**Project Overview**

Goal: Build a micro-job board where users can post and find local services based on their location. Users can search for jobs within a specific radius from their current location.

**Technical Requirements**

1. **Database**: Store job posts with associated location data (latitude, longitude).
2. **Geolocation:** Capture user coordinates and filter job posts based on proximity.
3. **Backend Logic:** Use Laravel for location-based querying and distance calculation.
4. **Frontend Mapping:** Display jobs on a map using a mapping API (e.g., Google Maps or OpenStreetMap).

**Core Functionalities**

1. **Location-Based Job Search:**

* Users can search for job posts near their current location or any specified location.
* Jobs within a certain radius (e.g., 10 km) are displayed.

1. **Job Posting with Location Data:**

* Users posting a job must enter an address, which is converted to coordinates using a Geocoding API.

1. **Interactive Map Display:**

* Show job locations on a map with pins, allowing users to view job details by clicking on a pin.

**Steps to Plan the Project**

**1. Database Design**

* User Table: For storing user data and preferences.
* **Job Post Table:**
* **id:** Job ID.
* **title:** Job title.
* **description:** Detailed description of the job.
* **latitude & longitude:** Coordinates for the job location.
* **created\_by:** Reference to the user who posted the job.
* **created\_at, updated\_at:** Timestamps.
* A Location Table for regions or cities, which can simplify location queries.

**2. Capture User Location (Frontend)**

* Use JavaScript’s Geolocation API to get the user’s current latitude and longitude upon their consent.
* Once coordinates are captured, display nearby jobs based on the chosen radius.

**3. Convert Addresses to Coordinates (Geocoding)**

* When users post jobs, allow them to enter an address or pinpoint a location on a map.
* Use a Geocoding API (like Google Maps Geocoding or OpenStreetMap’s Nominatim API) to convert addresses into latitude and longitude.
* Save these coordinates in the job post’s database record.

**4. Distance Calculation for Proximity Search (Backend)**

* To find nearby jobs, calculate the distance between the user’s coordinates and each job’s coordinates.
* Use the Haversine formula for calculating distances based on coordinates, which is ideal for this use case.

**5. Searching for Nearby Jobs (Backend)**

* Create an API endpoint in Laravel that accepts:
* **User’s Coordinates** (latitude and longitude).
* **Radius** (e.g., 10 km).
* Query jobs within the radius using the distance formula and return the result as JSON.

**6. Displaying Jobs on a Map (Frontend)**

* Integrate a mapping API (Google Maps or OpenStreetMap) to visually represent job locations.
* For each job in the results, add a marker on the map with the job’s location.
* Enable users to click on markers to see brief job details in a popup or side panel.

**7. Implementing User Interface with Bootstrap**

* **Search Form:** Allow users to enter a radius, view results as cards, and toggle between list and map view.
* **Map and Card View:** Use Bootstrap’s grid system to display the map on one side and a list of jobs on the other.
* **Job Detail Popup:** Show job details on marker click in a Bootstrap modal or custom info window on the map.