**Charity Web App**

Department of Computer Science Engineering

**ARYABHATTA COLLEGE OF ENGINEERING AND RESEARCH CENTER**  
Approved by AICTE new Delhi

Chatari Road, Chachiyawas, behind Mahila Kalyan Vikas Mandal, Ajmer, Rajasthan 305004

**Submitted by:**

Bhagwan Singh 19EAJCS007

**Guided by:**

Prakash sir (HOD)



SESSION: 19-23

**Acknowledgement**

Throughout the project I go through many references, libraries, documentation and much more, all this stuff, that what to search, how to do, how to get the things done via documentation is the best and important part for development and I learn all these important basics from some great teachers and good courses.

I express my gratitude to Internshala, Deepak Sir, Sanket sir and some youtube channels like code with Harry, and Thapa Technial.

I learned a lot from them and continue learning and developing stuff.

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| Sr.no | Topic | Page No. |
| 1 | Project Overview   * key features | 3-4 |
| 2 | Objective and Scope   * Objectives * Scope | 5-7 |
| 3 | Theoretical Background | 7-9 |
| 4 | System Analysis and Design   * Requirements Analysis * Design of System * React components used * Backend Server * APIs (localhost) | 9-37 |
| 5 | Hardware and Software Details | 37-38 |
| 6 | ER Diagram and DFD   * Entity Relationship (ER) Diagram * Data Flow Diagram | 38-43 |
| 7 | Database Schema   * Payment Collection Schema | 44-46 |
| 8 | Screenshots of Input/Output Interfaces/Forms/Web Pages | 47-49 |
| 9 | List of references, sources, and research materials | 50-51 |

**Project Overview:**

The purpose of this project is to demonstrate the integration of Razorpay, a popular online payment gateway, into a web application. The project focuses on creating a simple payment flow that allows users to transfer Rs 100 in a test environment. It serves as a tutorial to showcase the key features and functionalities of Razorpay integration.

Key Features:

1. Razorpay Integration:   
   The project demonstrates the integration of the Razorpay payment gateway into the web app. It covers the necessary steps, such as setting up a Razorpay account, obtaining API keys, and configuring the required parameters.
2. Test Mode:   
   The project operates in test mode, allowing users to make transactions without using real money. This ensures that the integration can be tested and validated without incurring any actual financial transactions.
3. Payment Flow:   
   The web app showcases a simplified payment flow, where users can initiate a transaction to transfer Rs 100. It guides users through the payment process, from selecting the payment method to completing the transaction.
4. Transaction Handling:   
   The project demonstrates how to handle the transaction response from Razorpay. It covers aspects such as verifying the transaction status, updating the app's database or records accordingly, and displaying appropriate success or failure messages to the user.
5. Error Handling:   
   The tutorial project includes error handling mechanisms, ensuring that users are provided with meaningful error messages in case of any issues during the payment process. This allows for a smooth user experience and helps in troubleshooting potential integration issues.
6. Security Considerations:   
   The project emphasizes the importance of security when handling payments. It highlights best practices for securely transmitting sensitive data, such as using secure communication protocols (HTTPS) and storing API keys securely.
7. Overall,   
   this tutorial project serves as a practical introduction to integrating Razorpay into a web app, providing a foundation for developers to incorporate secure and reliable payment functionalities into their own projects.

**Objective and Scope:**

**Objectives:**

1. Integration Proficiency:   
   The project aims to enhance your understanding of integrating third-party payment gateways, specifically Razorpay, into a web application. By completing this project, you will gain hands-on experience in working with payment APIs, handling transactions, and managing the payment flow.
2. Test Environment Familiarity:   
   The project focuses on utilizing Razorpay's test mode to create a safe and controlled environment for learning and experimentation. It allows you to become familiar with the various features and functionalities of Razorpay without involving real financial transactions.
3. Payment Flow Implementation:   
   The primary objective of the project is to implement a simplified payment flow that enables users to transfer Rs 100. This involves designing the user interface, handling the transaction process, and appropriately handling the response from Razorpay.
4. Error Handling and Security:   
   The project aims to educate you about the importance of error handling and security considerations when integrating payment gateways. It helps you develop skills in handling errors gracefully and implementing security measures to protect sensitive data during the payment process.

**Scope:**

1. Integration Scope:   
   The project focuses solely on the integration of Razorpay as the chosen payment gateway. It covers the necessary steps to set up and configure Razorpay API integration within the web app environment.
2. Test Environment Limitation:   
   Since the project operates in test mode, it does not involve real financial transactions. The scope is limited to simulating the payment process and validating the integration's functionality without actual money transfers.
3. Minimal Functionality:   
   The project emphasizes simplicity by implementing a basic payment flow for transferring Rs 100. It does not delve into complex features like subscription billing, multiple payment methods, or advanced customization. The primary goal is to provide a fundamental understanding of Razorpay integration.
4. Front-end Focus:   
   The project's scope predominantly revolves around the front-end implementation of the payment flow. It covers aspects such as designing the user interface, handling user interactions, and displaying transaction results. Back-end functionality, such as updating databases or integrating with other systems, may not be extensively covered.
5. Learning-oriented:   
   The project's primary aim is to facilitate learning and provide a tutorial experience. While it may not encompass every possible scenario or edge case, it focuses on delivering a clear understanding of the essential concepts and workflows involved in integrating Razorpay.

**Theoretical Background:**

1. Payment Gateway:   
   Explain the concept of a payment gateway and its role in facilitating online transactions. Elaborate on how Razorpay serves as a payment gateway, allowing businesses to securely accept payments from customers through various payment methods.
2. API Integration:   
   Describe the process of integrating the Razorpay API into your web application. Explain the steps involved, such as obtaining API keys, setting up webhooks for transaction notifications, and making API calls to initiate and verify payments.
3. Test Mode:   
   Highlight the significance of test mode in your project. Explain how test mode enables developers to simulate transactions without using real money, allowing them to thoroughly test the integration and ensure its functionality before going live.
4. Payment Flow:   
   Provide an overview of the payment flow implemented in your project. Explain how users interact with the donation button, how the transaction is initiated and processed, and how the response from Razorpay is handled.
5. Transaction Status:   
   Clarify the concept of transaction status and the different states a transaction can have, such as "created," "authorized," "captured," or "refunded." Describe how to retrieve and interpret the transaction status from the Razorpay API response.
6. Error Handling:   
   Discuss common errors and issues that users may encounter during the integration process. Provide guidance on how to handle and troubleshoot these errors, including appropriate error messages and suggestions for resolution.
7. Customization Options:   
   Describe any customization options available in the Razorpay integration. For example, explain how to customize the payment form, add additional fields, or implement advanced features like coupon codes or recurring payments.
8. Webhooks:   
   Introduce the concept of webhooks and their role in receiving real-time transaction notifications from Razorpay. Explain how to set up and handle webhooks in your application, ensuring that you receive and process transaction-related events effectively.
9. Documentation and Resources:   
   Provide links and references to official Razorpay documentation, API references, and other helpful resources. Encourage users to explore these resources for more in-depth information and guidance on specific integration scenarios.

**System Analysis and Design:**

**Requirements Analysis:**

1. Payment Integration:   
   The primary requirement is to integrate Razorpay as the payment gateway into the web application. This includes implementing a seamless payment flow that allows users to make donations securely.
2. User Interface:   
   The system should have an intuitive and user-friendly interface to facilitate easy navigation and donation processing. It should include a donation button, contact page, and an about us page to provide information about the charity.
3. Test Mode Functionality:   
   The system should operate in test mode, allowing developers to simulate transactions without using real money. This ensures a safe and controlled environment for learning and testing purposes.
4. Error Handling:   
   The system should handle errors gracefully and provide meaningful error messages to users in case of transaction failures or other issues.

**Design of the System:**

1. Frontend Design:   
   The frontend of the system will be developed using React, a popular JavaScript library for building user interfaces. It will consist of components such as the donation button, contact form, and pages for about us and contact information.
2. Backend Design:   
   The backend of the system will be built using Node.js, a runtime environment for server-side JavaScript execution. It will handle requests from the frontend, interact with the Razorpay API, and manage the payment flow.
3. Payment Integration:   
   The system will utilize the Razorpay API for payment integration. It will make API calls to initiate and verify transactions, handle webhooks for transaction notifications, and update the transaction status in the system.
4. Database Integration:   
   A database (such as MongoDB or MySQL) can be integrated into the backend to store transaction details, user information, and other relevant data if needed for future expansion (currently we used mongodb and capture a successful payment).
5. Methodology Adopted:   
   The methodology adopted for this project can be Agile or Iterative development. These methodologies emphasize frequent iterations, continuous feedback, and collaboration between the development team and stakeholders. It allows for incremental development and flexibility in adapting to changing requirements or enhancements during the project lifecycle.

**React Components used:**

* Home
* About
* Contact
* Card
* FlipCon
* Nav
* paymentSuccess
* TotalFund
* Footer

Home Component:

const Home = () => {

    const checkoutHandler = async (amount) => {

        const { data: { key } } = await axios.get("http://www.localhost:4000/api/getkey")

        const { data: { order } } = await axios.post("http://localhost:4000/api/checkout", {

            amount

        })

        const options = {

            key,

            amount: order.amount,

            currency: "INR",

            name: "A little change",

            description: "razorpay Payment Integration",

            image: "https://thumbs.dreamstime.com/b/charity-word-cloud-heart-concept-56405290.jpg",

            order\_id: order.id,

            callback\_url: "http://localhost:4000/api/paymentverification",

            notes: {

                "address": "Razorpay Corporate Office"

            },

            theme: {

                "color": "#9376E0"

            }

        };

        const razor = new window.Razorpay(options);

        razor.open();

    }

    return (

        <section id="main">

            <Nav/>

            <div>

                <div>

                    <h1>A Little Change</h1>

                    <p>Always give without <br /> remembering and always receive <br /> without forgetting</p>

                </div>

                <Card amount={100} img={"./charity.png"} checkoutHandler={checkoutHandler} />

            </div>

            <Footer/>

        </section>

    )

}

About Component

const About = () => {

  return (

    <section id='about'>

        <Nav/>

        <h1 id='head'>About</h1>

        <FlipCon image={"https://images.unsplash.com/photo-1488521787991-ed7bbaae773c?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxzZWFyY2h8NHx8Y2hhcml0eXxlbnwwfHwwfHx8MA%3D%3D&w=1000&q=80"} content={['A Little Change" charity is dedicated to making a positive impact on communities by focusing on small-scale projects that create lasting change. Their mission is to address social issues, improve lives, and foster sustainable development through targeted initiatives', 'The charity follows a grassroots approach, recognizing that even small actions can make a significant difference. They believe in the power of collective efforts and work closely with local communities, listening to their needs, and involving them in the decision-making process', '"A Little Change" charity undertakes a wide range of projects across various areas of social impact. Their initiatives may include providing access to education, healthcare, clean water, sanitation, empowering marginalized groups, supporting local economies, and promoting environmental sustainability']} flip={false}></FlipCon>

        <FlipCon image={"https://bsmedia.business-standard.com/\_media/bs/img/article/2022-04/07/full/1649270628-8781.jpg?im=FeatureCrop,width=826,height=465"} content={['Transparency is a core value of the charity. They strive to maintain open communication with donors, stakeholders, and the public, ensuring that every dollar donated is used effectively and efficiently. Regular reporting and updates are provided to showcase the impact of their projects', 'While the charity focuses on small-scale projects, they also adopt a scalable model. By meticulously planning and executing initiatives, they aim to create sustainable change that can be replicated in other communities, inspiring wider social transformation.']} flip={true}></FlipCon>

        <Footer/>

    </section>

  )

}

Contact Component

const Contact = () => {

  return (

    <section id="contact">

        <Nav/>

        <h1 id='head'>Contact</h1>

        <div>

            <button onClick={() => window.location = 'tel:9999999999'}>+91-7424970024</button>

            <button onClick={() => window.location = 'mailto:rohitsinghsignin@gmail.com'}>rohitsinghsignin@gmail.com</button>

        </div>

        <Footer/>

    </section>

  )

}

Card Component

const Card = ({ amount, img, checkoutHandler }) => {

    return (

        <section id='card'>

            <h3>Hi! <br /> Razorpay payment <br /> Integration</h3>

                <img src={img} alt='Hi' />

            <button onClick={() => checkoutHandler(amount)}>Donate</button>

        </section>

    )

}

FlipCon Component

const FlipCon = ({image, content, flip}) => {

  return (

    <div>

      {!flip && <img src={image} alt='charity gallery' />}

      {!flip && <div>

        {content.map((val) => {

          return (

            <p fontSize={"1.3rem"} marginBottom={"1rem"}>{val}</p>

            );

        })}

    </div>}

  {flip &&

  <div>

    {content.map((val) => {

      return (

        <p fontSize={"1.3rem"} marginBottom={"1rem"}>{val}</p>

      );

    })}

  </div>

  }

  {flip && <img src={image} alt="Charity gallery" />}

    </div>

  )

}

Nav Component

const Nav = () => {

  return (

    <nav>

        <Link className='navLink' to={"/"}><h1>Charity</h1></Link>

        <div>

            <Link className='navLinkHome' to={"/"}>Home</Link>

            <Link className='navLink' to={"/total-fund"}>Fund</Link>

            <Link className='navLink' to={"/about"}>About</Link>

            <Link className='navLink' to={"/contact"}>Contact</Link>

        </div>

    </nav>

  )

}

paymentSuccess Component

const PaymentSuccess = () => {

    const seachQuery = useSearchParams()[0]

    const referenceNum = seachQuery.get("reference")

    return (

        <section id='payment-success'>

            <Nav/>

            <div>

                <h1 textTransform={"uppercase"}> Order Successful</h1>

                <p>

                    Reference No.{referenceNum}

                </p>

            </div>

        </section>

    )

}

TotalFund Component

const TotalFund = () => {

    const [totalFund, setFund] = useState(0);

    useEffect( () => {

        axios.get("http://www.localhost:4000/api/total-fund")

        .then ((res) => setFund(res.data));

    }, []);

  return (

    <section id="total-fund">

        <Nav/>

        <h1 id="head">Total Fund</h1>

        <div>

            <h1>Total: {totalFund\*100} Rs</h1>

        </div>

        <Footer/>

    </section>

  )

}

Footer Component

const Footer = () => {

  return (

    <footer>&#169; All right reserved by Charity</footer>

  )

}

App Component

function App() {

  return (

    <Router>

      <Routes>

        <Route path="/" element={<Home />} />

        <Route path="/paymentsuccess" element={<PaymentSuccess />} />

        <Route path="/total-fund" element={<TotalFund />} />

        <Route path="/about" element={<About />} />

        <Route path="/contact" element={<Contact />} />

      </Routes>

    </Router>

  );

}

**Backend Server**

Package .json:

{

  "name": "server",

  "version": "1.0.0",

  "description": "",

  "type": "module",

  "main": "index.js",

  "scripts": {

    "dev": "nodemon server.js"

  },

  "author": "",

  "license": "ISC",

  "dependencies": {

    "cors": "^2.8.5",

    "dotenv": "^16.0.1",

    "express": "^4.18.1",

    "mongoose": "^6.4.2",

    "razorpay": "^2.8.2"

  }

}

Server.js

import { app } from "./app.js";

import Razorpay from "razorpay";

import { connectDB } from "./config/database.js";

connectDB();

export const instance = new Razorpay({

  key\_id: process.env.RAZORPAY\_API\_KEY,

  key\_secret: process.env.RAZORPAY\_APT\_SECRET,

});

app.listen(process.env.PORT, () =>

  console.log(`Server is working on ${process.env.PORT}`)

);

app.js

import express from "express";

import { config } from "dotenv";

import paymentRoute from "./routes/paymentRoutes.js";

import cors from "cors";

config({ path: "./config/config.env" });

export const app = express();

app.use(cors());

app.use(express.json());

app.use(express.urlencoded({ extended: true }));

app.use("/api", paymentRoute);

app.get("/api/getkey", (req, res) =>

  res.status(200).json({ key: process.env.RAZORPAY\_API\_KEY })

);

Routes > paymentRoutes.js

import express from "express";

import {

  checkout,

  paymentVerification,

  fetchPayment

} from "../controllers/paymentController.js";

const router = express.Router();

router.route("/checkout").post(checkout);

router.route("/paymentverification").post(paymentVerification);

router.route("/total-fund").get(fetchPayment);

export default router;

Controllers > paymentController.js

import { instance } from "../server.js";

import crypto from "crypto";

import { Payment } from "../models/paymentModel.js";

export const checkout = async (req, res) => {

  const options = {

    amount: Number(req.body.amount \* 100),

    currency: "INR",

  };

  const order = await instance.orders.create(options);

  res.status(200).json({

    success: true,

    order,

  });

};

export const paymentVerification = async (req, res) => {

  const { razorpay\_order\_id, razorpay\_payment\_id, razorpay\_signature } =

    req.body;

  const body = razorpay\_order\_id + "|" + razorpay\_payment\_id;

  const expectedSignature = crypto

    .createHmac("sha256", process.env.RAZORPAY\_APT\_SECRET)

    .update(body.toString())

    .digest("hex");

  const isAuthentic = expectedSignature === razorpay\_signature;

  if (isAuthentic) {

    // Database comes here

    await Payment.create({

      razorpay\_order\_id,

      razorpay\_payment\_id,

      razorpay\_signature,

    });

    res.redirect(

      `http://localhost:3000/paymentsuccess?reference=${razorpay\_payment\_id}`

    );

  } else {

    res.status(400).json({

      success: false,

    });

  }

};

export const fetchPayment = async(req, res) => {

  try{

      const tempData = await Payment.count();

      res.status(200).json(tempData);

  }catch(err){

      res.status(400).json({"messgae": err.message})

  }

};

Models > paymentModel.js

import mongoose from "mongoose";

const paymentSchema = new mongoose.Schema({

  razorpay\_order\_id: {

    type: String,

    required: true,

  },

  razorpay\_payment\_id: {

    type: String,

    required: true,

  },

  razorpay\_signature: {

    type: String,

    required: true,

  },

});

export const Payment = mongoose.model("Payment", paymentSchema);

config > config.env

PORT = 4000

RAZORPAY\_API\_KEY= "rzp\_test\_eWqc5ryDsG2s3z"

RAZORPAY\_APT\_SECRET="y6ltxUVRZnBkbzunMWSkin49"

MONGO\_URI="mongodb+srv://rohit:boom@raz-pay-intg.ikt7zgy.mongodb.net/?retryWrites=true&w=majority"

Config > database.js

import mongoose from "mongoose";

export const connectDB = async () => {

  const { connection } = await mongoose.connect(process.env.MONGO\_URI);

  console.log(`Mongodb is connected with ${connection.host}`);

};

**APIs** (localhost):  
[**http://localhost:4000/api/checkout**](http://localhost:4000/api/checkout)[**http://localhost:4000/api/paymentverification**](http://localhost:4000/api/paymentverification)[**http://localhost:4000/api/total-fund**](http://localhost:4000/api/total-fund)[**http://localhost:4000/api/getkey**](http://localhost:4000/api/getkey)

**CSS (Cascading Style Sheet):**

:root{

    --fontColor1: #2d2d2d;

    --fontColor2: #4a4a4a;

    --fontDonate: tomato;

    --hoverColor: #fff;

    --purplish: #9376E0;

    --pinkish: #E893CF;

    --lightpinkish: #F3BCC8;

    --yellowish: #F6FFA6;

    --whitist: #f8f6f6;

}

html{

    box-sizing: border-box;

}

\*, \*::after, \*::before{

    box-sizing: inherit;

    user-select: none;

}

body{

    margin: 0;

    padding: 0;

}

nav{

    display: flex;

    justify-content: space-between;

    align-items: center;

    align-content: center;

    border-bottom: 5px solid var(--pinkish);

    background-color: var(--whitist);

}

nav > .navLink{

    font-size: 2rem;

    color: var(--fontColor1);

    font-weight: 600;

    letter-spacing: 0.4rem;

    margin-left: 1rem;

}

nav div > .navLink{

    padding: 0 0.3rem;

    font-size: 1.3rem;

    color: var(--fontColor2);

    margin-right: 1rem;

    letter-spacing: 0.2rem;

    transition: border-Bottom 0.3s;

}

nav > .navLink::first-letter{

    color: var(--purplish);

}

nav div > .navLinkHome{

    font-size: 1.3rem;

    color: var(--purplish);

    margin-right: 1rem;

    padding: 0.3rem 0.5rem;

    border-radius: 0.4rem;

    letter-spacing: 0.2rem;

    transition: background-color 0.3s, color 0.3s;

}

/\* Nav Hover part  \*/

nav div > .navLink:hover{

    border-bottom: 6px solid var(--purplish);

    border-radius: 10px;

}

nav div > .navLinkHome:hover{

    background-color: var(--purplish);

    color: var(--hoverColor);

}

/\* -- Home --  \*/

#main{

    overflow: hidden;

    height: max-content;

    min-height: 100vh;

    background-color: var(--purplish);

}

#main > div{

    height: max-content;

    display: flex;

    justify-content: space-around;

    align-items: center;

    align-content: center;

    flex-wrap: wrap;

    gap: 1rem;

    margin: 4rem 1rem 1rem 1rem;

}

#main > div > div > h1{

    font-size: 5rem;

    letter-spacing: 0.4rem;

    font-weight: 600;

    color: var(--whitist);

}

#main > div > div > p{

    font-size: 1.4rem;

    letter-spacing: 0.2rem;

}

#main #card h3{

    width: 100%;

    text-align: center;

    font-size: 1.4rem;

    letter-spacing: 0.4rem;

    font-weight: 600;

    color: var(--whitist);

    margin-bottom: 2rem;

}

#main #card button{

    width: 100%;

    margin: 2rem auto;

    font-size: 1.4rem;

    background-color: var(--whitist);

    color: var(--purplish);

    padding: 0.3rem 0.5rem;

    border-radius: 0.4rem;

    letter-spacing: 0.2rem;

    font-weight: 600;

    transition: background-color 0.3s, color 0.3s;

}

#main #card img{

    animation: hello 1s ease-in-out infinite;

}

/\* Home Hover Part  \*/

#main #card button:hover{

    background-color: var(--pinkish);

    color: var(--whitist);

}

@keyframes hello {

    0%{

        rotate: 15deg;

    }

    65%{

        rotate: -15deg;

    }

    100%{

        rotate: 15deg;

    }

}

/\* -- About --   \*/

#about > div > img{

    width: 400px;

    height: 400px;

    border-radius: 1rem;

    margin: 1rem;

    box-shadow: 9px 9px 5px -5px rgba(0,0,0,0.75);

}

#about > div{

    padding: 1rem;

    display: flex;

    justify-content: center;

    align-items: center;

}

#about > div p{

    font-size: 1.4rem;

    margin: 1rem;

    color: var(--fontColor2);

}

/\* -- Contact --  \*/

#contact{

    height: 100vh;

}

#contact > div {

    display: flex;

    justify-content: space-around;

    align-items: center;

    align-content: center;

}

#contact > div button{

    margin-top: 6rem;

    font-size: 2rem;

    padding: 0.8rem 1rem;

    color: var(--whitist);

    background-color: var(--purplish);

    border-radius: 0.5rem;

    font-weight: 100;

}

footer{

    width: 100%;

    position: absolute;

    bottom: 0;

    text-align: center;

    background-color: var(--purplish);

    color: var(--whitist);

}

/\* -- Payment Successful --  \*/

#payment-success{

    width: 100%;

    height: 100vh;

}

#payment-success > div{

    height: 80vh;

    display: flex;

    flex-direction: column;

    justify-content: center;

    align-items: center;

}

#payment-success > div h1{

    font-size: 4rem;

    font-weight: 600;

}

#payment-success > div h1::first-letter{

    color: #9376E0;

}

/\* -- fund page --  \*/

#total-fund{

    width: 100%;

    height: 100vh;

    overflow: hidden;

}

#total-fund > div{

    height: 100vh;

    display: flex;

    justify-content: center;

    align-items: center;

    align-content: center;

}

#total-fund > div > h1{

    margin-top: -12rem;

    background-color: var(--purplish);

    font-size: 3rem;

    color: var(--whitist);

    padding: 1rem 2.5rem;

    border-radius: 1rem;

    box-shadow: 10px 9px 11px -8px rgba(0,0,0,0.45);

    transition: transform 1s;

    cursor: pointer;

    letter-spacing: 0.2rem;

}

#total-fund > div > h1::first-letter{

    color: var(--pinkish);

}

#total-fund > div > h1:hover{

    transform: scale(1.1);

}

/\* -- Common -- \*/

#head{

    width: 100%;

    text-align: center;

    font-size: 2rem;

    padding: 2rem;

    background: rgb(147,118,224);

    background: linear-gradient(133deg, rgba(147,118,224,1) 31%, rgba(9,100,121,1) 100%, rgba(255,255,255,1) 100%);

    color: var(--whitist);

}

/\* -- Media Queries -- \*/

@media only screen and (max-width: 1087px){

    /\* -- Home Page --  \*/

    #main > div{

        flex-direction: column-reverse;

    }

    #main > div > div{

        text-align: center;

        margin-bottom: 2rem;

    }

    #main > div > div > h1,p{

        text-align: center;

    }

    /\* -- About Page --  \*/

    #about > div{

        flex-wrap: wrap;

    }

    /\* -- Contact Page -- \*/

    #contact > div{

        flex-wrap: wrap;

    }

}

**Hardware and Software Details:**

1. Hardware:   
   The system can be deployed on standard hardware infrastructure, including a web server to host the frontend and backend components, a database server if required, and sufficient computing resources to handle the expected user load.
2. Software:   
   The software components include:
3. Frontend:   
   React, HTML, CSS, JavaScript
4. Backend:   
   Node.js, Express.js (framework for building web applications), Razorpay API
5. Database:   
   MongoDB, MySQL, or any other suitable database management system (In this we use mongodb and capture a successful payment)

**ER Diagram and DFD**

**ER Diagram:**The ER diagram represents the database structure for the "Payment" entity in your system. The "Payment" entity stores information related to payments made through Razorpay integration. Let's explore the key components of the ER diagram:

**Entity:**

* "Payment": This entity represents the main table or collection that stores payment-related information.

**Attributes:**

* "\_id": The "\_id" attribute serves as the unique identifier for each payment entry. It ensures that each payment record is uniquely identifiable.
* "razorpay\_order\_id": This attribute stores the unique order ID generated by Razorpay for each payment transaction. It helps in associating the payment with the corresponding order.
* "razorpay\_payment\_id": The "razorpay\_payment\_id" attribute stores the unique payment ID generated by Razorpay for each payment transaction. It acts as a reference to identify and retrieve specific payment details.
* "razorpay\_signature": The "razorpay\_signature" attribute stores the signature associated with the payment. The signature is a security measure used to verify the authenticity and integrity of the payment data.
* "\_\_v": The "\_\_v" attribute represents the version number or version control information. It can be used for tracking and managing changes to the payment records.
* This ER diagram provides a visual representation of the database structure for managing payment-related data in your system. It showcases the attributes that are crucial for capturing and storing information related to payments made through Razorpay integration.

+----------------------------------+

| Payment |

+----------------------------------+

| \_id |

| razorpay\_order\_id |

| razorpay\_payment\_id |

| razorpay\_signature |

| \_\_v |

+----------------------------------+

Payment

* By referring to this ER diagram, you can better understand the structure of "Payment" entity and the attributes associated with it, aiding in the management and retrieval of payment-related data in system.

**Data Flow Diagram:**

* Donor: The Donor represents the user who interacts with the website.
* Website: The Website serves as the user interface where the donor can initiate the payment by clicking the donation button.
* Frontend: The Frontend component, implemented in React, handles the user interface and captures the donor's inputs, such as the donation amount and other necessary information.
* Backend: The Backend component, implemented in Node.js, serves as the intermediary between the frontend and other components. It receives the donation details from the frontend and performs the necessary processing.
* Razorpay: The Razorpay integration component is responsible for interacting with the Razorpay API to initiate and process the payment transaction. It communicates with the backend to receive the necessary data for payment processing.
* Database: The Database component stores the payment-related information received from the backend, such as payment IDs, order IDs, signatures, and other relevant data, for future reference or retrieval.
* The data flow in this system starts with the donor interacting with the website's frontend. The frontend captures the donor's inputs and sends them to the backend for further processing. The backend communicates with the Razorpay integration component to handle the payment transaction, and the necessary data is passed back and forth between the backend and the Razorpay integration. Finally, the relevant payment details are stored in the database for record-keeping and retrieval.

DONOR

FRONTEND

BACKEND

WEBSITE

RAZORPAY

DATABASE

**Database Schema:**

**Payment Collection Schema:**

const paymentSchema = new mongoose.Schema({

  razorpay\_order\_id: {

    type: String,

    required: true,

  },

  razorpay\_payment\_id: {

    type: String,

    required: true,

  },

  razorpay\_signature: {

    type: String,

    required: true,

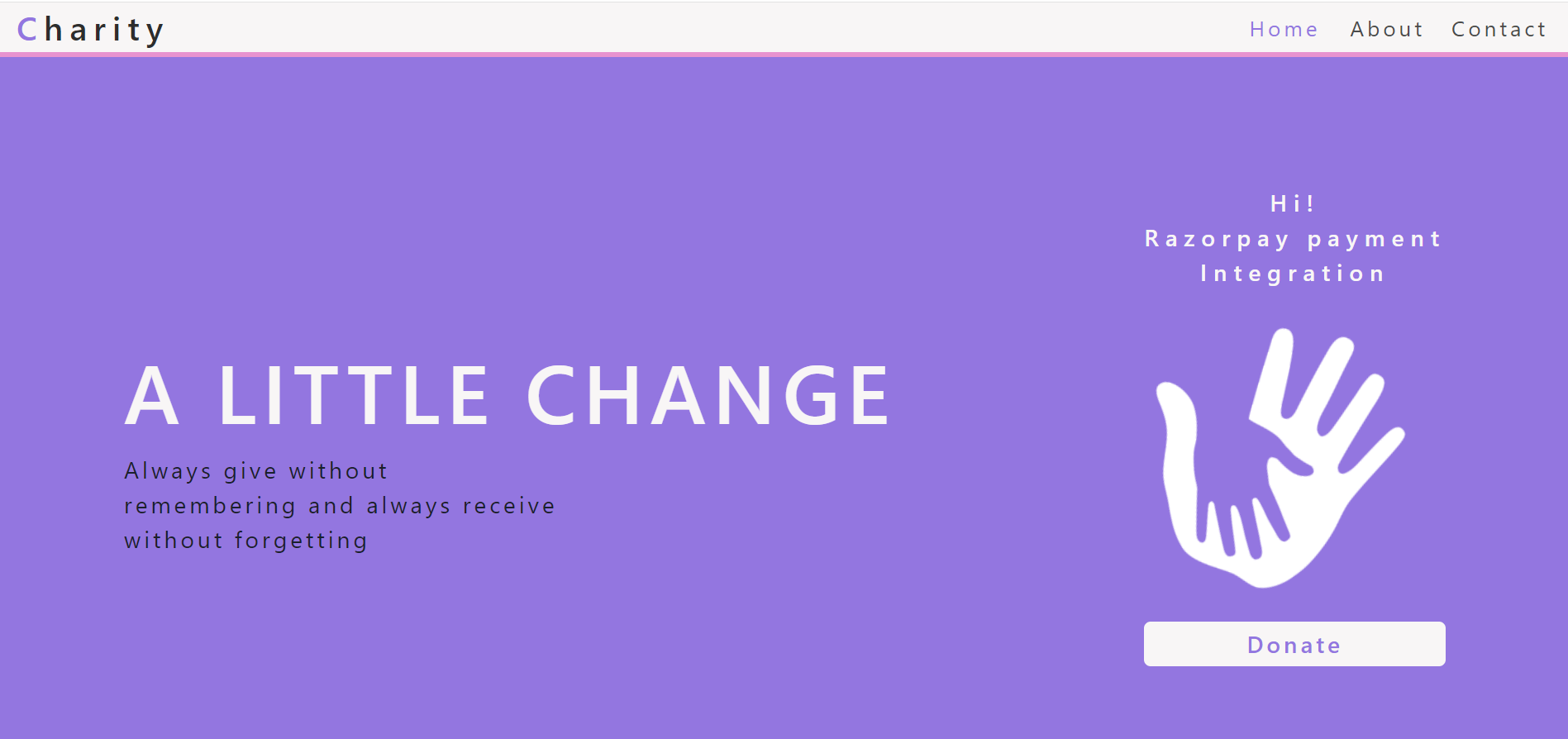
  },

});



1. Database Schema Overview:   
   The screenshot provides an overview of the database schema used in the payment integration project. It showcases the structure and organization of the database, including the tables, fields, and relationships.
2. Payment Table:   
   The screenshot displays the structure of the "Payment" table. It includes fields such as "\_id," "razorpay\_order\_id," "razorpay\_payment\_id," "razorpay\_signature," and "\_\_v." These fields store the relevant information associated with each payment transaction.
3. Field Descriptions:   
   This screenshot presents a detailed view of the fields within the "Payment" table. It provides descriptions for each field, explaining their purpose and usage in the system. This helps in understanding the data stored in the table and its significance in relation to payment processing.
4. Sample Data:   
   The screenshot showcases a set of sample data entries in the "Payment" table. It demonstrates how the fields are populated with actual values, including unique identifiers, Razorpay order IDs, payment IDs, signatures, and version numbers. This sample data helps visualize the structure and content of the database in action.

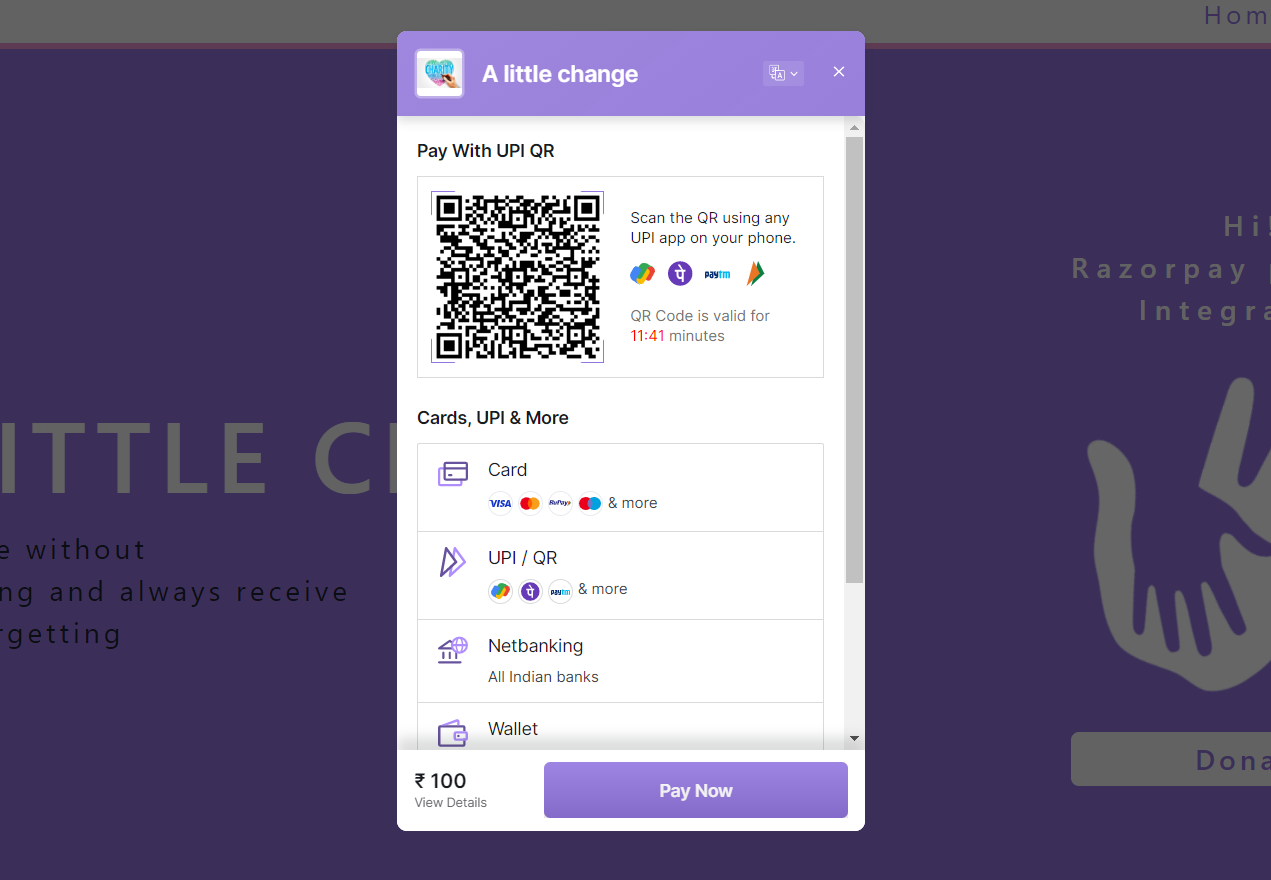
**Screenshots of Input/Output Interfaces/Forms/Web Pages:**



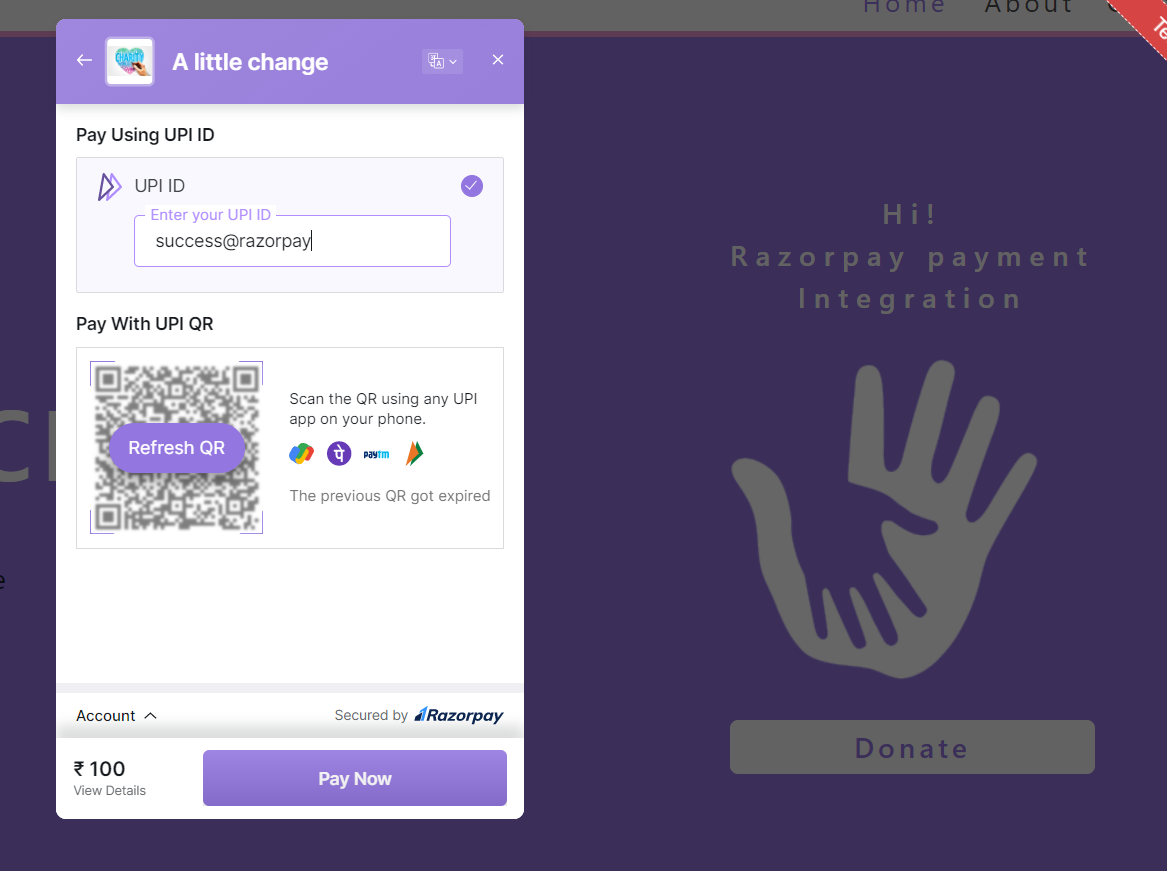
Navigation -> Home, About, Contact

Home page have Razorpay Integration

Click on Donate to call Razorpay script and to start payment process



Here you find all payment options  
Choose any one  
for e.g. we choose UPI / QR

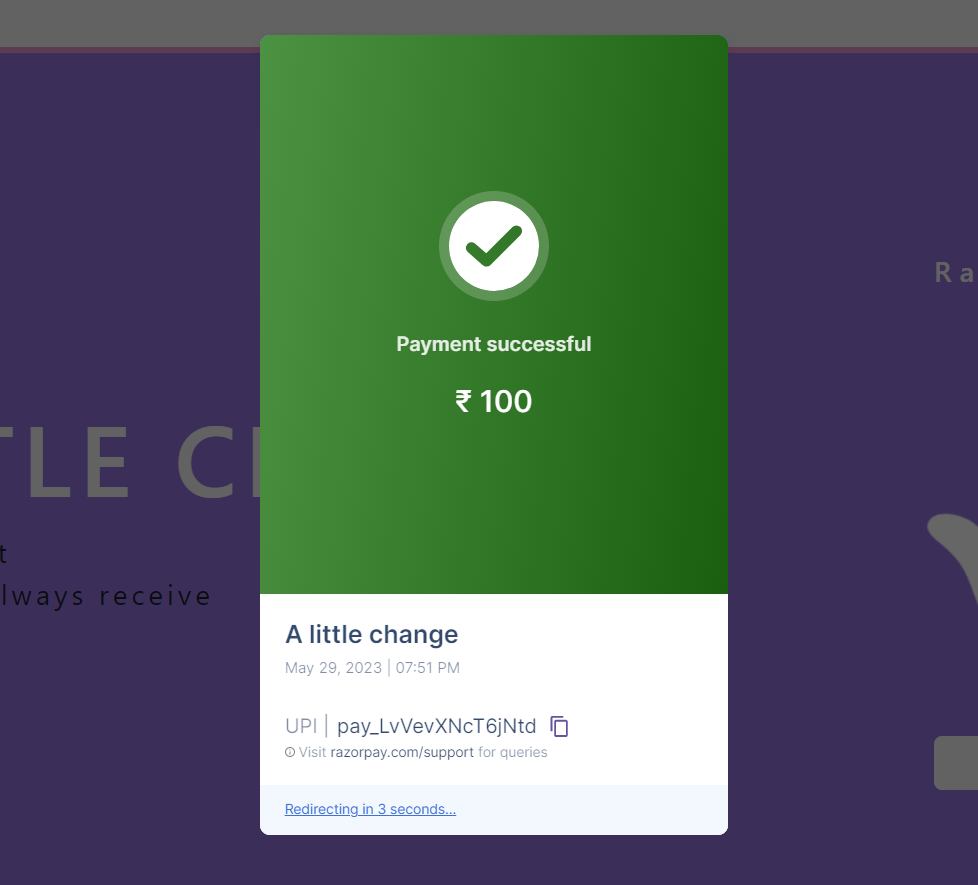


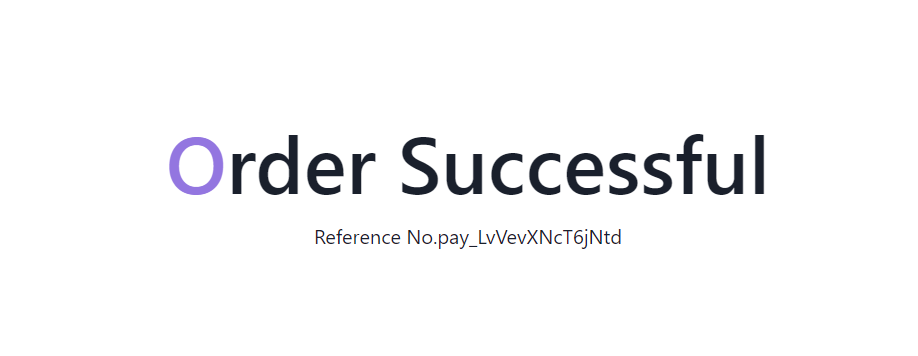
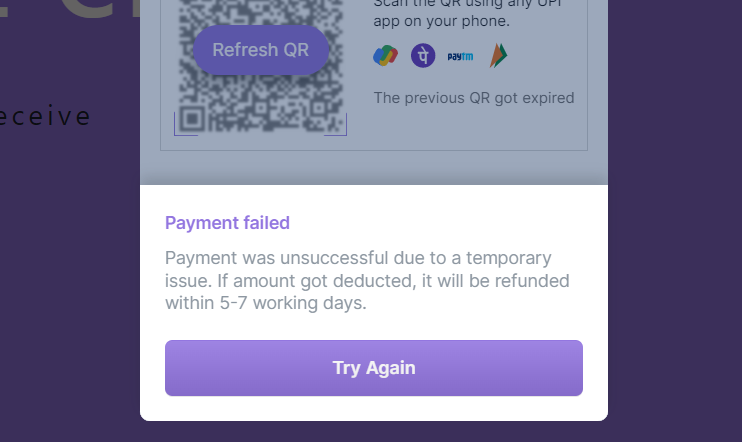
As we’re know that we using Test mode show enter ‘success@razorpay’

Click on Pay Now

(If you want to test payment failure that is ‘failure@razorpay’)

(The amount is fixed 100Rs)



**** ****

**On Payment Failure**

**On Payment Success:** Redirect to payment successful Page

**List of references, sources, and research materials:**

1. Razorpay Documentation:   
   The official documentation provided by Razorpay (<https://razorpay.com/docs/>) was a valuable resource throughout the project. It provided detailed information on integrating Razorpay into web applications, handling payment transactions, and verifying signatures.
2. React Documentation:  
   The React documentation (<https://reactjs.org/docs/>) was a fundamental reference for understanding and implementing the frontend component of the project. It provided comprehensive information on React concepts, components, and best practices.
3. Node.js Documentation:  
   The Node.js documentation (<https://nodejs.org/en/docs/>) served as a valuable resource for developing the backend component of the project. It provided guidance on setting up a Node.js server, handling HTTP requests, and integrating external APIs.
4. Express.js Documentation:  
   The Express.js documentation (<https://expressjs.com/>) was extensively used for building the backend server using the Express.js framework. It provided in-depth explanations of Express.js concepts, routing, middleware, and server-side request handling.
5. MongoDB Documentation:  
   The MongoDB documentation (<https://docs.mongodb.com/>) was referenced for understanding database concepts and implementing database operations. It provided information on schema design, querying data, and managing the MongoDB database.
6. Online Tutorials and Blogs:  
   Various online tutorials and blog posts were referenced during the project for specific implementation details and troubleshooting. Some notable sources include:

Medium (<https://medium.com/>)

Stack Overflow (<https://stackoverflow.com/>)

END