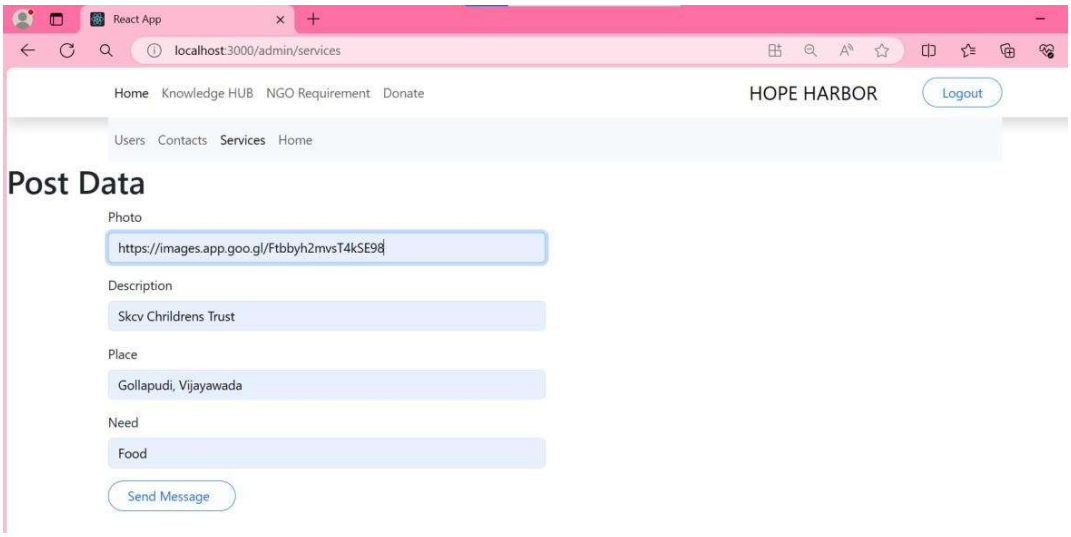

Razorpay | OTP Page - Profile 1 - Microsoft Edge
about:blank
Razorpay
Verified by VISA
PAYING TO
sowjanya
TOTAL AMOUNT
₹ 200
DATE
30 Mar 2024
ENTER OTP
Secured by Razorpay
OTP sent to the phone number linked to your card ending with 1111.
OTP
Submit
Cancel
Resend OTP
OR
Complete on bank's page
You will be redirected to the bank's 3D-Secure page.


Razorpay | OTP Page - Profile 1 - Microsoft Edge
about:blank
Razorpay
Verified by VISA
PAYING TO
sowjanya
TOTAL AMOUNT
₹ 200
DATE
30 Mar 2024
ENTER OTP
Secured by Razorpay
OTP sent to the phone number linked to your card ending with 1111.
1111
Submit
Cancel
Resend OTP
OR
Complete on bank's page
You will be redirected to the bank's 3D-Secure page.


Payment successful
₹ 200
child aid foundation
Mar 30, 2024 | 01:21 AM
Card | pay_NsID97Ghuo7dMb
Visit razorpay.com/support for queries


React App
localhost:3000/donate/clothing
Home Knowledge HUB NGO Requirement Donate
HOPE HARBOR
Logout
Food
Clothing
Health Checkups
Books
Others
Food
Child Aid Orphanage
Place: patamatalanka,Vijayawada
Need: clothing
Donate Clothing
Donate Money
mangwende orphanage care trust
Place: Amaravati,Guntur
Need: clothing
Donate Clothing
Donate Money
→ CLICK HERE ←


React App
localhost:3000/admin/users
Home Knowledge HUB NGO Requirement Donate
HOPE HARBOR
Logout
Users Contacts Services Home
Admin Users Data
Name
Email
Update
Delete
sandvi
sandvi@gmail.com
edit
delete
harsh
harsh@gmail.com
edit
delete
pavithra
pavithra@gmail.com
edit
delete


React App
localhost:3000/admin/services
Home Knowledge HUB NGO Requirement Donate
HOPE HARBOR
Logout
Users Contacts Services Home
Post Data
Photo
https://images.app.goo.gl/Ftbbyh2mvsT4kSE98
Description
Skcv Childrens Trust
Place
Gollapudi, Vijayawada
Need
Food
Send Message