TECHNICAL MANUAL & IMPLEMENTATION GUIDE

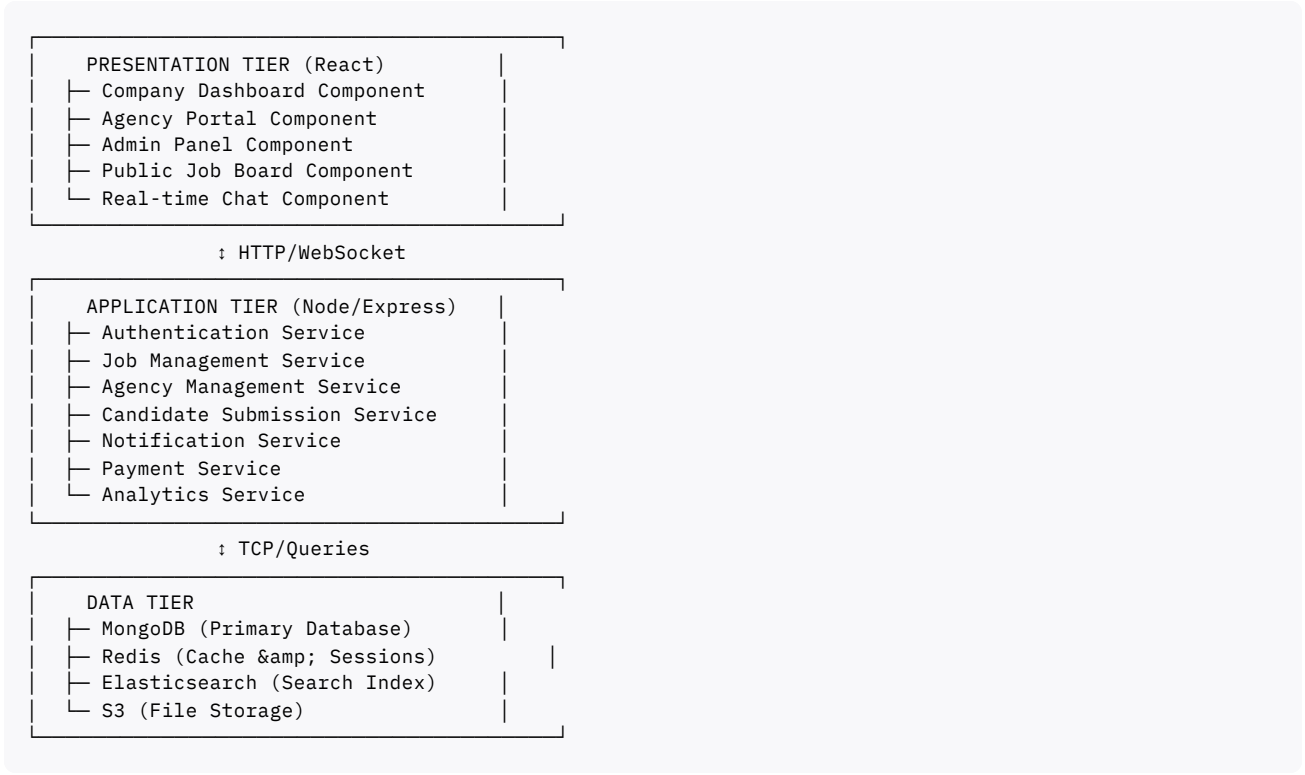
Table of Contents

Vendor Management System + Job Portal Hybrid Platform

Table of Contents

1. System Architecture Deep Dive

1.1 Three-Tier Architecture



1.2 Microservices Architecture (Future Scalability)

Monolithic → Microservices Evolution:

Phase 1: Monolithic API (Fast deployment)
Phase 2: Extract job & agency services
Phase 3: Extract payment & notification services
Phase 4: Full microservices with event-driven architecture

Core Services:

- **User Service:** Authentication, profiles, roles
- **Job Service:** CRUD operations on vacancies
- **Agency Service:** Vendor management and onboarding
- **Submission Service:** Candidate submissions and tracking
- **Notification Service:** Email, SMS, push notifications
- **Payment Service:** Commission calculation and processing
- **Analytics Service:** KPI generation and reporting

2. Database Design & Schemas

2.1 Collections Structure (MongoDB)

Users Collection

```
{
  "_id": ObjectId,
  "email": "user@company.com",
  "phone": "+919999999999",
```

```
  "passwordHash": "bcrypt_hash",
  "role": ["CLIENT", "AGENCY", "ADMIN"],
  "firstName": "John",
  "lastName": "Doe",
  "avatar": "url_to_avatar",
  "status": "ACTIVE|INACTIVE|SUSPENDED",
  "twoFactorEnabled": true,
  "twoFactorSecret": "secret",
  "lastLogin": ISODate,
  "createdAt": ISODate,
  "updatedAt": ISODate,
  "deletedAt": ISODate
}
```

Companies Collection

```
{
  "_id": ObjectId,
  "userId": ObjectId,
  "companyName": "Tech Corp Inc.",
  "industry": "IT/Software",
  "companySize": "1001-5000",
  "location": "Mumbai",
  "country": "India",
  "website": "www.techcorp.com",
  "gstNumber": "18ABCDE1234F2Z0",
  "logo": "url_to_logo",
  "verificationStatus": "APPROVED|PENDING|REJECTED",
  "verificationDocuments": [
    { "type": "GST_CERTIFICATE", "url": "..." }
  ],
  "subscriptionPlan": "PROFESSIONAL",
  "subscriptionEndDate": ISODate,
  "annualBudget": 500000,
  "primaryContact": "hr@techcorp.com",
  "apiKey": "unique_api_key_for_integrations",
  "createdAt": ISODate
}
```

Agencies Collection

```
{
  "_id": ObjectId,
  "userId": ObjectId,
  "agencyName": "Talent Hunters Pvt Ltd",
  "gstNumber": "27ABCDE5678F1Z0",
  "panNumber": "ABCDE1234F",
  "registrationNumber": "U74110MH2020...",
  "specializations": ["IT", "HR", "Finance"],
  "location": "Bangalore",
  "country": "India",
  "phone": "+919876543210",
  "officeAddress": "123 Tech Park...",
  "kycStatus": "APPROVED|PENDING|REJECTED",
  "kycDocuments": [
    { "type": "GST_CERT", "url": "..." },
    { "type": "PAN_CERT", "url": "..." },
    { "type": "REG_CERT", "url": "..." }
  ],
  "bankDetails": {
    "accountName": "...",
    "accountNumber": "...",
    "bankName": "...",
    "ifsc": "..."
  },
  "approvalDate": ISODate,
  "performanceRating": 4.5,
  "totalPlacements": 150,
  "successRate": 0.65,
```

```
  "createdAt": ISODate
}
```

Vacancies Collection

```
{
  "_id": ObjectId,
  "companyId": ObjectId,
  "jobTitle": "Senior Software Engineer",
  "description": "We are hiring...",
  "requirements": [
    "5+ years experience",
    "Node.js, React expertise",
    "System design knowledge"
  ],
  "skills": ["JavaScript", "Node.js", "React", "MongoDB"],
  "salaryMin": 1200000,
  "salaryMax": 1800000,
  "location": "Mumbai",
  "employmentType": "FULL_TIME|CONTRACT",
  "urgencyLevel": "NORMAL|HIGH|CRITICAL",
  "postingDate": ISODate,
  "deadline": ISODate,
  "anonymousPosting": true,
  "status": "ACTIVE|CLOSED|ON_HOLD",
  "assignedAgencies": [ObjectId, ObjectId],
  "totalSubmissions": 25,
  "totalShortlisted": 8,
  "createdAt": ISODate,
  "updatedAt": ISODate
}
```

Candidates Collection

```
{
  "_id": ObjectId,
  "firstName": "Raj",
  "lastName": "Patel",
  "email": "raj.patel@email.com",
  "phone": "+918765432109",
  "location": "Pune",
  "currentJobTitle": "Software Developer",
  "currentCompany": "XYZ Solutions",
  "yearsOfExperience": 6,
  "skills": ["JavaScript", "Python", "AWS"],
  "resume": "s3_url_to_resume",
  "profileScore": 85,
  "source": "AGENCY|DIRECT_APPLY|IMPORTED",
  "sourceAgencyId": ObjectId,
  "availability": "IMMEDIATE|2_WEEKS|1_MONTH",
  "noticePeriod": "15 days",
  "expectedSalary": 1500000,
  "openToRelocation": false,
  "createdAt": ISODate
}
```

Submissions Collection

```
{
  "_id": ObjectId,
  "vacancyId": ObjectId,
  "agencyId": ObjectId,
  "candidateId": ObjectId,
  "status": "SUBMITTED|UNDER_REVIEW|SHORTLISTED|INTERVIEWED|OFFERED|REJECTED",
  "submittedDate": ISODate,
  "remarks": "Strong candidate with relevant experience",
  "reason": "Not meeting salary expectations",
  "companyFeedback": "Great fit, proceeding to interview",
```

```
  "candidateRating": 4.5,
  "agencyRating": 5,
  "interviewDate": ISODate,
  "offerDate": ISODate,
  "joiningDate": ISODate,
  "outcome": "PLACED|NOT_PLACED|ONGOING",
  "updatedAt": ISODate
}
```

Ratings Collection

```
{
  "_id": ObjectId,
  "raterId": ObjectId,
  "raterRole": "COMPANY|AGENCY|CANDIDATE",
  "rateeId": ObjectId,
  "rateeRole": "AGENCY|COMPANY|CANDIDATE",
  "score": 5,
  "category": "QUALITY|SPEED|PROFESSIONALISM|COMMUNICATION",
  "feedback": "Excellent work, will definitely recommend",
  "submissionId": ObjectId,
  "createdAt": ISODate
}
```

2.2 Indexing Strategy

```
// Users collection
db.users.createIndex({ email: 1 }, { unique: true });
db.users.createIndex({ phone: 1 });
db.users.createIndex({ role: 1 });

// Vacancies collection
db.vacancies.createIndex({ companyId: 1, status: 1 });
db.vacancies.createIndex({ skills: 1 });
db.vacancies.createIndex({ location: 1 });
db.vacancies.createIndex({ urgencyLevel: 1 });
db.vacancies.createIndex({ createdAt: -1 });
db.vacancies.createIndex({ "text": "text" },
  { default_language: "english" });

// Submissions collection
db.submissions.createIndex({ vacancyId: 1, status: 1 });
db.submissions.createIndex({ agencyId: 1 });
db.submissions.createIndex({ candidateId: 1 });
db.submissions.createIndex({ submittedDate: -1 });
```

3. API Endpoints & Integration Guide

3.1 Authentication Endpoints

```
POST   /api/v1/auth/register
       Body: { email, password, role, firstName, lastName }
       Response: { userId, token, refreshToken }

POST   /api/v1/auth/login
       Body: { email, password }
       Response: { userId, token, refreshToken, role }

POST   /api/v1/auth/refresh
       Body: { refreshToken }
       Response: { token }

POST   /api/v1/auth/logout
       Response: { success: true }
```

```
POST   /api/v1/auth/two-factor/enable
       Response: { qrCode, secret }

POST   /api/v1/auth/two-factor/verify
       Body: { code }
       Response: { success: true }
```

3.2 Job Management Endpoints

```
POST   /api/v1/jobs
       Headers: { Authorization: "Bearer token" }
       Body: { title, description, skills, salary, location, urgency }
       Response: { jobId, status: "ACTIVE" }

GET    /api/v1/jobs/:jobId
       Response: { job details }

GET    /api/v1/jobs?location=Mumbai&amp;amp;skills=Node.js
       Response: [jobs array]

PUT    /api/v1/jobs/:jobId
       Body: { updates }
       Response: { updated job }

DELETE /api/v1/jobs/:jobId
       Response: { success: true }

POST   /api/v1/jobs/:jobId/assign-agencies
       Body: { agencyIds: [ObjectId] }
       Response: { assigned: true }

GET    /api/v1/jobs/:jobId/submissions
       Response: [submissions for this job]
```

3.3 Agency Endpoints

```
POST   /api/v1/agencies/register
       Body: { agencyName, gstNumber, phone, location }
       Response: { agencyId, kycStatus: "PENDING" }

GET    /api/v1/agencies/:agencyId
       Response: { agency details, performance metrics }

GET    /api/v1/agencies?specialization=IT&amp;amp;location=Mumbai
       Response: [agencies array]

PUT    /api/v1/agencies/:agencyId/kyc
       Body: FormData with documents
       Response: { kycStatus: "SUBMITTED" }

POST   /api/v1/agencies/:agencyId/submit-candidate
       Body: { jobId, candidate: { name, email, resume } }
       Response: { submissionId }

GET    /api/v1/agencies/:agencyId/submissions
       Response: [all submissions from this agency]

GET    /api/v1/agencies/:agencyId/performance
       Response: { totalPlacements, successRate, rating }
```

3.4 Candidate Endpoints

```
POST   /api/v1/candidates
       Body: { firstName, lastName, email, skills, yearsOfExp }
       Response: { candidateId }

GET    /api/v1/candidates/:candidateId
       Response: { candidate profile }

POST   /api/v1/candidates/:candidateId/apply
       Body: { jobId }
       Response: { applicationId }

GET    /api/v1/candidates/matching-jobs?skills=Node.js
       Response: [matching jobs]
```

3.5 Submission Endpoints

```
POST   /api/v1/submissions
       Body: { vacancyId, agencyId, candidateId, remarks }
       Response: { submissionId, status: "SUBMITTED" }

PUT    /api/v1/submissions/:submissionId/status
       Body: { status: "SHORTLISTED|REJECTED", feedback: "..." }
       Response: { submission with new status }

PUT    /api/v1/submissions/:submissionId/rate
       Body: { score: 5, feedback: "..." }
       Response: { updated submission }

GET    /api/v1/submissions?vacancyId=ObjectId
       Response: [submissions array]
```

3.6 Notification Endpoints

```
GET    /api/v1/notifications
       Response: [{ id, type, message, read, createdAt }]

POST   /api/v1/notifications/:notificationId/read
       Response: { success: true }

POST   /api/v1/notifications/subscribe
       Body: { deviceToken, endpoint }
       Response: { subscriptionId }
```

3.7 Payment Endpoints

```
POST   /api/v1/payments/process
       Body: { agencyId, submissionId, amount }
       Response: { paymentId, status: "PENDING" }

GET    /api/v1/payments/:paymentId/status
       Response: { status: "COMPLETED|FAILED", transactionId }

GET    /api/v1/payments/agency/:agencyId
       Response: [payment history]

POST   /api/v1/payments/calculate-commission
       Body: { salaryAmount, isUrgent, agencyRating }
       Response: { commission, totalAmount }
```

4. Frontend Implementation

4.1 Project Structure

```
frontend/
├── public/
│   ├── index.html
│   └── favicon.ico
├── src/
│   ├── components/
│   │   ├── Company/
│   │   │   ├── Dashboard.jsx
│   │   │   ├── PostJob.jsx
│   │   │   ├── CandidateList.jsx
│   │   │   └── ChatWindow.jsx
│   │   ├── Agency/
│   │   │   ├── Dashboard.jsx
│   │   │   ├── AvailableJobs.jsx
│   │   │   ├── SubmitCandidate.jsx
│   │   │   └── Performance.jsx
│   │   ├── Admin/
│   │   │   ├── Dashboard.jsx
│   │   │   ├── UserManagement.jsx
│   │   │   ├── Analytics.jsx
│   │   │   └── PaymentManagement.jsx
│   │   └── Common/
│   │       ├── Header.jsx
│   │       ├── Sidebar.jsx
│   │       └── Login.jsx
│   ├── services/
│   │   ├── api.js (Axios instance)
│   │   ├── jobService.js
│   │   ├── agencyService.js
│   │   └── authService.js
│   ├── store/
│   │   ├── actions/
│   │   ├── reducers/
│   │   └── index.js (Redux setup)
│   ├── hooks/
│   │   ├── useAuth.js
│   │   ├── useFetch.js
│   │   └── useForm.js
│   ├── styles/
│   │   ├── global.css
│   │   └── theme.js
│   ├── App.jsx
│   └── index.js
├── package.json
└── .env
```

4.2 Key Frontend Libraries

```
{
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.8.0",
    "axios": "^1.3.0",
    "redux": "^4.2.0",
    "react-redux": "^8.0.0",
    "@reduxjs/toolkit": "^1.9.0",
    "socket.io-client": "^4.5.0",
    "@mui/material": "^5.11.0",
    "react-hook-form": "^7.43.0",
    "yup": "^1.0.0",
    "date-fns": "^2.29.0"
  }
}
```

4.3 Sample React Component

```
// components/Company/PostJob.jsx
import React, { useState } from 'react';
import { useForm } from 'react-hook-form';
import { yupResolver } from '@hookform/resolvers/yup';
import * as yup from 'yup';
import jobService from '../../services/jobService';

const schema = yup.object({
  jobTitle: yup.string().required('Job title is required'),
  description: yup.string().min(50).required(),
  skills: yup.array().min(1),
  salaryMin: yup.number().required(),
  location: yup.string().required(),
});

const PostJob = () =&gt; {
  const { register, handleSubmit, formState: { errors } } =
    useForm({ resolver: yupResolver(schema) });
  const [loading, setLoading] = useState(false);

  const onSubmit = async (data) =&gt; {
    setLoading(true);
    try {
      const response = await jobService.createJob(data);
      console.log('Job posted:', response);
      // Show success message
    } catch (error) {
      console.error('Error posting job:', error);
      // Show error message
    } finally {
      setLoading(false);
    }
  };

  return (
    <div>
      <h2>Post New Job</h2>
      &lt;form onSubmit={handleSubmit(onSubmit)}&gt;
        &lt;input
          {...register('jobTitle')}
          placeholder="Job Title"
        /&gt;
        {errors.jobTitle &amp;&amp; <span>{errors.jobTitle.message}</span>}

        &lt;textarea
          {...register('description')}
          placeholder="Job Description"
        /&gt;

        &lt;button type="submit" disabled={loading}&gt;
          {loading ? 'Posting...' : 'Post Job'}
        &lt;/button&gt;
      &lt;/form&gt;
    </div>
  );
};

export default PostJob;
```

5. Backend Implementation

5.1 Project Structure

```
backend/
├── config/
│   ├── database.js (MongoDB connection)
│   ├── jwt.js (JWT configuration)
│   ├── payment.js (Stripe config)
│   └── email.js (Email service)
├── models/
│   ├── User.js
│   ├── Company.js
│   ├── Agency.js
│   ├── Vacancy.js
│   ├── Candidate.js
│   ├── Submission.js
│   └── Rating.js
├── routes/
│   ├── auth.js
│   ├── jobs.js
│   ├── agencies.js
│   ├── submissions.js
│   ├── payments.js
│   └── admin.js
├── controllers/
│   ├── authController.js
│   ├── jobController.js
│   ├── agencyController.js
│   ├── submissionController.js
│   └── paymentController.js
├── services/
│   ├── emailService.js
│   ├── notificationService.js
│   ├── paymentService.js
│   └── analyticsService.js
├── middleware/
│   ├── auth.js (JWT verification)
│   ├── rbac.js (Role-based access)
│   ├── errorHandler.js
│   └── logger.js
├── utils/
│   ├── validators.js
│   ├── helpers.js
│   └── constants.js
├── app.js
├── server.js
├── package.json
└── .env
```

5.2 Backend Key Libraries

```
{
  "dependencies": {
    "express": "^4.18.0",
    "mongoose": "^7.0.0",
    "dotenv": "^16.0.0",
    "bcryptjs": "^2.4.0",
    "jsonwebtoken": "^9.0.0",
    "cors": "^2.8.0",
    "helmet": "^7.0.0",
    "express-rate-limit": "^6.7.0",
    "multer": "^1.4.0",
    "aws-sdk": "^2.1300.0",
    "nodemailer": "^6.8.0",
    "stripe": "^11.0.0",
    "socket.io": "^4.5.0",
    "redis": "^4.5.0",
```

```
    "elasticsearch": "^7.16.0"
  }
}
```

5.3 Sample Express Controller

```
// controllers/jobController.js
const Job = require('../models/Vacancy');
const Agency = require('../models/Agency');
const notificationService = require('../services/notificationService');

exports.createJob = async (req, res) =&gt; {
  try {
    const { jobTitle, description, skills, salaryMin, location, urgency } = req.body;
    const companyId = req.user.companyId;

    // Validate input
    if (!jobTitle || !description) {
      return res.status(400).json({ error: 'Missing required fields' });
    }

    // Create job
    const newJob = new Job({
      companyId,
      jobTitle,
      description,
      skills,
      salaryMin,
      location,
      urgencyLevel: urgency || 'NORMAL',
      status: 'ACTIVE',
    });

    await newJob.save();

    // If urgency is high, notify top agencies
    if (urgency === 'HIGH' || urgency === 'CRITICAL') {
      const topAgencies = await Agency.find({
        specializations: { $in: skills },
        location: location,
        kycStatus: 'APPROVED',
      }).sort({ performanceRating: -1 }).limit(10);

      for (let agency of topAgencies) {
        await notificationService.sendNotification(
          agency.userId,
          `New ${urgency} job opportunity: ${jobTitle}`,
          'NEW_JOB',
          { jobId: newJob._id }
        );
      }
    }

    res.status(201).json({
      success: true,
      jobId: newJob._id,
      message: 'Job posted successfully',
    });
  } catch (error) {
    console.error('Error creating job:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
};

exports.getJobs = async (req, res) =&gt; {
  try {
    const { location, skills, urgency, page = 1, limit = 10 } = req.query;

    const filter = { status: 'ACTIVE' };
    if (location) filter.location = location;
```

```
    if (urgency) filter.urgencyLevel = urgency;
    if (skills) filter.skills = { $in: skills.split(',') };

    const jobs = await Job.find(filter)
      .skip((page - 1) * limit)
      .limit(limit)
      .populate('companyId', 'companyName logo');

    const total = await Job.countDocuments(filter);

    res.json({
      success: true,
      data: jobs,
      pagination: { page, limit, total },
    });
  } catch (error) {
    res.status(500).json({ error: 'Internal server error' });
  }
};
```

6. Security Implementation

6.1 Authentication & Authorization

```
// middleware/auth.js
const jwt = require('jsonwebtoken');
const User = require('../models/User');

exports.authenticateToken = async (req, res, next) =&gt; {
  try {
    const authHeader = req.headers['authorization'];
    const token = authHeader &amp;&amp; authHeader.split(' ')[^1];

    if (!token) {
      return res.status(401).json({ error: 'Token not provided' });
    }

    jwt.verify(token, process.env.JWT_SECRET, async (err, user) =&gt; {
      if (err) return res.status(403).json({ error: 'Invalid token' });

      const dbUser = await User.findById(user.id);
      if (!dbUser || dbUser.status === 'SUSPENDED') {
        return res.status(403).json({ error: 'User not found or suspended' });
      }

      req.user = dbUser;
      next();
    });
  } catch (error) {
    res.status(500).json({ error: 'Authentication error' });
  }
};

exports.authorize = (...roles) =&gt; {
  return (req, res, next) =&gt; {
    if (!roles.includes(req.user.role)) {
      return res.status(403).json({ error: 'Insufficient permissions' });
    }
    next();
  };
};
```

6.2 Password Hashing

```
// Before saving user
const bcrypt = require('bcryptjs');

const salt = await bcrypt.genSalt(10);
user.passwordHash = await bcrypt.hash(password, salt);
await user.save();

// During login
const isMatch = await bcrypt.compare(password, user.passwordHash);
```

6.3 Environment Variables

```
# .env<a></a>
NODE_ENV=production
PORT=5000

# Database<a></a>
MONGODB_URI=mongodb+srv://username:password@cluster.mongodb.net/dbname

# JWT<a></a>
JWT_SECRET=your_super_secret_key_here
JWT_EXPIRE=7d

# AWS S3<a></a>
AWS_ACCESS_KEY_ID=your_access_key
AWS_SECRET_ACCESS_KEY=your_secret_key
AWS_REGION=ap-south-1
AWS_S3_BUCKET=your_bucket_name

# Email Service<a></a>
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=your_email@gmail.com
SMTP_PASS=your_app_password

# Payment<a></a>
STRIPE_SECRET_KEY=sk_live_...
STRIPE_PUBLISHABLE_KEY=pk_live_...

# Redis<a></a>
REDIS_URL=redis://localhost:6379

# Elasticsearch<a></a>
ELASTICSEARCH_URL=http://localhost:9200
```

7. Deployment & DevOps

7.1 Docker Setup

```
# Dockerfile (Backend)<a></a>
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./
RUN npm install

COPY . .

EXPOSE 5000

CMD ["npm", "start"]
```

```
# Dockerfile (Frontend)<a></a>
FROM node:18-alpine as build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

7.2 Docker Compose

```
# docker-compose.yml<a></a>
version: '3.8'

services:
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    ports:
      - "5000:5000"
    environment:
      - MONGODB_URI=mongodb://mongo:27017/vms
      - REDIS_URL=redis://redis:6379
    depends_on:
      - mongo
      - redis

  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    ports:
      - "3000:80"
    depends_on:
      - backend

  mongo:
    image: mongo:6.0
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"

volumes:
  mongo_data:
```

7.3 CI/CD Pipeline (GitHub Actions)

```
# .github/workflows/deploy.yml<a></a>
name: Deploy to Production

on:
  push:
    branches: [main]

jobs:
```

```
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v2
        with:
          node-version: '18'
      - run: npm install
      - run: npm run test
      - run: npm run lint

  build:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - run: docker build -t vms-backend:latest ./backend
      - run: docker push ${{ secrets.DOCKER_REGISTRY }}/vms-backend:latest

  deploy:
    needs: build
    runs-on: ubuntu-latest
    steps:
      - name: Deploy to AWS
        run: |
          aws elasticbeanstalk create-environment-version \
            --region ap-south-1
```

8. Troubleshooting & Support

8.1 Common Issues & Solutions

Issue 1: Database connection timeout

- Solution: Check MongoDB connection string, firewall rules, IP whitelisting

Issue 2: JWT token expired

- Solution: Implement token refresh mechanism, use `refreshToken`

Issue 3: CORS errors

- Solution: Configure CORS middleware with allowed origins

Issue 4: File upload failures

- Solution: Check S3 bucket permissions, file size limits, allowed MIME types

Issue 5: Email delivery failed

- Solution: Verify SMTP credentials, enable less secure apps

8.2 Performance Optimization

Database Queries:

- Use indexes for frequently filtered fields
- Implement pagination for large datasets
- Use projection to fetch only required fields

API Response:

- Compress responses (gzip)
- Implement rate limiting
- Use Redis caching for frequently accessed data

Frontend:

- Code splitting with React.lazy()
- Image optimization and lazy loading
- Minification and bundling

Conclusion

This technical manual provides comprehensive implementation guidance for the VMS hybrid platform. Follow the architecture, database design, and API patterns outlined here to build a scalable, secure, and efficient recruitment platform.

Support Resources:

- GitHub Repository: [your-repo-url]
- Documentation: [your-docs-url]
- Issue Tracker: [your-issues-url]
  [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19]

⁂

1. https://www.semanticscholar.org/paper/2df4d53762ee49a3b91b581397e75072d668c9d1
2. https://aquentscout.com/blog/what-is-a-recruitment-marketplace-and-why-its-valuable/
3. https://www.yo-kart.com/blog/multi-vendor-marketplace-development-guide/
4. https://miracuves.com/blog/job-portal-clone-revenue-model/
5. https://simtechdev.com/blog/how-to-build-a-multi-vendor-marketplace/
6. https://www.magnaquest.com/the-modern-subscription-business-model-for-innovative-hiring-portals/
7. https://www.sparxitsolutions.com/blog/multi-vendor-ecommerce-marketplace-development-guide/
8. https://www.ismartrecruit.com/blogs/recruitment-marketplace
9. https://www.sharetribe.com/how-to-build/multi-vendor-marketplace/
10. https://nyusoft.com/how-white-label-passive-job-portals-can-generate-new-business-opportunities/
11. https://arxiv.org/abs/2502.05090
12. https://support.sas.com/resources/papers/proceedings-archive/SUGI95/Sugi-95-222 Cates.pdf
13. https://ieeexplore.ieee.org/document/11208313/
14. https://rgs-ibg.onlinelibrary.wiley.com/doi/10.1111/area.70039
15. https://aacrjournals.org/cancerres/article/85/8_Supplement_1/6309/757951/Abstract-6309-CliPP-on-Web-a-dynamic-platform-for
16. https://aacrjournals.org/mct/article/24/10_Supplement/C102/766537/Abstract-C102-MCATT-A-3D-cancer-functional
17. https://arxiv.org/abs/2507.13575
18. https://aacrjournals.org/cancerres/article/85/8_Supplement_1/7463/759625/Abstract-7463-A-linker-platform-for-antibody-drug
19. https://www.mdpi.com/2076-3417/15/22/11882