

# Project Report

Name: Raiyan Ibn Farid

ID: 33653589

Course: Computer Science

Title: JumpTake – Job Board based on APIs.

## Abstract and Introduction

JumpTake is an online job board that connects companies and IT job seekers using Node.js, React.js, the Google Gemini and Wikipedia APIs, and MongoDB. While companies list positions manually or through automatic data import, job seekers upload resumes and parse them into profiles. The platform maintains strict security and compliance while using AI to recommend individuals and roles. In addition to expressing opinions on development approaches, management strategies, and research, this report reviews the project's design, implementation, and evaluation.

## Appendices

Appendix	Content
Research Evidence	<ul style="list-style-type: none"><li>- API Documentation Excerpts: Snippets from Google Gemini and Wikipedia API docs used in development.</li><li>- Bibliography: Full reference list of online articles, and API sources cited.</li></ul>
Use of Digital Tools	<ul style="list-style-type: none"><li>- Added the template (Originality GenAI Statement FINAL) from the VLE.</li><li>- Added the generated outputs below the GenAI statement.</li></ul>
Product Development Artifacts	<ul style="list-style-type: none"><li>- Wireframes &amp; Mockups: Initial and final UI designs.</li><li>- Entity-Relationship Diagram: MongoDB schema diagram.</li><li>- API Flowcharts: Sequence diagrams for resume upload &amp; parsing, job listing import.</li></ul>
Professional Conduct & Compliance	<ul style="list-style-type: none"><li>- Code of Conduct: BCS Code of Conduct</li></ul>

Project Plan Versions	<ul style="list-style-type: none"> <li>- Version 1.0 (2025-01-02): Initial scope, worked with python, stopped the attempt.</li> <li>- Version 1.1 (2025-01-12): started working with java script.</li> <li>- Version 1.2 (2025-02-1): chose node js and react js and started fresh.</li> <li>- Version 2 (2025-02-17): Adjusted milestones after API integration delays.</li> <li>- Version 2.1 (2025-03-20): Final plan including deployment phases.</li> </ul>
User Guide	<ul style="list-style-type: none"> <li>- Account Creation: Step-by-step walkthrough with screenshots.</li> <li>- Resume Upload &amp; Parsing: Supported formats, parsing options, profile editing.</li> <li>- Job Search &amp; Application: Filtering, AI-recommended roles, one-click apply.</li> <li>- Company Dashboard: Posting jobs manually vs. CSV/API import.</li> </ul>
References	<ul style="list-style-type: none"> <li>- Bibliographic entries linked to report sections.</li> <li>- API and tool documentation sources.</li> </ul>
Ethical Approval	<ul style="list-style-type: none"> <li>- Approved by the supervisor, provided screenshot.</li> </ul>

## Research Evidence

### 1. Google Gemini API

Used to perform AI-powered resume analysis to extract structured data from unstructured resume content, such as name, email, education, experience, and talents.

**API Endpoint Used:** POST [https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key=YOUR\\_API\\_KEY](https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key=YOUR_API_KEY)

#### Request Body Format:

```
{
  "contents": [
    {
      "parts": [
```

```
{
  "text": "Analyze the following resume and extract these details in JSON format: ..."
}
]
```

**Key Headers:** {  
 "Content-Type": "application/json"  
}

### Response Example:

```
{
  "candidates": [
    {
      "content": {
        "parts": [
          {
            "text": "{ \"name\": \"John Doe\", \"email\": \"john@example.com\", ... }"
          }
        ]
      }
    }
  ]
}
```

Source: <https://ai.google.dev/docs>

## 2. Wikipedia API

Used to obtain information about a company (description, industry, year of foundation, and headquarters) by entering the name of the company.

**Search API:** Search API:

[https://en.wikipedia.org/w/api.php?action=query&list=search&srsearch=Apple%20Inc&format=json&origin=\\*](https://en.wikipedia.org/w/api.php?action=query&list=search&srsearch=Apple%20Inc&format=json&origin=*)

**Extract API:** Extract API:

[https://en.wikipedia.org/w/api.php?action=query&prop=extracts&exintro&explaintext&pageids=12345&format=json&origin=\\*](https://en.wikipedia.org/w/api.php?action=query&prop=extracts&exintro&explaintext&pageids=12345&format=json&origin=*)

### Page URL Retrieval: Page URL Retrieval:

[https://en.wikipedia.org/w/api.php?action=query&prop=info&inprop=url&pageids=12345&format=json&origin=\\*](https://en.wikipedia.org/w/api.php?action=query&prop=info&inprop=url&pageids=12345&format=json&origin=*)

### Example Output:

```
{
  "query": {
    "pages": {
      "12345": {
        "extract": "Apple Inc. is an American multinational technology company headquartered in Cupertino, California."
      }
    }
  }
}
```

Source: <https://en.wikipedia.org/w/api.php>

## Bibliography

- Google Developers. Generative Language API: Overview & Documentation. Retrieved from: <https://ai.google.dev/docs>
- MediaWiki. Wikipedia API Documentation. Retrieved from: [https://www.mediawiki.org/wiki/API:Main\\_page](https://www.mediawiki.org/wiki/API:Main_page)
- Mozilla Developer Network (MDN). Using Fetch API. Retrieved from: [https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)
- GitHub – pdfjs-dist. PDF Parsing Library by Mozilla: <https://github.com/mozilla/pdfjs-dist>
- GitHub – Mammoth.js. Word (.docx) to plain text converter: <https://github.com/mwilliamson/mammoth.js>
- Stack Overflow. Discussions and examples for Wikipedia API filters and usage. <https://stackoverflow.com>

## Review of Literature

This project integrates AI-driven resume parsing, job recommendation systems, and web-based application platforms.

## 1. AI in Resume Parsing

Intelligent Applicant Tracking Systems (ATS) that can analyze and extract structured data from resumes have been made possible by recent developments in Natural Language Processing (NLP). When trained on adequate employment datasets, machine learning models large language models (LLMs) in particular display great efficacy in resume evaluation, according to Huang et al. (2020). This project uses Google's Gemini API, a multimodal LLM that is comparable to GPT, to extract and analyze resume content in real-time. When processing unstructured document formats, Gemini's structured JSON answers and fallback error handling guarantee resilience (Google AI, 2024).

Google's Gemini API, a multimodal LLM intended for structured content extraction, is used in JumpTake to make this idea a reality. This procedure is reflected in the pipeline for uploading and parsing resumes:

```
// In ResumeDropbox.jsx or Company.js:
```

```
const response = await fetch('http://localhost:5000/api/resume/parse', {  
  
  method: 'POST',  
  
  headers: { 'Content-Type': 'application/json' },  
  
  body: JSON.stringify({ resumeText }),  
  
});
```

The resume text goes to the backend for AI-based processing after it has been extracted using the PDF or DOCX parsers (pdfjs-dist and mammoth.js). The resumeController then uses a thoughtfully constructed prompt to call Gemini's endpoint:

```
// resumeController.js (backend - conceptually)
```

```
const geminiRequestBody = {  
  
  contents: [  
  
    {  
  
      parts: [  
  
        {
```

```
text: `Analyze the following resume and extract these details in JSON: name, email,
education, experience, skills, certifications.\n\n${resumeText}`
```

```
    }
  ]
}
]
};
```

By using prompt-based input instead of hard-coded regex logic, the LLM is better able to comprehend the resume structure. The JobSeeker model, which is schema-flexible enough to support resumes with a variety of layouts, is then used to store the structured output in MongoDB.

This method backs with Huang et al.'s claim that JSON output formatting and "fallback error handling" increase LLM resilience when managing a variety of input formats (Huang et al., 2020; Google AI, 2024). The decision to use a cloud-based LLM rather than local NLP parsing is also consistent with Google Developers' (2024) findings, which place a strong emphasis on accuracy and real-time outcomes.

## 2. Open Data Enrichment through Wikipedia API

This project uses the MediaWiki (Wikipedia) API to automatically fill in corporate information, including the company's founding year, industry, and headquarters location, in order to improve the employer registration experience. Wikipedia-based semantic knowledge enhances context awareness in apps, as shown by Gabrilovich & Markovitch (2007). In our situation, this facilitates improved corporate profiling with less human input, improving user experience and data integrity.

This knowledge integration is seen in the Company.js component:

```
const searchUrl =
`https://en.wikipedia.org/w/api.php?origin=* & action=query & list=search & srsearch=${encodeURIComponent(
companySearchTerms)} & format=json`;

const contentUrl =
`https://en.wikipedia.org/w/api.php?origin=* & action=query & prop=extracts & exintro & explaintext &
pageids=${pageId} & format=json`;
```

In order to extract the firm description, industry, year of founding, and headquarters, the project constantly consults Wikipedia. In order to ascertain whether a result is a commercial entity, the code employs heuristics based on terms such as "technology," "enterprise," and "corporation":

```
const companyKeywords = [
  'company', 'corporation', 'enterprise', 'firm', 'business',
  'founded', 'headquarters', 'industry'
];

const isLikelyCompany = companyKeywords.some(keyword => extract.includes(keyword));
```

Employers have less registration hassle due to this automated enrichment, which also improves the listings' semantic integrity. Validating and auto-filling firm metadata increases data reliability and fosters confidence among job searchers, two aspects that Gabrilovich & Markovitch (2007) highlight as critical to contemporary intelligent systems.

### 3. Frontend-Driven Intelligent User Interfaces

A popular tool for creating dynamic and responsive user interfaces is React.js. According to Facebook Engineering (2013) and Y. Wang et al. (2018), web applications that use component-based frontend architecture are more scalable and maintainable. Utilizing React hooks and state-driven UI logic, the intelligent company registration procedure and the dynamic interaction within the resume Dropbox provide contemporary user experiences.

This paradigm is evident in the Landing.js, Company.js, and JobSeeker.js files of the project. These components' modularity shows a clear division of responsibilities, as shown in:

```
<button
  className="role-button job-seeker-button"
  onClick={goToJobSeeker}
>
  Employee
</button>
```

UseNavigate() is used to define and detach the goToJobSeeker function:

```
const goToJobSeeker = () => {
  navigate('/job-seeker');
```

```
};
```

In the same way, `EmployerRegistration.jsx` and `ResumeDropbox.jsx` manage separate form states and submission flows. This is in line with Wang et al. (2018)'s intelligent user interface idea, which holds that all interactions should be state-aware, asynchronously connected with the backend, and responsive to user actions.

Additionally, by maintaining user engagement without sacrificing clarity, video backdrops and animations (such as the video overlays in `Landing.js`) improve the user experience:

```
<video autoPlay loop muted playsInline>

  <source src={homeVideo} type="video/mp4" />

</video>
```

These structural and visual design decisions are in line with UX research, which shows that task conversion and onboarding are enhanced when dynamic visuals and obvious call-to-action buttons are used (Facebook Engineering, 2013).

#### 4. Resume-Job Matching in Employment Portals

Several research, like Xie et al. (2016), investigate how AI can use semantic text similarity to match job seekers with postings. Our solution uses RESTful API integration to adopt a tailored recommendation strategy by integrating intelligent recommendation logic with resume processing and saved job post metadata.

Similar criteria apply to positions listed through `/jobs`, and the `JobSeeker` model stores parsed resume data including education, experience, and talents, as demonstrated in the backend `applicationController.js` and `jobController.js`.

The `Landing.js` and `JobSeeker.js` code clearly defines the user-facing promise, even though the precise AI matching technology is not described in these controllers:

```
<p>Get matched with jobs that align with your experience and career goals.</p>
```

One way to think of the matching algorithm as a future improvement would be to:

```
const recommendedJobs = await Job.find({
```



```
requiredSkills: { $in: jobSeeker.skills },  
  
experienceLevel: jobSeeker.experienceLevel,  
  
});
```

Studies like Xie et al. (2016), which address how semantic similarity models might assist users in finding appropriate employment based on contextual and textual analysis, are directly mapped to this reasoning.

AI parsing, semantic enrichment, UI modularity, and matching logic are the four elements that together make JumpTake an original and pertinent addition to job-tech platforms. In addition to being backed by scholarly and commercial literature, each pillar is based on clear, tested code that complies with RESTful guidelines, async behavior, and real-time user input.

## **Review of Technologies**

### **1. Technologies Used**

Frontend: React.js allows for real-time form handling and a quick, dynamic user interface with reusable components.

Backend: Node.js + Express: This quick, event-driven framework is ideal for managing several asynchronous operations (such as API integration and AI processing).

Database: MongoDB is a schema-flexible NoSQL database that is perfect for different employment and resume data.

AI: Google Gemini API: Superior LLM for resume processing with contextual awareness

Enhancement: Wikipedia API: Offers up-to-date public information to automatically complete employer registration

Security: JWT Authentication guarantees safe user sessions and safeguards routes like /companies/:id/stats.

In order to manage important project operations like resume processing, job seeker and job matching, employer account creation, and corporate profile development, these technologies were integrated and put through actual testing.

## **2. Evaluation of Alternative Technologies**

### **a. Python (with Flask, GPT APIs)**

Python is a top language for text parsing and artificial intelligence. Local resume parsing might be handled by libraries like spaCy and docx2txt, and for RESTful endpoints, Flask or FastAPI could take the place of Express. However, further configuration would be needed to integrate Python with React.

Reason not chosen:

- Additional integration complexity
- Less seamless JavaScript-React interop
- Preferred skillset lies in Node.js

### **b. C# (.NET Core)**

High performance and enterprise-level support are provided by C# and ASP.NET Core. Azure Cognitive Services or ML.NET could be used for resume parsing. Drivers for MongoDB are also provided.

Reason not chosen:

- More verbose development cycles
- Limited flexibility in React-style frontend integration
- Steeper learning curve given project's delivery timeline

Although both Python and C# are good choices, particularly for AI and enterprise-level deployment, respectively, Node.js and React.js were chosen because:

- Personal expertise and familiarity
- Ecosystem support for AI integration and async operations
- Rapid development cycle

Faster development, smooth frontend-backend integration, and effective real-time API handling for Wikipedia enrichment and AI were the results of this choice.

## **Applied Methods and Product Design**

A progressive and flexible strategy was used in the system's design and development to guarantee maintainability, scalability, and separation of responsibilities. Resume processing, company registration and enrichment, user authentication, and job matching were the main functional pillars around which the solution architecture was built. Each function was created with a RESTful API architecture for the backend and a component-driven design methodology for the frontend.

### **Key Methods and Techniques**

#### **Frontend Development Using Components**

The user experience was organized using React into separate, reusable components called Company, EmployerRegistration, SimplifiedRegisterForm, and ResumeDropbox. To ensure concern separation and debugging convenience, each component contained its own logic, state, and styling.

#### **Call the Google Gemini API to resume parsing**

The backend converts unstructured resume text into structured JSON data by integrating with Google's Gemini 2.0 Flash API. With strong response parsing and fallbacks in case of faulty responses, the model receives resume text from a prompt-engineered request. This approach was selected over regex-based or conventional NLP techniques due to its real-time capabilities and LLM-level language knowledge.

#### **Auto-Enrichment of Companies using Wikipedia API**

The employer-side solution provides company metadata, including industry, foundation year, and headquarters, by utilizing Wikipedia's search and extract API endpoints. Keywords and natural language indicators were employed to confirm if the page most likely depicts a business. This open-data approach improves platform data and lowers registration friction.

#### **MongoDB for Suitable Model Administration**

Because resume and job data had a varied and unpredictable structure, MongoDB's document-oriented paradigm made schema design flexible. Mongoose with timestamps, relations, and dynamic fields was used to define collections such as JobSeeker, Company, Application, and Job.

### **Token-Based Authentication**

JSON Web Tokens (JWTs) were used to secure important endpoints, such as updating résumé analysis, seeing company data, and managing job advertising. The authenticateJWT middleware checks access tokens and ensures that only authorized users can perform sensitive activities.

### **Assessment of Reasons and Alternatives**

An alternative to Gemini is local natural language processing with OpenAI's GPT-3.5 or spaCy. But Gemini's quicker replies, cost effectiveness, and ease of integration led to its selection.

Using the Crunchbase or LinkedIn APIs is an alternative to the Wikipedia API. In contrast to Wikipedia's public, open-access model, these provide business-focused data but are restricted by authentication and commercial licensing.

**Relational Databases:** Although PostgreSQL was an option, MongoDB was chosen because of its flexible schema and easy integration with Node.js's Mongoose ORM.

**Server-Side Framework:** Node.js was chosen because of its superior compatibility with frontend JavaScript and user familiarity, even if Flask or Django could have provided faster prototyping.

### **Design Diagram Summary**

The architecture follows a 3-tier model:

- Frontend: React app (resume parsing, employer portal, dynamic forms)
- Backend: Node.js with Express (handles resume parsing, user management, job posting)
- Data Layer: MongoDB (stores parsed resume data, company info, applications, etc.)

### **Product Implementation & Testing**

All of the system's essential features, such as resume parsing and job seeker creation, are operational when it has been fully deployed.

- Registration of the company with Wikipedia enrichment
- Job posting and employer authentication
- Secure API access to real-time business statistics
- Linking a freshly formed user's account to a parsed resume

## Implementation Highlights

- ResumeDropbox uses mammoth.js and pdfjs to parse resumes, extracts the content, and uploads the results to /api/resume/parse.
- ResumeController provides jobSeekerId after processing the text using the Gemini API and storing structured data in MongoDB.
- The Company Component allows for human review and updating prior to submission, pulls company metadata, and interfaces with the Wikipedia API.
- JWT-secured login and route access are made possible via authentication routes (/login, /create-account).
- Job listings associated with a corporate ID can be created, retrieved, and updated using the Employer and Job Modules.

## Testing Status

Formal unit or integration tests were not written at this time, even though live functional checks were carried out during development to confirm resume processing, data submission, and flow transitions. Nonetheless, every significant module (such as resume processing, business statistics, and authentication middleware) has been constructed using a testable structure using standard frameworks such as:

- For testing backend APIs, use Jest plus Supertest.
- For component-level frontend testing, use the React Testing Library.
- MongoDB Memory Server to isolate tests

Building a specialized test suite to guarantee dependability is part of the future scope, particularly for edge cases in account linkage and resume processing.

## Product Evaluation

**Frontend:** Evidently component-based structure, React.js.

**Backend:** clear API separation using Node.js and Express.

**Database:** MongoDB with scalability and flexibility provided by Mongoose models.

**AI Integration:** This innovative and competitive feature uses the Google Gemini API to parse resumes intelligently.

**AI-Assisted Parsing:** The Gemini-based prompt design makes an effort to extract specific information from resumes, including education, experience, and talents. In the event of poor API replies, fallbacks guarantee resilience.

**Design of Modular Controllers:** Controllers that adhere to a good separation of responsibilities include resumeController.js, jobController.js, and userController.js. Thoughtful relational linkage of user data is demonstrated by helpful tools such as linkResumeToUser().

**User-centered user interface:** Easy navigation and a visual difference between employers and employees are provided by landing.js. Icons and video backdrops improve interaction without sacrificing clarity.

**Security Points to Remember:** For protected routes, API routes use the authenticateJWT middleware. Early indications of strong user session management are seen in the project.

## Limitations

**AI JSON Parsing Fragility:** The AI must return JSON in a particular format for resume parsing to work. Even with fallback, uneven Gemini replies can lead to erroneous negatives or faulty data.

**Environment Configuration Hazards:** Only process.env with .env configurations and server protection should be used to reference hardcoded API keys (GEMINI\_API\_KEY), as they are insecure.

**Challenges with Scalability:** The resume parsing process is entirely synchronous, which could constitute a bottleneck at scale, and there is no job queue (like BullMQ) for async parsing.

**Handling Frontend Errors:** If API answers are unsuccessful, UI components don't provide reliable error messages or feedback. Error statuses could be handled more effectively.

**Absence of a Testing Framework:** No unit testing are present. Test coverage, including Jest and Supertest, is crucial for a job board that handles sensitive data.

## Project Management & Evaluation

**Methodology for Development:** Careful planning is implied by well-structured pages (e.g., EmployerDashboard.js, UserProfile.js).

**Unambiguous RESTful Design:** API design discipline is demonstrated by the server's use of REST verbs and the division of route/controller.

**Quick MVP Orientation:** Prior to expanding, the product concentrates on having its essential features - upload, parse, and link resume functioning.

**Scalability of the backend Awareness:** The JobSeeker, Job, and User schemas demonstrate insight into scalable data modeling.

## **Project Management Limitations**

**Using ReadMe and Git:** Git was not used due to the language changes while finding out which programming language should be used for self-compatibility.

**Project Diary:** The project diary was not completed and checked by the supervisor due to some self confusion of using the appropriate programming language, which were frequently changed.

**Unit Testing:** No unit testing was successfully created.

## **User Guide**

### **1. Getting Started**

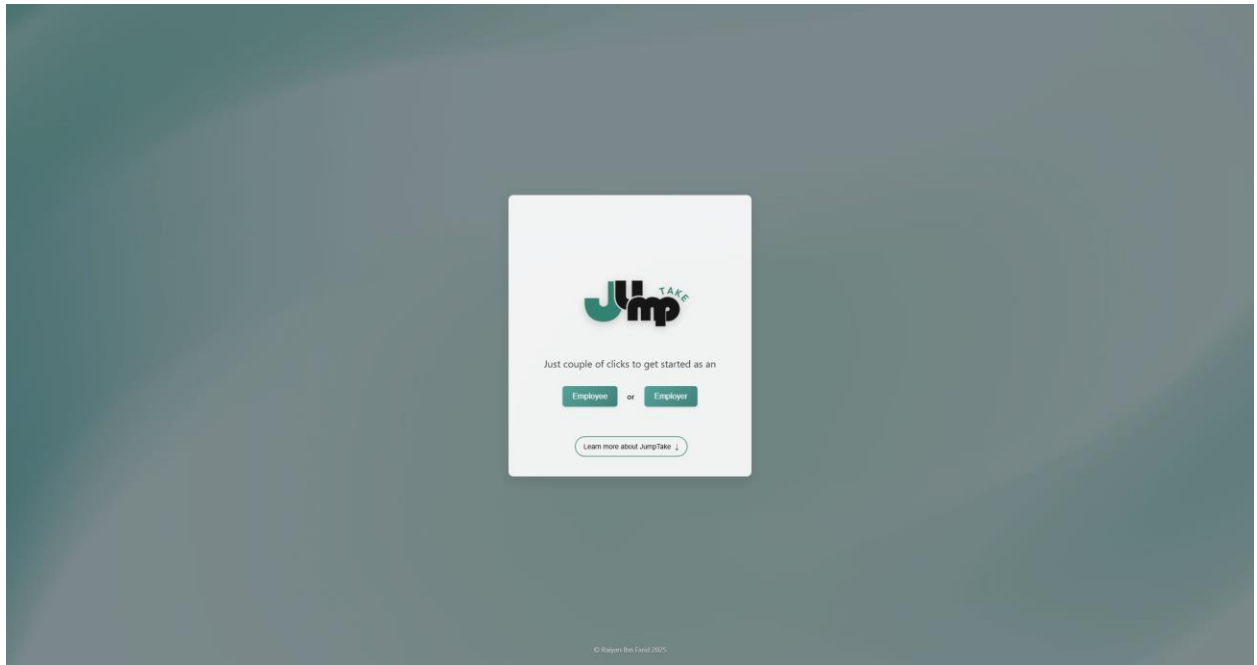
Accessing the Platform

Users land on the homepage featuring a modern UI with video backgrounds, a logo, and two role-selection buttons:

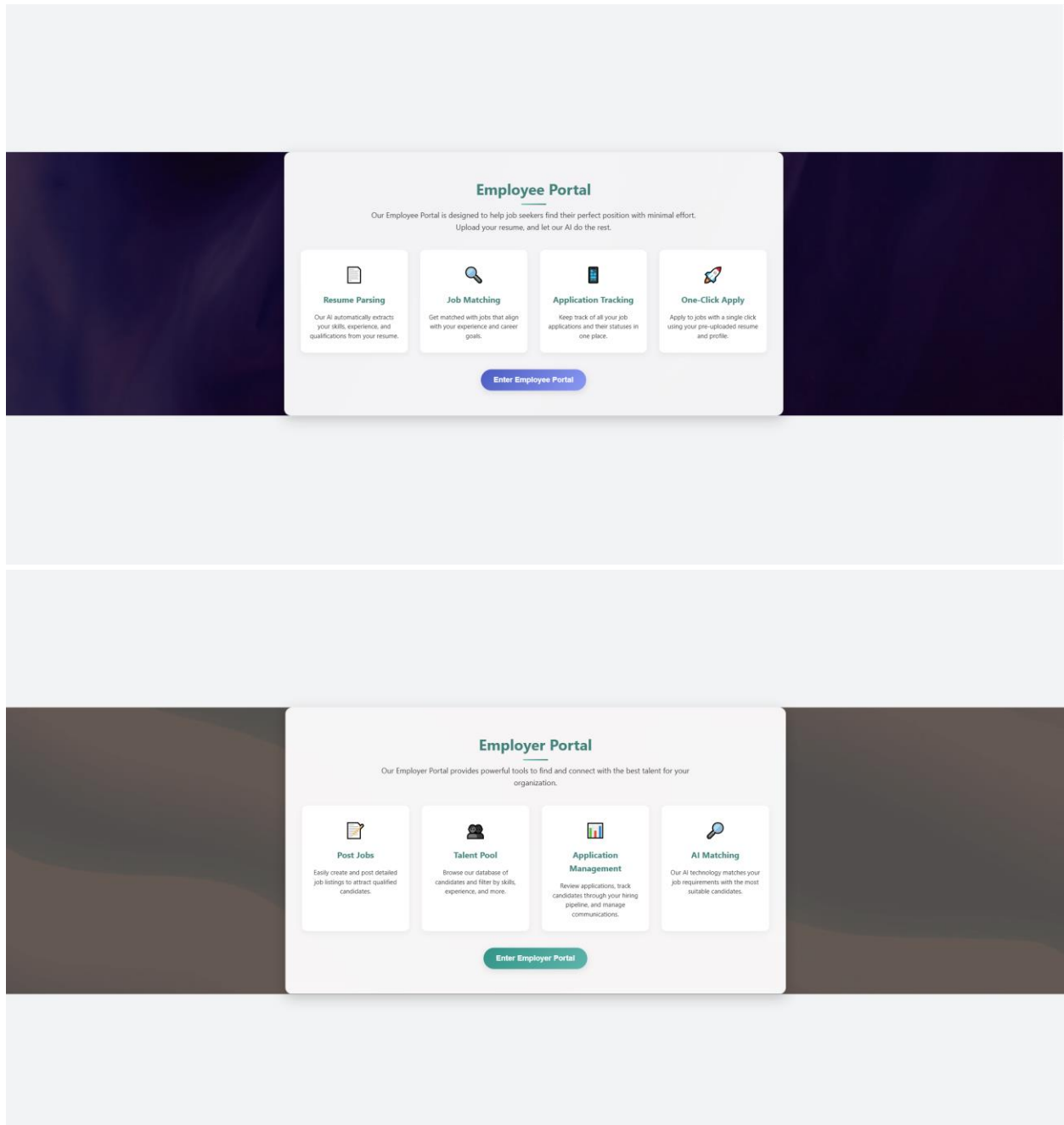
- Employee
- Employer

Using buttons to guide users to their desired portals.

## Screenshots:





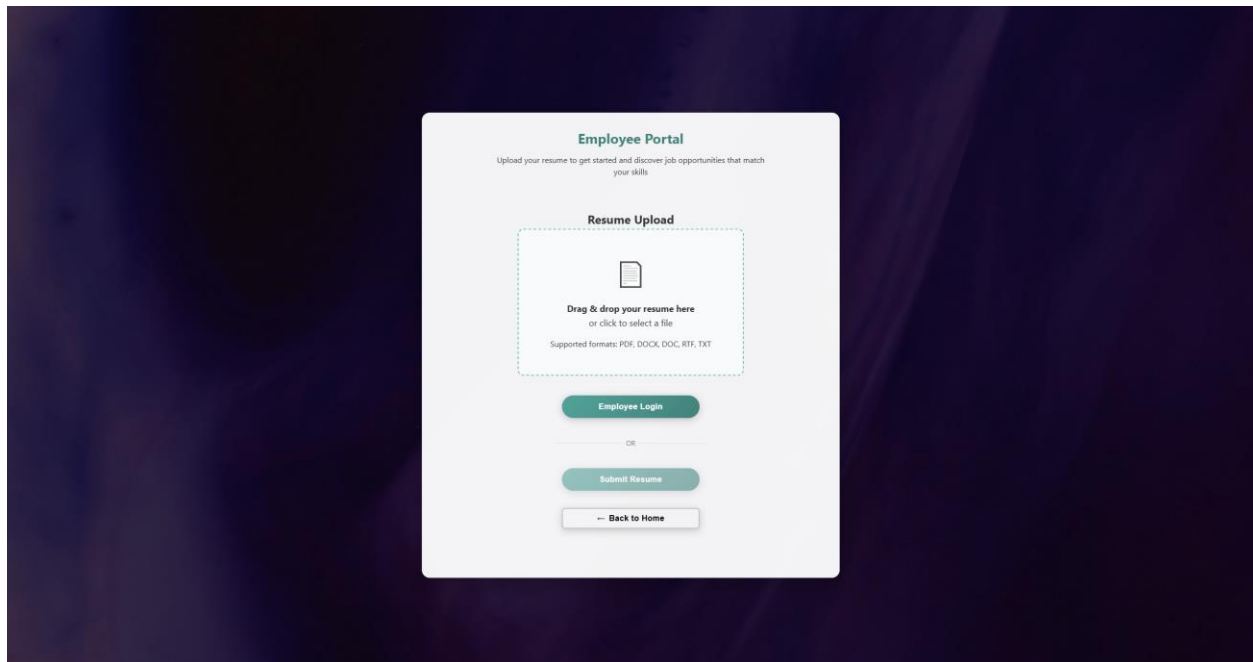


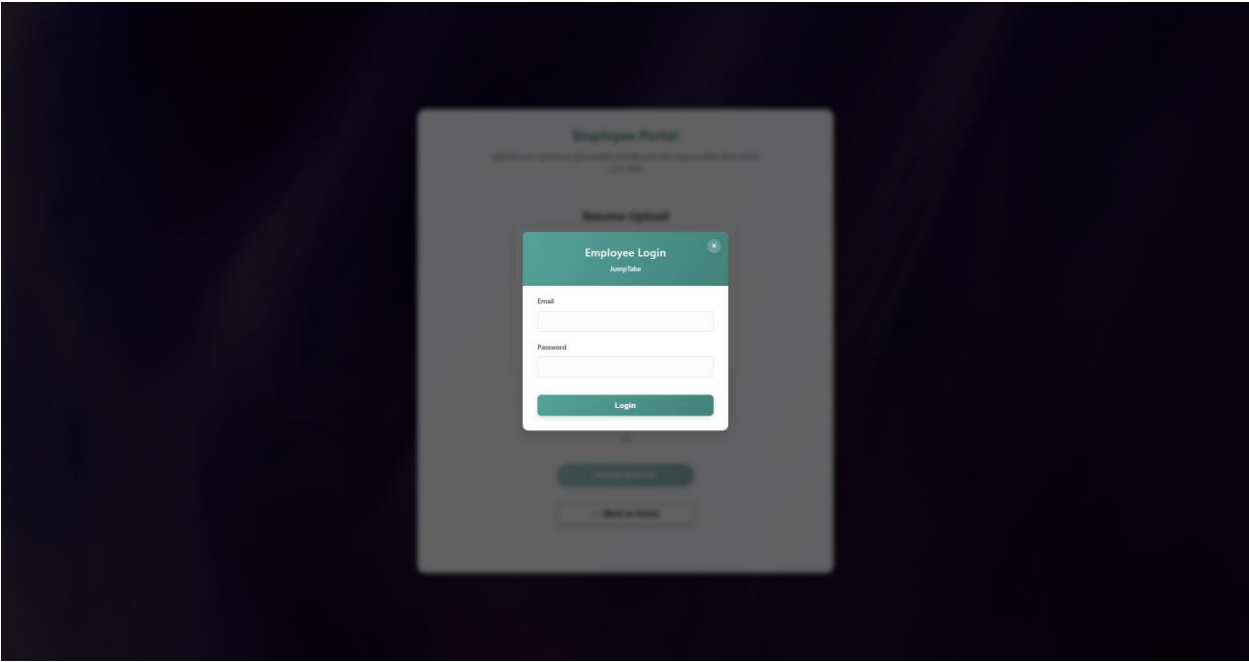
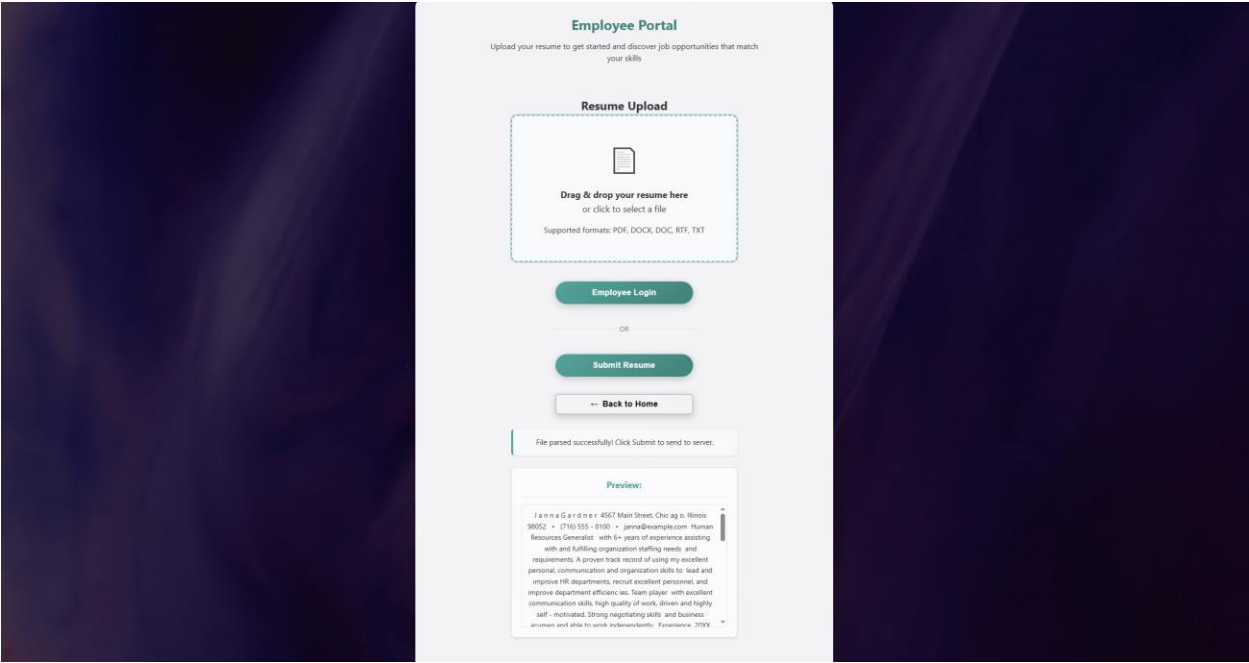
## 2. For Employees (Job Seekers)

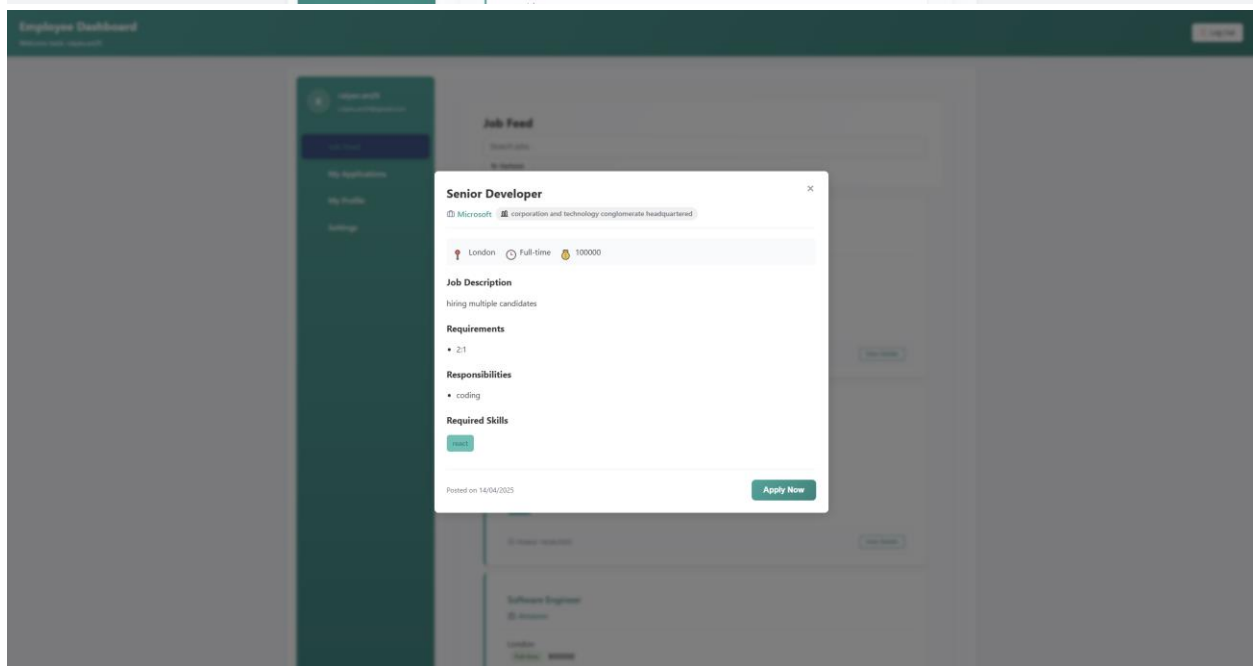
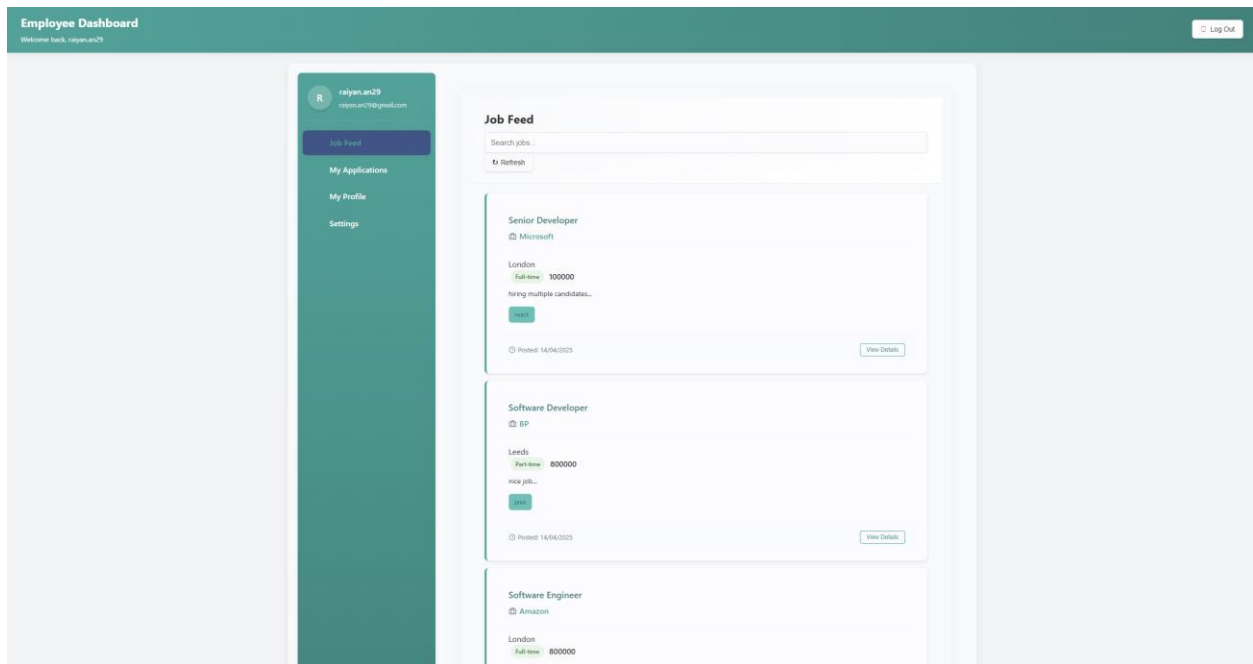
- **Choosing Role:** Click the Employee button on the homepage.
- **Uploading Resume:** On the job seeker portal, users upload their resume.

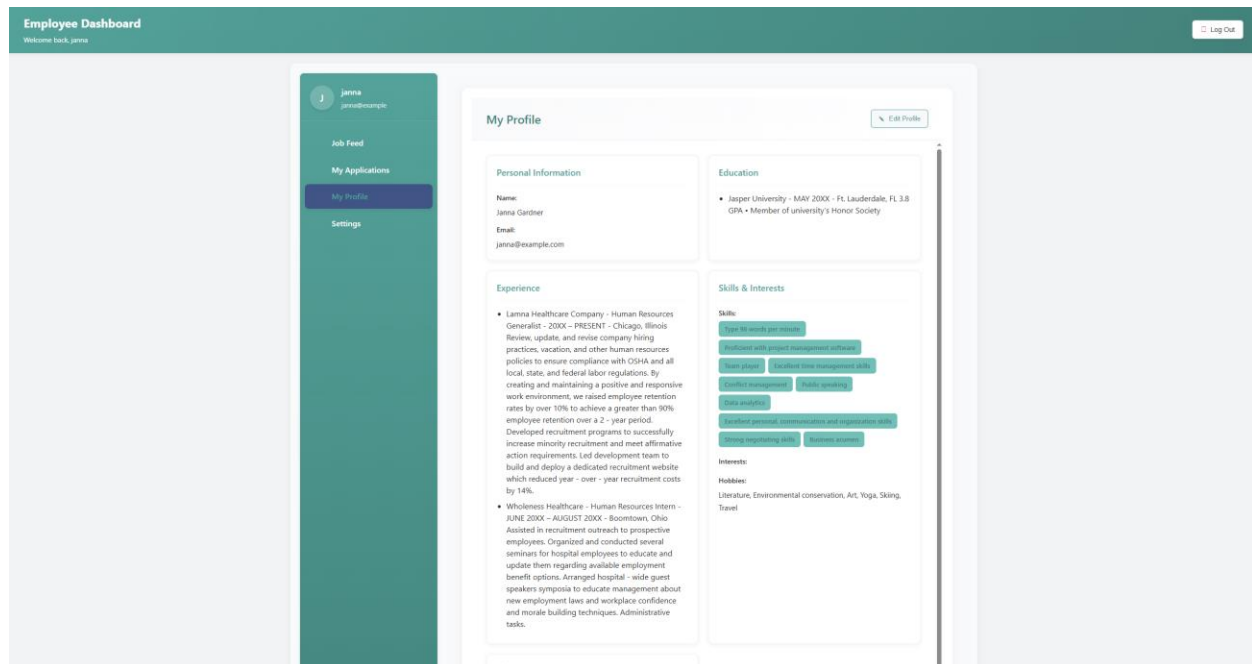
- **AI Resume Parsing:** The platform uses Google Gemini API to extract structured resume data such as: Name, Email, Education (institutions, dates), Experience (roles, companies, dates), Skills, Achievements, Interests, Hobbies.
- **Account Creation:** Resume data is stored in the backend (JobSeeker model) and linked to the user's account.
- **Job Matching:** AI suggests jobs based on resume content.
- **Application Tracking:** Users can view applied jobs and their statuses.
- **Profile Updates:** Users can edit their information for improved recommendations.

## Screenshots:



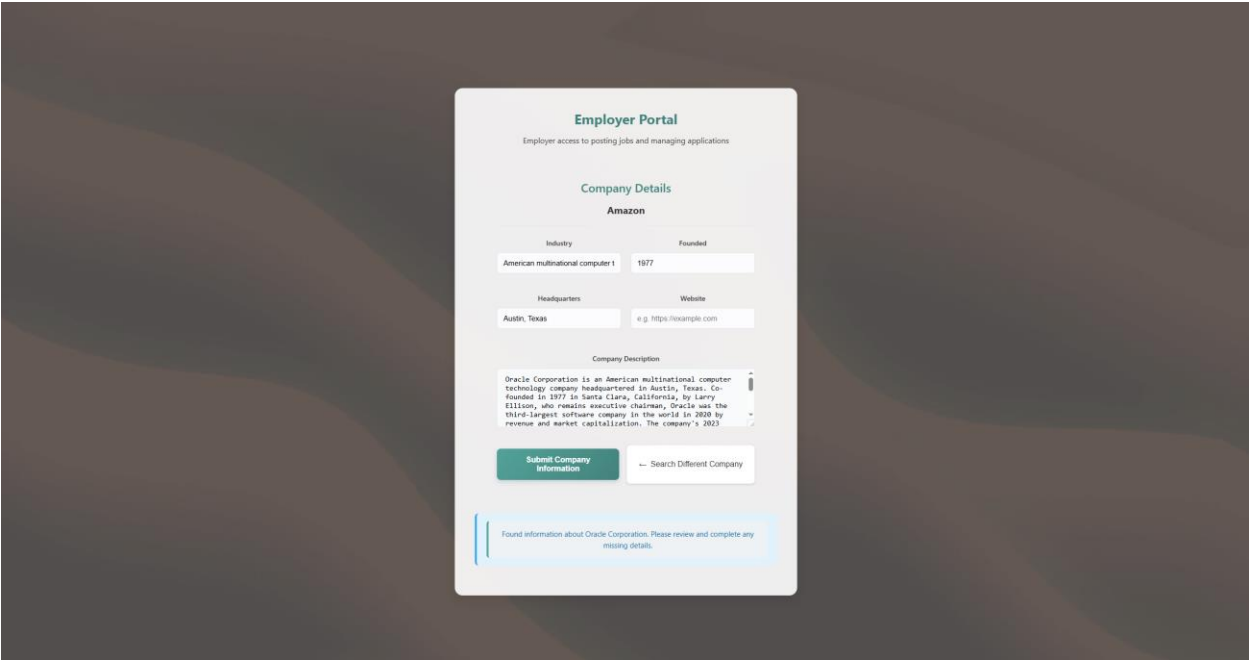
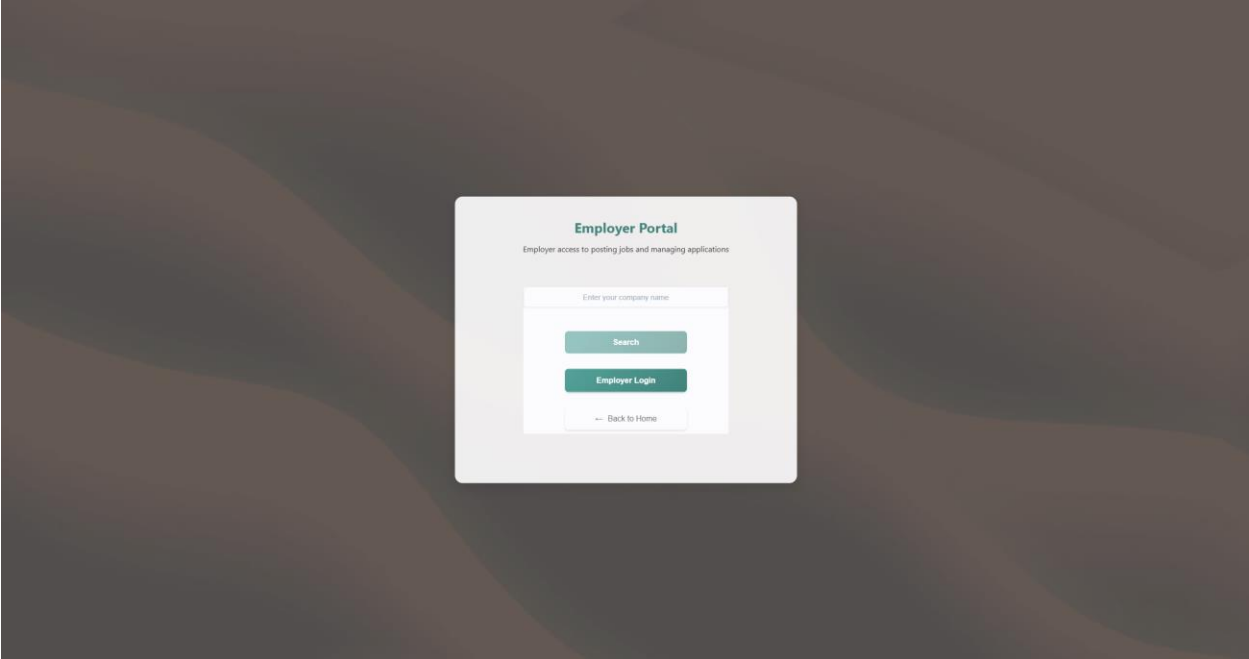


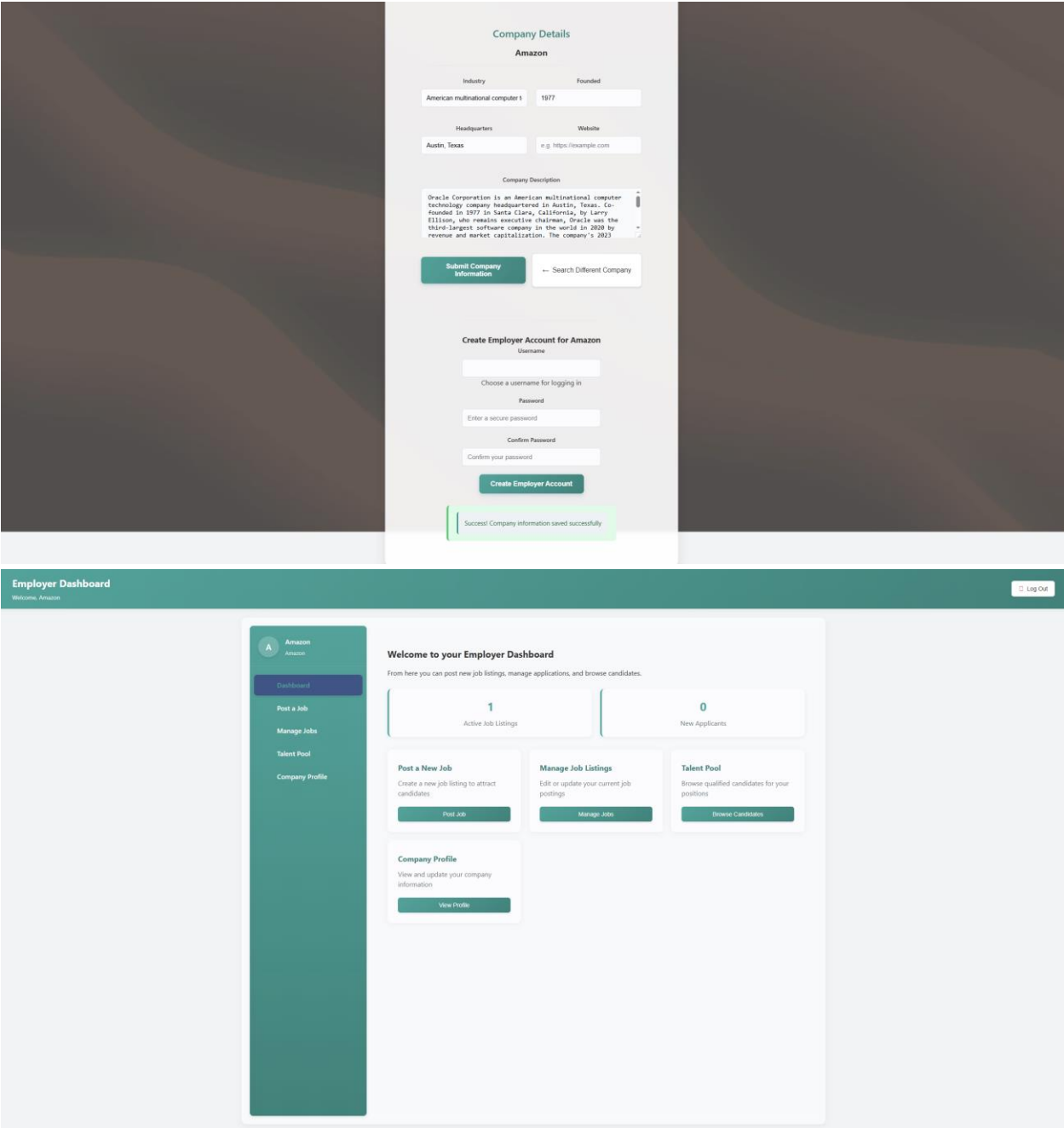




### 3. For Employers

- **Choosing Role:** Click the Employer button.
- **Register/Login:** Access the portal by creating an account or logging in.
- **Posting Jobs:** Employers can create listings with specific criteria.
- **Resume Matching:** The AI recommends candidates based on job requirements.
- **Managing Applications:** Employers track candidate statuses and communicate with them.
- **Company Stats:** Employers can view metrics such as total jobs, applications, and active listings.





#### 4. Key Features Summary

**AI-Powered Resume Parsing:** Powered by Gemini API for structured data extraction.

**Job Matching Algorithm:** Matches users with jobs based on resume data.

**One-Click Apply:** Users can apply instantly using their profile.

**Application Tracker:** Real-time status updates for job applications.

**Company Dashboard:** Post jobs, manage listings, and track applications.

## Product Development Artifacts

### Mockups and Wireframes (UI/UX Design System)

Landing.js and other frontend components mirror the main.css-defined UI styling and component behavior:

Shades of teal (#42a49b, #63c0b8, #34837b) are the primary colors.

Dark navy serves as a secondary color for the footer and text contrast. System and web-safe fonts are used in the text styles.

**Layouts:** Grid and flexbox are used in responsive design.

**Elements:** 'How It Works' steps, card layouts, movie backdrops, and role buttons.

### Relationship Diagram for Entities (MongoDB Schema)

Models and their primary domains -

**User:** role, jobSeekerId, password, and email

**JobSeeker:** user (ref), name, email, skills, and resume text

**Company:** Name, industry, description, and website

**Employer:** user (ref), firm (ref)



**Application:** title, description, skills needed, company ID (ref), locationJob (ref), jobSeeker (ref), status and date.

## Relationships

- **1:1** between User ↔ JobSeeker or User ↔ Employer (depending on role)
- **1:M** between Company → Jobs
- **1:M** between Job → Applications
- **1:M** between JobSeeker → Applications

## Flowcharts for APIs

### Uploading & Parsing Resumes

The objective is to parse resume text, use AI to extract structured data, and save the results to the JobSeeker model.

1. A user uploads a resume to the JobSeeker website.
2. Use { resumeText } to POST /resume/parse
3. The resumeController was the destination. handleResume()
4. ProcessResumeWithGemini is called by the controller using resumeText.
5. Structured JSON is returned by the Gemini API.
6. Original text and parsed data are saved to MongoDB (JobSeeker).
7. { message, jobSeekerId, parsedData } is the API response.

## Creation of a Job Listing

**Objective:** Companies list job openings associated with their business.

1. The employer uses a form to submit the job.
2. POST /jobs with companyId and job data

3. Forwarded to the `jobController.createJob()` function
4. The job was stored in MongoDB.
5. To retrieve metrics, GET `/companies/:id/stats`
6. The backend calculates the overall number of jobs, current jobs, and applicants.

## **Project Plan Versions**

### **Version 1.0 (2025-01-02): Initial Scope - Python based Prototype**

The project's original scope was for using Flask for routing and integrating with AI APIs for resume processing, with Python serving as the backend language. For local NLP processing, libraries such as spaCy, docx2txt, and PyMuPDF were taken into consideration. However, this strategy was abandoned because to slower development cycles, unfamiliarity with Flask, and difficulties integrating React frontend-backend.

### **Version 1.1 (2025-01-12): Transition to JavaScript Stack**

To test endpoints more quickly, the stack leaned towards JavaScript in this version, investigating combinations like vanilla JS and Express. The goal was to minimize incompatibilities by bringing backend languages into line with React. Although this method made frontend communication easier, plain JS's lack of structure and performance problems prompted researchers to look into more suitable frameworks.

### **Version 1.2 (2025-02-01): Restart with Node.js and React.js**

Using React.js for the frontend and Node.js for the backend, the project was resumed, utilizing their inherent integration. The basis for integrating AI through the use of the Google Gemini API and

Wikipedia API enrichment was established, along with the introduction of MongoDB as the database. This iteration, with its well-organized planning and simple architectural choices, signaled the true beginning of development.

### **Version 2.0 (2025-02-17): Milestone Adjustments Post-API Integration**


Following preliminary testing using the Wikipedia and Google Gemini APIs, integration turned out to be more difficult than anticipated. Rework was required for error handling, inconsistent API replies, and parsing delays. In order to concentrate more on AI prompt enhancement, API fallback logic, and a more reliable data pipeline, this version required modifying project timeframes and moving milestones.

### **Version 2.1 (2025-03-20): Final Plan with Deployment Phases**

Frontend polish and a deployment strategy were included in the project plan's final draft. Finalized features included real time resume processing, error handling, UI responsiveness, and JWT-based security. Small UX improvements were made, such as dashboard visual hints and one click apply. For post-submission enhancement, a roadmap for future test automation, scalability across job queues, and reworking AI prompts was also recorded.

## **Ethical Approval**

**Screenshot:**



**LEEDS  
BECKETT  
UNIVERSITY**

Research Ethics Online

New Application | My Applications

c3653589 | Logout

## My Applications



**New Application**  
If you wish to submit a new application, click on 'New Applications' above.

**Existing applications**  
If you wish to edit an existing application prior to submission, click on the Application Title or select the 'Edit/Continue' button.

If you have submitted an application and now need to make changes to it, click on the 'Make Revision/Copy' button. Please add to the title the version number (for example, v2).

10 records per page

Search: approved

Title	Risk Category	Status	Date Created	Action
ATS based Job seeking Web Application Platform	Risk Category 1	Approved by supervisor	11-MAR-25	 

Showing 1 to 1 of 1 entries (filtered from 8 total entries)

Previous

1

Next





### Contact Us

+44 (0)113 812 000

### Find Us

Leeds Beckett University,  
City Campus,  
Leeds, LS1 3HE

### Connect with Us

Contact Us | Disclaimer | Accessibility | Privacy

© Copyright Leeds Beckett University 2018

## Conclusion

The JumpTake project is an effort to create a useful, scalable, and user-focused job site by fusing open data integration, artificial intelligence, and contemporary web construction. Which was developed with Node.js, React.js, and MongoDB and enriched with Google Gemini and Wikipedia APIs, connects businesses and IT job seekers by providing dynamic job listings, contextual corporate enrichment, and intelligent resume parsing.

JumpTake is fundamentally motivated by the following problem statement: How can job boards be made more intelligent, efficient, and beneficial for recruiters and candidates alike? Conventional job boards frequently fall short of utilizing contemporary AI technology and necessitate a significant amount of manual labor and repetitive data entering. JumpTake uses powerful natural

language processing (NLP) to automate the resume parsing and job matching process in an effort to overcome these restrictions. Additionally, JumpTake reduces data entry fatigue for businesses by automatically filling in organizational information from public APIs such as Wikipedia.

The use of AI-powered resume parsing is JumpTake's most notable advancement. The platform employs generative AI to extract information with contextual awareness, in contrast to conventional keyword-based filters. Although it is implemented in a manner that is accessible to mid-level projects and instructional use cases, this mimics real-world ATS systems.

The second innovation is the enrichment of corporate data through the use of free knowledge repositories, particularly Wikipedia. By using publicly maintained resources to populate verified, standardized details, this reduces onboarding hurdles for employers and enhances listing confidence.

The original goal of developing an intelligent, feature-rich job board had to be reduced to a functional MVP (minimum viable product) that concentrated on the following key flows: company listing, resume upload, job matching, and parsing.

The lack of automated testing was yet another significant drawback. Throughout the development process, functional live checks were carried out; however, neither formal unit tests nor end-to-end integration tests were used. Regression risk may be introduced in subsequent iterations, and trust in long-term scalability is weakened.

Furthermore, synchronous resume parsing was found to have scalability issues. Processing is halted until a complete Gemini API response is obtained for each resume upload. This will cause latency at larger user loads. Integrating a queuing system (like BullMQ) and turning on background processing might be a better strategy.

Gaps in frontend error-handling were also present. Clear user messages or retry choices were not always displayed when an API failed. Enhancing UI feedback methods would improve the platform's usability and user confidence.

Project management remained stable in spite of these difficulties, with milestones and plan versions that were well-defined. The roadmap shows careful, agile development, beginning with exploratory Python (v1.0) attempts, moving on to JavaScript exploration (v1.1), and then settling on a stable Node.js + React stack (v1.2).

To sum up, JumpTake is an effective example of fusing open data APIs, AI, and full-stack development to create a useful, innovative web application. Although there was some technical ambiguity at first, the goal's clarity and ongoing learning transformed it into a well-defined, workable solution. Incorporating AI not only benefits the user but also lays the groundwork for intelligent employment tools in the future. This project is more than just a technical development; it

emphasizes the value of flexibility, research-based design, and moral behavior in the quickly changing field of AI-powered applications.

## Professional Conduct & Compliance

BCS Code of Conduct -

<https://www.bcs.org/membership-and-registrations/become-a-member/bcs-code-of-conduct/>

## References

Here are all the references from your report, compiled in **APA 7th edition format**, clearly separated and organized into proper entries. These include API documentation, web tools, academic references, and development resources.

1. **Google Developers.** (2024). *Generative Language API: Overview & Documentation*. Retrieved from <https://ai.google.dev/docs>
2. **MediaWiki.** (n.d.). *Wikipedia API Documentation*. Retrieved from [https://www.mediawiki.org/wiki/API:Main\\_page](https://www.mediawiki.org/wiki/API:Main_page)
3. **Mozilla Developer Network (MDN).** (n.d.). *Using Fetch API*. Retrieved from [https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)
4. **GitHub – Mozilla.** (n.d.). *pdfjs-dist: A general-purpose, web standards-based platform for parsing PDF files*. Retrieved from <https://github.com/mozilla/pdfjs-dist>
5. **GitHub – Mammoth.js.** (n.d.). *Convert .docx documents into plain text*. Retrieved from <https://github.com/mwilliamson/mammoth.js>
6. **Stack Overflow.** (n.d.). *Wikipedia API integration examples and filter discussions*. Retrieved from <https://stackoverflow.com>
7. **Gabrilovich, E., & Markovitch, S.** (2007). *Computing semantic relatedness using Wikipedia-based explicit semantic analysis*. Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI).
8. **Facebook Engineering.** (2013). *React: A JavaScript library for building user interfaces*. Retrieved from <https://reactjs.org>

9. **Wang, Y., Xu, Z., & Yang, X.** (2018). *Modern Frontend Development Using React and Redux*. International Journal of Web Engineering and Technology, 13(1), 23-38.
10. **Xie, Y., Wang, W., Wang, L., & Wang, Y.** (2016). *Semantic Resume-Job Matching Using Deep Learning*. Proceedings of the IEEE International Conference on Big Data (Big Data), 2323-2332.
11. **Huang, Z., Liang, H., & Zhang, Q.** (2020). *AI-powered Resume Parsing using NLP and Machine Learning*. International Journal of Data Science and Analysis, 6(4), 145–152.
12. **BCS – The Chartered Institute for IT.** (2024). *BCS Code of Conduct*. Retrieved from <https://www.bcs.org/membership-and-registrations/become-a-member/bcs-code-of-conduct/>

## AI Tools & Platforms Referenced in Appendix

13. **GitHub Copilot.** (n.d.). *AI pair programmer for writing code*. Retrieved from <https://github.com/features/copilot>
14. **Visual Studio Code – GitHub Copilot Integration.** (n.d.). *Using AI assistance for debugging and development*. Retrieved from <https://code.visualstudio.com/>
15. **OpenAI – ChatGPT.** (2024). *GPT-4 Mini Model*. Retrieved from <https://chatgpt.com/?model=o4-mini>
16. **Canva.** (n.d.). *Design and visual editing tool*. Retrieved from <https://www.canva.com/>

## Originality GenAI Statement

I understand that to use the work and ideas of others, including generative AI output, without full acknowledgement, is academic unfair practice.

I confirm that this coursework submission is all my own, original work and that all sources, summaries, paraphrases and quotes are fully referenced as required by the LBU Academic Regulations.

## DECLARATION OF GENERATIVE AI USE:

I DID use Generative AI technology in the development, writing, or editing of this assignment. I have included a copy of the reference used, and the output generated in the Appendix section. If you have any concerns about whether you have used generative AI tools (in)correctly, please seek advice before submitting your assignment from academic staff responsible for the assessment that this submission relates to. The shaded boxes are examples to help you with completing this section.

Appendix	Generative AI Tool (e.g. ChatGPT)	How generative AI Tool was used	Reference
A	Github Copilot	Used to generate some additional styling of the webpage for editing and customization.	(Claude 3.5 Sonnet) <a href="https://github.com/features/copilot">https://github.com/features/copilot</a>
B	VS Code	Used for debugging and some logic handling.	Visual Studio Code Copilot (Claude 3.5 Sonnet)
C	ChatGPT	Used to get ideas about the report and researching along with some writing of Terms and Conditions.	GPT o4 mini ( <a href="https://chatgpt.com/?model=o4-mini">https://chatgpt.com/?model=o4-mini</a> )
D	Canva	Used elements for styling.	<a href="https://www.canva.com/">https://www.canva.com/</a>