

DHA SUFFA UNIVERSITY

Department of Computer Science



OpenHands: Full-Stack Fundraising Web Application

Submitted By (Team Members):

Name	Roll No
Muhammad Maaz Khan	Se-221053
Shayan Adnan Hasan	Se-221060
Mahnoor Arshad	Se-221027
Farrukh Iqbal	Se-221005
Syed Minhal Ali	Se-221018

Submission Date: 24 june 2025

Introduction

Overview

OpenHands is a web-based platform that allows individuals to create and manage fundraising campaigns online. The platform is designed to help users raise money for medical needs, education, community causes, or personal emergencies. It offers a user-friendly interface where users can share their stories and accept donations securely via Stripe. Admins review and approve fundraisers, ensuring legitimacy and trust. A special feature allows donors to receive SMS confirmation after donating.

Problem Statement

In many developing regions, people face difficulties raising emergency funds for health or personal crises. Traditional fundraising is time-consuming, untrustworthy, and lacks reach. There's a need for a secure, simple, and reliable platform where genuine users can request help and generous individuals can contribute directly.

Objectives

- To provide an easy-to-use platform for creating and managing fundraising campaigns.
- To ensure each campaign is reviewed and verified by an admin.
- To allow donors to contribute securely using online payment (Stripe).
- To enhance transparency and trust through SMS confirmations using Twilio.

Scope

This project focuses on creating a secure full-stack web application that allows:

- Users to register, log in, and submit fundraiser requests.
- Admins to review, approve, or reject fundraiser submissions.

- Users and guests to view verified fundraisers and donate online.
- Donors to receive confirmation via SMS after successful payment.

This version supports only verified users on Twilio trial for SMS and uses a test environment in Stripe.

Technologies Used

- **Backend:** Node.js, Express.js
- **Frontend:** EJS (Embedded JavaScript Templates), HTML, CSS
- **Database:** MySQL with Prisma ORM
- **Payments:** Stripe API (Test Mode)
- **SMS Notifications:** Twilio (Trial Mode)

SRS – Software Requirements Specification

The Software Requirements Specification outlines what the system will do and how it will behave. This is divided into two parts: Functional and Non-Functional requirements.

a. Functional Requirements (What the system should do)

User Registration/Login - Users can sign up with their details: name, email, username, password, city, and country. - Passwords are securely hashed using bcrypt. - After successful login, a session is maintained using cookies and JWT.

Submit Fundraiser - Logged-in users can fill out a form to create a new fundraiser request. - Form accepts: Fundraiser title, description, goal amount, PNG/JPEG image, PDF for verification. - Submitted requests are saved in a PendingRequests table.

Admin Approval Panel - Admin logs in from a separate /adminLogin page. - Admin views all pending fundraiser requests in a dashboard. - Admin can: - Accept → moves request to Fundraisers table and marks as “Verified”. - Reject → deletes or marks the request as rejected.

Donation via Stripe - Users can view verified fundraisers and click “Donate”. - A donation form appears asking for: Donor name, amount, message (optional), phone number. - Stripe Checkout is launched for secure card payment in test mode.

SMS Confirmation via Twilio - After successful payment, a confirmation SMS is sent to the donor's phone using Twilio API. - Message example: "Thank you Maaz Khan for your generous donation to OpenHands!"

View Approved Fundraisers - Home page displays all fundraisers that are marked as "Verified".
- Each card shows image, title, goal, and verification badge.

b. Non-Functional Requirements (How the system should perform)

Security - JWT tokens protect user sessions. - Sensitive data (like passwords) are hashed.
- Only authenticated users can submit fundraisers. - Only admins can approve or reject fundraisers.

Availability - The system is designed to be always available locally. - Runs on localhost with minimal setup using npm run dev.

Performance - Optimized SQL queries via Prisma ORM. - Images and documents are stored locally for fast access. - Stripe and Twilio APIs integrated with async calls.

Scalability - Clean MVC folder structure allows future upgrades. - Can scale to production by switching from local DB to hosted MySQL.

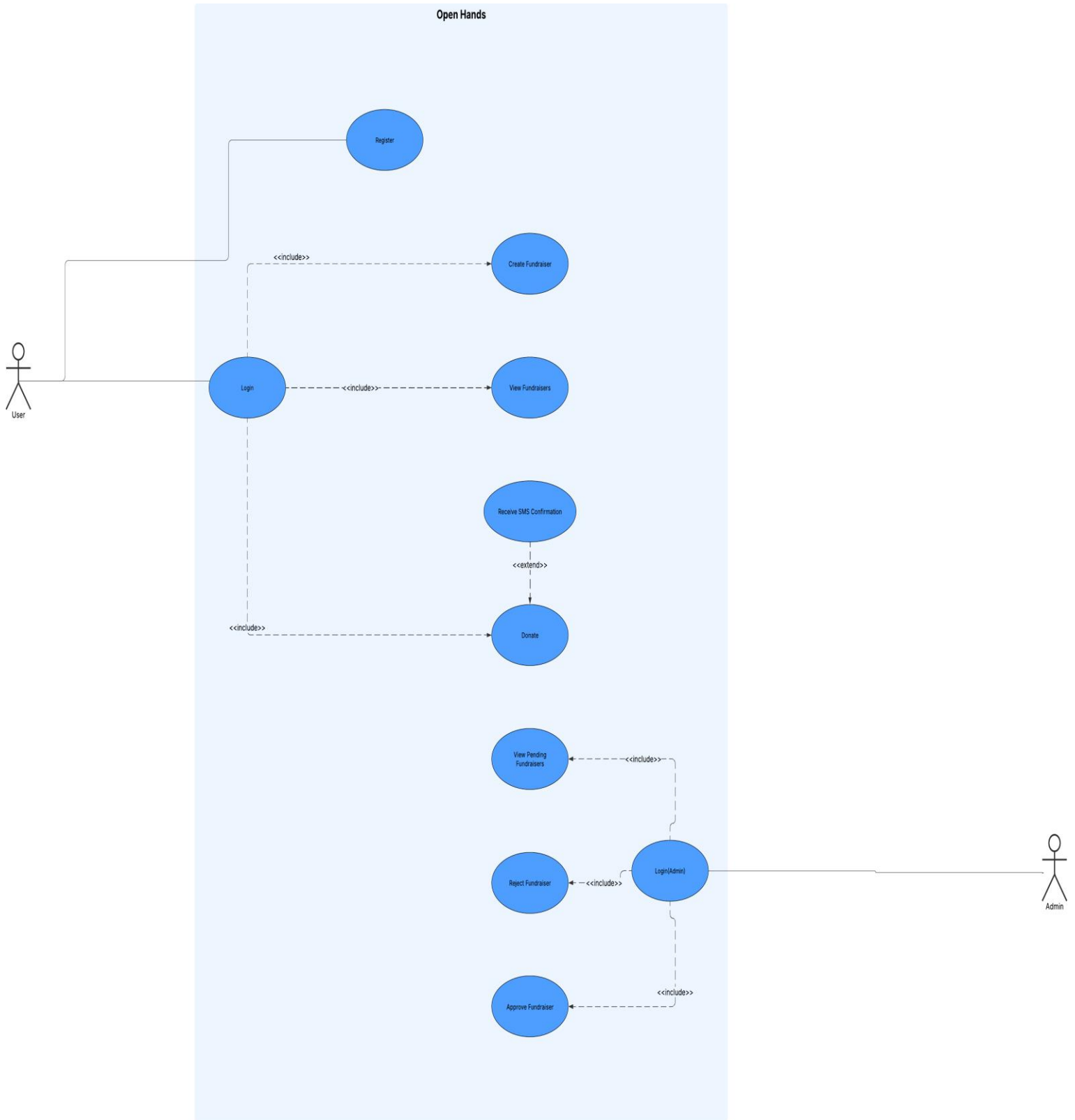
Maintainability - Code is modularized into routes, services, controllers, and models. - Prisma schema makes database changes easy to manage.

Usability - Responsive UI using EJS templates and Bootstrap. - User-friendly forms and navigation. - Clear error/success messages throughout the flow.

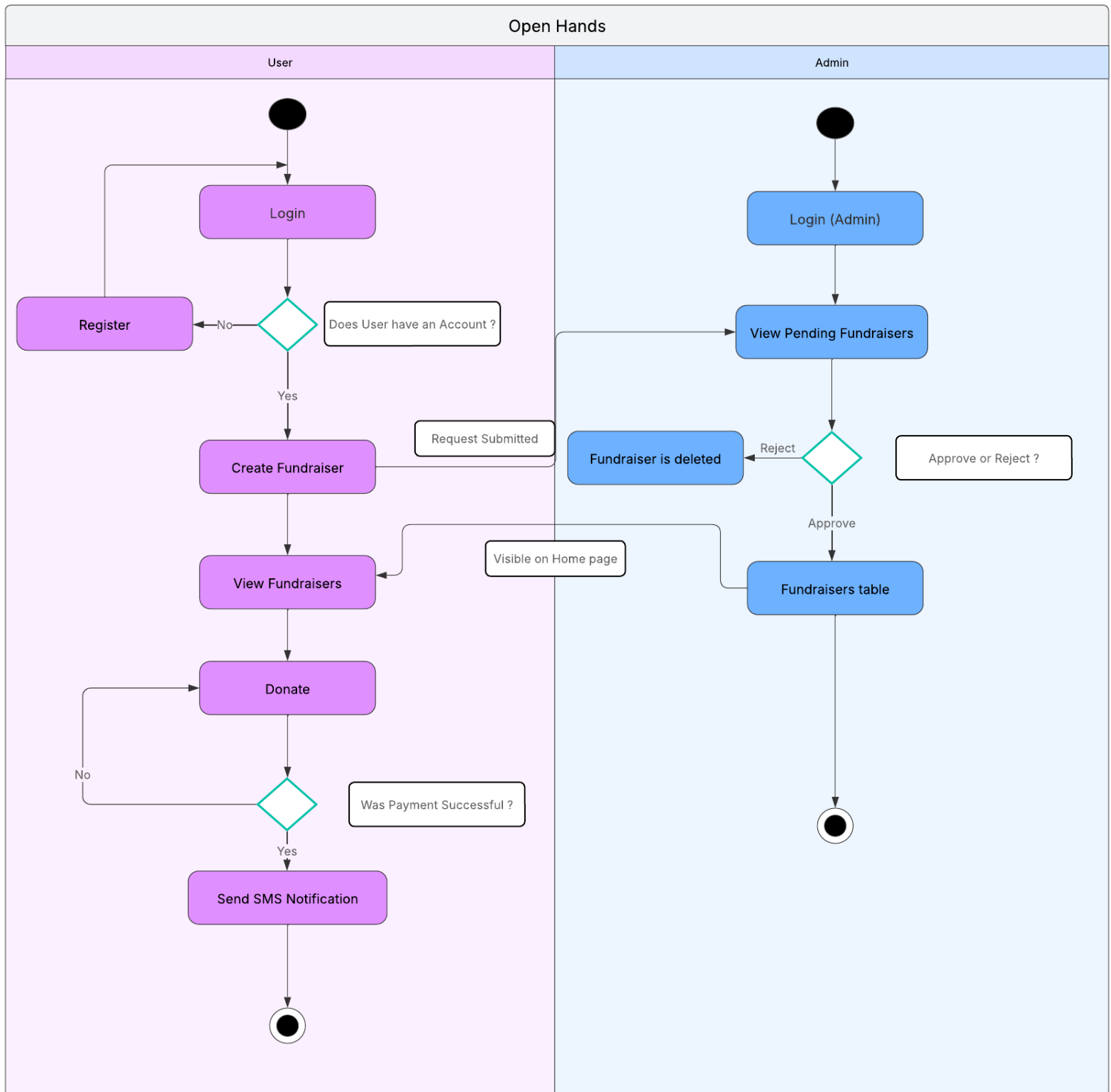
3. DDS – Design Documents

This section includes design visuals that describe the structure and flow of the OpenHands platform.

Use Case Diagram

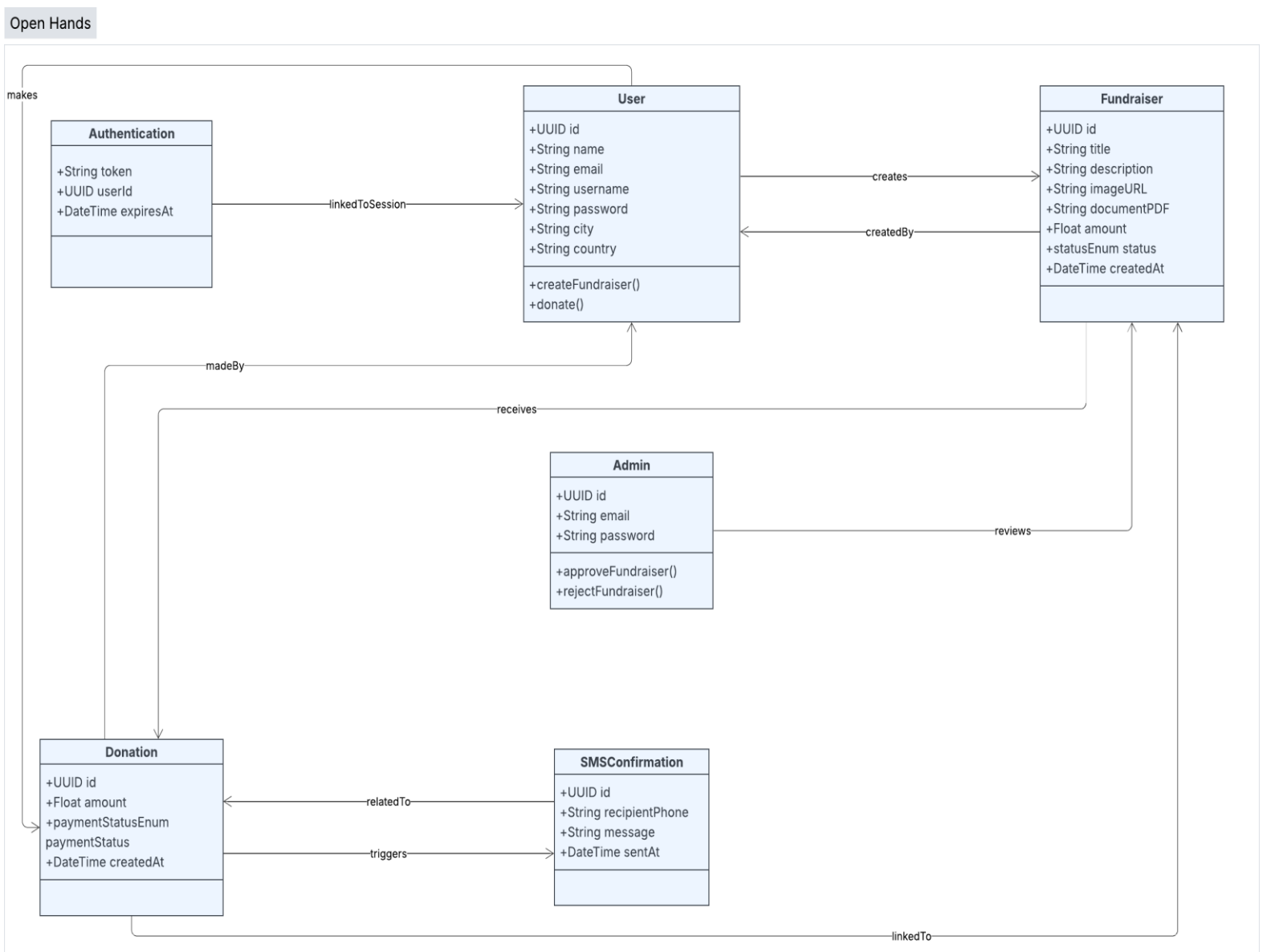


Activity Diagram



Class Diagram

(Insert Class Diagram here – showing models like User, Fundraiser, Donation)



4. Appendix

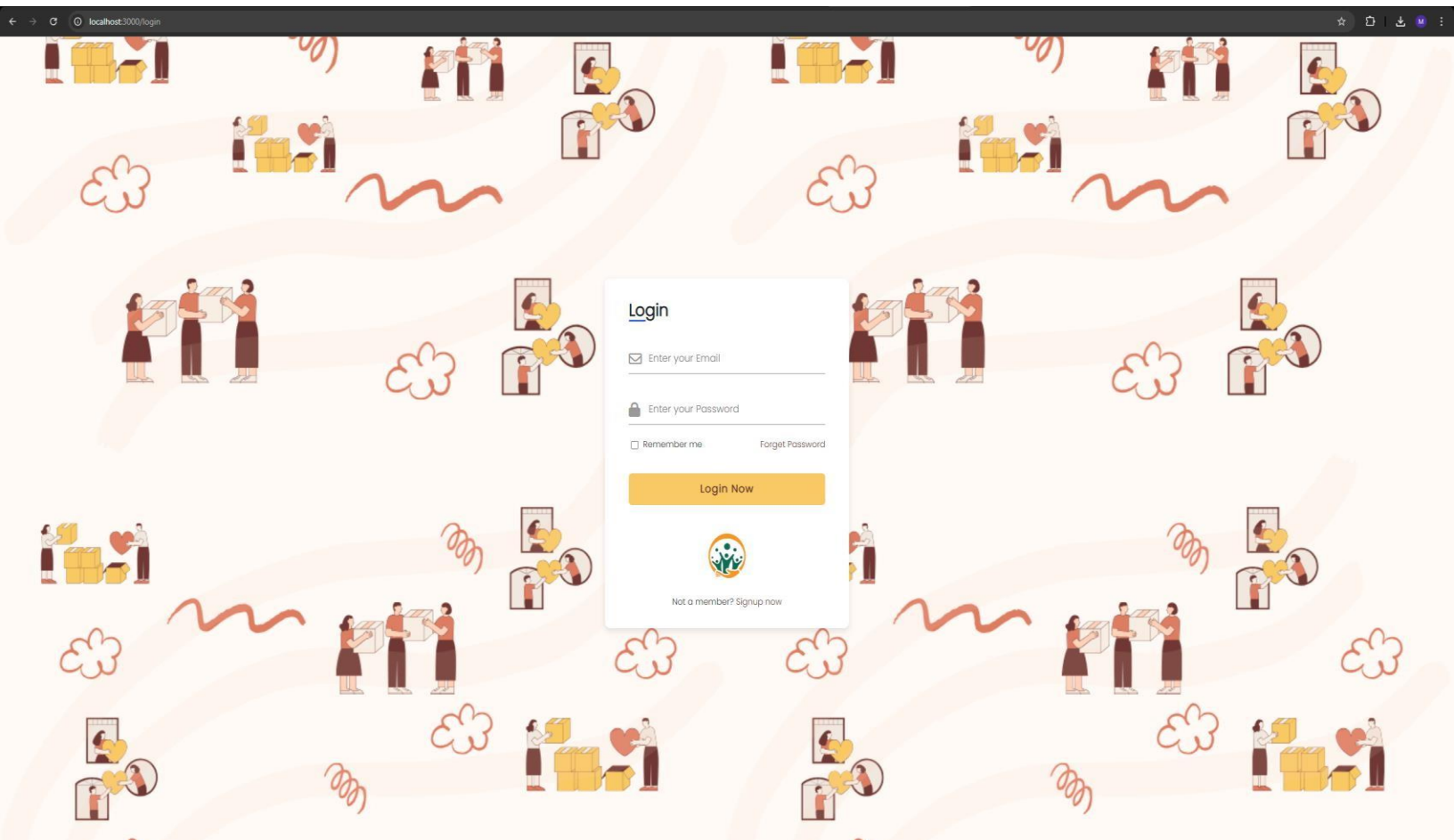
Appendix A – Abbreviations

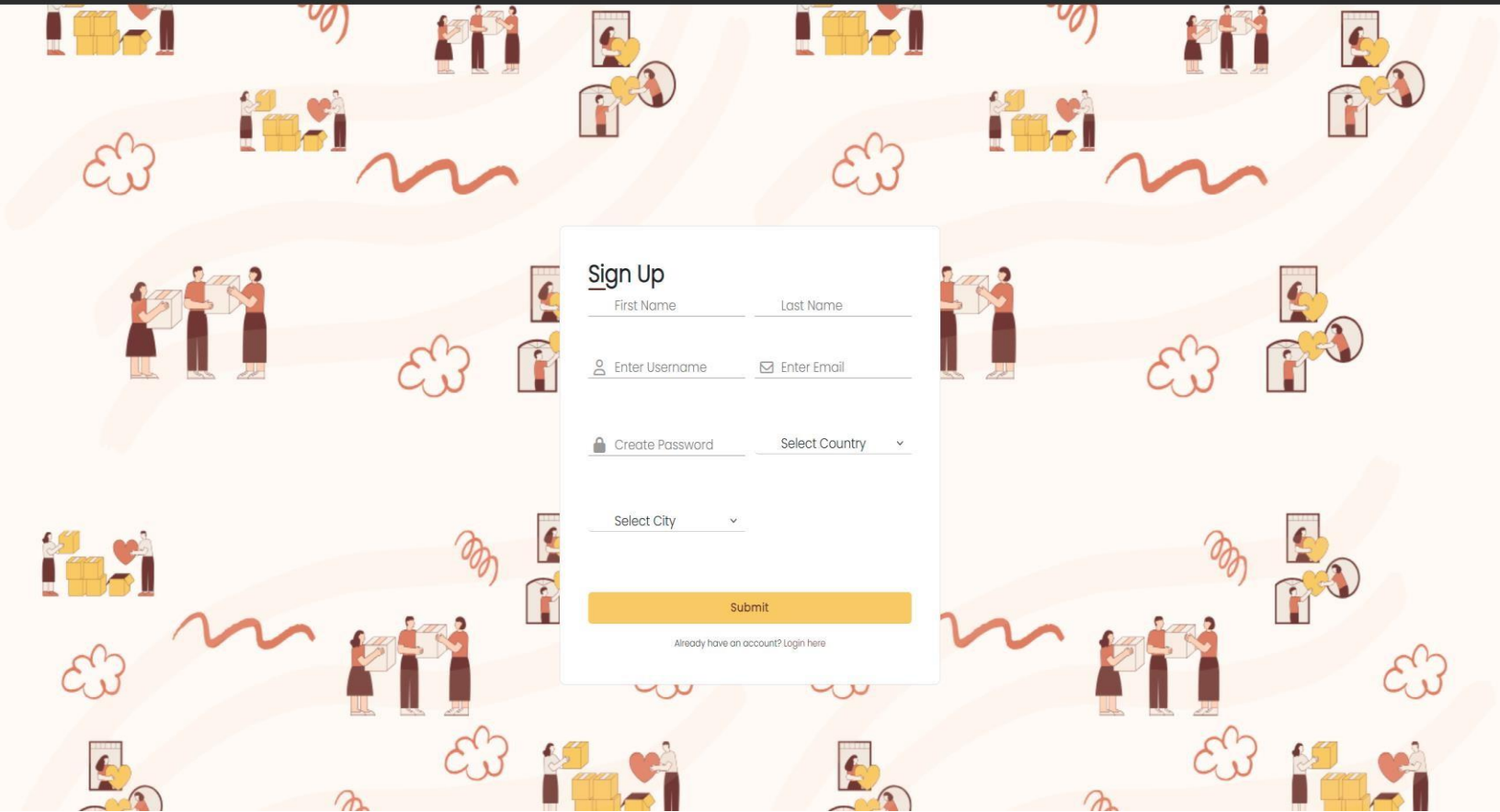
Abbreviation	Meaning
JWT	JSON Web Token
SRS	Software Requirements Specification
DDS	Design Description Specification
ORM	Object Relational Mapping
UI	User Interface
SMS	Short Message Service
API	Application Programming Interface
etc.	As required

Appendix B – Screenshots

1. **Login/Register Screen**

Description: Interface for users to sign up or sign in.





Sign Up

First Name

Last Name



Enter Username



Enter Email



Create Password

Select Country



Select City



Submit

Already have an account? [Login here](#)

2. Fundraiser Form

Description: Users submit fundraiser details, image, and PDF document.

localhost:3000/createFundraiser

OpenHands Home About Contact

Create a Fundraiser

First & Last Name:

First name Last name

Email address:

name@example.com

Title of Fundraiser:

Provide details of your situation:

Describe your fundraiser

Attach image (PNG/JPEG):

Choose File No file chosen

Attach document for situation verification (PDF):

Choose File No file chosen

Add amount needed:

PKR 0.00

Send Fundraiser Request

United States - English

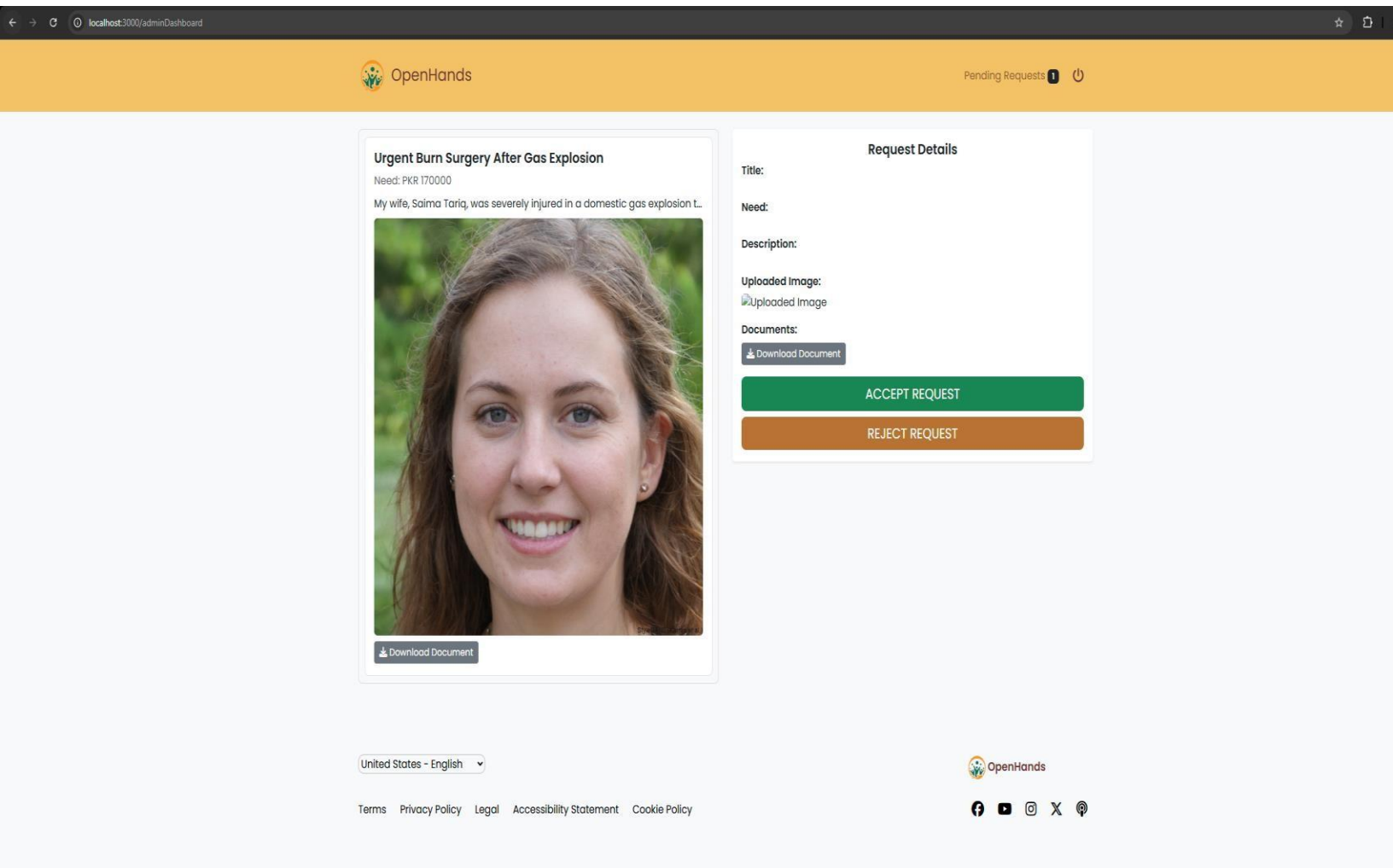
OpenHands

Terms Privacy Policy Legal Accessibility Statement Cookie Policy

f y i x p

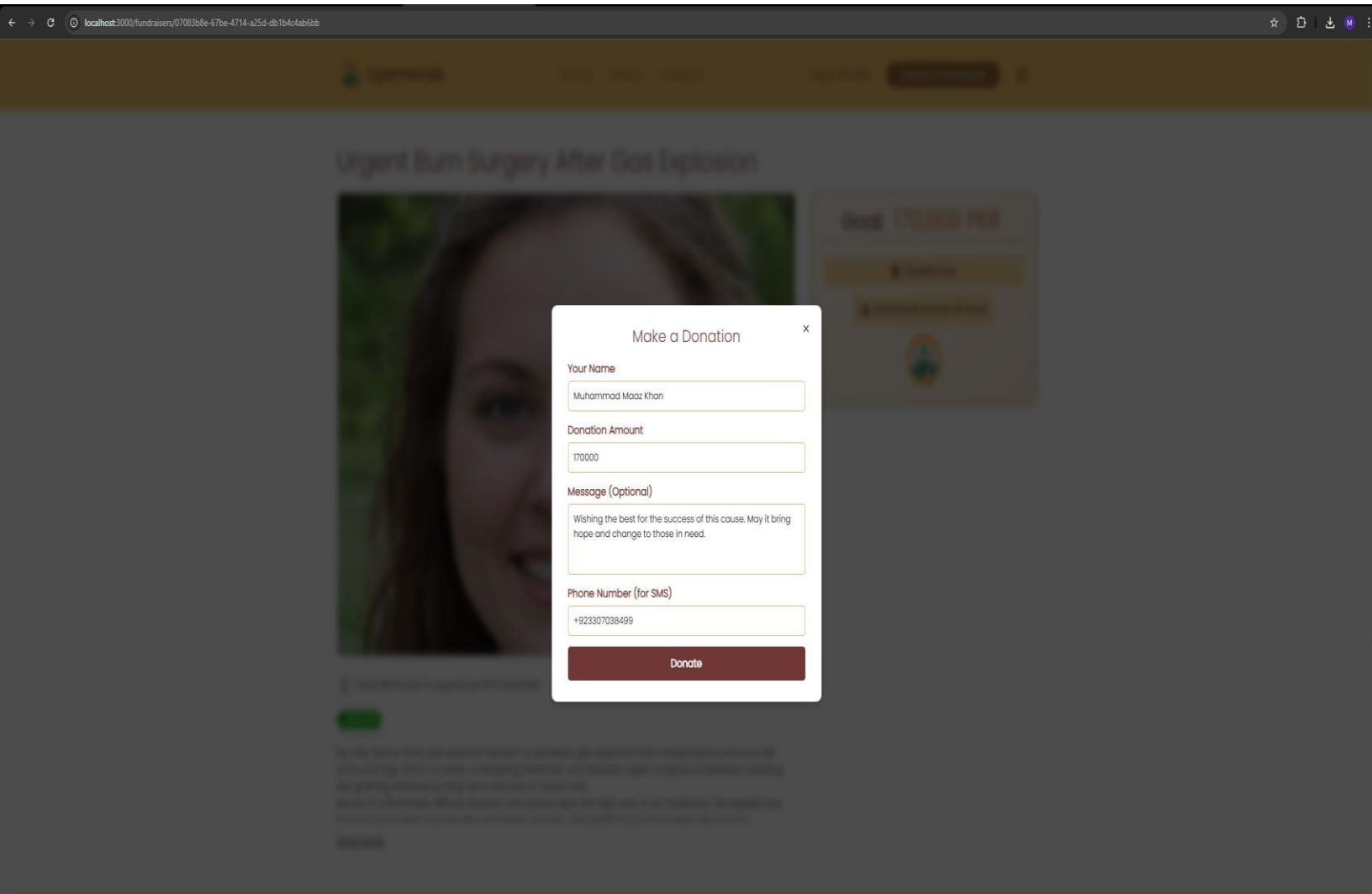
3. Admin Dashboard

Description: Admin reviews fundraiser submissions and approves/rejects.



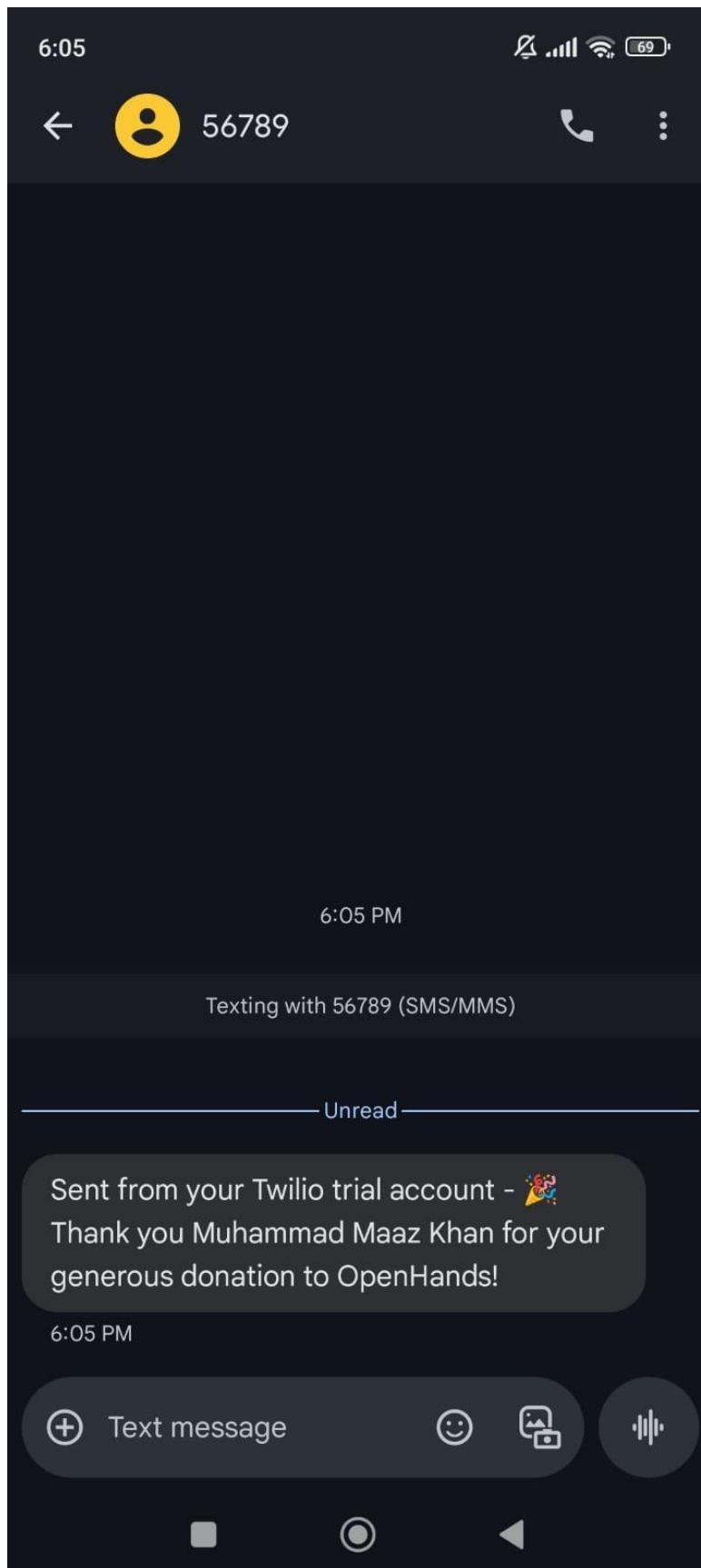
4. **Donation Modal**

Description: Collects donor details for Stripe Checkout.



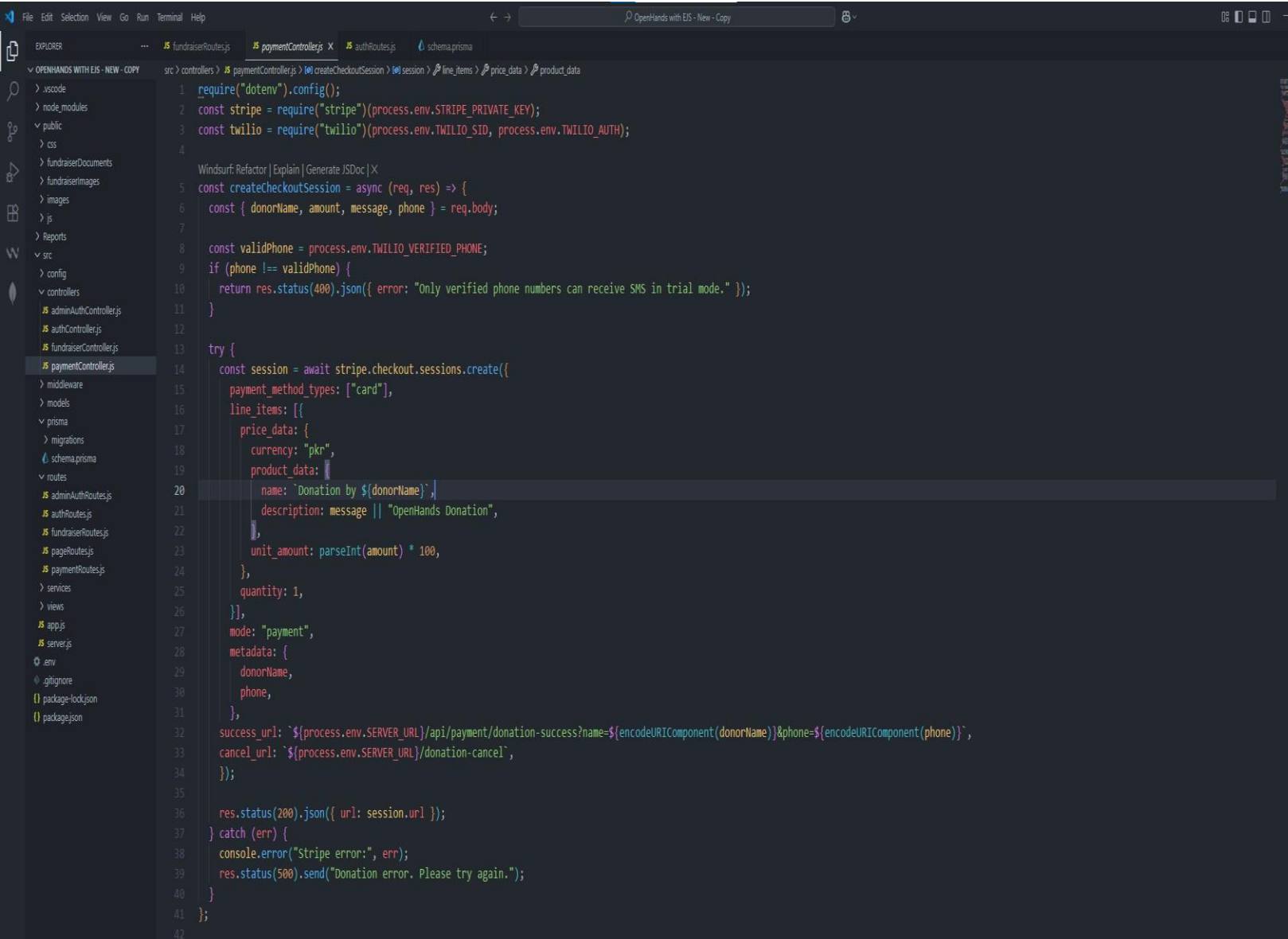
5. Twilio Confirmation SMS

Description: Message received after successful donation.



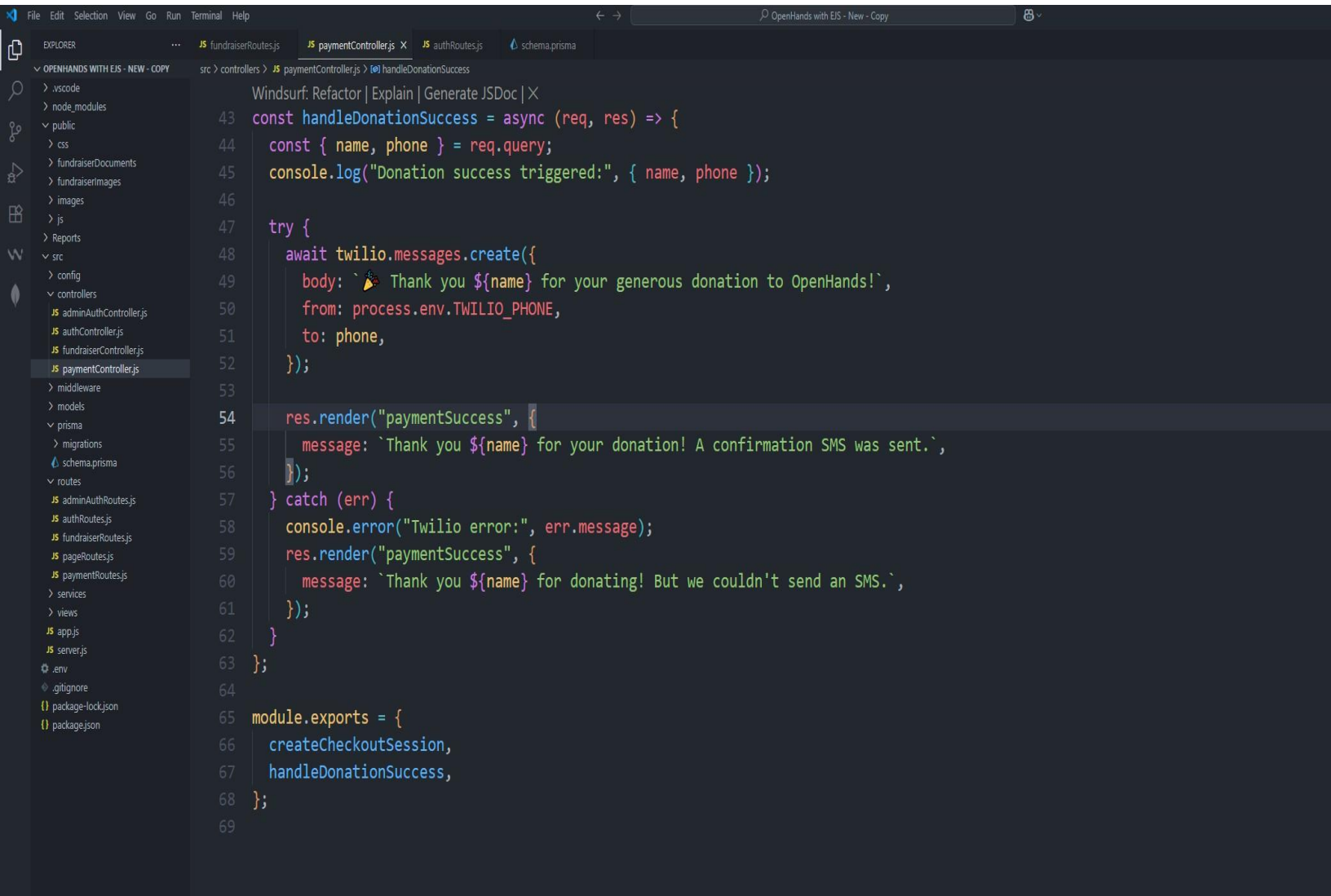
4. Appendix C – Code Snippets

1. Stripe Payment Code (Handles donation checkout via Stripe API)



```
1  require("dotenv").config();
2  const stripe = require("stripe")(process.env.STRIPE_PRIVATE_KEY);
3  const twilio = require("twilio")(process.env.TWILIO_SID, process.env.TWILIO_AUTH);
4
5  Windsurf: Refactor | Explain | Generate JSDoc | X
6  const createCheckoutSession = async (req, res) => {
7    const { donorName, amount, message, phone } = req.body;
8
9    const validPhone = process.env.TWILIO_VERIFIED_PHONE;
10   if (phone !== validPhone) {
11     return res.status(400).json({ error: "Only verified phone numbers can receive SMS in trial mode." });
12   }
13
14   try {
15     const session = await stripe.checkout.sessions.create({
16       payment_method_types: ["card"],
17       line_items: [
18         {
19           price_data: {
20             currency: "pkr",
21             product_data: {
22               name: `Donation by ${donorName}`,
23               description: message || "OpenHands Donation",
24             },
25             unit_amount: parseInt(amount) * 100,
26           },
27           quantity: 1,
28         },
29       ],
30       mode: "payment",
31       metadata: {
32         donorName,
33         phone,
34       },
35       success_url: `${process.env.SERVER_URL}/api/payment/donation-success?name=${encodeURIComponent(donorName)}&phone=${encodeURIComponent(phone)}`,
36       cancel_url: `${process.env.SERVER_URL}/donation-cancel`,
37     });
38     res.status(200).json({ url: session.url });
39   } catch (err) {
40     console.error("Stripe error:", err);
41     res.status(500).send("Donation error. Please try again.");
42   }
43 }
```

2. Twilio Integration



```
43 const handleDonationSuccess = async (req, res) => {
44   const { name, phone } = req.query;
45   console.log("Donation success triggered:", { name, phone });
46
47   try {
48     await twilio.messages.create({
49       body: `👋 Thank you ${name} for your generous donation to OpenHands!`,
50       from: process.env.TWILIO_PHONE,
51       to: phone,
52     });
53
54     res.render("paymentSuccess", {
55       message: `Thank you ${name} for your donation! A confirmation SMS was sent.`,
56     });
57   } catch (err) {
58     console.error("Twilio error:", err.message);
59     res.render("paymentSuccess", {
60       message: `Thank you ${name} for donating! But we couldn't send an SMS.`,
61     });
62   }
63 };
64
65 module.exports = {
66   createCheckoutSession,
67   handleDonationSuccess,
68 };
69
```

3. Routes (User/Admin)

```
src > routes > fundraiserRoutes.js > ...
1  const express = require("express");
2  const router = express.Router();
3  const fundraiserController = require("../controllers/fundraiserController");
4  const { upload } = require("../middleware/multerConfig");
5
6  router.post(
7    "/createFundraiserRequest",
8    upload.fields([ { name: "image" }, { name: "document" } ]),
9    fundraiserController.createFundraiserRequest
10 );
11
12 router.get(
13   "/fetchFundraiserRequests",
14   fundraiserController.fetchFundraiserRequests
15 );
16
17 router.post(
18   "/approveFundraiserRequest",
19   fundraiserController.approveFundraiserRequest
20 );
21
22 router.post(
23   "/rejectFundraiserRequest",
24   fundraiserController.rejectFundraiserRequest
25 );
26
27 router.get(
28   "/fetchApprovedFundraisers",
29   fundraiserController.fetchApprovedFundraisers
30 );
31
32 router.get("/fetchFundraiser/:id", fundraiserController.fetchFundraiser);
33
34 module.exports = router;
```

```
src > routes > pageRoutes.js > @ router.get("adminDashboard") callback
1  const express = require("express");
2  const path = require("path");
3  const { checkAuthorization } = require("../middleware/authMiddleware");
4  //const { checkIfUserIsAdmin } = require("../services/adminAuthService");
5
6  const router = express.Router();
7
8  router.get("/", checkAuthorization, (req, res) => {
9    res.render("index", { user: req.user, isAdmin: req.isAdmin }); //sending user data to homepage to display data accordingly
10 });
11
12 router.get("/login", checkAuthorization, (req, res) => {
13   if (req.user) {
14     res.redirect("/"); //if already logged in, redirect to home page
15   } else {
16     res.render("login");
17   }
18 });
19
20 router.get("/signup", checkAuthorization, (req, res) => {
21   if (req.user) {
22     res.redirect("/"); //if already logged in, redirect to home page
23   } else {
24     res.render("signup");
25   }
26 });
27
28 router.get("/adminlogin", checkAuthorization, (req, res) => {
29   if (req.user) {
30     res.redirect("/"); //if already logged in, redirect to home page
31   } else {
32     res.render("adminLogin");
33   }
34 });
35
36 router.get("/adminDashboard", checkAuthorization, async (req, res) => {
37   if (req.user && req.isAdmin) {
38     res.render("adminDashboard");
39   } else if (req.user && !req.isAdmin) {
40     res.redirect("/"); //if a user is logged in but is not an admin, this prevents them from accessing the admin dashboard
41   } else {
42     res.redirect("adminLogin");
43   }
44 });
45
46 router.get(
```


4. Prisma Models (schema.prisma)

```
src > prisma > schema.prisma
16  model User {
17    id String @id @default(uuid())
18    email String @unique
19    password String
20    firstName String
21    lastName String
22    username String
23    country String
24    city String
25    accCreatedAt DateTime @default(now())
26    pendingRequests PendingRequests[]
27    processedRequests ProcessedRequests[]
28  }
29
30  model PendingRequests {
31    id String @id @default(uuid())
32    firstName String
33    lastName String
34    email String
35    title String
36    details String @db.LongText
37    imageName String
38    documentName String
39    amountNeeded Int
40    requestCreatedAt DateTime @default(now())
41    user User @relation(fields: [userId], references: [id])
42    userId String
43  }
44
45  model ProcessedRequests {
46    id String @id @default(uuid())
47    firstName String
48    lastName String
49    email String
50    title String
51    details String @db.LongText
52    imageName String
53    documentName String
54    amountNeeded Int
55    status Status
56    processedAt DateTime @default(now())
57    user User @relation(fields: [userId], references: [id])
58    userId String
59  }
60
61  enum Status {
62    Approved
63    Rejected
64  }
65
66  model Admin {
67    id String @id @default(uuid())
68    email String @unique
69    password String
70    firstName String
71    lastName String
72    username String
73    country String
74    city String
75    accCreatedAt DateTime @default(now())
76  }
77
```

5. References

- 1. <https://stripe.com/docs>
- 2. <https://www.twilio.com/docs>
- 3. <https://www.prisma.io/docs>
- 4. <https://nodejs.org/en/docs>