

# A PROJECT REPORT ON “JOB LISTNING APP”

Submitted in partial fulfillment of the requirement for the award of the degree  
Bachelor of Computer Applications (BCA)

SESSION :2024-2025

Project Guided By:	Submitted By:
<b>Internal Guide:</b>	Name: Utsav Nagar Enrollment No.: AY147401004 Institute code No.: 9028  Name: Uma Rajput Enrollment No.: AY147401002 Institute code No.: 9028



Makhanlal Chaturvedi National University of Journalism and  
Communication, Bhopal  
STUDY CENTER  
INFOTECH COMPUTER ACADEMY, INDORE  
(CODE-9028)

# Indexing

• ACKNOWLEDGMENT	3
• PROJECT WORK EVALUATION	4-5
• CERTIFICATE	6
• SELF-CERTIFICATE	7-8
• ABSTRACT	9
• INTRODUCTION	10-13
• REQUIREMENT SPECIFICATION	14
• SOFTWARE CONFIGURATION	15-16
• DESIGN AND ANALYSIS	17-19
• USE CASE DIAGRAM	20-26
• IMPLEMENTATION & SYSTEM TESTING	27-29
• DESIGNING & CODING	30-53
• EVALUATION	54-63
• CONCLUSION	64
• KEY ACHIEVEMENTS	64
• CHALLENGES FACED	65
• FUTURE ENHANCEMENTS	65
• FINAL THOUGHTS	66
• CONCLUSION	66

## **Acknowledgment**

This Major Project is the result of the contribution of many minds. I would like to acknowledge and thank my project guide Mr. Ashok Sharma for his valuable support and guidance. He guided me through the process from conception till the completion of this project.

I would also like to thank our Director Mr. Ravi Gattani and all my faculties at Infotech Computer Academy for their consistent support and encouragement throughout this journey. I extend my gratitude to the lab staff members and other non-teaching members for their assistance whenever needed.

I am very thankful for the open-handed support extended by many people. While no list would be complete, it is my pleasure to acknowledge the help of my friends who provided encouragement, knowledge, and constructive suggestions. This experience has enriched my learning and enhanced my skills in web development.

**(Utsav Nagar)**

**Enrollment No: AY147401004**

# PROJECT WORK EVALUATION

1. project title:- Job Listning App
2. Software Base :- **React & Firebase**
3. Submmited for Course & Year :- **BCA 6th Semester MAY-JUNE 2025.**
4. Group Evaluation (Project reports & Floppy py Evaluation Marks Planning + Marks Development + marks execution).

Examier - 1	Examier - 2	Group Evaluation (M. G.)

## 5. Individual Student Evaluation:

Regd. No.	Roll No.	Name Of Candidate	Int. Marks (MI) Given By Head of study center	Int. Marks (ME) Given By Head of Univer. Examiner	Total Marks (Max. 160) MT(MG+MI+ME)
		Utsav Nagar			
		Uma Rajput			

- Study Center Code & Name: MCRPSV/B.C.A..9028, INFOTECH COMPUTER ACADEMY.
- Forwarding By Head of Study Center:(Signature,
- name & study center seal).
- Remarks.

**DIRECTOR  
SIGNATURE**

**EXAMINER  
SIGNATURE**

# CERTIFICATE

This is to certify that this Major Project entitled "Job Listing App" submitted in partial fulfillment of the requirements for the award of the degree of Bachelor of Computer Application (BCA) in session 2022 to 2025 to the Makhanlal Chaturvedi National University of Journalism & Communication, Bhopal, done by Utsav Nagar & Uma Rajput is an authentic work carried out by them at Infotech Computer Academy, Indore under my guidance.

The matter and software embodied in this project work has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief.

Signature of BCA Teacher  
(Project Guide)  
Mr. Ashok Sharma

Signature of BCA Teacher

## **SELF-CERTIFICATE**

This is to certify that the Major Project report entitled “Job Listing App” is done by me, and it is authentic work carried out for the partial fulfillment of the requirements

for the award of the degree of Bachelor of Computer Application (BCA) under the guidance of **Mr. Ashok Sharma**. The matter and software embodied in this project has not been submitted earlier for award of any degree or diploma to the best of my knowledge and belief.

(Utsav Nagar)

Enrollment No: AY147401004

## **Abstract**

The Job Listing App is a web-based platform designed to connect job seekers with employers efficiently. It features job postings, filtering options, user profiles, and a job application system, ensuring a seamless experience for both candidates and recruiters.

Developed using React.js, the application enhances accessibility and usability with an intuitive interface. It simplifies the hiring process, making job searching and recruitment more streamlined. This project demonstrates expertise in frontend development and user-centric web solutions.



# **Introduction**

## **Introduction:**

The Job Listing App is a web-based platform designed to bridge the gap between job seekers and employers. It allows users to post jobs, filter listings, manage applications, and create profiles, making the job search and hiring process more efficient and accessible.

Built using React.js, the application provides a user-friendly interface with smooth navigation. It focuses on enhancing the recruitment experience by offering structured job listings and intuitive features, ensuring a seamless and productive interaction for both employers and job seekers.

## **User Modules:**

- Log In & Log Out
- Users can securely register and log in using their email and password. The authentication system ensures data privacy, allowing users to access their accounts and log out when needed.
- Finding Jobs
- Job seekers can browse available job listings posted by employers. The platform provides an easy-to-navigate interface for users to explore relevant job opportunities.
- Filtering Jobs

- Users can refine their job search by applying filters such as job title, location (work from home, onsite, hybrid), and salary range, helping them find suitable positions efficiently.
- Posting Jobs
- Employers can create and post job listings by providing essential details such as job title, salary, position, and location, making the hiring process straightforward.
- Open Profile (Self & Other Users)
- Users can view their own profiles to track posted and applied jobs. They can also visit other users' profiles to explore job postings and professional details.
- Log Out
- Users can securely log out of their accounts to end their sessions, ensuring account security and privacy.

## **Purpose:**

The Job Listing App aims to provide a seamless platform for job seekers and employers to connect. It simplifies job searching, filtering, and posting, ensuring an efficient recruitment process with an easy-to-use interface for both candidates and recruiters.

**Scope:**

The application offers features like job posting, filtering, profile management, and job applications. Future enhancements may include AI-based job recommendations, resume screening, mobile app integration, and employer-candidate communication, making the platform more robust and widely accessible.

# **Requirement Specification**

## **Hardware Configuration**

### **1. Server-Side:**

- Processor: Intel Core i5 or higher
- RAM: Minimum 8GB (Recommended: 16GB)
- Storage: Minimum 100GB SSD
- Network: High-speed internet connection
- Hosting: Cloud server (AWS, DigitalOcean, or any VPS)

### **2. Client-Side:**

- Processor: Intel Core i3 or higher
- RAM: Minimum 4GB (Recommended: 8GB)
- Storage: Minimum 20GB free space
- Display: Minimum 1366x768 resolution

## **Software Configuration**

### **1. Server-Side:**

- Operating System: Linux (Ubuntu 20.04+ recommended) or Windows Server
- Backend Framework: Node.js with Express.js
- Database: MongoDB / MySQL
- Web Server: NGINX / Apache
- Hosting: AWS, DigitalOcean, or any cloud provider

### **2. Client-Side:**

- Operating System: Windows / macOS / Linux
- Browser: Google Chrome, Mozilla Firefox, Edge (Latest versions recommended)
- Frontend Framework: React.js
- Code Editor: VS Code / WebStorm
- Dependencies: Node.js (for package management), npm or yarn

## Firebase

1. Firebase Authentication – Provides user authentication with email/password, Google, Facebook, GitHub, and more, making sign-ins secure and easy.
2. Firestore Database – A NoSQL cloud database that allows real-time syncing of data between users, making it ideal for chat apps, live updates, and dynamic applications.
3. Firebase Realtime Database – Another NoSQL database that provides real-time updates and stores data as JSON objects, ensuring low-latency synchronization.
4. Firebase Hosting – A free and fast hosting service for web apps with support for SSL, CDN, and custom domains.
5. Firebase Cloud Functions – Enables serverless computing, allowing developers to write backend logic without managing servers.
6. Firebase Cloud Messaging (FCM) – Used to send push notifications across Android, iOS, and web apps.
7. Firebase Analytics – Provides deep insights into user behavior and app performance to help improve user experience and engagement.
8. Firebase Storage – Securely stores and serves files like images, videos, and documents with easy integration into applications.
9. Firebase Remote Config – Allows app updates without republishing by remotely changing configurations and UI elements.

## Why Use Firebase?

- Real-time data synchronization
- Highly scalable and flexible
- Secure authentication
- Serverless backend
- Google-powered analytics & hosting

# **Design and Analysis**

## **Analysis**

The Job Listing App streamlines job searching and recruitment by providing job postings, filtering, and profile management. Built with React.js, it ensures a user-friendly experience. Future improvements like AI-based recommendations and mobile integration can enhance its efficiency and reach.

## **Disadvantages of the Current Software**

- Limited Backend Features – The current system may lack advanced backend functionalities like automated resume screening or AI-based job recommendations.
- No Real-time Notifications – Users do not receive instant alerts for new job postings or application status updates, which can reduce engagement.
- No Built-in Chat System – Employers and job seekers cannot communicate directly within the platform, requiring external contact methods.



- Scalability Issues – If hosted on a basic server, the app may struggle to handle a large number of users simultaneously.
- No Mobile App – The absence of a dedicated Android or iOS app limits accessibility and convenience for mobile users.
- Security Concerns – Without robust encryption and multi-factor authentication (MFA), user data may be vulnerable to breaches.
- Limited Resume Parsing – The system does not analyze or rank applicants based on skills and experience, making the hiring process less efficient.

## **Design Introduction**

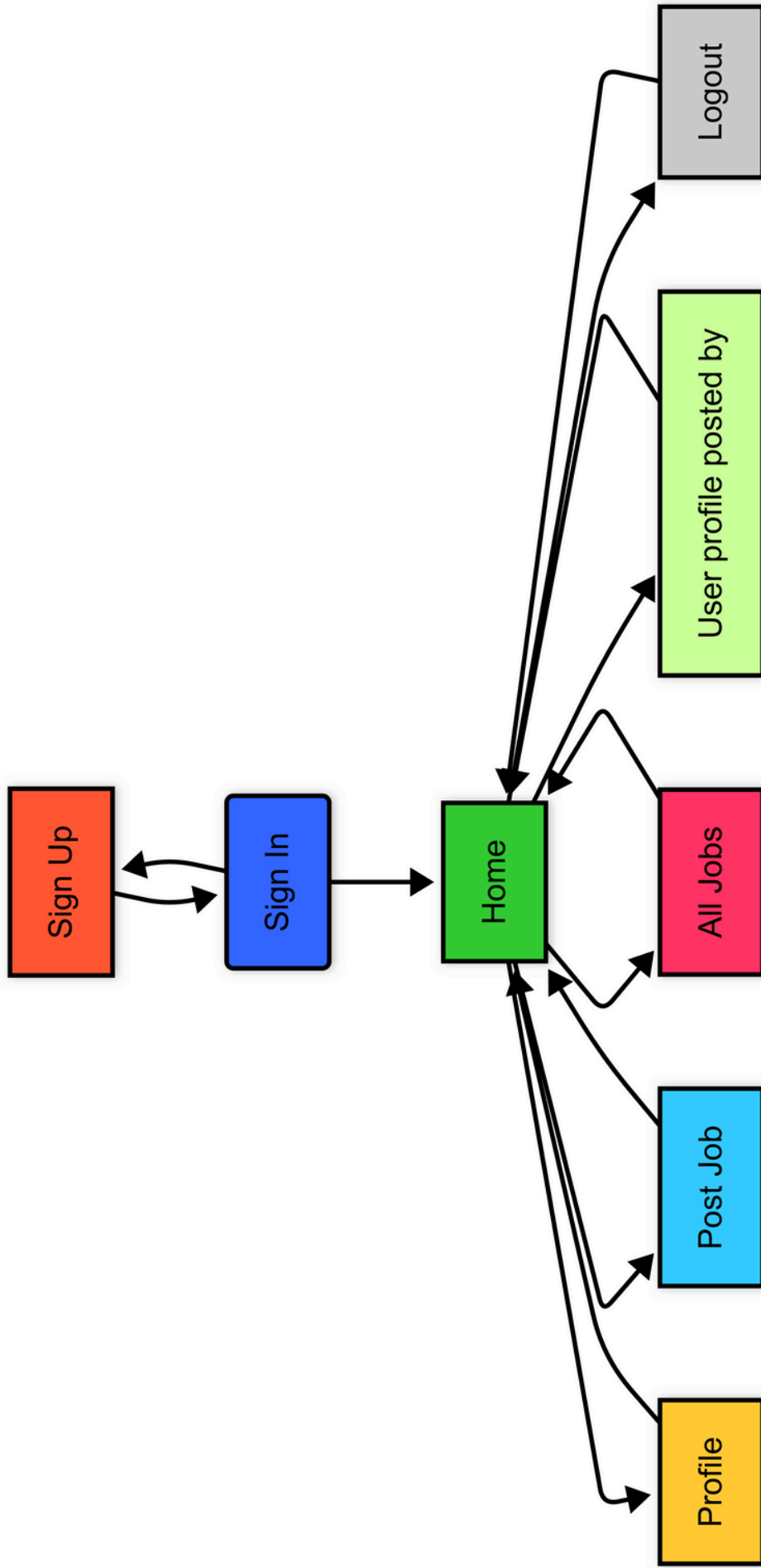
The Job Listing App is designed with a user-friendly interface using React.js for the frontend and a secure backend. It ensures seamless job posting, filtering, and profile management, focusing on efficiency, accessibility, and scalability for both job seekers and employers.

The design of the Job Listing App focuses on user-friendly navigation, responsiveness, and an intuitive interface to enhance the job-seeking and hiring experience. The frontend is developed using React.js, ensuring a dynamic and interactive UI, while the backend efficiently handles job postings, applications, and user authentication. The application follows a modular design approach, allowing scalability and easy future enhancements. A clean and structured layout improves usability, enabling job seekers and employers to navigate seamlessly. The responsive design ensures compatibility across different devices, including desktops, tablets, and mobile phones, making the platform accessible to a broader audience.

# Use Case Diagram

- Sign Up:Represents the initial registration process.
- Connects to Sign In.
- Sign In:Represents the login process.
- Connects to Home and Sign Up (allowing users to switch).
- Home:The main dashboard or landing page after login.
- Connects to all the main features: Profile, Post Job, All Jobs, User Profile (by posted), and Logout.
- All features connect back to the Home page.
- Profile:User's personal profile section.
- Post Job:Functionality to create and publish job listings.

- All Jobs:A list of all available job listings.
- User Profile (by posted):The profile of the user that posted a job.
- Logout:Logs the user out of the system.
- Arrows:Indicate the flow of navigation between the different sections.
- Styling:The styling is added to help distinguish the boxes.



## Entities & Relationships

1. User (User\_ID, Name, Email, Password, Role)
  - A user can be either a job seeker or an employer.
  - One user can post multiple jobs.
  - One user can apply for multiple jobs.
2. Job (Job\_ID, Title, Description, Salary, Location, Posted\_By)
  - A job is posted by one user.
  - A job can have multiple applicants.
3. Application (Application\_ID, Job\_ID, User\_ID, Status, Applied\_Date)
  - Tracks job applications made by users.
  - Links Users with Jobs.
4. Profile (Profile\_ID, User\_ID, Bio, Skills, Experience)
  - Stores additional details of users.
5. Job Filters (Filter\_ID, Job\_ID, Location, Job\_Type, Salary\_Range)
  - Allows users to filter jobs based on criteria.

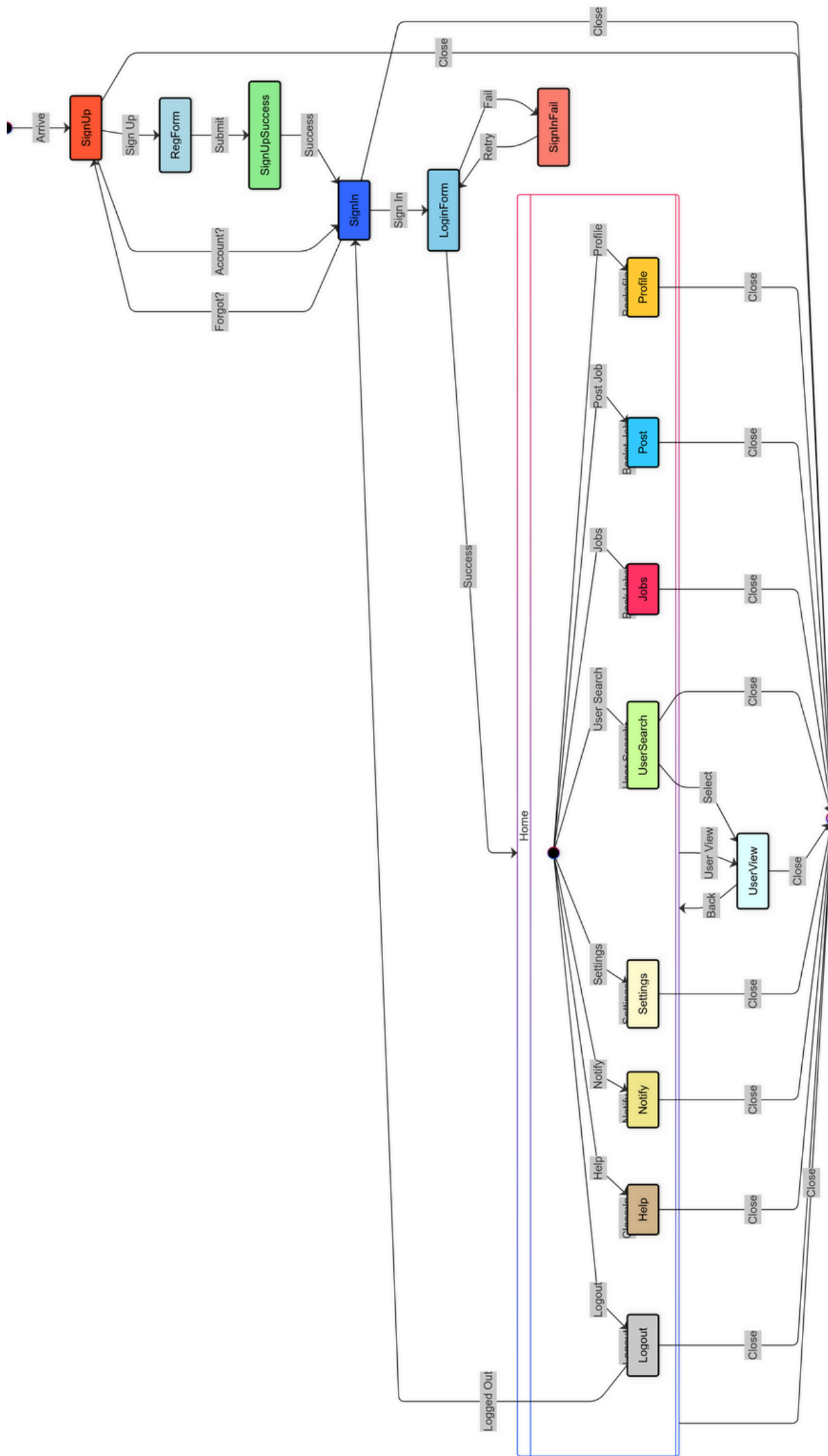
## Relationships

- User (1) → (M) Job → One user can post multiple jobs.
- User (1) → (M) Application → One user can apply for multiple jobs.
- Job (1) → (M) Application → One job can have multiple applicants.
- User (1) → (1) Profile → Each user has one profile.

## Entities & Attributes

1. User (User\_ID, Name, Email, Password, Role)
  - Primary Key: User\_ID
  - Role: Employer or Job Seeker
  - Relationships: Can post jobs, apply for jobs, and has one profile
2. Job (Job\_ID, Title, Description, Salary, Location, Posted\_By)
  - Primary Key: Job\_ID
  - Foreign Key: Posted\_By (References User\_ID)
  - Relationships: A job belongs to a user and receives applications

- Application (Application\_ID, Job\_ID, User\_ID, Status, Applied\_Date)
  - Primary Key: Application\_ID
  - Foreign Keys: Job\_ID (References Job), User\_ID (References User)
  - Relationships: A job can have multiple applicants
- Profile (Profile\_ID, User\_ID, Bio, Skills, Experience)
  - Primary Key: Profile\_ID
  - Foreign Key: User\_ID (References User)
  - Relationships: A user has one profile
- Job Filters (Filter\_ID, Job\_ID, Location, Job\_Type, Salary\_Range)
  - Primary Key: Filter\_ID
  - Foreign Key: Job\_ID (References Job)
  - Relationships: Helps filter jobs based on criteria





# **Implementation & System Testing**

## **Implementation**

The implementation of the Job Listing App follows a modular approach, ensuring smooth development, scalability, and maintainability. The project is built using React.js for the frontend and Node.js with Express.js for the backend. The MongoDB/MySQL database is used to store user details, job listings, and applications. The implementation process involves:

### **1. Frontend Development**

- Designing a responsive UI with React.js & Tailwind CSS.
- Implementing job searching, filtering, and profile management.

### **2. Backend Development**

- Setting up RESTful APIs using Node.js & Express.js.
- Managing user authentication (JWT-based login/signup).

- Database Management
  - Storing user details, job posts, and applications.
  - Implementing CRUD operations for job postings.
- Integration
  - Connecting frontend with backend APIs.
  - Implementing state management (Redux/Context API) for seamless navigation.

## **System Testing**

System testing ensures that the application functions correctly across different use cases. The testing process includes:

### **1. Functional Testing**

- Login/Signup: Verifying user authentication.
- Job Posting: Checking if employers can successfully post jobs.
- Job Search & Filtering: Ensuring job listings appear correctly based on applied filters.

### **2. Performance Testing**

- Testing response time and API efficiency under high traffic conditions.

- Compatibility Testing
  - Ensuring the application runs smoothly across different browsers (Chrome, Firefox, Edge) and devices (mobile, tablet, desktop).
- Security Testing
  - Checking for SQL Injection, Cross-Site Scripting (XSS), and Data Encryption to protect user data.
- User Acceptance Testing (UAT)
  - Conducting real-user testing to validate ease of use and bug-free experience.

#### Final Deployment

After testing, the application is hosted on cloud platforms like Vercel (Frontend) and AWS/DigitalOcean (Backend) for real-world access.

# **DESIGNING & CODING**

# Home.jsx

```
import JobList from "../JobList";  
import NavBar from "../NavBar"
```

```
const Home = () => {  
  return (  
    <div>  
      <NavBar></NavBar>  
      <JobList />  
    </div>  
  )  
}
```

```
export default Home;
```

# NavBar.jsx

```
import React, { useState } from 'react';  
import { AppBar, Toolbar, Typography, Button, Menu,  
MenuItem, IconButton, Box } from '@mui/material';  
import AccountCircle from '@mui/icons-  
material/AccountCircle';  
import { useNavigate } from 'react-router-dom';  
import { logout } from '../Firebase/database';  
import { auth } from '../Firebase/firebase';
```

```
export default function NavBar() {  
  const [anchorEl, setAnchorEl] = useState(null);  
  const navigate = useNavigate();  
  
  const handleMenu = (event) => {  
    setAnchorEl(event.currentTarget);  
  };  
  
  const handleClose = () => {  
    setAnchorEl(null);  
  };  
  
  const handleLogout = () => {  
    const status = logout();  
    if(status)  
      navigate("/")  
    else  
      alert("logout failed")  
    // Implement logout logic here  
    handleClose();  
    // Implement logout logic here  
  };  
  
  const handleProfile = () => {  
    navigate('/profile/'+auth.currentUser?.uid);  
    handleClose();  
  };  
  
  const handlePostJob = () => {  
    navigate('/post-job');  
  };  
}
```

```

return (
  <AppBar position="static">
    <Toolbar>
      <Typography variant="h6" sx={{ flexGrow: 1 }}>
        Open Job Listings
      </Typography>
      <Box display="flex" alignItems="center">
        <Button color="inherit" onClick={handlePostJob}>Post a
Job</Button>
        <IconButton
          size="large"
          edge="end"
          color="inherit"
          onClick={handleMenu}
        >
          <AccountCircle />
        </IconButton>
        <Menu
          anchorEl={anchorEl}
          open={Boolean(anchorEl)}
          onClose={handleClose}
        >
          <MenuItem onClick={handleProfile}>Profile</MenuItem>
          <MenuItem onClick={handleLogout}>Logout</MenuItem>
        </Menu>
      </Box>
    </Toolbar>
  </AppBar>
);
}

```

# JobListning.jsx

```
import React, { useEffect, useState } from 'react';
import {
  Grid,
  Card,
  CardContent,
  Typography,
  Button,
  Box,
  Dialog,
  DialogTitle,
  DialogContent,
  DialogActions,
  Link,
  Select,
  MenuItem,
  FormControl,
  InputLabel,
} from '@mui/material';
import { fetchJobsFromDatabase, applyToJob } from
'../Firestore/database';
import { useNavigate } from 'react-router-dom';
import { auth } from '../Firestore/firebase';
```



```
const JobList = () => {  
  const [jobs, setJobs] = useState([]);  
  const [filteredJobs, setFilteredJobs] = useState([]);  
  const [selectedJob, setSelectedJob] = useState(null);  
  const [open, setOpen] = useState(false);  
  const [locationFilter, setLocationFilter] = useState('');  
  const [jobTypeFilter, setJobTypeFilter] = useState('');  
  const navigate = useNavigate();
```

```
  useEffect(() => {  
    const fetchJobs = async () => {  
      try {  
        const fetchedJobs = await fetchJobsFromDatabase();  
        setJobs(fetchedJobs);  
        setFilteredJobs(fetchedJobs);  
      } catch (error) {  
        console.error('Error fetching jobs:', error);  
      }  
    };  
  });
```

```
  fetchJobs();  
}, []);
```

```
  useEffect(() => {  
    let filtered = jobs;
```

```
    if (locationFilter) {  
      filtered = filtered.filter((job) => job.location ===  
locationFilter);  
    }  
  });
```

```
if (jobTypeFilter) {  
  filtered = filtered.filter((job) => job.jobType ===  
jobTypeFilter);  
}
```

```
setFilteredJobs(filtered);  
, [locationFilter, jobTypeFilter, jobs]);
```

```
const handleShowDetails = (job) => {  
  setSelectedJob(job);  
  setOpen(true);  
};
```

```
const handleShowDetails = (job) => {  
  setSelectedJob(job);  
  setOpen(true);  
};
```

```
const handleClose = () => {  
  setOpen(false);  
  setSelectedJob(null);  
};
```

```

const handleApply = async (jobId) => {
  if (!auth.currentUser) {
    alert('Please log in to apply.');
```

```

    return;
  }

  try {
    await applyToJob(auth.currentUser.uid, jobId);
    alert('You have successfully applied for this job!');
  } catch (error) {
    console.error('Error applying for job:', error);
    alert('Failed to apply. Try again later.');
```

```

  }
};

return (
  <Box sx={{ padding: 3 }}>
    <Typography variant="h4" gutterBottom>
      Available Jobs
    </Typography>

    { /* Filter Section */ }
    <Box
      sx={{
        display: 'flex',
        flexWrap: 'wrap',
        gap: 3,
        alignItems: 'center',
        mb: 4,
      }}
    >

```

```
<FormControl variant="outlined" sx={{ minWidth:
250 }}>
  <InputLabel>Location</InputLabel>
  <Select
    value={locationFilter}
    onChange={(e) => setLocationFilter(e.target.value)}
    label="Location"
  >
    <MenuItem value="">All Locations</MenuItem>
    <MenuItem value="Remote">Remote</MenuItem>
    <MenuItem value="Office">Office</MenuItem>
    <MenuItem value="Hybrid">Hybrid</MenuItem>
  </Select>
</FormControl>
```

```
<FormControl variant="outlined" sx={{ minWidth:
250 }}>
  <InputLabel>Job Type</InputLabel>
  <Select
    value={jobTypeFilter}
    onChange={(e) => setJobTypeFilter(e.target.value)}
    label="Job Type"
  >
```

```
<MenuItem value="">All Job Types</MenuItem>
<MenuItem value="Full Time">Full Time</MenuItem>
<MenuItem value="Part Time">Part Time</MenuItem>
<MenuItem value="Internship">Internship</MenuItem>
</Select>
</FormControl>
</Box>
```

```
{/* Job Cards */}
<Grid container spacing={3}>
  {filteredJobs.length > 0 ? (
    filteredJobs.map((job) => (
      <Grid item xs={12} sm={6} md={4} key={job.id}>
        <Card sx={{ boxShadow: 3 }}>
          <CardContent>
            <Typography variant="h6" gutterBottom>
              {job.jobTitle}
            </Typography>
            <Typography variant="body2" color="textSecondary">
              paragraph>
                {job.description.substring(0, 100)}...
            </Typography>
            <Typography variant="body2" color="textPrimary">
              paragraph>
                <strong>Salary:</strong> {job.salaryRange.from} -
                {job.salaryRange.to}
            </Typography>
            <Typography variant="body2" paragraph>
              <strong>Location:</strong> {job.location}
            </Typography>
```

```

<Button
  variant="contained"
  color="primary"
  fullWidth
  onClick={() => handleApply(job.id)}
>
  Apply
</Button>
<Button
  variant="outlined"
  color="secondary"
  fullWidth
  onClick={() => handleShowDetails(job)}
  sx={{ mt: 1 }}
>
  Show More Details
</Button>
</CardContent>
</Card>
</Grid>
))
):(
  <Typography variant="body1">No jobs found
  matching the filters.</Typography>
  )}
</Grid>

```

```

{ /* Job Details Dialog */
  <Dialog open={open} onClose={handleClose}
  maxWidth="sm" fullWidth>
    <DialogTitle>Job Details</DialogTitle>
    <DialogContent>
      {selectedJob && (
        <>
          <Typography variant="h6">{selectedJob.jobTitle}
        </Typography>
        <Typography variant="body1" paragraph>
          {selectedJob.description}
        </Typography>
        <Typography variant="body2">
          <strong>Salary:</strong> {selectedJob.salaryRange.from} -
          {selectedJob.salaryRange.to}
        </Typography>
        <Typography variant="body2">
          <strong>Job Type:</strong> {selectedJob.jobType}
        </Typography>
        <Typography variant="body2">
          <strong>Location:</strong> {selectedJob.location}
        </Typography>
        <Typography variant="body2">
          <strong>Technologies:</strong>
          {selectedJob.technologies.join(', ')}
        </Typography>
        <Typography variant="body2" sx={{ mt: 2 }}>
          <strong>Posted By:</strong>{' '}
          <Link

```

```

    component="button"
    variant="body2"
    onClick={() =>
      navigate(`/profile/${selectedJob.postedByUid}`)}
    >
    {selectedJob.postedByName}
  </Link>
</Typography>
</>
)}
</DialogContent>
<DialogActions>
  <Button onClick={handleClose} color="secondary">
    Close
  </Button>
  <Button
    onClick={() => handleApply(selectedJob.id)}
    color="primary"
    variant="contained"
  >
    Apply
  </Button>
</DialogActions>
</Dialog>
</Box>
);
};

export default JobList;

```



# JobPostForm.jsx

```
import React, { useState } from 'react';
import { TextField, Button, MenuItem, Select, InputLabel,
FormControl, Checkbox, ListItemText, Input, Grid } from
'@mui/material';
import { Container, Box } from '@mui/system';
import { addJobDataToDatabase } from
'../Firestore/database';
import { useNavigate } from 'react-router-dom'; // Use
useNavigate instead of useHistory
import { auth } from '../Firestore/firebase';

const JobPostForm = () => {
  const [jobTitle, setJobTitle] = useState("");
  const [description, setDescription] = useState("");
  const [salaryFrom, setSalaryFrom] = useState("");
  const [salaryTo, setSalaryTo] = useState("");
  const [selectedTechnologies, setSelectedTechnologies] =
useState([]);
  const [jobType, setJobType] = useState(""); // New state for
job type
  const [location, setLocation] = useState(""); // New state for
job location

  // Technology options for multi-select
  const techOptions = [
    'firebase',
    'reactjs',
    'node js',
    'express js',
    'vue js',
    ,
```

```
Java',
'python',
'javascript',
'C++',
'game development',
'graphic designer',
'other'
];

// Job type options
const jobTypeOptions = ['Full Time', 'Part Time',
'Internship'];

// Location options
const locationOptions = ['Remote', 'Office', 'Hybrid'];

const navigate = useNavigate(); // Initialize useNavigate

// Handle form submission
const handleSubmit = (e) => {
e.preventDefault();

// Constructing the job data object
const jobData = {
jobTitle,
description,
salaryRange: {
from: salaryFrom,
to: salaryTo
},
```

```

technologies: selectedTechnologies,
jobType, // Add jobType to the job data
location, // Add location to the job data
postedByName: auth.currentUser?.displayName,
postedByUid: auth.currentUser?.uid
};

// Logging the formatted data to the console
console.log('Job Posted:', JSON.stringify(jobData, null, 2));

addJobDataToDatabase(jobData)
.then(docId => {
  alert('Job posted with ID:', docId);
})
.catch(error => {
  alert("Error posting job");
  console.error('Error posting job:', error);
});

// Reset the form (optional)
setJobTitle("");
setDescription("");
setSalaryFrom("");
setSalaryTo("");
setSelectedTechnologies([]);
setJobType("");
setLocation("");
};

// Handle back button click
const handleBack = () => {
  navigate(-1); // Navigates to the previous page
};

```

```
return (  
<Container maxWidth="sm">  
<Box my={3}>  
<form onSubmit={handleSubmit}>  
<Grid container spacing={3}>  
  { /* Job Title */ }  
  <Grid item xs={12}>  
    <TextField  
      label="Job Title"  
      variant="outlined"  
      fullWidth  
      value={jobTitle}  
      onChange={(e) => setJobTitle(e.target.value)}  
      required  
    />  
  </Grid>  
</form>  
</Box>  
</Container>  
)
```

```
    { /* Job Description */  
    <Grid item xs={12}>  
    <TextField  
    label="Job Description"  
    variant="outlined"  
    fullWidth  
    multiline  
    rows={4}  
    value={description}  
    onChange={(e) => setDescription(e.target.value)}  
    required  
    />  
    </Grid>
```

```
    { /* Salary Range */  
    <Grid item xs={6}>  
    <TextField  
    label="Salary From"  
    variant="outlined"  
    fullWidth  
    value={salaryFrom}  
    onChange={(e) => setSalaryFrom(e.target.value)}  
    type="number"  
    required  
    />  
    </Grid>  
    <Grid item xs={6}>  
    <TextField  
    label="Salary To"  
    variant="outlined"  
    fullWidth
```

```

value={salaryTo}
onChange={(e) => setSalaryTo(e.target.value)}
type="number"
required
/>
</Grid>

```

```

{/* Technologies Selection */}
<Grid item xs={12}>
<FormControl fullWidth>
<InputLabel>Technologies Required</InputLabel>
<Select
multiple
value={selectedTechnologies}
onChange={(e) => setSelectedTechnologies(e.target.value)}
input={<Input />}
renderValue={(selected) => selected.join(', ')}
>
{techOptions.map((tech) => (
<MenuItem key={tech} value={tech}>
<Checkbox checked={selectedTechnologies.indexOf(tech) >
-1} />
<ListItemText primary={tech} />
</MenuItem>
)))}
</Select>
</FormControl>
</Grid>

```

```

{/* Job Type Selection */}
<Grid item xs={12}>
<FormControl fullWidth>
<InputLabel>Job Type</InputLabel>
<Select
value={jobType}
onChange={(e) => setJobType(e.target.value)}
label="Job Type"
required
>
{jobTypeOptions.map((type) => (
<MenuItem key={type} value={type}>
{type}
</MenuItem>
))}
</Select>
</FormControl>
</Grid>

```

```

{/* Location Selection */}
<Grid item xs={12}>
<FormControl fullWidth>
<InputLabel>Location</InputLabel>
<Select
value={location}
onChange={(e) => setLocation(e.target.value)}
label="Location"
required
>
{locationOptions.map((loc) => (
<MenuItem key={loc} value={loc}>

```

```

{loc}
</MenuItem>
)}}
</Select>
</FormControl>
</Grid>

{/* Post Job Button */}
<Grid item xs={12}>
  <Button variant="contained" color="primary" fullWidth
type="submit">
    Post Job
  </Button>
</Grid>

{/* Back Button */}
<Grid item xs={12}>
  <Button variant="outlined" color="secondary" fullWidth
onClick={handleBack}>
    Back
  </Button>
</Grid>
</Grid>
</form>
</Box>
</Container>
);
};

export default JobPostForm;

```



# UserProfile.jsx

```
import React, { useEffect, useState } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import { doc, getDoc, collection, getDocs, query, where } from
'firebase/firestore';
import { db, auth } from '../Firebase/firebase';
import { Avatar, Box, Typography, Grid, Card, CardContent,
Divider, Button } from '@mui/material';

const UserProfile = () => {
  const { userId } = useParams(); // Get userId from URL params
  const navigate = useNavigate(); // Initialize navigate hook
  const [userData, setUserData] = useState(null);
  const [postedJobs, setPostedJobs] = useState([]);
  const [appliedJobs, setAppliedJobs] = useState([]);
  const [loading, setLoading] = useState(true);

  // Fetch user data from Firestore
  useEffect(() => {
    const fetchUserData = async () => {
      try {
        const usersCollection = collection(db, 'Users');
        const q = query(usersCollection, where('uid', '==', userId));
        const userSnapshot = await getDocs(q);

        if (!userSnapshot.empty) {
          const userDoc = userSnapshot.docs[0];
          const data = userDoc.data();
          setUserData(data);
        }
      } catch (error) {
        console.error('Error fetching user data:', error);
      }
    };
    fetchUserData();
  }, [userId]);
}
```

```

// Fetch posted jobs
if (data.postedJobs && data.postedJobs.length > 0) {
  const jobsPromises = data.postedJobs.map(jobId =>
getJobDetails(jobId));
  const jobs = await Promise.all(jobsPromises);
  setPostedJobs(jobs);
}

// Fetch applied jobs if viewing own profile
if (auth.currentUser?.uid === userId && data.appliedInJobs) {
  const appliedJobsPromises = data.appliedInJobs.map(jobId
=> getJobDetails(jobId));
  const jobs = await Promise.all(appliedJobsPromises);
  setAppliedJobs(jobs);
}
}
setLoading(false);
} catch (error) {
  console.error('Error fetching user data:', error);
  setLoading(false);
}
};

fetchUserData();
}, [userId]);

// Fetch job details by ID
const getJobDetails = async (jobId) => {
  const jobRef = doc(db, 'jobs', jobId);
  const jobSnapshot = await getDoc(jobRef);

```

```
if (jobSnapshot.exists()) {  
  return { id: jobSnapshot.id, ...jobSnapshot.data() };  
}  
return null;  
};  
  
// Navigate back to the previous page  
const handleBack = () => {  
  navigate(-1); // Go back to the previous page  
};  
  
// Loading state  
if (loading) {  
  return <Typography>Loading...</Typography>;  
}  
  
// User not found  
if (!userData) {  
  return <Typography>User not found.</Typography>;  
}
```

# EVALUATION

## Login

LOGIN

Don't have an account? [Create a new account here](#)

# Sign Up

Name \*

Email \*

Password \*

SIGN UP



SIGN IN WITH GOOGLE

Already have an account? [Log In](#)



# Available Jobs

Location



Job Type



## Python dev

dev dev dev...

**Salary:** 12313 - 13223**Location:** Remote

APPLY

SHOW MORE DETAILS

## Java dev

dev dev de v...

**Salary:** 100 - 100**Location:** Remote

APPLY

SHOW MORE DETAILS

## python developer

sjkbadv skjdvb nsljdk...

**Salary:** 1000 - 1000**Location:** Remote

APPLY

SHOW MORE DETAILS

## java developer

sdbvj sd.kjbv bkdsif ...

**Salary:** 37678 - 56758**Location:** Office

APPLY

SHOW MORE DETAILS



## Available Jobs

Location



Job Type



Python dev

dev dev dev

Java dev

dev dev dev

### Job Details

Python dev

dev dev dev

**Salary:** 12313 - 13223**Job Type:** Part Time**Location:** Remote**Technologies:** firebase, python, game development, C++**Posted By:** [Utsav Nagar](#)

CLOSE

APPLY

**Salary:** 1000 - 1000**Location:** Remote

APPLY

SHOW MORE DETAILS

**Salary:** 37678 - 56758**Location:** Office

APPLY

SHOW MORE DETAILS



Profile

Logout

## Available Jobs

Location



Job Type



### Python dev

dev dev dev...

**Salary:** 12313 - 13223**Location:** Remote

APPLY

SHOW MORE DETAILS

### Java dev

dev dev de v...

**Salary:** 100 - 100**Location:** Remote

APPLY

SHOW MORE DETAILS

### python developer

sjkbadv skjdvb nsljdk...

**Salary:** 1000 - 1000**Location:** Remote

APPLY

SHOW MORE DETAILS

### java developer

sdbvj sd.kjbv bkdsif ...

**Salary:** 37678 - 56758**Location:** Office

APPLY

SHOW MORE DETAILS



BACK



Utsav Nagar

un@gmail.com

## Posted Jobs

Java dev

Remote

dev dev de v

Salary: 100 - 100

Python dev

Remote

dev dev dev

Salary: 12313 - 13223

python developer

Remote

sjkbadv skjdvb nsldjk

Salary: 1000 - 1000

java developer

Office

sdbvj sd.kjbv bkdsIf

Salary: 37678 - 56758

## Applied Jobs

Java dev

Remote

dev dev de v

Salary: 100 - 100

Python dev

Remote

dev dev dev

Salary: 12313 - 13223

Job Title \*

Job Description \*



Salary From \*

Salary To \*

Technologies Required



Job Type



Location



POST JOB

BACK

Panel viewQuery builder

Users

nDTIhBQxJKXly..

More in Google Cloud

(default)

Users

nDTIhBQxJKXlyWC1pJOS

+ Start collection

Users

jobs

+ Add document

nDTIhBQxJKXlyWC1pJOS

tN9LD714z7qWRMfhSpeR

zx4t8eigF60WLIQ88ZS5

+ Start collection

+ Add field

appliedInJobs

0 "D7fqKCzUMZFx092sxYUa"

1 "xr4RwgGzWMTUTdSjGwHg"

email: "hemantnagar5446@gmail.com"

name: "Hemant Nagar"

uid: "CFLhtwb8Sff9LSCrYEMMTgxiq2D2"

Database location: asia-south1

Panel viewQuery builder

jobs

D7fqKCzUMZFx..

More in Google Cloud

(default)

jobs

D7fqKCzUMZFx092sxYUa

+ Start collection

Users

jobs

+ Add document

D7fqKCzUMZFx092sxYUa

YxMSIhyE4tPMrX1bCrXv

jLXRgr6LQQULvTh0Trf1

xr4RwgGzWMTUTdSjGwHg

+ Start collection

+ Add field

Applicants

0 "UTCaoHeHgHXekgAWCZdelbUtgYY2"

1 "CFLhtwb8Sff9LSCrYEMMTgxiq2D2"

description: "dev dev dev"

jobTitle: "Python dev"

jobType: "Part Time"

location: "Remote"

postedByName: "Utsav Nagar"

postedByUid: "UTCaoHeHgHXekgAWCZdelbUtgYY2"

salaryRange

from: "12313"

to: "13223"

technologies

0 "firebase"

1 "python"



2 "game development"



Database location: asia-south1


59


Panel view


Query builder

 More in Google Cloud 

 D7fqKCzUMZFx092sxYUa 

 Start collection

 Add field

 Applicants

0

"UTCaoHeHgHXekgAWCZdelbUtgYY2"

1

"CFLhtwb8Sff9LSCrYEMMTgxiq2D2"

description: "dev dev dev"


jobTitle: "Python dev"

jobType: "Part Time"

location: "Remote"


postedByName: "Utsav Nagar"

postedById: "UTCaoHeHgHXekgAWCZdelbUtgYY2"

 salaryRange

from: "12313"

to: "13223"

 technologies

0

"firebase"

1

"python"

2

"game development"

Search by email address, phone number, or user UID

Add user

Identifier	Providers	Created <div>↓</div>	Signed In	User UID
hemantnagar5446@gm...	<div></div>	Jan 10, 2025	Feb 1, 2025	CFLhtwb8Sff9LScrYEMMTgxl...
hemant@gmail.com	<div></div>	Jan 5, 2025	Jan 31, 2025	vypIF6f43dfNmse2H1mPfFV...
un@gmail.com	<div></div>	Jan 4, 2025	Feb 20, 2025	UTCaoHeHgHXekgAWCZdelb...

Rows per page:

50 

▼

1 – 3 of 3

<

>

## Conclusion

The Job Listing App successfully bridges the gap between job seekers and employers by providing an efficient, user-friendly platform for job searching, filtering, and posting. The system ensures seamless interaction, allowing users to create profiles, apply for jobs, and manage listings with ease. The use of React.js, Node.js, and a robust database enables scalability, responsiveness, and efficient data handling.

## Key Achievements

- ✅ User-Friendly Interface – A simple, intuitive design makes job searching and posting effortless.
- ✅ Advanced Filtering – Users can refine job searches based on title, location, and type (remote, hybrid, onsite).
- ✅ Secure Authentication – Implementing JWT-based authentication ensures data security.
- ✅ Efficient Data Management – A well-structured database stores job postings, applications, and user details securely.
- ✅ Scalability – The system is designed to handle a growing number of users and job listings.

## Challenges Faced

- ⚠ Real-Time Communication – Lack of direct messaging between job seekers and recruiters.
- ⚠ Resume Parsing – No automated resume ranking or AI-based recommendations.
- ⚠ Mobile Optimization – While responsive, a dedicated mobile app could enhance accessibility.

## Future Enhancements

- 🚀 AI-Based Job Recommendations – Implementing AI to match job seekers with relevant positions.
- 🚀 In-App Messaging System – Enabling direct communication between recruiters and applicants.
- 🚀 Mobile App Development – Creating an Android/iOS app for a smoother mobile experience.

## **Final Thoughts**

The Job Listing App is a well-structured and scalable solution for modern job seekers and employers. With future improvements like AI-powered recommendations, mobile applications, and real-time communication, it has the potential to become a leading job search platform

## **Conclusion**

The Job Listing App provides a structured and efficient platform for job seekers and employers to connect. With features like job posting, filtering, profile management, and applications, the system ensures a smooth user experience. The use of modern web technologies such as React.js, Node.js, and a secure database makes the platform scalable, responsive, and user-friendly.

While the app meets its core objectives, some areas require further enhancement, such as real-time communication, AI-driven job recommendations, and a dedicated mobile application. Addressing these challenges will enhance usability and engagement.

In the future, integrating advanced search algorithms, AI-based resume screening, and an intuitive mobile experience can help make the platform more dynamic and competitive. The Job Listing App has great potential to revolutionize online job searches and recruitment processes