💼

Job Portal System — Technical Documentation

Purpose:

This document serves as the technical reference for submission, covering:

• System Architecture
• Application Flow
• API Specification
• File-Level Responsibilities

For setup and run instructions, refer to README.md.

1️⃣ System Overview

1.1 Architecture

Layer	Description
Frontend	Flutter application (Web/Mobile) consuming REST APIs. Single-page flow with role-based routes.
Backend	Node.js + Express REST API, MongoDB (Mongoose), JWT Authentication, Role-based Authorization Middleware
Database	MongoDB
Authentication	JWT (Bearer Token)
Real-time	Socket.io for live notifications & updates

1.2 User Roles

👩‍💼 **Candidate**
• Login via /login
• Can register
• Access candidate dashboard
• Apply to jobs
• Track application status

🏢 **Recruiter**
• Login via /login
• Can register
• Access recruiter dashboard
• Post jobs
• Review applications

👨‍💼 **Admin (Optional)**
• Login via /admin-login
• No public registration
• Manage users & jobs

2️⃣ Technology Stack & Libraries

2.1 Backend (Node.js + Express)

Core Packages

Package	Purpose
express	HTTP server, routing
mongoose	MongoDB ODM
jsonwebtoken	JWT generation & verification
bcryptjs	Password hashing
cors	Cross-origin requests
dotenv	Environment variable loading

multer	File upload handling
cloudinary	Cloud file storage
socket.io	Real-time communication

Entry Point

backend/index.js

Responsibilities:

• Load environment variables
• Connect to MongoDB
• Mount route modules under /api/*
• Start server on PORT (default: 8000)

2.2 Frontend (Flutter + Dart)

Built using:

• Flutter Framework
• Dart Language
• Provider for state management
• GoRouter for routing

Main Packages

Package	Purpose
provider	Global state (AuthProvider)
go_router	Declarative routing
http	API requests
intl	Date & number formatting
shared_preferences	Persistent local storage

Entry File

frontend/lib/main.dart

Responsibilities:

• Wrap app with ChangeNotifierProvider
• Configure router
• Set theme
• Initial route: /login

---

3 Application Flow

3.1 Backend Startup Flow

index.js
→ connectDB()
→ mount routes
→ app.listen(PORT)

3.2 Frontend Startup Flow

main()
→ JobPortalApp
→ Provider initialized
→ Router created
→ Redirect to /login

---

4 Authentication Flow

4.1 Candidate / Recruiter

1. Login → POST /api/auth/login
2. Store JWT token
3. Store profile

4. Redirect to role dashboard

If email not found:
→ Redirect to /register
→ POST /api/auth/register
→ Redirect after success

---

5 Route Guards

Each screen checks:
auth.isLoggedIn && auth.user.role == 'candidate' or 'recruiter'

If not:
→ Redirect /login

Backend middleware verifies JWT before protected routes.

---

6 Backend Structure

6.1 Configuration

index.js – Bootstrap application
config/db.js – MongoDB connection
config/cloudinary.js – Cloudinary setup

6.2 Middleware

middleware/auth.js – JWT verification + role protection
middleware/upload.js – Multer config for uploads

6.3 Models

Core Identity

User

Domain Models

Job
Application
Post
Connection

---

7 API Overview

Base URL:
http://localhost:8000/api

Authenticated routes require:
Authorization: Bearer <token>

7.1 Auth APIs

GET /auth/check-email – Check email existence
POST /auth/register – Register user
POST /auth/login – Login
GET /auth/me – Get current user

7.2 Job APIs

POST /jobs – Create job
GET /jobs – Get all jobs
PUT /jobs/:id – Update job
DELETE /jobs/:id – Delete job

7.3 Application APIs

POST /applications – Apply for job
GET /applications – View applications

---

8 Frontend Architecture

8.1 Core Layer

core/config/api_config.dart – Base API URL
core/theme/app_theme.dart – App theme
core/models/user_model.dart – User model
core/providers/auth_provider.dart – Auth state management
core/services/api_service.dart – API calls
core/router/app_router.dart – Routing config

8.2 Feature Modules

Auth

login_screen
register_screen

Candidate

candidate_dashboard
job_detail_screen
my_applications_screen

Recruiter

recruiter_dashboard
create_job_screen
job_applications_screen
analytics_screen

9️⃣ Data Flow Summary

1. User logs in
2. Token stored in AuthProvider
3. ApiService attaches Bearer token
4. Backend validates JWT
5. Controller processes request
6. JSON returned
7. UI updates

Logout:
Clear token
Redirect to login

🔟 Environment Configuration

Backend .env

Must include:
MONGO_URI=
JWT_SECRET=
PORT=8000
CLOUDINARY_*

Frontend

Update:
lib/core/config/api_config.dart

Set baseUrl to production server when deployed.

1️⃣1️⃣ Health Check

GET /api/health

Response:
{ ok: true }

📌 Conclusion

This documentation provides a complete architectural and functional overview of the Job Portal System.

It is intended for:
• Academic Submission
• Technical Evaluation
• Developer Reference