hi i want to make a design document for person a (contractor) to person b(worker)

1. where worker can a carpenters ,labour, plumber etc.

2. where contractor can be any one who hire workers.

3.Contracture and worker can find each one.

4. Contracture and worker can get suggestion on location and job type base for adding.

5.worker can add contractures in his contact list

6.contructure can add worker in his contact list.

7.if contractor wants worker then he will active his status for three hours that he required type of worker.

8.if worker is free and wants job then he will active his status that he required a work.

9.Sign in and Sign out are required.

10. as a backend we can use express typescript or Nestjs with PostgreSQL and pgadmin.

11. we will make separate backend and frontend for each.

12.logger and error handler required.

13. Contracture and worker minimal details are required for profile.

14. app should be simpler .

15.we will use solid principal .  
16.they can contact or call each others.  
17. Can we choose a froentend which can work in web app and android also.

can you generate a design document so that i can make a MicroService after read and analysis this document.

Design Document  
Got it ✅ You want a **design document** for a contractor–worker platform, based on the requirements you listed. I’ll create a structured document that you can use as a foundation for analyzing and later implementing as **microservices**.

**Design Document – Contractor & Worker Matching Platform**

**1. Introduction**

This platform connects **Contractors** (individuals or companies who hire) with **Workers** (e.g., carpenters, plumbers, laborers).  
Both parties can:

* Discover each other.
* Maintain contacts.
* Share availability status.
* Communicate directly.

The system will have **separate backend and frontend applications**, designed to be **scalable with microservices**.

**2. Objectives**

* Enable **contractor-worker matching** based on location, job type, and availability.
* Provide a **simple, lightweight app** for both sides.
* Support **cross-platform frontend** (web + mobile).
* Ensure **scalable backend** with logging, error handling, and modular design.

**3. Users & Roles**

**Contractor**

* Can search/find workers.
* Can add workers to their contact list.
* Can activate "need worker" status for 3 hours.
* Can call or message workers directly.

**Worker**

* Can search/find contractors.
* Can add contractors to their contact list.
* Can activate "need job" status.
* Can call or message contractors directly.

**4. Core Features**

1. **Authentication**
   * Sign up / Sign in / Sign out.
   * JWT-based sessions.
2. **Profiles (minimal details required)**
   * Contractor: name, contact info, company (optional).
   * Worker: name, skill type (carpenter/plumber/etc.), contact info, availability.
3. **Discovery & Suggestions**
   * Match contractors and workers by:
     + Location (GPS or city-based).
     + Job type (skill category).
4. **Contacts**
   * Contractors can save workers.
   * Workers can save contractors.
5. **Availability Status**
   * Contractor: "Looking for worker" (active for 3 hours).
   * Worker: "Looking for job" (active until manually disabled).
6. **Communication**
   * In-app chat or call option.
7. **Logging & Error Handling**
   * Centralized logging service.
   * Error handler middleware.

**5. System Architecture**

We adopt **microservice-based architecture** with **Domain-Driven Design** and **SOLID principles**.

**Tech Stack**

* **Backend:**
  + Express.js (TypeScript) or NestJS
  + PostgreSQL + pgAdmin
  + REST APIs (possibly extend to GraphQL later)
* **Frontend:**
  + **React Native** → works for both Web + Android (Expo can help in development).
* **Other Tools:**
  + Docker for containerization.
  + Nginx / API Gateway.
  + Redis (optional for session caching).

**Microservices (suggested separation)**

1. **Auth Service**
   * Handles sign in/out, JWT, role management.
2. **User Service**
   * Profile management (contractor/worker).
   * Contacts management.
3. **Matching Service**
   * Location & job-based recommendations.
   * Availability status tracking.
4. **Communication Service**
   * Messaging & call initiation (possibly integrate Twilio/VoIP).
5. **Notification Service**
   * Push notifications, job requests, reminders.
6. **Logging & Error Handling Service**
   * Centralized logs (e.g., Winston + ELK stack).

**6. Database Design (PostgreSQL)**

**Tables**

* **users**
  + id (PK)
  + role (contractor/worker)
  + name
  + phone/email
  + password\_hash
  + location
  + created\_at, updated\_at
* **worker\_profiles**
  + id (PK, FK → users.id)
  + skill\_type
  + experience\_years
  + availability\_status
* **contractor\_profiles**
  + id (PK, FK → users.id)
  + company\_name (nullable)
  + need\_worker\_status
* **contacts**
  + id (PK)
  + owner\_id (FK → users.id)
  + contact\_user\_id (FK → users.id)
* **jobs**
  + id (PK)
  + contractor\_id (FK → users.id)
  + worker\_id (FK → users.id)
  + job\_type
  + status (requested/accepted/completed)
  + created\_at
* **logs**
  + id
  + service
  + level (info/warn/error)
  + message
  + timestamp

**7. API Endpoints (Sample)**

* **Auth Service**
  + POST /auth/signup
  + POST /auth/login
  + POST /auth/logout
* **User Service**
  + GET /users/:id
  + PUT /users/:id
  + GET /users?role=worker&skill=plumber&location=delhi
* **Contacts Service**
  + POST /contacts/add
  + GET /contacts/list
* **Matching Service**
  + GET /match/suggestions?role=worker&location=xyz
* **Availability Service**
  + POST /status/activate
  + POST /status/deactivate

**8. Non-Functional Requirements**

* **Scalability:** Microservices with Docker + Kubernetes.
* **Security:** JWT auth, hashed passwords (bcrypt/argon2).
* **Performance:** Caching with Redis.
* **Reliability:** Graceful error handling + logging.
* **Simplicity:** Minimal UI/UX for fast adoption.

**9. Future Enhancements**

* Ratings & reviews for workers/contractors.
* Payment gateway for contract work.
* AI-based matching for better recommendations.
* Offline mode (job postings saved until internet resumes).

✅ This design document gives you the **high-level system design**, database schema, microservices breakdown, and technical stack.  
From here, you can break it down into **microservice-specific documents** (Auth, User, Matching, etc.) before implementation.

Would you like me to also create a **sequence diagram** (e.g., how a contractor finds and hires a worker) so it’s easier to visualize workflow?