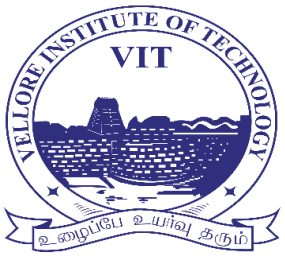


VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE ENGINEERING

AND INFORMATION SYSTEMS

PROGRAM: MCA

PMCA601L – Full Stack Web Development

Food Wastage Reduction Management System

Course Project

SUBMITTED ON: 11 - APR - 2025

Submitted By:

Alan K Sijo – 24MCA0227

Karan Maurya – 24MCA0260

Yash Bishnoi – 24MCA0263

Rahul Mittal – 24MCA0252

Akash Kumar Banik – 24MCA0242

Submitted To:

Prof. Sumangali K

Objective

The primary objective of this project is to create a Food Sharing Platform that connects donors with individuals or organizations in need of food. The platform aims to reduce food wastage and promote social welfare by enabling users to donate surplus food and request food donations efficiently.

Features

i. User Authentication:

Users can register and log in to the platform.

Authentication is implemented using JWT (JSON Web Tokens) for secure access.

ii. Food Donation:

Users can add food donations by providing details such as title, food item, quantity, location, expiry date, and an image of the food.

Images are uploaded to Cloudinary for secure storage.

iii. Food Requests:

Users can request food donations from available listings.

Donors can accept or reject requests.

iv. Smart Recipe Suggestions:

Users can input ingredients, and the platform provides AI-generated recipe suggestions using a generative language model.

v. Notifications:

Users receive notifications for food requests and updates on their donations.

vi. Responsive Design:

The platform is fully responsive and optimized for both desktop and mobile users.

Software Requirement

- **Frontend:**
  - React.js (v18.0 or higher)
  - Tailwind CSS
  - Node.js (v16.x or higher)
- **Backend:**
  - **MongoDB:** NoSQL database for storing user data, food donations, requests, and notifications.

- **Cloudinary:** Cloud-based service for storing and managing uploaded images.
- **JWT (JSON Web Tokens):** Secure token-based authentication for user sessions.
- **Google Generative Language API:** Used for generating recipe suggestions based on user-provided ingredients.

- **Development Tools:**
  - Visual Studio Code (IDE)
  - Postman (for API testing)
  - Git (for version control)

2. Layout Design

A. User Authentication

Registration:

Users register by providing their email, password, and other details.

Passwords are hashed before being stored in the database.

Login:

Users log in with their email and password.

A JWT token is generated and stored in the browser's local storage for authentication.

B. Food Donation

Add Donation:

Users fill out a form to add a food donation, including details like title, food item, quantity, location, expiry date, and an image.

The image is uploaded to Cloudinary, and the secure URL is stored in the database.

View Donations:

All available donations are displayed on the homepage for users to browse.

C. Food Requests

Request Food:

Users can request food donations by clicking on a donation listing and submitting a request.

Manage Requests:

Donors can view all requests for their donations and either accept or reject them.

Notifications:

Users receive notifications when their requests are accepted or rejected.

D. Smart Recipe Suggestions

Input Ingredients:

Users enter a list of ingredients they have.

AI-Generated Recipes:

The platform uses the Google Generative Language API to generate recipe suggestions based on the provided ingredients.

Display Recipes:

Recipes are displayed with details like preparation time, cooking time, and instructions.

E. Notifications

Notifications are sent to users for the following events:

When a food request is made.

When a request is accepted or rejected.

Updates on donations.

F. Responsive Design

Desktop View:

Sidebar navigation is always visible.

Main content is displayed alongside the sidebar.

Mobile View:

Sidebar is hidden by default and can be toggled using a hamburger menu.

Main content adjusts to fit smaller screens.

Source Code

1. Frontend (client)

client/

├── public/

├── src/

│ ├── components/ Reusable components (e.g., Sidebar, Navbar)

│ ├── pages/ Page components (e.g., Home, Add, Profile)

│ ├── auth/ Authentication-related components (e.g., PrivateRoute)

```
│   ├── App.jsx          Main application component
│   ├── index.js         Entry point for the React app
│   └── styles/          Tailwind CSS configuration
└── package.json         Frontend dependencies
```

1. **Package.json**

```
{
  "name": "client",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint .",
    "preview": "vite preview"
  },
  "dependencies": {
    "@tailwindcss/vite": "^4.0.17",
    "axios": "^1.8.4",
    "client": "file:",
    "lucide-react": "^0.483.0",
    "react": "^19.0.0",
    "react-dom": "^19.0.0",
    "react-router-dom": "^7.4.0",
    "tailwindcss": "^4.0.17"
  },
  "devDependencies": {
    "@eslint/js": "^9.21.0",
    "@types/react": "^19.0.10",
    "@types/react-dom": "^19.0.4",
    "@vitejs/plugin-react": "^4.3.4",
    "eslint": "^9.21.0",
    "eslint-plugin-react-hooks": "^5.1.0",
    "eslint-plugin-react-refresh": "^0.4.19",
    "globals": "^15.15.0",
    "vite": "^6.2.0"
  }
}
```

2. **Main.jsx**

```
import { createRoot } from "react-dom/client";
import "./index.css";
import App from "./App.jsx";
import { BrowserRouter as Router } from "react-router-dom";

createRoot(document.getElementById("root")).render(
  <Router>
    <App />
  </Router>
);
```

3. **App.jsx**

```
import React, { useState } from "react";
import { Link, useLocation, Route, BrowserRouter as Router, Routes } from "react-router-dom";
import { Home, Add, Profile, LoginOrRegister, Requests } from "./pages/index";
import PrivateRoute from "./auth/PrivateRoute";
import FoodDetails from "./pages/FoodDetails";
import { LucideHome, PlusCircle, ClipboardList, User, NotebookText, Menu, X, } from "lucide-
react";
import SmartRec from "./pages/SmartRec";

const App = () => {
  const location = useLocation();
  const user = localStorage.getItem("email");
  const [isSidebarOpen, setIsSidebarOpen] = useState(false);

  if (!user) {
    return <LoginOrRegister />;
  }

  const links = [
    { to: "/", label: "Home", icon: <LucideHome size={20} /> },
    { to: "/add", label: "Add", icon: <PlusCircle size={20} /> },
    { to: "/requests", label: "Requests", icon: <ClipboardList size={20} /> },
    { to: "/smartrecipe", label: "Smart Recipe", icon: <NotebookText size={20} /> },
    { to: "/profile", label: "Profile", icon: <User size={20} /> },
  ];

  return (
    <div className="flex h-screen">
      {/* Sidebar */}
      <div
        className={`fixed top-0 left-0 z-50 h-full bg-black text-white shadow-lg transition-
transform transform ${isSidebarOpen ? "translate-x-0" : "-translate-x-full"
```

```
        } md:translate-x-0 md:relative md:w-64`}
      >
        <button
          className="absolute top-4 right-4 md:hidden text-white"
          onClick={() => setIsSidebarOpen(false)}
        >
          <X size={24} />
        </button>
        <div className="flex flex-col gap-4 items-center pt-16 px-2 border-r border-white/30">
          {links.map(({ to, label, icon }) => (
            <Link
              key={to}
              to={to}
              className={`flex items-center gap-3 w-full px-6 py-3 rounded-lg transition
${location.pathname === to
                  ? "bg-white text-black"
                  : "hover:bg-gray-700"
                }`}
              onClick={() => setIsSidebarOpen(false)} // Close sidebar on link click
            >
              {icon}
              <span className="text-base">{label}</span>
            </Link>
          ))}
        </div>
      </div>

      {/* Hamburger Menu for Mobile */}
      <button
        className="fixed top-4 left-4 z-50 md:hidden bg-black text-white p-2 rounded-full
shadow-lg"
        onClick={() => setIsSidebarOpen(true)}
      >
        <Menu size={24} />
      </button>

      {/* Main Content */}
      <div className="flex-1 h-screen overflow-y-scroll bg-gray-100 p-0">
        <Routes>
          {/* Public Route */}
          <Route path="/" element={<Home />} />

          {/* Protected Route */}
          <Route
            path="/add"
```

```
            element={
              <PrivateRoute>
                <Add />
              </PrivateRoute>
            }
          />
          <Route path="/profile" element={<Profile />} />
          <Route path="/login" element={<LoginOrRegister />} />
          <Route path="/food/:id" element={<FoodDetails />} />
          <Route path="/requests" element={<Requests />} />
          <Route path="/smartrecipe" element={<SmartRec />} />
        </Routes>
      </div>
    </div>
  );
};

export default App;
```

4. **Home.jsx**

```
import React, { useEffect, useState } from "react";
import axios from "axios";
import { Link } from "react-router-dom";

const Home = () => {
  const [items, setItems] = useState([]);

  const getItems = async () => {
    const email = localStorage.getItem("email");
    // console.log(import.meta.env.VITE_DEV_URL);
    try {
      const { data } = await axios.post(`${import.meta.env.VITE_DEV_URL}/food`, {
        email,
      });
      setItems(data);
    } catch (error) {
      console.error("Error fetching data:", error);
    }
  };

  useEffect(() => {
    getItems();
  }, []);
```

```
  return (
    <div className="min-h-screen bg-gray-900 text-gray-200">

      {/* Food Items Section */}
      <main className="py-12 px-6" id="food-items">
        <h2 className="text-2xl font-semibold mb-8 text-center">
          Available Food Items
        </h2>

        {items.length > 0 ? (
          <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-8">
            {items.map((item, index) => (
              <FoodItem key={index} item={item} />
            ))}
          </div>
        ) : (
          <p className="text-center text-gray-500">
            No food items available at the moment.
          </p>
        )}
      </main>
    </div>
  );
};

export default Home;

const FoodItem = ({ item }) => {
  const expirydate = new Date(item.expiryDate);
  const today = new Date();
  const timediff = expirydate - today;
  const daysleft = Math.ceil(timediff / (1000 * 60 * 60 * 24));

  const getExpiryStatus = () => {
    if (daysleft <= 0) return { text: "Expired", color: "bg-red-500" };
    if (daysleft <= 3) return { text: "Expiring Soon", color: "bg-yellow-500" };
    return { text: `Expires in ${daysleft} days`, color: "bg-green-500" };
  };

  const { text, color } = getExpiryStatus();

  return (
    <Link to={`/food/${item._id}`} className="bg-gray-800 rounded-lg shadow-md overflow-
hidden hover:shadow-lg transition transform hover:scale-105">
      {/* Food Image */}
```

```
        <div className="w-full h-40 bg-gray-700">
          <img
            src={item.images}
            alt={item.foodItem}
            className="w-full h-full object-cover"
          />
        </div>

        {/* Food Details */}
        <div className="p-4">
          <h3 className="text-lg font-semibold mb-2">{item.foodItem}</h3>
          <p className="text-sm text-gray-400 mb-1">Quantity: {item.quantity}</p>
          <p className="text-sm text-gray-400 mb-1">Location: {item.location}</p>
          <p className="text-sm text-gray-400 mb-3">
            Expiry Date: {new Date(item.expiryDate).toLocaleDateString()}
          </p>

          {/* Expiry Badge */}
          <span
            className={`inline-block px-3 py-1 text-xs font-medium text-white rounded-full ${color}`}
          >
            {text}
          </span>
        </div>
      </Link>
    );
  };
```

2. Backend (server)

server/

```
├── models/             Mongoose models (e.g., User, Food, Request)
├── routes/             Express routes (e.g., foodRoutes, userRoutes)
├── middleware/         Middleware (e.g., authentication)
├── uploads/            Temporary storage for uploaded files
├── server.js           Main server file
├── .env                Environment variables
└── package.json        Backend dependencies
```

1. **Package.json**

```
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "dev": "nodemon server.js",
    "start": "node server.js",
    "build": "npm run build"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@google/generative-ai": "^0.24.0",
    "axios": "^1.8.4",
    "bcryptjs": "^3.0.2",
    "body-parser": "^1.20.3",
    "cloudinary": "^2.6.0",
    "cookie-parser": "^1.4.7",
    "cors": "^2.8.5",
    "dotenv": "^16.4.7",
    "express": "^4.21.2",
    "jsonwebtoken": "^9.0.2",
    "mongoose": "^8.12.1",
    "multer": "^1.4.5-lts.2",
    "nodemon": "^3.1.9",
    "server": "file:"
  }
}
```

2. **Server.js**

```
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
const cookieParser = require("cookie-parser");
const foodRoutes = require("./routes/foodRoutes.js");
const userRoutes = require("./routes/userRoutes.js");
require("dotenv").config();
const app = express();
app.use(
  cors({
    origin: "http://localhost:5173",
```

```
    credentials: true,
    methods: ["GET", "POST", "PUT", "DELETE"],
  })
);
app.use(express.json());
app.use(cookieParser());
app.use("/food", foodRoutes);
app.use("/auth", userRoutes);

const PORT = process.env.PORT || 5000;

// MongoDB Connection
mongoose
  .connect(process.env.MONGO_URI)
  .then(() => console.log("MongoDB Connected"))
  .catch((err) => console.error(err));

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

3. **userRoutes.js**

```
const express = require("express");
const User = require("../models/User");
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
const router = express.Router();
const authenticateUser = require("../middleware/auth");
const Notification = require("../models/Notif");
const axios = require("axios");
const { GoogleGenerativeAI } = require("@google/generative-ai");


// Register user
router.post("/register", async (req, res) => {
  try {
    const { name, email, password } = req.body;
    if (!name || !email || !password) {
      return res.status(400).json({ message: "All fields are required" });
    }
    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(400).json({ message: "User already exists" });
    }
```

```
    const hashedPassword = await bcrypt.hash(password, 10);
    const user = new User({ name, email, password: hashedPassword });
    await user.save();
    res.status(201).json({ message: "User registered successfully" });
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});

// Login user
router.post("/login", async (req, res) => {
  try {
    const { email, password } = req.body;
    console.log("Data:", { email, password });
    if (!email || !password) {
      return res
        .status(400)
        .json({ message: "Email and password are required" });
    }

    const user = await User.findOne({ email });
    console.log("User:", user);
    if (!user || !(await bcrypt.compare(password, user.password))) {
      return res.status(400).json({ message: "Invalid email or password" });
    }

    const token = jwt.sign({ id: user._id }, "secret", {
      expiresIn: "7d",
    });
    console.log("Token", token);

    res.cookie("token", token, {
      httpOnly: true,
      secure: process.env.NODE_ENV === "production", // Set to true in production
      maxAge: 7 * 24 * 60 * 60 * 1000, // 7 days
    });

    res.json({ message: "Login successful", token, user });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

router.post("/jwt", (req, res) => {
  const token = req.cookies.token;
```

```
  if (!token) {
    res.clearCookie("token");
    return res.status(401).json({ message: "Unauthorized" });
  }
  try {
    const user = jwt.verify(token, "secret");
    console.log("User", user);
    res.status(200).json({ user });
  } catch (err) {
    res.status(401).json({ message: "Unauthorized" });
  }
})

router.post("/logout", async (req, res) => {
  const { email } = req.body;
  const user = await User.updateOne({ email }, { token: "" });
  res.clearCookie("token").json({ message: "Logged out" });
});

router.post("/notifications", authenticateUser, async (req, res) => {
  try {
    const { email } = req.body;
    const notifications = await Notification.find({ recipient: email }).sort({ createdAt: -1 });
    res.status(200).json(notifications);
  } catch (err) {
    console.log(err);
    res.status(500).json({ error: err.message });
  }
});

router.delete("/notifications/:id", authenticateUser, async (req, res) => {
  try {
    const { id } = req.params;
    const notification = await Notification.findByIdAndDelete(id);

    if (!notification) {
      return res.status(404).json({ message: "Notification not found" });
    }

    res.status(200).json({ message: "Notification deleted successfully" });
  } catch (err) {
    console.log(err);
    res.status(500).json({ error: err.message });
  }
});
```

```
router.put("/notifications/:id", authenticateUser, async (req, res) => {
  try {
    const { id } = req.params;
    const notification = await Notification.findById(id);

    if (!notification) {
      return res.status(404).json({ message: "Notification not found" });
    }

    notification.isRead = true;
    await notification.save();

    res.status(200).json({ message: "Notification marked as read" });
  } catch (err) {
    console.log(err);
    res.status(500).json({ error: err.message });
  }
});


router.post("/generate-content", async (req, res) => {
  // 1. Extract Ingredients from Request Body
  const { ingredients } = req.body;

  // 2. Validate Input
  if (!ingredients || typeof ingredients !== 'string' || ingredients.trim() === "") {
    // Check if ingredients exist, are a string, and are not just whitespace
    return res.status(400).json({ message: "Ingredients (string) are required in the request
body." });
  }
  const trimmedIngredients = ingredients.trim(); // Use trimmed version

  // 3. Get API Key and Validate
  const apiKey = process.env.GEN_API_KEY;
  if (!apiKey) {
    console.error("FATAL ERROR: Google Generative AI API key (GEN_API_KEY) is missing in
environment variables.");
    return res.status(500).json({ error: "Server configuration error: API key is missing." });
  }
  // Optional: Log only that the key is present, not the key itself for security
  console.log("Using Google Generative AI API Key: Present");

  try {
    // 4. Initialize Google Generative AI Client
```

```
    const genAI = new GoogleGenerativeAI(apiKey);


    const model = genAI.getGenerativeModel({ model: "gemini-1.5-pro-latest" });
    const prompt = `
      I have these leftover ingredients: ${trimmedIngredients}.

      Create a creative recipe that primarily uses them, minimizing waste.

      **Format the entire response STRICTLY as a single, valid JSON object.**
      **Do NOT include any introductory text, explanations, markdown formatting (like \`\`\`json),
or any characters outside the JSON object itself.**

      The JSON object should have the following structure:
      {
        "dishName": "String",
        "prepTime": "String (e.g., '10 minutes')",
        "cookingTime": "String (e.g., '15 minutes')",
        "ingredients": ["String ingredient 1", "String ingredient 2", ...],
        "instructions": ["String step 1", "String step 2", ...],
        "additionsSubstitutions": "String (or null if none)",
        "servingSize": "String (e.g., '2 servings')",
        "variations": "String (or null if none)"
      }

      Ensure all text values within the JSON are properly escaped strings.
      Generate the recipe details based on the provided ingredients: ${trimmedIngredients}.
    `;

    console.log(`Generating content for ingredients: ${trimmedIngredients}`);

    const generationConfig = {
      temperature: 0.7,        // Controls randomness (creativity vs. predictability)
      candidateCount: 1,       // Number of response candidates to generate
      maxOutputTokens: 1024,     // Increased max tokens for potentially longer recipes
      topP: 0.8,               // Nucleus sampling parameter
      topK: 40                 // Top-K sampling parameter
    };

    const result = await model.generateContent({
       contents: [{ role: "user", parts: [{ text: prompt }] }],
       generationConfig: generationConfig // Pass the config here
    });
```

```
    const generatedText = result?.response?.candidates?.[0]?.content?.parts?.[0]?.text;

    if (!generatedText) {
      console.error("API Error: No text content received in the response.", JSON.stringify(result,
null, 2));
      return res.status(500).json({ error: "Failed to generate content: Empty response from AI
model." });
    }

    console.log("Successfully generated content.");
    console.log(generatedText);

    res.status(200).json({ recipe: JSON.parse(generatedText) });

  } catch (error) {
    // 11. Handle Errors during API call or processing
    console.error("API Call Error:", error.message); // Log the specific error message
    // Log the full error for server-side debugging, but don't send it all to the client
    console.error("Full Error Details:", error);

    // Send a generic error message to the client
    res.status(500).json({
        error: "Failed to generate content due to an internal server error.",

    });
  }
});
module.exports = router;
```

4. **foodRoutes.js**

```
const express = require("express");
const router = express.Router();
const Food = require("../models/Food");
const authenticateUser = require("../middleware/auth");
const Request = require("../models/Request");
const Notification = require("../models/Notif");
const cloudinary = require('cloudinary').v2;
const multer = require("multer");
const fs = require("fs");
const upload = multer({ dest: "uploads/" });
const path = require("path");

router.post("/requests", async (req, res) => {
  try {
```

```
    const { userMail } = req.body;
    console.log("email:", userMail);
    const requests = await Request.find({ donorMail: userMail });
    console.log("requests", requests);
    const claimed = requests.filter((item) => item.status === "Accepted");
    const available = requests.filter((item) => item.status === "Pending");
    console.log("claimed", claimed);
    console.log("available", available);
    res.status(200).json({ claimed, available });
  } catch (error) { }
});

router.post("/requestDonation", async (req, res) => {
  try {
    const {title, userMail, donorMail, foodItemId, requestedQuantity, message } =
      req.body;

    if (!title || !userMail || !foodItemId || !requestedQuantity || !donorMail) {
      console.log("error here ");
      return res.status(400).json({ error: "Missing required fields" });
    }

    const newRequest = new Request({
      title,
      userMail,
      donorMail,
      foodItemId,
      requestedQuantity,
      message,
    });
    await newRequest.save();
    res.status(201).json({ success: true, message: "Request submitted successfully" });
  } catch (error) {
    console.error("Error requesting donation:", error);
    res.status(500).json({ error: "Internal Server Error" });
  }
});

// module.exports = router;

// Add food (Only for logged-in users)
router.post("/add", authenticateUser, upload.single("file"), async (req, res) => {
  try {
    console.log("Request body:", req.body);
    const { title, foodItem, quantity, description, location, expiryDate, donorMail } =
```

```
  req.body;
const file = req.file;
console.log("File:", file);
if (!file) {
  console.log("File not found in request");
  return res.status(400).json({ error: "No file uploaded" });
}

const filePath = path.join(__dirname, "../uploads", file.filename);
cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});

console.log("Cloudinary config:", {
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
});
const uploadFile = await cloudinary.uploader.upload(filePath, {
  folder: "food_images",
  use_filename: true,
  unique_filename: false,
}).catch((err) => {
  console.error("Error uploading file to Cloudinary:", err);
  return res.status(500).json({ error: "Error uploading file" });
});

const imageUrl = uploadFile.secure_url;

const newFood = new Food({
  donorId: req.user?.id,
  donorName: req.user.name,
  title,
  donorMail,
  foodItem,
  description,
  quantity,
  location,
  expiryDate,
  images: imageUrl,
});
await newFood.save();
fs.unlink(filePath, (err) => {
  if (err) {
```

```
        console.error("Error deleting file:", err);
      } else {
        console.log("File deleted successfully");
      }
    });
    res.status(201).json({ message: "Food added successfully", food: newFood });
  } catch (err) {
    console.log(err);
    res.status(400).json({ error: err.message });
  }
});

router.get("/getItem/:id", async (req, res) => {
  try {
    const foodItem = await Food.find({ _id: req.params.id });
    res.status(200).json(foodItem);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// Get all available food
router.post("/", async (req, res) => {
  const { mail } = req.body;
  try {
    const foodItems = await Food.find({
      status: "Available",
      donorMail: { $ne: mail },
    });

    res.status(200).json(foodItems);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// Claim food
router.post("/acceptReq", authenticateUser, async (req, res) => {
  try {

    const { id, mail } = req.body;

    const foodItem = await Food.findById({ _id: id });

    if (!foodItem || foodItem.status !== "Available") {
```

```
      console.log("Returned from line no. /claim > 107");
      return res.status(404).json({ message: "Food not available" });
    }

    const freq = await Request.findOne({ foodItemId: id });

    foodItem.status = "Claimed";
    foodItem.claimedBy = mail;
    await foodItem.save();
    freq.status = "Accepted";
    await freq.save();
    const notification = new Notification({
      title: "Food Claimed",
      recipient: freq.userMail,
      message: `Your request for ${foodItem.foodItem} has been accepted.`,
    });
    await notification.save();
    res.status(200).json({ message: "Food claimed successfully" });
  } catch (err) {
    console.log(err);
    res.status(500).json({ error: err.message });
  }
});

router.post("/rejectReq", authenticateUser, async (req, res) => {

  try {
    const { id, mail } = req.body;

    const freq = await Request.findOne({ foodItemId: id });
    if (!freq) {
      return res.status(404).json({ message: "Request not found" });
    }
    freq.status = "Rejected";
    await freq.save();
    const notification = new Notification({
      title: "Request Rejected",
      recipient: freq.userMail,
      message: `Your request for ${freq.foodItemId} has been rejected.`,
    });
    await notification.save();
    res.status(200).json({ message: "Request rejected successfully" });
  }
  catch (err) {
    console.log(err);
```

```
    res.status(500).json({ error: err.message });
  }
});

module.exports = router;
```

Models

5. **Food.js**

```
const mongoose = require("mongoose");
const FoodSchema = new mongoose.Schema({
  donorId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
    required: true,
  }, // Stores donor's ID
  donorMail: { type: String, required: true },
  donorName: { type: String, required: true }, // Redundant but useful for quick access
  title: { type: String, required: true },
  foodItem: { type: String, required: true },
  description: { type: String, required: true },
  quantity: { type: Number, required: true },
  location: { type: String, required: true },
  expiryDate: { type: Date, required: true },
  images: { type: String },
  claimedBy: { type: String, default: null },
  status: {
    type: String,
    enum: ["Available", "Claimed"],
    default: "Available",
  },
  createdAt: { type: Date, default: Date.now },
});

module.exports = mongoose.model("Food", FoodSchema);
```

6. **Request.js**

```
const mongoose = require("mongoose");

const requestSchema = new mongoose.Schema({
  title:{
    type: String,
    required: true,
```

```
  },
  userMail: {
    type: String,
    ref: "User",
    required: true,
  },
  foodItemId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "FoodItem",
    required: true,
  },
  donorMail: {
    type: String,
    ref: "User",
    required: true,
  },
  requestedQuantity: { type: Number, required: true },
  message: { type: String },
  status: {
    type: String,
    enum: ["Pending", "Accepted", "Rejected"],
    default: "Pending",
  },
  requestDate: { type: Date, default: Date.now },
});

module.exports = mongoose.model("Request", requestSchema);
```

7. **Notifications.js**

```
const mongoose = require('mongoose');
const notificationSchema = new mongoose.Schema({
    title: {
        type: String,
        required: true,
        trim: true,
    },
    message: {
        type: String,
        required: true,
    },
    recipient: {
        type: String,
```

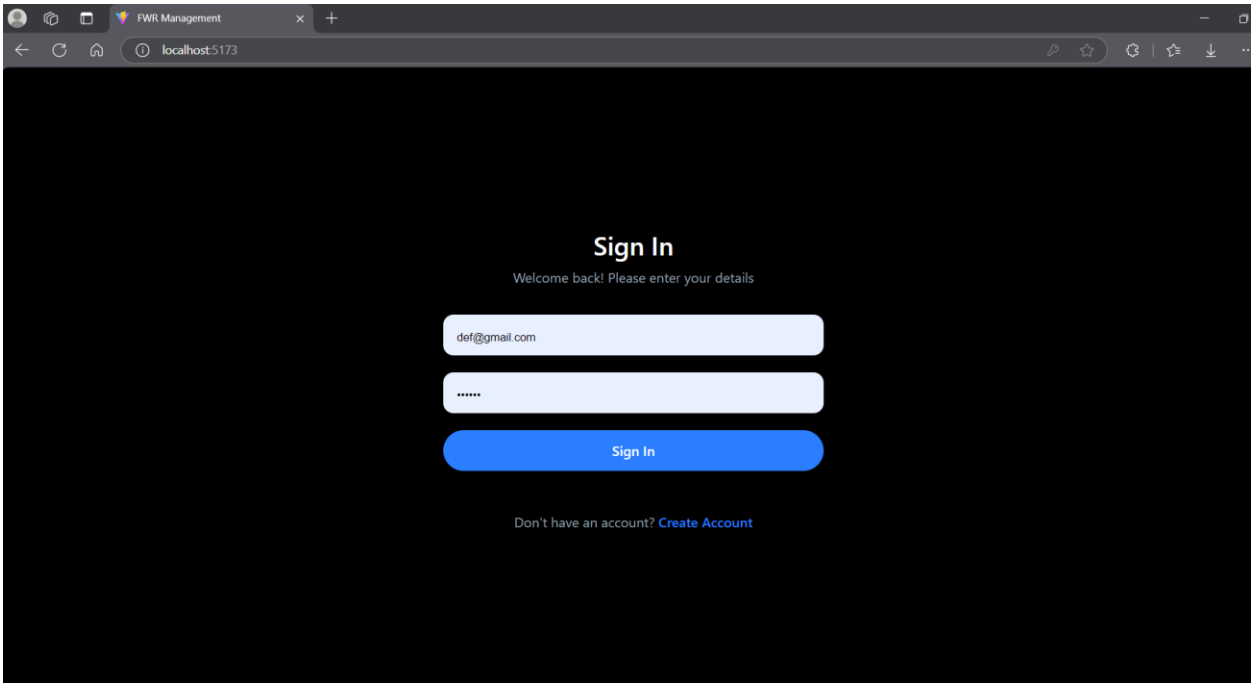
```
        required: true,
    },
    isRead: {
        type: Boolean,
        default: false,
    },
    createdAt: {
        type: Date,
        default: Date.now,
    },
});

module.exports = mongoose.model('Notification', notificationSchema);
```

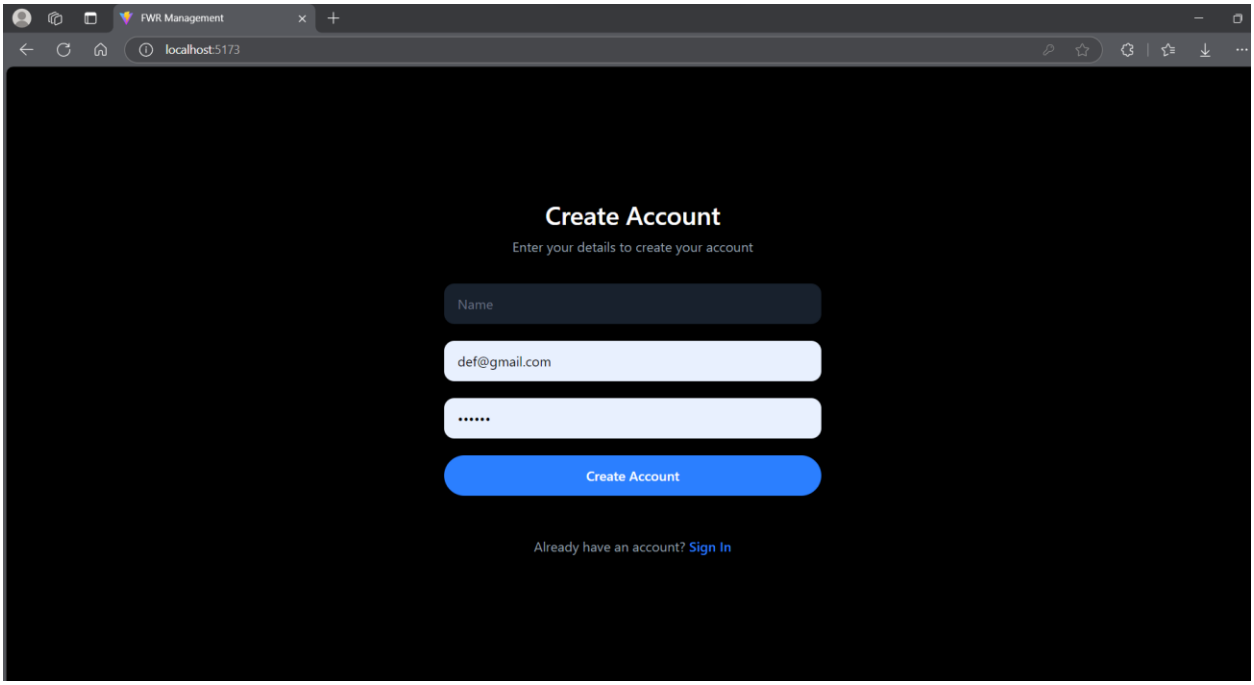
8. **User.js**

```
const mongoose = require("mongoose");
const UserSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
});
module.exports = mongoose.model("User", UserSchema);
```

4. Execution Screenshots

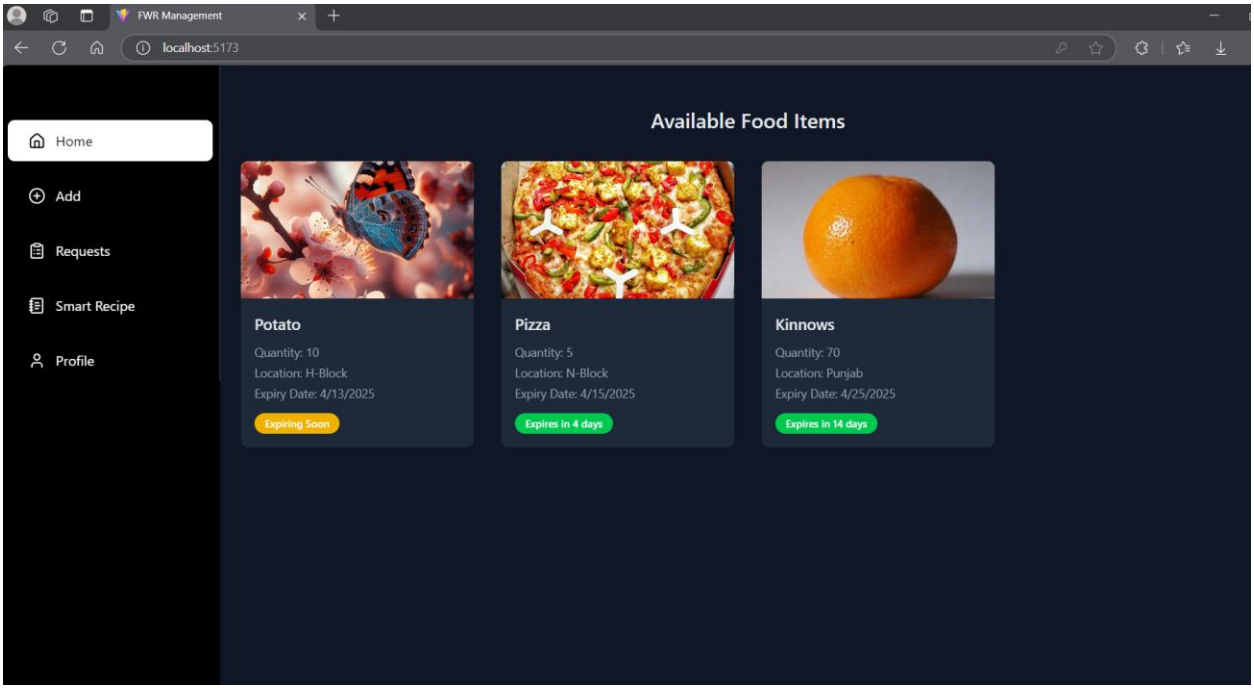
1. Login/Signup Page:





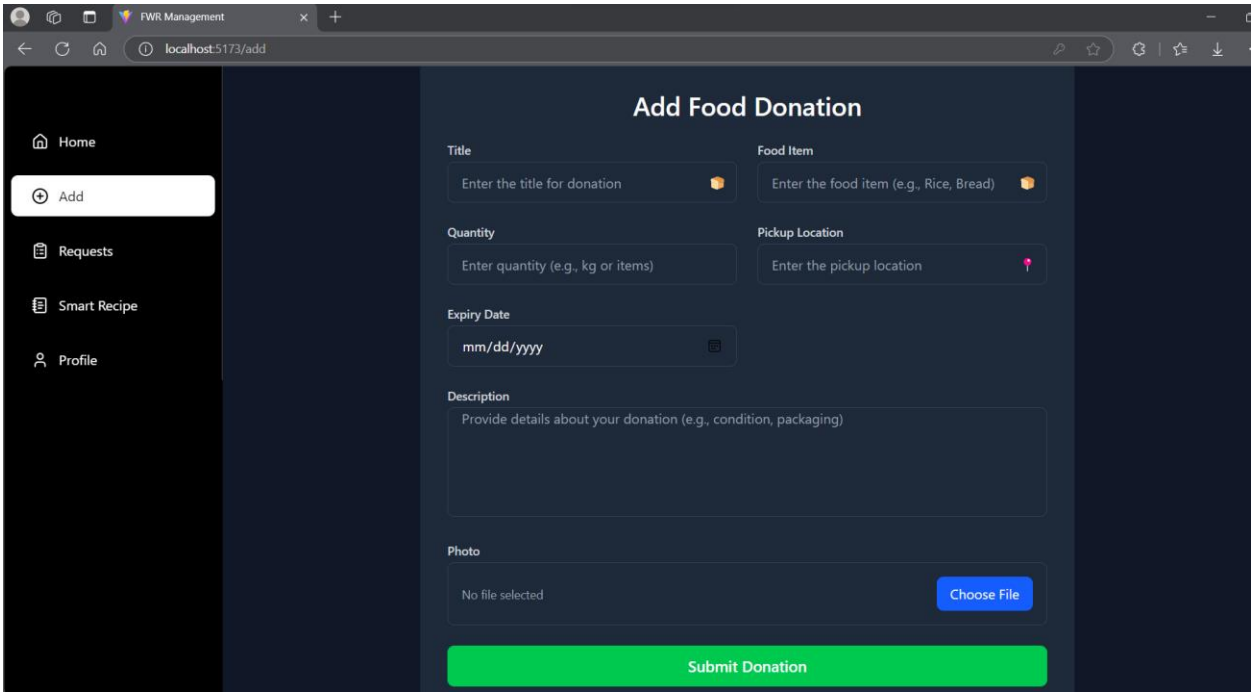
2. Homepage:

Displays all available food donations with details like title, quantity, and location.



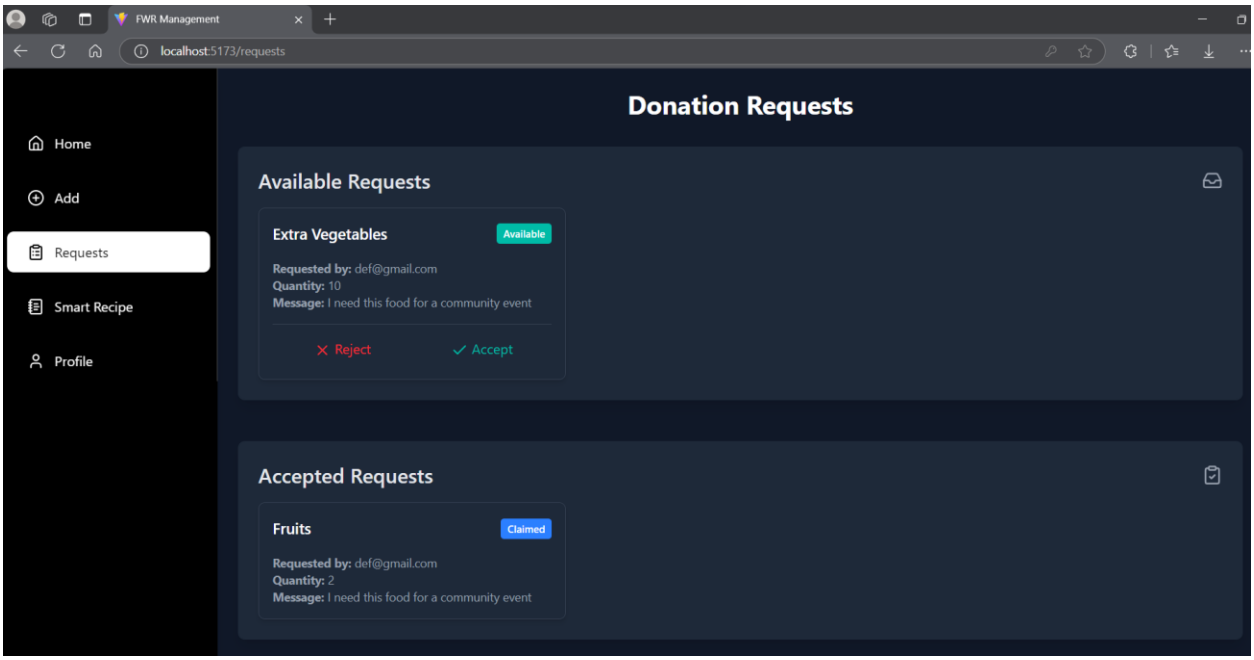
3. Add Food Donation:

A form for adding food donations with an image upload option.



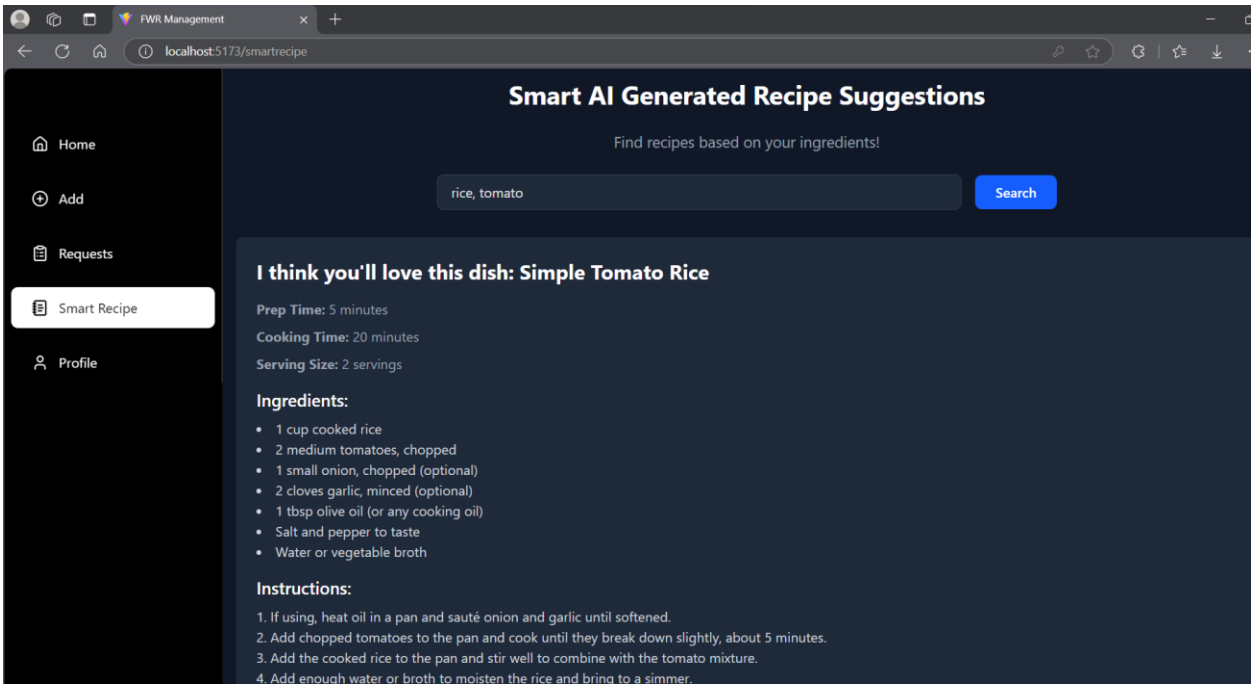
4. Requests Page:

Shows request for donation by other individuals to the user.



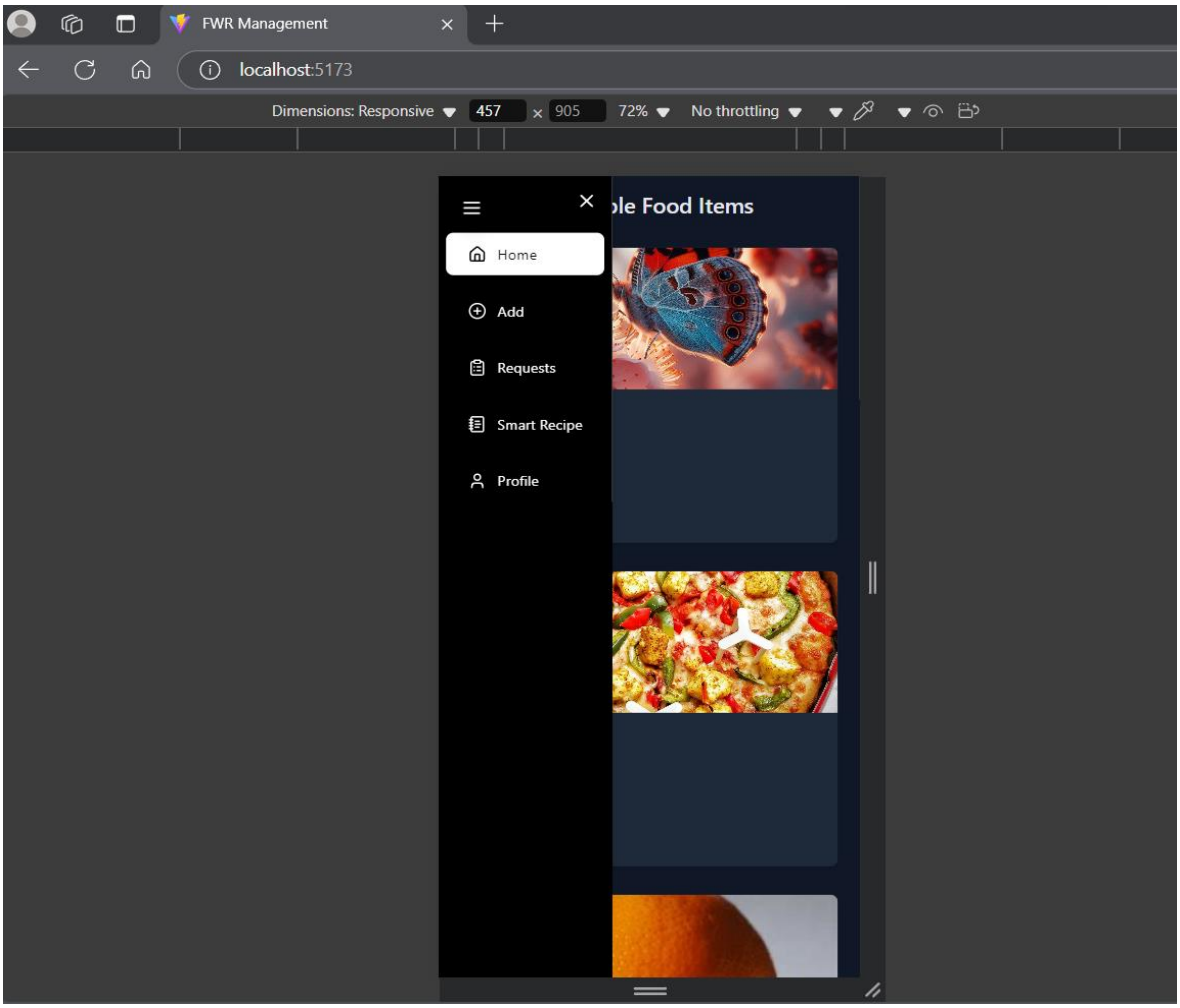
5. Smart Recipe Suggestions:

Input ingredients and view AI-generated recipes.



6. Mobile View:

Collapsible sidebar and responsive layout for smaller screens.



Conclusion and Future Work

- **Conclusion:**

The Food Sharing Platform successfully connects donors with individuals or organizations in need of food. It reduces food wastage and promotes social welfare by providing an easy-to-use interface for food donations and requests. The integration of AI for recipe suggestions adds value to the platform, making it more engaging and useful for users.

- **Future Work:**

1. Report Generation: Creating reports of donations done by user or organization.
2. Advanced Search and Filters: Add search and filter options for food donations.
3. Multi-Language Support: Provide support for multiple languages to cater to a global audience.
4. Mobile App: Develop a mobile application for better accessibility.