# Job Recruitment Platform – Backend System

## Introduction

The Job Recruitment Platform is a web-based backend system designed to simplify the hiring process for both job seekers and employers. This backend is built using Node.js, Express.js, and MongoDB, and includes user authentication, job posting, and application handling features.  
  
The system allows:  
- Users to register and log in  
- Recruiters to post jobs  
- Applicants to apply for jobs  
- Admins to manage users, jobs, and applications  
  
This backend also integrates authentication using JWT (JSON Web Tokens) and password encryption using bcrypt, ensuring that the platform is secure and scalable for real-world usage.

## Technology Stack

- Node.js – JavaScript runtime for backend  
- Express.js – Web framework for routing and middleware  
- MongoDB – NoSQL database  
- Mongoose – ODM for MongoDB  
- JWT – For secure user authentication  
- bcryptjs – For password encryption  
- dotenv – For environment variable management  
- Cloudinary (optional) – For file/image uploads

## Key Features

- User registration and login  
- Secure authentication system  
- Role-based access (admin, recruiter, applicant)  
- Job posting and listing  
- Job application submission  
- API-based architecture for frontend integration

Config inside db.js

const mongoose = require('mongoose');

require('dotenv').config();

const connectDB = async () => {

  try {

    await mongoose.connect(process.env.MONGO\_URI);

    console.log('MongoDB Connected');

  } catch (error) {

    console.error(error.message);

    process.exit(1);

  }

};

module.exports = connectDB;

controller /applicationController.js/jobController.js/

userController.js

const { User } = require('../models');

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

require('dotenv').config();

// ✅ Register New User

exports.registerUser = async (req, res) => {

  try {

    const { name, email, password, role } = req.body;

    // Check if user already exists

    const existingUser = await User.findOne({ email });

    if (existingUser) {

      return res.status(400).json({ message: 'User already exists' });

    }

    // Hash password

    const hashedPassword = await bcrypt.hash(password, 10);

    // Save user

    const newUser = new User({ name, email, password: hashedPassword, role });

    await newUser.save();

    res.status(201).json({ message: 'User registered successfully' });

  } catch (err) {

    res.status(500).json({ message: err.message });

  }

};

// ✅ Login User

exports.loginUser = async (req, res) => {

  try {

    const { email, password } = req.body;

    // Find user

    const user = await User.findOne({ email });

    if (!user || !(await bcrypt.compare(password, user.password))) {

      return res.status(400).json({ message: 'Invalid credentials' });

    }

    // Create token

    const token = jwt.sign({ id: user.\_id, role: user.role }, process.env.JWT\_SECRET, {

      expiresIn: '7d'

    });

    res.json({ token });

  } catch (err) {

    res.status(500).json({ message: err.message });

  }

};

Model/Application.js

const mongoose = require('mongoose');

const ApplicationSchema = new mongoose.Schema({

job: { type: mongoose.Schema.Types.ObjectId, ref: 'Job' },

jobSeeker: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },

status: { type: String, enum: ['applied', 'reviewed', 'accepted', 'rejected'], default: 'applied' }

});

module.exports = mongoose.model('Application', ApplicationSchema);

modela/index.js

// models/index.js

const User = require('./user');

const Job = require('./job');

const Application = require('./application');

module.exports = {

  User,

  Job,

  Application

};

model/job.js

const mongoose = require('mongoose');

const JobSchema = new mongoose.Schema({

title: String,

description: String,

requirements: String,

location: String,

employer: { type: mongoose.Schema.Types.ObjectId, ref: 'User' }

});

module.exports = mongoose.model('Job', JobSchema);

model/user.js

const mongoose = require('mongoose');

const UserSchema = new mongoose.Schema({

name: String,

email: { type: String, unique: true },

password: String,

role: { type: String, enum: ['job\_seeker', 'employer'], required: true }

});

module.exports = mongoose.model('User', UserSchema);

routes/applicationRoutes.js

const express = require('express');

const router = express.Router();

const { applyJob, getApplications } = require('../controllers/applicationController');

router.post('/', applyJob);

router.get('/', getApplications);

module.exports = router;

routes/authRoutes.js

const express = require('express');

const router = express.Router();

// Example route

router.get('/', (req, res) => {

  res.send('Auth route working');

});

module.exports = router;

routes/jobRoutes

const express = require('express');

const router = express.Router();

const { createJob, getJobs, getJobById } = require('../controllers/jobController');

router.post('/', createJob);

router.get('/', getJobs);

router.get('/:id', getJobById);

module.exports = router;

routes/userRoutes

const express = require('express');

const router = express.Router();

// POST /api/users

router.post('/', (req, res) => {

  const { name, email } = req.body;

  if (!name || !email) {

    return res.status(400).json({ error: 'Name and email are required' });

  }

  res.status(201).json({ message: `User ${name} with email ${email} created.` });

});

module.exports = router;

server.js

const express = require('express');

const connectDB = require('./config/db'); // MongoDB connection

require('dotenv').config();

const app = express();

// Connect to MongoDB

connectDB();

// Middleware

app.use(express.json());

// Import routes

const authRoutes = require('./routes/authRoutes');

const userRoutes = require('./routes/userRoutes');

const jobRoutes = require('./routes/jobRoutes');

const applicationRoutes = require('./routes/applicationRoutes');

// Use routes

app.use('/api/auth', authRoutes);

app.use('/api/users', userRoutes);

app.use('/api/jobs', jobRoutes);

app.use('/api/applications', applicationRoutes);

// Start the server

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

OUTPUT

