# Colbin Recruitment Platform Prototype

*Full-stack Node/React/Tailwind & MongoDB implementation*

---

## 1. Architecture

| Layer | Tech / Responsibility |
|---|---|
| **Client** | React 18 (Vite) + React-Router-Dom 6 + Tailwind CSS |
| **API Gateway** | Express 4 (Node 18) – REST JSON endpoints |
| **Auth** | JWT (30 day expiry) via `Authorization: Bearer …` header |
| **Data** | MongoDB (Mongoose 7) – `User`, `Job`, `Application`, `Interview` collections |
| **Dev Tooling** | Nodemon, ESLint hints, Vite live-reload |

All requests flow through Express middleware (CORS, Helmet, morgan logger, `express-validator`, custom error handler). Front-end communicates via Axios with a central `AuthContext` that persists tokens in `localStorage`.

---

## 2. Database Schema (Mongoose)

```
User { name, email*, passwordHash, createdAt }
Job  { title, description, location, salaryRange, status, createdBy→User, createdAt }
Application { job→Job, applicantName, applicantEmail, resumeUrl, status, createdAt }
Interview { application→Application, scheduledFor, status, createdAt }
```

*`email` is unique; passwords are salted/hashed with bcrypt.*

All schemas include timestamps and validation rules (min/max, regex, enums). Relational references (`ObjectId`) enable efficient population queries.

---

## 3. Key API End-points

| Method & Path | Purpose | Auth |
|---|---|---|
| `POST /api/auth/register` | Create account (name, email, password) | Public |
| `POST /api/auth/login` | Obtain JWT token | Public |
| `GET /api/auth/profile` | Current user profile | Bearer |
| `GET /api/jobs` | List open jobs | Public |

| Method & Path | Purpose | Auth |
| --- | --- | --- |
| `POST /api/jobs` | Create job | Bearer |
| `GET /api/jobs/:id` | Job details | Public |
| `POST /api/jobs/:id/apply` | Submit application | Public |
| `POST /api/interviews` | Schedule interview | Bearer |
| `GET /api/summary` | Dashboard counts (jobs, apps, interviews) | Bearer |

All protected routes use `authMiddleware` (JWT verify & user hydration). Validation middleware returns `400` with structured error arrays.

---

# 4. Front-end Features

- **AuthContext** – handles login/register, token storage, auto-attach `Authorization` header.
- **Dashboard** – shows live counts from `/summary`.
- **JobListings** – fetches `/jobs`; authenticated users can **Apply** in one click.
- **PostJob** – protected form to create a new job.
- **Profile** – displays user info, logout button.
- **Navbar** – responsive links & logout; visibility driven by auth state.

Tailwind provides clean, mobile-friendly UI with zero custom CSS.

---

# 5. Security & Error Handling

- Bcrypt password hashing (10 salt rounds).
- JWT signed with `JWT_SECRET`; secret & `MONGO_URI` kept in `.env` (not committed).
- Helmet sets common secure HTTP headers. CORS default allows localhost dev; restrict in prod.
- Central `errorMiddleware` standardises JSON error responses; uncaught async errors captured by `express-async-handler`.

---

# 6. Scaling & Improvements

1. **Testing** – add Jest + Supertest for unit/integration coverage.
2. **Pagination & Filters** – for `/jobs` and `/applications` queries.
3. **Role-based Auth** – recruiter vs applicant roles; route guards.
4. **File uploads** – S3/Cloud storage for resumes.
5. **CI/CD** – GitHub Actions + Netlify/Vercel (front) & Render/Fly.io (API) with Docker.
6. **Monitoring** – Winston logging + Prometheus/Grafana dashboards.

---

# 7. Local Setup

```
# Backend
cd backend && npm i && cp .env.example .env  # add secrets
npm run dev

# Frontend
cd ../frontend && npm i
npm run dev  # http://localhost:3000
```

The API defaults to `http://localhost:5010` (configurable via `.env` and `VITE_API_URL`).

---