**Common Corporate Social Responsibility (CSR) Portal**

Corporate Social Responsibility (CSR) is an essential part of corporate culture, enabling companies to contribute to social welfare initiatives. However, many corporations, especially Small and Medium-sized Enterprises (SMEs), face difficulties in executing large-scale CSR activities due to limited resources, knowledge, or connections with suitable NGOs. Conversely, NGOs and smaller social organizations struggle to secure resources and corporate partnerships. A centralized and collaborative CSR portal is required to bridge this gap, facilitating collaboration between corporations and NGOs, pooling resources, and achieving a broader societal impact.

## **Key Features**

### **1. NGO Registration & Needs Listing**

* NGOs can create profiles, outline their social welfare objectives, and specify their requirements (funds, resources, volunteers).

### **2. Corporate Sector Registration & Budget Allocation**

* Corporations can register, define their CSR budgets, and list the types of initiatives they support (education, healthcare, environment, etc.).

### **3. Matching Algorithm**

* The system automatically connects NGOs with corporate contributors based on sector alignment, budget allocation, and geographical location to foster effective collaborations.

### **4. Collaboration Dashboard**

* A centralized dashboard enables corporations and NGOs to track ongoing and past collaborations, monitor initiative statuses, and manage their CSR efforts efficiently.

### **5. Reporting & Impact Analytics**

* Automated reports measure CSR impact through metrics such as funds allocated, lives impacted, and resources delivered.

### **6. Geographical Search Filters**

* Users can filter initiatives based on region, emphasizing support for underserved communities and local causes.

## **Bonus Features**

### **1. Tax Incentives Tracker**

* Integration with India’s tax policies to monitor CSR-related tax benefits for corporations.

### **2. Payment Integration & Financial Transparency**

* A secure payment gateway for donations with detailed financial tracking to ensure transparency in fund allocation.

### **3. Real-Time Updates**

* Push notifications and alerts for corporations and NGOs regarding new partnership opportunities, events, and relevant CSR trends.

### **4. CSR Campaigns & Crowdfunding**

* A dedicated feature for corporations to initiate CSR campaigns or crowdfunding efforts, facilitating urgent or niche social causes.

By centralizing CSR efforts, this portal enables corporations and NGOs to work together more effectively, driving meaningful social change while ensuring transparency and efficiency in resource distribution.

+

## **1️⃣ Tech Stack (All Free Tools)**

### **Backend (Python + Flask)**

* **Flask**: Lightweight and free for creating REST APIs and serving pages.
* **Flask-JWT-Extended**: For JWT-based authentication.
* **PyMongo or MongoEngine**: To connect with your locally installed MongoDB.
* **Flask-SocketIO**: For real-time messaging and notifications.
* **Celery with Redis (Optional)**: For background tasks (if needed). Both Redis and Celery are free and can run locally.

### **Database**

* **MongoDB Community Edition**: Install and run it locally on your machine.

### **Frontend (React + Tailwind CSS)**

* **React.js**: Use [Create React App](https://create-react-app.dev/) (free) to set up the frontend.
* **React Router**: For page navigation.
* **Tailwind CSS**: Can be used via a CDN or locally set up with PostCSS (both options are free).
* **Redux (optional)**: For global state management if needed.

### **Development & Testing Tools**

* **Visual Studio Code** (or any free code editor)
* **Git**: For version control.
* **Postman/Insomnia**: For testing your APIs.
* **ngrok (Optional)**: If you need to expose your local server temporarily for a demo (has a free tier).

## **2️⃣ Project Structure**

### **Backend (Flask) Structure**

Organize your Flask app into modular blueprints to keep things tidy. For example:

bash

CopyEdit

/csr\_portal

/app

/auth

\_\_init\_\_.py

routes.py # Signup, login, JWT auth

/ngo

\_\_init\_\_.py

routes.py # NGO-specific endpoints (profile, posts, matching, events)

/company

\_\_init\_\_.py

routes.py # Company-specific endpoints (profile, posts, matching, events)

/messaging

\_\_init\_\_.py

routes.py # Real-time chat endpoints (using SocketIO)

/models

user.py

ngo.py

company.py

event.py

\_\_init\_\_.py # App factory and configuration

config.py # Configuration settings (local MongoDB URI, JWT secret, etc.)

run.py # Script to start the Flask server

### **Frontend (React) Structure**

Using Create React App, your project structure might look like:

pgsql

CopyEdit

/csr-portal-frontend

/public

/src

/components

Header.js // Persistent navbar with Sign In/Sign Up

Footer.js

/pages

Home.js // Homepage with info about CSR

NgoDashboard.js // NGO home feed, search, match, profile, events, messaging

CompanyDashboard.js // Company home feed, search, match, profile, events, messaging

Login.js

Signup.js

/services

api.js // API calls to your Flask backend

App.js

index.js

package.json

## **3️⃣ Database Schema (MongoDB)**

Since you’re running MongoDB locally, you can use a flexible schema. Example collections:

### **Users Collection**

json

CopyEdit

{

"\_id": "ObjectId",

"email": "user@example.com",

"password": "hashed\_password",

"user\_type": "NGO" // or "Company",

"profile": {

"name": "Name of NGO or Company",

"description": "Description and details",

"location": "City, Country",

// additional fields as required

}

}

### **Posts Collection**

For Company posts (NGOs see these in their feed):

json

CopyEdit

{

"\_id": "ObjectId",

"company\_id": "CompanyId",

"content": "Here's how we can help!",

"timestamp": "ISODate",

"media\_url": "optional\_image\_or\_video\_url"

}

### **Events Collection**

json

CopyEdit

{

"\_id": "ObjectId",

"title": "Event Title",

"organizer": "NGO or Company ID",

"date": "2025-02-15T00:00:00Z",

"location": "City, Country",

"description": "Event details",

"participants": ["UserId1", "UserId2"]

}

### **Messages Collection**

You can either embed messages in a user’s document or use a separate collection:

json

CopyEdit

{

"\_id": "ObjectId",

"sender\_id": "UserId",

"receiver\_id": "UserId",

"message": "Your message content",

"timestamp": "ISODate"

}

## **4️⃣ API Endpoints (Flask)**

### **Authentication**

* POST /api/auth/signup – Register as NGO or Company.
* POST /api/auth/login – Log in and receive a JWT token.
* GET /api/auth/user – Retrieve current user details.

### **NGO Endpoints**

* GET /api/ngo/<id> – Retrieve NGO profile.
* PUT /api/ngo/<id> – Update NGO profile.
* GET /api/ngo/posts – Fetch posts from followed companies.
* GET /api/ngo/match – Get matching companies (using your Hinge-like algorithm).
* POST /api/ngo/follow/<company\_id> – Follow a company.
* POST /api/ngo/event – Create an event.

### **Company Endpoints**

* GET /api/company/<id> – Retrieve Company profile.
* PUT /api/company/<id> – Update Company profile.
* POST /api/company/post – Create a post.
* GET /api/company/match – Get matching NGOs.
* POST /api/company/follow/<ngo\_id> – Follow an NGO.
* GET /api/company/events – Get list of events (for discovery).

### **Messaging (Using SocketIO)**

* Socket events for joining rooms, sending messages, and receiving real-time chat updates.

## **5️⃣ Matching Algorithm (Hinge-like)**

A simple score-based algorithm can be implemented in Python:

python

CopyEdit

def match\_ngo\_to\_company(ngo):

# Example criteria: matching focus areas, geographical proximity, etc.

# ngo is a dictionary representing the NGO's details.

query = {

"focus\_areas": {"$in": ngo.get("causes", [])},

# You can add more filtering criteria here.

}

matching\_companies = list(db.companies.find(query))

# Optionally, sort or score the results here.

return matching\_companies

Similarly, you can build a matching function for companies looking for NGOs.

## **6️⃣ Running Locally**

* **Flask App:** Use Flask’s built-in development server (flask run or python run.py).
* **MongoDB:** Run your MongoDB instance locally (default URI: mongodb://localhost:27017/yourdbname).
* **React App:** Use npm start from your Create React App directory.
* **Integration:** Ensure your React app points to your Flask backend’s local URL (e.g., http://localhost:5000/api/).

## **7️⃣ Next Steps for Your Hackathon Demo**

1. **Set Up Your Environment:**
   * Install Python, MongoDB, Node.js, and your preferred code editor.
   * Set up a virtual environment for Python.
2. **Develop the Backend:**
   * Start by building user authentication and basic CRUD endpoints.
   * Test endpoints using Postman.
   * Integrate SocketIO for messaging if time permits.
3. **Develop the Frontend:**
   * Build a simple homepage with Sign In/Sign Up options.
   * Create basic layouts for NGO and Company dashboards.
   * Consume your Flask API endpoints to display posts, profiles, and matching data.
4. **Local Testing:**
   * Ensure everything works together locally.
   * Use sample data to populate MongoDB for demo purposes.
5. **Documentation:**
   * Prepare a brief demo script explaining the flow (from signup/login to matching and messaging).