

**EUROPEAN UNIVERSITY OF LEFKE**

**FACULTY OF ENGINEERING**

**Graduation Project 2**

# Online Recruitment System

**Dias Mukash**

**194072**

An **online recruitment system** is a web application that allows organizations to efficiently manage their hiring process by posting job openings, receiving and reviewing job applications, and communicating with applicants entirely online. The main idea behind an online recruitment system is to streamline and automate the recruitment process, making it easier for organizations to find and hire the best candidates for their open positions. This type of system can be used by businesses of all sizes, in any industry, to manage their recruitment needs. The goals of my recruitment solution includes the improving the efficiency and speed of the recruitment hiring process, with increasing the number of qualified candidates that apply for open positions, and reducing the workload of HR staff. Online recruitment system provides a convenient and user-friendly experience for applicants and employers so that they communicate in the best way, allowing them to easily apply for jobs and track/manage the applications.

**Supervisor**

Asst. Prof. Dr. Zafer Erenel

12.06.2023

## Table Of Contents

Dias Mukash .....	i
194072.....	i
Asst. Prof. Dr. Zafer Erenel .....	i
12.06.2023 .....	i
1. Introduction.....	1
1.1    Problem definition .....	1
1.2    Goals .....	2
2. Literature Survey .....	3
3. Background Information.....	4
3.1 Required & Used software.....	4
3.2 Other software.....	5
4. Design Documents .....	6
4.1    Data flow diagram.....	6
4.2    Context Diagram.....	7
5. Methodology .....	8
6. Conclusion .....	31
6.1    Benefits .....	31
a.    Benefits to users : .....	31
b.    Benefits to me : .....	31
6.2    Ethics.....	32
6.3    Future Works .....	33
7. References.....	34

# 1.Introduction

The developing digital landscape and the need for remote operations have increased the demand for effective online recruitment systems. In this context, I have developed a comprehensive, user-friendly, and efficient recruitment system using the MERN stack (MongoDB, Express.js, React.js, and Node.js). This system facilitates a seamless hiring process by allowing employers to post jobs, review applications, and manage the hiring pipeline, while simultaneously enabling job seekers to apply for jobs, track their application status, and communicate with potential employers. Also, I decided to add a feature like AI chat widget, it's an interface for communicating with ChatGPT, an OpenAI's product by using their API, this feature can be useful in many ways, both by applicants and employers and also will demonstrate the possibilities of modern AI systems.

## 1.1 Problem definition

Traditional recruitment processes are often time-consuming and fraught with inefficiencies, making them costly for businesses (Frye, 2019). The increasing need for remote working and hiring has amplified these challenges. Some of the problems that my project aims to address are:

- **Fragmented Communication:** Traditional recruitment methods often involve different platforms for job postings, application tracking, and candidate communication, leading to fragmented and uncoordinated interactions.
- **Inefficient Application Review:** Reviewing applications and shortlisting candidates often require manual processes, which can be time-consuming and susceptible to human error.
- **Lack of Transparency for Candidates:** Candidates often have limited visibility into their application status, leading to uncertainty and potential dissatisfaction.
- **Limited Accessibility:** Traditional job posting platforms may not be accessible to all potential candidates, limiting the diversity of the applicant pool.
- **Lack of CV viewing features.** Almost all of the traditional recruitment web applications on the market don't have a simple and user-friendly interface or even functionality to view and access candidates CV. My application solves that problem.

## 1.2 Goals

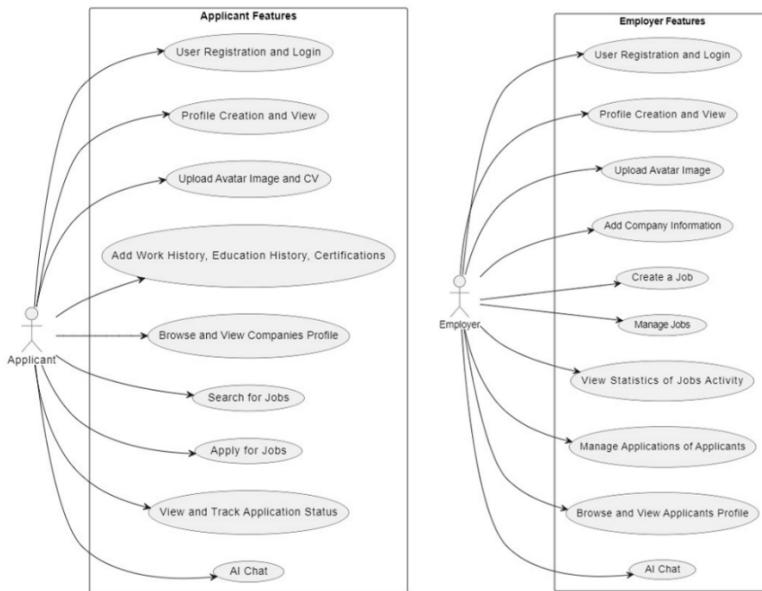
The goals of a project:

- To build an online web application for organizations to post jobs and for applicants to apply to these positions;
- To make the application process better for both companies and job seekers by allowing the job submission and review of applications and CV online;
- Create a user-friendly and minimal interface that makes the process of interaction between the two parties better;
- Ensure equal access to all populations, races, nationalities, ages and orientations (Gurchiek, 2021);
- To allow for the tracking of applicants and the status of their application;
- Build a web app using the right architecture, principles and methodologies;
- Use the modern software dev frameworks and technologies to build the fully working and industry standard web application;
- Provide the good cybersecurity and confidentiality of users info and data;

Also, one of the main principles of my online recruitment system is to follow the good right methods of the application development, with the good architecture and principles, software development methods in order to make it easy to maintain the application in the future.

Overall, my project aims to create a comprehensive recruitment system that addresses the current shortcomings of existing solutions and helps companies and job seekers connect more effectively.

**Use case diagram:**



## 2. Literature Survey

There are many online recruitment systems that have been developed in the past, such as LinkedIn and Indeed. These systems allow companies to post jobs and for applicants to search these jobs and apply for them. However, there are some limitations to these systems. LinkedIn, for example, has a focus on networking, blog posts and job search rather than the actual application process, and may not have all the necessary features for a comprehensive recruitment system (Koch, Gerber, & J. De Klerk, 2018). But still recruiters cannot perform effective recruitment process without using LinkedIn. Indeed, allows for the posting and searching of job openings, but the application process may not be as streamlined as desired.

My project's purpose is to improve upon these existing online recruitment systems by including all necessary features for efficient recruitment process and providing user-friendly interface for both companies for both companies and job seekers. Also, this project is different because I've included some additional features, such as Resume tools and AI chat to enhance the user experience.

**Comparing with LinkedIn:** LinkedIn is a professional social network platform for networking that allows its users, customers to connect with other users, search for a colleagues, search for jobs, apply for positions, etc., also it has blogs, so users can create blog posts and share it. Comparing to LinkedIn my online recruitment system is focused more on the job postings and all the necessary instruments for applying for jobs and tracking the applications.

**Comparing with Indeed:** Indeed is a web app that stores jobs from different sources, works like an aggregator, including company websites, recruitment systems, etc. It allows customers to search jobs by location, job title, and so on, like filters. In comparison with Indeed my online recruitment system has the same features like search by location, job titles and company, including results by job type, the filtering and also has a user-friendly interface, which is making the process easy and smooth.

**Comparing with Glassdoor:** Glassdoor is a job search web app that offers companies reviews, salary information, and job listings, etc. Glassdoor also offers employer branding and recruitment ads services. My online recruitment system in comparison with Glassdoor offers more comprehensive and user-friendly recruitment solution for many organizations.

While LinkedIn, Indeed, Glassdoor and other apps and platforms focus on job search and employer branding mainly, my project makes it a more integrated and streamlined recruitment process, including the ability to manage and track candidates throughout the hiring process.

### **3. Background Information**

My online recruitment system is using a combination of modern tools, technologies and programming languages. On the frontend side of the online recruitment system I used modern and robust framework ReactJS and Material UI (MUI) design system to make a user-friendly interface (Banks & Porcello, 2020). As an addition the Redux Toolkit is used for state management with ReactJS. React allows to create reusable components and its easier to manage and maintain them later, Redux Toolkit makes the state management more efficient, it's an alternative way of storing the data and actions and dispatching them. In the good frontend applications there're 2 abstract layers, one layer is called BLL (Business Logic Layer), which is responsible for business-oriented functionality, other layer can be called as UI (User Interface), which consists of the components responsible for rendering the layout of the application. According to the principle of Single Responsibility the data from the business layer is not recommended to be much used in the UI layer.

For the backend side of my application I chose NodeJS platform, cause it gives the code consistency, since it's using JavaScript language and it is a powerful framework for creating asynchronous functions, making the app work faster and effective (Casciaro & Mammino, 2020). NodeJS is based on Libuv engine and V8. Together they create the core of the framework. NodeJS is using cross-platform I/O operations and non-blocking algorithm, which allows to process a large number of queries per unit time.

#### **3.1 Required & Used software**

- ReactJS:**

React is a widely used JavaScript library for creating user interfaces and it offers several benefits for developers. One benefit is the ability to efficiently update and render components, which can improve the performance of applications. Another plus is React's flexibility and reusable components, that might save time and resources in the software dev. process.

- NodeJS:**

Node.js is a runtime environment for building server-side applications, and it offers several benefits for developers. One plus is its ability to handle a big number of concurrent connections with its high performance, making it very suitable for real-time applications. Another plus is its huge ecosystem of libraries and tools, which can reduce the time and effort required to build and deploy the applications.

- MongoDB:**

MongoDB is an open-source DBMS (database management) system that uses a document-oriented model to store data. It offers several benefits for developers. One benefit is its scalability, which allows it to easily handle large amounts of data and support the growth of applications. Another plus is its flexibility, since it allows developers to store and manipulate data in a variety of formats, including JSON and BSON (Bradshaw, Brazil, & Chodorow, 2019).

- **ExpressJS:**  
A simple and adaptable open-source Node.js framework called Express.js is used for web applications. It can be applied over Node.js to improve web functionality. The most widely used Node.js web framework is Express.
- **Visual Studio Code:**  
Good choice for any sorts of projects, in my case it's best suitable for building the web apps.

### 3.2 Other software

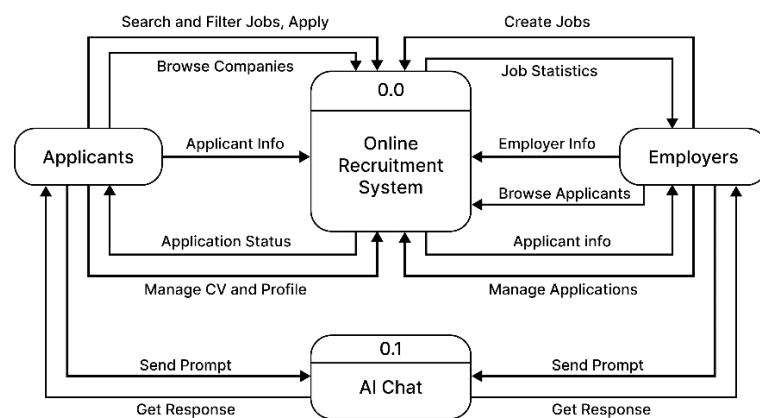
- **Figma:**  
Figma is a cloud-based design and prototyping tool that offers several benefits, including real-time collaboration, a wide range of design features, and the ability to easily share and collaborate with team members and stakeholders.
- **Git:**  
Git is a version control system that allows developers to track and manage changes to their code, and one of its main benefits is its ability to allow multiple developers to work on the same project concurrently while keeping track of all changes.
- **Postman:**  
Postman is a software service for testing API (Application Programming Interface). Basically, it's a HTTP-client for sending request and getting response from a server, interface for data exchange between two applications of software components. I used Postman to test my API (server side of my project)

## 4. Design Documents

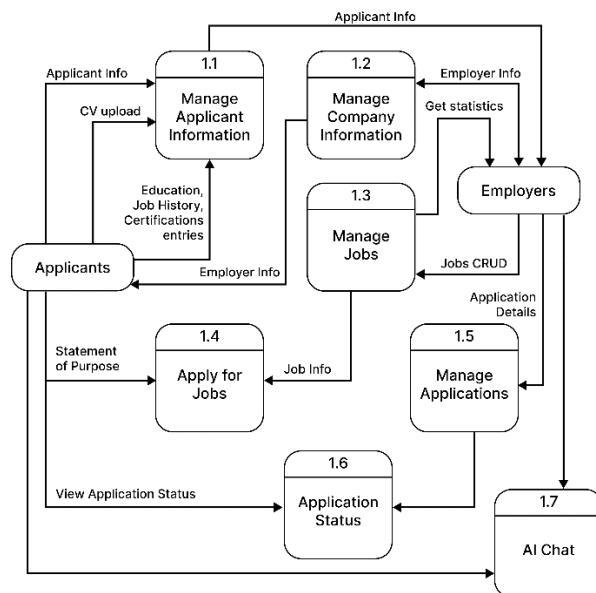
### 4.1 Data flow diagram

This is the application module that's represented in the DFD diagrams, you can see the applicants and employers. As it's seen here the main system is the Online Recruitment System itself along with the processes for both roles in my application. Also, there's a second application here which is AI Chat. From these diagrams you can obtain the basic overview of the entire system.

**Online Recruitment System DFD level 0**

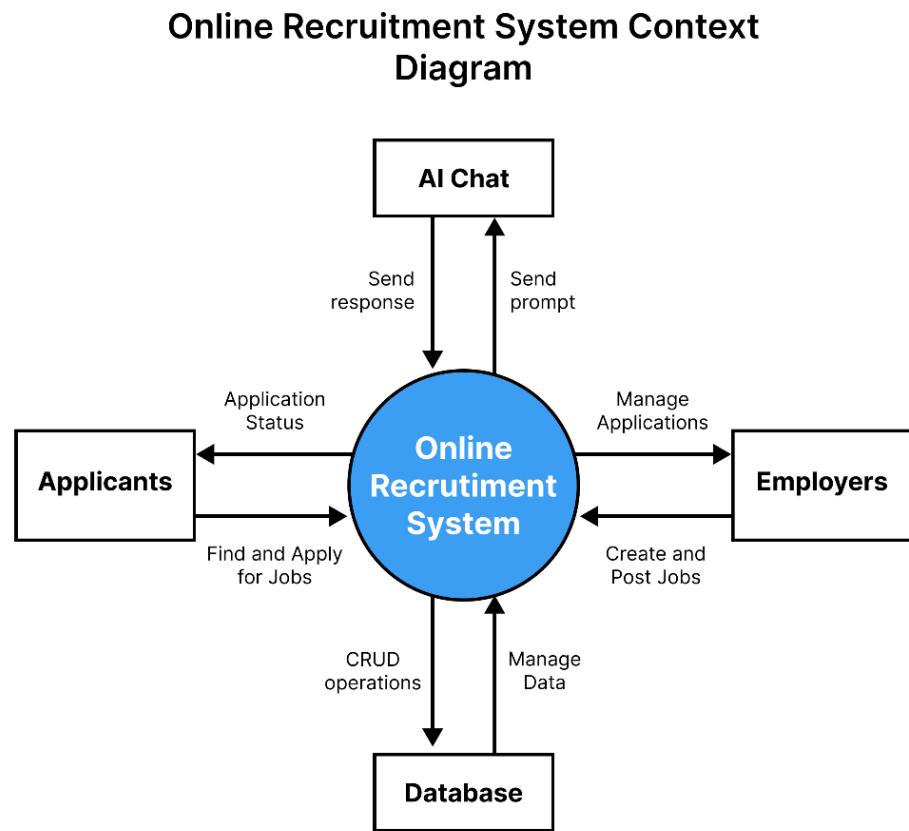


**Online Recruitment System DFD level 1**



## 4.2 Context Diagram

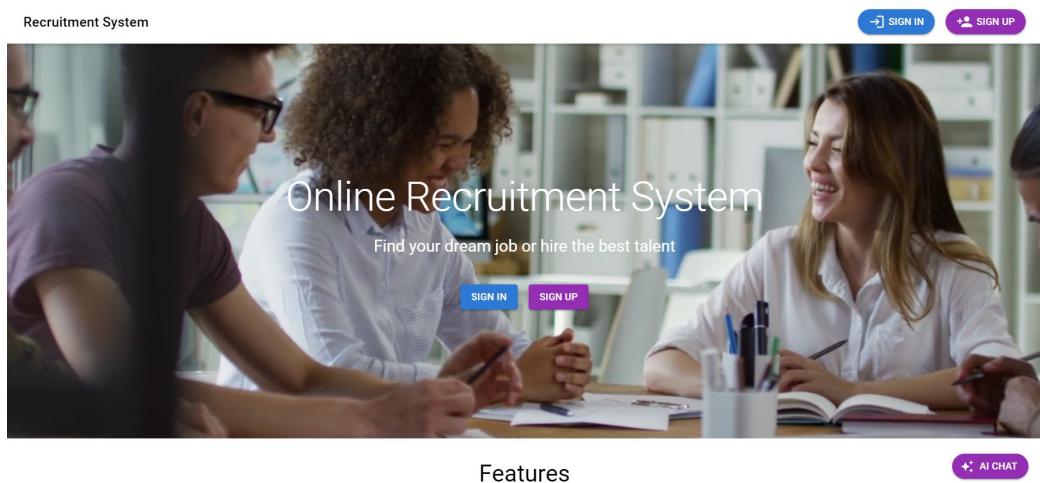
From this context diagram it is visible how the external entities can interact with the online recruitment system. This high-level abstraction makes it easy to display the main processes of an application.



## 5. Methodology

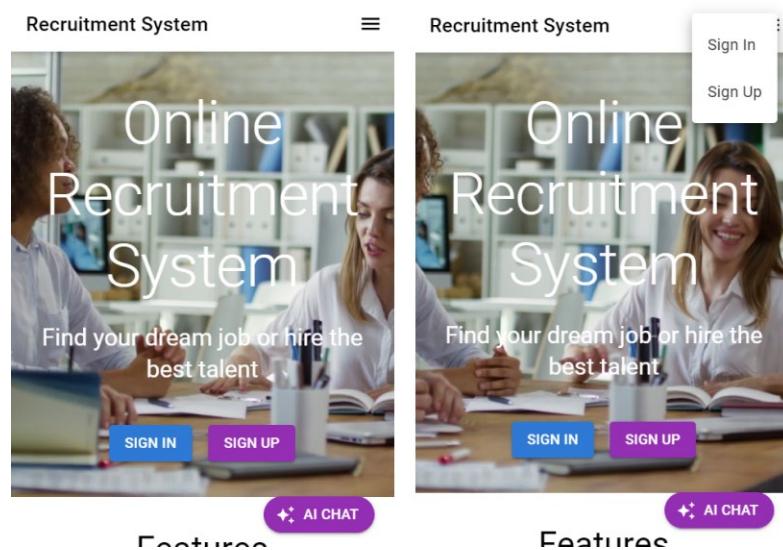
As I mentioned before, for my online recruitment system I chose MERN stack. The common approach in web development is to start from the backend development, but I chose the simultaneous approach to develop the frontend and backend at the same time. This makes the process clear and less error prone, since I can see the result of my work faster and debug the code and test it.

Here's my welcome page. I added the animated hero section and CTA (Call to action) buttons, like "Sign In" and "Sign Up".



Features

One of the main focuses for building a user-friendly interface is responsiveness, so that users can use the application from their smartphones and the layout will change accordingly, making the user experience more like using a mobile application (What is Responsive Design?, 2022).



For frontend design I used the Material UI design system, since it's an industry standard way of designing the user interfaces together with ReactJS, and it reduces the boilerplate code written in CSS, making the app consistent and for developers to maintain the code better and focus more on the functional aspects of a project.

Here are the Sign In and Sign Up forms, where you can choose the roles, whether it's an Applicant or Employer:

For the user input I implemented data validation both on the frontend and backend side to follow the best practices. Regular expressions are used to ensure that user entered the email in correct form, for the frontend validation I used the react-hook-form library.

Below is the code snippet responsible for the frontend data validation:

```

53   <Box component="form" noValidate sx={{ mt: 1 }} onSubmit={handleSubmit(onSubmit)}>
54     <TextField
55       margin="normal"
56       required
57       fullWidth
58       id="email"
59       label="Email Address"
60       name="email"
61       autoComplete="email"
62       autoFocus
63       {...register("email", {
64         required: "Email is required",
65         pattern: {
66           value: /^[\\w-]+(\\.\\w-)*@[\\w-]*\\.[a-zA-Z]{2,7}$/,
67           message: "Wrong email format",
68         },
69       })}
70       error={!errors.email}
71       helperText={(errors.email && errors.email.message)}
72     />
73     <TextField
74       margin="normal"
75       required
76       fullWidth
77       name="password"
78       label="Password"
79       type={showPassword ? "text" : "password"}
80       id="password"
81       autoComplete="current-password"
82       {...register("password", {
83         required: "Password is required",
84       })}
85       error={!errors.password}
86       helperText={(errors.password && errors.password.message)}
87       InputProps={{
88         endAdornment: (
89           <InputAdornment position="end">
90             <IconButton
91               onClick={togglePasswordVisibility}
92               edge="end"
93             >
94               {showPassword ? <VisibilityOff /> : <Visibility />}
95             </IconButton>
96           </InputAdornment>
97         ),
98       })
99     />

```

If user entered wrong credentials the app will notify the user.

The screenshot shows the 'Sign in' page of the 'Recruitment System'. At the top, there is a navigation bar with tabs for 'APPLICANT' and 'EMPLOYER'. Below the navigation bar is a form with two input fields: 'Email Address \*' containing 'applicant1@example.com' and 'Password \*' containing '\*\*\*\*\*'. A blue 'SIGN IN' button is located below the form. Above the form, a pink notification bar displays the message 'Invalid email or password' with an exclamation mark icon and a close button ('X').

Here's the additional form validation:

The screenshot shows the 'Sign up' page of the 'Recruitment System'. The page includes fields for 'First Name \*', 'Last Name \*', 'Email Address \*', and 'Password \*'. Below the 'SIGN UP' button, a link says 'Already have an account? Sign in' and an 'AI CHAT' button is visible. A callout box labeled 'Additional password validation to make it secure' points to the 'Password \*' field, which is highlighted with a red border. Two arrows point from this callout to two separate validation messages: 'Password must be at least 8 characters' and 'Password must contain at least 1 uppercase letter and 1 number'. The original 'Password \*' field contains '\*\*\*\*\*'.

User can choose roles to Sign Up and Sign In. Let's start from an employer's perspective. Once employers have successfully logged in into the application, they're getting to the dashboard. The page has navbar displaying the "Employer Dashboard" and the user information on the top right with is the dropdown menu with "Profile", "Settings" and "Logout" options.

The screenshot shows the "Employer Dashboard" interface. At the top, there's a navigation bar with the title "Employer Dashboard" and a user profile icon for "Creative Solutions, Inc." and "employer1@example.com". Below the navigation bar is a cartoon illustration of a person sitting at a desk with a laptop, another person pointing at a screen displaying three people, and a person jumping in excitement. A large "Welcome to your dashboard, Creative Solutions, Inc.! " message is centered below the illustration. On the left side, there's a vertical sidebar with icons and text for "Create Job", "Manage Jobs", "Applications", "Applicants", "Profile", "Settings", "Additional Features", and "HR Resources". At the bottom left is a "Log Out" button, and at the bottom right is an "AI CHAT" button.

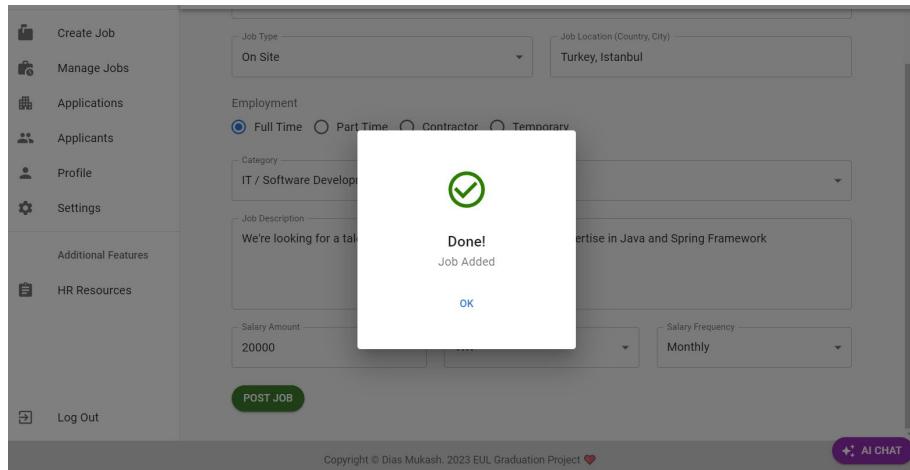
**Create jobs page.** Here employers can post jobs. There're many properties for making a job post, like "Job Title", "Job Type", "Job Location", "Employment", "Category", "Job Description", "Salary Amount", "Currency", "Salary Frequency" and "Post Job" button.

The screenshot shows the "Create Job" page within the "Employer Dashboard". The page title is "Create Job". It features several input fields: "Job Title" (text input), "Job Type" (dropdown menu with "On Site" selected), "Job Location (Country, City)" (text input), "Employment" (radio buttons for "Full Time" (selected), "Part Time", "Contractor", and "Temporary"), "Category" (dropdown menu with "IT / Software Development" selected), and "Job Description" (text area). At the bottom right is an "AI CHAT" button. The left sidebar is identical to the one in the dashboard, and a "Log Out" button is at the bottom left.

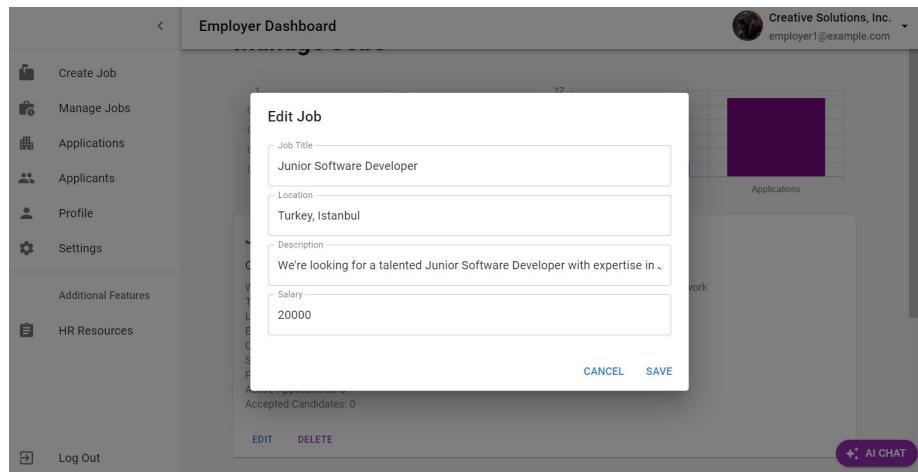
These fields are under the validation control as well, since employers cannot post an empty job posting, and for the "Salary Amount" field it strictly validates the value to be a number. All the fields are required, so when employer press the "Post Job" button, the new job posting is created.

A close-up view of the job posting form. It includes three dropdown menus: "Salary Amount" (with placeholder "Enter salary amount"), "Currency" (with "GBP" selected), and "Salary Frequency" (with "Anually" selected). Below these is a green "POST JOB" button.

Once the job is posted, the user gets the confirmation success dialog box:



**Manage jobs page.** In this page employers can see and manage the jobs that they've created along with the job information. If employers click on edit button, they can edit the necessary information for the particular job position.



Edit job form cannot be empty if an employer wants to change the information, again it's the form validation which makes the application secure and reduces potential errors and bugs.

A screenshot of the "Edit Job" form. The fields are: Job Title (Junior Software Developer), Location (Turkey, Istanbul), Description (empty, with validation error "Description is required."), and Salary (empty, with validation error "Salary is required."). At the bottom are CANCEL and SAVE buttons.

In the manage jobs page employers can see the statistics of the job activity, the number of accepted, shortlisted and rejected applicants, the number of jobs posted and the total applications count.

**Manage Jobs**

Junior Software Developer  
Creative Solutions, Inc.  
We're looking for a talented Junior Software Developer with expertise in Java and Spring Framework.  
Type: On Site  
Location: Turkey, Istanbul  
Employment: Full Time  
Category: IT / Software Development  
Salary: 20000 TRY (Monthly)  
Posted on: 04.06.2023  
Active Applications: 0  
Accepted Candidates: 0

**Applications tab.** In this page employers can manage the applications. It's represented as the centralized table view, data grid for tracking applicants. When applicants successfully submit their applications, employers are getting their information and can decide whether to shortlist, accept or reject them. The features here include getting the applicant's name, email, job title, download CV, manage applications and view the statement of purpose.

Application ID	First Name	Last Name	Email	Job Title
647c798ae4b2386d243cc...	Karl	Luxembourg	applicant1@example.com	Junior Software Develop...
647c79a2e4b2386d243cc...	Karl	Luxembourg	applicant1@example.com	Senior Software Develop...
647c79b2e4b2386d243cc...	Karl	Luxembourg	applicant1@example.com	Senior Team Leader

Here's the continuation of the table, you can see the “Download”, “Shortlist”, “Reject” buttons and “View SOP” buttons.

	Job Title	Resume	Status	SOP	
.com	Junior Software Developer	<a href="#">DOWNLOAD</a>	<a href="#">SHORTLIST</a>	<a href="#">REJECT</a>	<a href="#">VIEW SOP</a>
.com	Senior Software Developer	<a href="#">DOWNLOAD</a>	<a href="#">SHORTLIST</a>	<a href="#">REJECT</a>	<a href="#">VIEW SOP</a>
.com	Senior Team Leader	<a href="#">DOWNLOAD</a>	<a href="#">SHORTLIST</a>	<a href="#">REJECT</a>	<a href="#">VIEW SOP</a>

1 row selected      Rows per page: 100 ▾ 1–3 of 3 < >

Log Out      + AI CHAT

Here's the continuation of the table, you can see the “Download”, “Shortlist”, “Reject” buttons and “View SOP” buttons. When employers click on “View SOP” button, they can see the statement of purpose sent by applicants, statement of purpose makes the application process clear for applicants and companies, since it's a recommended step during the application process.

I am writing to express my strong interest in applying for the open position within your organization. As a diligent professional with 5 years in the Software and IT industry, I believe that my unique blend of skills, professional experience, and personal qualities aligns well with the role and the broader goals of your team. Throughout my career, I have consistently demonstrated a commitment to excellence, a knack for solving complex problems, and the capacity to work well under pressure. My extensive background in IT, makes me a well-suited candidate for this role.

CLOSE

When employers click on the “Download” button in the “Resume” field, they can view and download the resume. These days companies allocate resources to have an access to a CV database, since it's the most valuable asset in the recruitment systems. My application solves that problem by providing the Resume viewing and downloading options. The application opens a new page with CV after clicking on the “Download” button.

Linda Harris  
Test engineer CV

**PERSONAL SUMMARY**

A highly skilled, ambitious and self-motivated test engineer, with a strong technical background who possesses self-discipline and the ability to work with the minimum of supervision. Having a proven ability to carry out the creation of test cases, appropriate implementation of individual tests, log outcomes and communicate the results effectively. Possesses excellent communication skills, including written and verbal communication skills alongside an ability to formulate, advise and implement testing strategy. Using initiative to develop effective solutions to problems with an active and dynamic approach to work and getting things done efficiently.

Keen to find a suitable position with a successful and ambitious company.

**WORK EXPERIENCE**

**IT & Telecoms Company - Coventry** TEST ENGINEER June 2008 - Present

Responsible for the whole test process from planning, through test plan development, execution & result reporting. Also involved in the development and improvement of the test functions, putting forward suggestions and implementing plans accordingly.

**Duties:**

- Organising, conducting and supporting test activities.
- Responsible for performance testing and integration testing.
- Rapidly response to equipment failures & implementing immediate repairs.
- Participation in audits and review of testing, highlighting areas for improvement.

**PROFESSIONAL**  
ISB certification

Employers can manage the application by choosing the shortlist, reject or accept options.

Email	Job Title	Resume	Status
applicant1@example.com	Junior Software Developer	<a href="#">DOWNLOAD</a>	REJECTED
applicant1@example.com	Senior Software Developer	<a href="#">DOWNLOAD</a>	<a href="#">ACCEPT</a> <a href="#">REJECT</a>
applicant1@example.com	Senior Team Leader	<a href="#">DOWNLOAD</a>	<a href="#">SHORTLIST</a> <a href="#">REJECT</a>

**Applicants page.** In the Applicants page employers can browse and search for applicants profiles. On the top of this page there's a search box.

- John Doe
- Diasik M
- Diteix Dasdsad
- Karl Luxembourg
- Applicant Bob
- Mark Duja
- Email Emailovich

If employers click on the applicant's name, they can view their profile and from the profile page they can download applicant's CV and view all the necessary information about the applicant.

**Senior Software Developer**  
Hello, I'm Mike! Senior Software Developer and Designer. I was born and raised in London. Feel free to write me anytime! Hooray! Waka chuku kuku!

**Positions**

Software Engineer	Meta, Inc.
2010 - 2014	
It was nice!	

**Education**

Oxford University	BSc, Computer Science
2004 - 2009	
Courses taken: ...	

**Profile page.** In the profile page employers can view their profile and view how it is going to look like from the applicant's perspective.

The screenshot shows the Employer Dashboard. On the left, a sidebar menu includes: Create Job, Manage Jobs, Applications, Applicants, Profile, Settings, Additional Features, and HR Resources. The main area displays the employer's profile under 'Company Bio'. It features a circular profile picture of a robot, the company name 'Creative Solutions, Inc.', the email 'employer1@example.com', the phone number '+90533875894', and the location 'Turkey, Istanbul'. At the top right, the user information 'Creative Solutions, Inc.' and 'employer1@example.com' is shown.

**Settings page.** In the settings page employers can change their information along with uploading their profile images. They can also delete the profile image if they like. The main information here is the Company Name, Location, Phone and Short Bio.

The screenshot shows the Employer Dashboard with the 'Settings' tab selected. The left sidebar remains the same. The main area allows the employer to manage their profile. It includes fields for 'Email' (employer1@example.com), 'Company Name' ('Creative Solutions, Inc.'), 'Location' ('Turkey, Istanbul'), 'Phone' ('+90533875894'), and a 'Short Bio' section with the same descriptive text as the profile page. Buttons for 'UPLOAD' (with a camera icon) and 'DELETE' (with a trash icon) are available above the profile picture. A blue 'UPDATE PROFILE' button is at the bottom. An 'AI CHAT' button is located in the bottom right corner.

**Applicant's side.** When users are signing in, they can choose their roles as I mentioned before. If users choose an applicant's role, they're signing in as applicants.

The screenshot shows the Applicant Dashboard. The left sidebar includes: Find Jobs, Applications, Profile, Companies, and Settings. The main area features a large, colorful illustration of a hand holding a tablet displaying three people's profiles with five-star ratings. Below the illustration, a welcome message reads 'Welcome to your dashboard, Mike!'. At the top right, the user information 'Mike Johnson' and 'applicant1@example.com' is shown. A blue 'AI CHAT' button is in the bottom right corner.

**Find jobs page.** In the “Find Jobs” page applicants can browse jobs by using the filter box on the left side of the page or the search field on the top. Filtering includes various parameters, which applicants can use to find the right job position for them, these parameters include: Filter by Job Type, Location, Category and Salary Range. The search element can find any jobs by Title, Company Name or any Job Keyword.

The job card’s layout is informative and well-designed so that the applicants can see the job details. It includes: Position Title, Company Name, Job Description, Job Type, Employment Type, Category, Location, Salary Amount, Salary Currency, Salary Frequency and the Number of Applicants applied and the Apply button.

To display the jobs in a convenient format the table pagination is implemented to view the specified number of job postings on a single page. Here users can choose rows per page and navigate between pages.

To apply for a job, applicants must send a statement of purpose (cover letter) to tell briefly about themselves and express why they want to apply for a particular job.

Apply to Middle Web Developer (React.js, Node.js)

**Company:** Creative Solutions, Inc.  
Job Title: Middle Web Developer (React.js, Node.js)  
Type: Remote  
Location: Turkey, Istanbul  
Employment: Part Time  
Category: IT / Software Development  
Description: We're looking for a talented and passionate web developer with expertise in web development, familiar with HTML, CSS and JavaScript and React.js or Vue.js framework. Highly appreciated if candidate has experience in Node.js and MongoDB.  
Thanks!  
Salary: 24000 TRY / Monthly  
0 people applied

**Statement of Purpose**

Hello! My name is Mike. I think that I'm a good choice for your company, since I have expertise in the field you mentioned and I have strong soft skills, so working with me is comfortable and effective for a team! Thanks for your attention!

CANCEL    APPLY NOW

Apply to Middle Web Developer (React.js, Node.js)

**Company:** Creative Solutions, Inc.  
Job Title: Middle Web Developer (React.js, Node.js)  
Type: Remote  
Location: Turkey, Istanbul  
Employment: Part Time  
Category: IT / Software Development  
Description: We're looking for a talented and passionate web developer with expertise in web development, familiar with HTML, CSS and JavaScript and React.js or Vue.js framework. Highly appreciated if candidate has experience in Node.js and MongoDB.  
Thanks!  
Salary: 24000 TRY / Monthly  
0 people applied

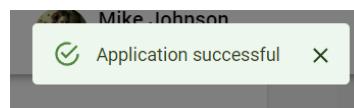
**Statement of Purpose**

Statement of Purpose is required.

Statement of purpose is a required field

CANCEL    APPLY NOW

If the application is successful, users get the alert.



**Applications page.** This is one of the main pages for applicants, since it provides a centralized table view for the applications and their status.

Job ID	Company Name	Job Title	Status	Application Date
6472a5a688eb5c4716cce2f3	Meta, Inc.	MERN stack developer	<span style="border: 1px solid red; border-radius: 5px; padding: 2px;">REJECTED</span>	28.05.2023
647bbb9ae4b2386d243cbfde	Creative Solutions, Inc.	Junior Software Developer	<span style="border: 1px solid red; border-radius: 5px; padding: 2px;">REJECTED</span>	04.06.2023
647bbc42e4b2386d243cbff0	Creative Solutions, Inc.	Senior Software Developer	<span style="border: 1px solid orange; border-radius: 5px; padding: 2px;">SHORTLISTED</span>	04.06.2023
647c78c5e4b2386d243cc07a	Creative Solutions, Inc.	Senior Team Leader	<span style="border: 1px solid gray; border-radius: 5px; padding: 2px;">APPLIED</span>	04.06.2023
647d29a0b89fa9787c06379c	Creative Solutions, Inc.	Middle Web Developer (React.js, Node.js)	<span style="border: 1px solid gray; border-radius: 5px; padding: 2px;">APPLIED</span>	05.06.2023

Rows per page: 100 ▾ 1–5 of 5 < >

AI CHAT

**Profile page.** Here applicants can see their profiles, with all the necessary information like profile image, personal data, CV, title, bio, job history, education and certifications fields.

Applicant Dashboard

- grid Find Jobs
- camera Applications
- user Profile
- building Companies
- gear Settings

Log Out

**Mike Johnson**  
applicant1@example.com  
+12154257858  
Canada, Montreal

**CV (Resume)** [View](#)

**Senior Software Developer**  
Hello, I'm Mike! Senior Software Developer and Designer. I was born and raised in London. Feel free to write me anytime! Hooray! Waka chuku kuku!

**Positions**

<b>Software Engineer</b>	August 2010 - October 2014
Meta, Inc.	
It was nice!	

**Education**

<b>BSc, Computer Science</b>	2004 - 2009
Oxford University	
Courses taken: ...	

AI CHAT

**Companies page.** The page where applicants can search companies. And to view the company's profile applicant can click on one of the company list elements.

The company profile.

**Settings page.** The page where applicants can upload their profile pictures, change name if needed, add personal data, upload and store resume, add job history, education and certification entries.

**Position 1**

Occupation/Title  
Software Engineer

Company  
Meta, Inc.

Location  
USA, San Jose

Start Month  
August

Start Year  
2010

End Month  
October

End Year  
2014

Description  
It was nice!

✖ REMOVE THIS POSITION 💾 SAVE THIS POSITION

+ ADD EDUCATION

+ AI CHAT

**Education 1**

University  
Oxford University

Degree  
BSc, Computer Science

Start Year  
2004

End Year  
2009

Description  
Courses taken: ...

✖ REMOVE THIS EDUCATION 💾 SAVE THIS POSITION

+ ADD CERTIFICATION

+ AI CHAT

**Certification 1**

Name  
SCRUM Master Program

Authority  
SCRUM Foundation

License Number  
REF202202020

Description  
SCRUM Master Program

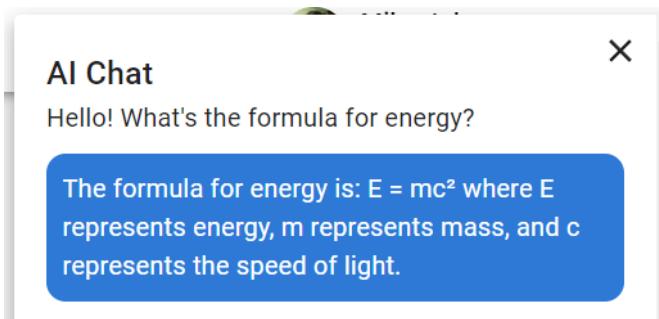
✖ REMOVE THIS CERTIFICATION 💾 SAVE THIS CERTIFICATION

UPDATE PROFILE

+ AI CHAT

In the rapidly evolving landscape of technology, cutting-edge tools such as OpenAI's ChatGPT have become increasingly prevalent. OpenAI has established accessible APIs, thus allowing developers to incorporate their advanced products into their own applications. Consequently, I decided to integrate an AI Chat feature into my system. This feature, embodied as an interactive chatbox, empowers users to converse with an AI entity, enabling them to obtain responses to a wide array of inquiries, thereby enhancing user engagement and accessibility.

Both employers and applicants can use this feature.

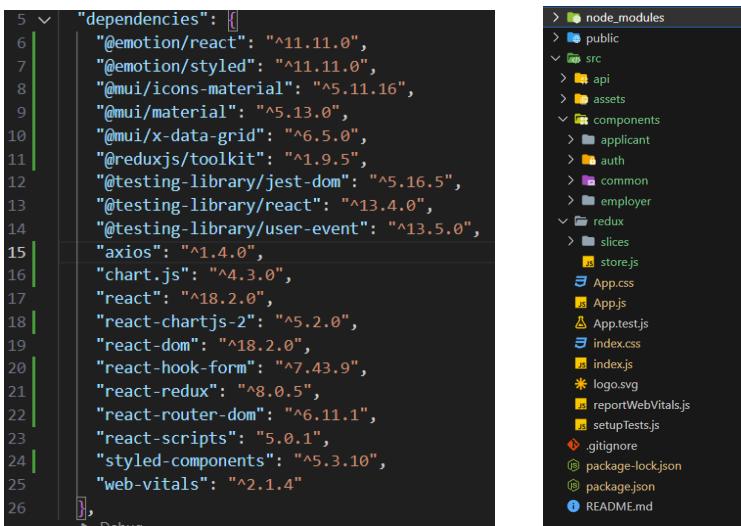


## Frontend:

As I have mentioned before for the frontend development I used the React.js framework along with Material UI design system and Redux Toolkit for state management (Garreau & Faurot, 2018). This combination of technologies provided a robust and efficient tools, enabling me to deliver a high-quality, user-friendly application that met all project requirements.

Material-UI is a popular React UI framework that provides a set of pre-built components, enabling developers to create visually appealing and responsive user interfaces with ease. It adheres to Google's Material Design guidelines, ensuring a modern and clean aesthetic that enhances user experience.

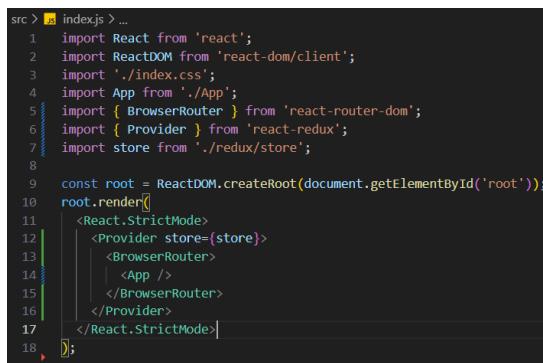
Here's the list of used packages and tools for the frontend and the folder structure:



The image shows two side-by-side screenshots. On the left is a terminal window displaying the 'dependencies' section of a package.json file. The list includes various React, Material-UI, and Redux-related packages. On the right is a file explorer showing the directory structure of the project, which includes node\_modules, public, src, api, assets, components, applicant, auth, common, employer, redux, slices, store.js, App.css, App.js, App.test.js, index.css, index.js, logo.svg, reportWebVitals.js, setupTests.js, .gitignore, package-lock.json, package.json, and README.md.

```
5 "dependencies": [
6   "@emotion/react": "^11.11.0",
7   "@emotion/styled": "^11.11.0",
8   "@mui/icons-material": "^5.11.16",
9   "@mui/material": "^5.13.0",
10  "@mui/x-data-grid": "^6.5.0",
11  "@reduxjs/toolkit": "^1.9.5",
12  "@testing-library/jest-dom": "^5.16.5",
13  "@testing-library/react": "^13.4.0",
14  "@testing-library/user-event": "^13.5.0",
15  "axios": "^1.4.0",
16  "chart.js": "^4.3.0",
17  "react": "^18.2.0",
18  "react-chartjs-2": "^5.2.0",
19  "react-dom": "^18.2.0",
20  "react-hook-form": "^7.43.9",
21  "react-redux": "^8.0.5",
22  "react-router-dom": "^6.11.1",
23  "react-scripts": "5.0.1",
24  "styled-components": "^5.3.10",
25  "web-vitals": "^2.1.4"
26 ]
```

The entry point of React app is index.js file. It imports the required libraries, including React itself, the ReactDOM library, which provides DOM-specific methods, the redux store and the CSS file. The BrowserRouter component is imported from react-router-dom, which is a third-party library used for adding navigation and routing to a React application. The Provider component is imported from react-redux. This component makes the Redux store available to the rest of the app.



The image shows a code editor with the index.js file open. The code defines the root component of the application. It imports React, ReactDOM, and other necessary components like App, Provider, and BrowserRouter. It then creates a root element using ReactDOM.createRoot and renders the App component within a Provider component that is passed the store as a prop. Finally, it wraps the entire structure in a StrictMode component.

```
src > index.js > ...
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import './index.css';
4 import App from './App';
5 import { BrowserRouter } from 'react-router-dom';
6 import { Provider } from 'react-redux';
7 import store from './redux/store';
8
9 const root = ReactDOM.createRoot(document.getElementById('root'));
10 root.render([
11   <React.StrictMode>
12   <Provider store={store}>
13     <BrowserRouter>
14       <App />
15     </BrowserRouter>
16   </Provider>
17 </React.StrictMode>
18]);
```

App.js defines the App component, which is the root component of the application. It's responsible for rendering all the other components and handling routing.

```

src > App.js > App
  import Footer from './components/common/Footer';
  import ApplicantDashboard from './components/applicant/.dashboard';
  import EmployerDashboard from './components/employer/.dashboard';
  import PrivateRoute from './components/auth/Private';
  import ChatWidget from './components/common/chat';
  ...
  import { useDispatch } from 'react-redux';
  import { rehydrate } from './redux/slices/authSlice';
  ...
function App() {
  const dispatch = useDispatch();
  ...
  useEffect(() => {
    dispatch(rehydrate());
  }, [dispatch]);
  ...
  const location = useLocation();
  const shouldRenderNavbar = ['/','/signin','/signup'].includes(location.pathname);
  ...
  return (
    <div className="App">
      {shouldRenderNavbar && <Navbar />}
      <div className="content">
        <Routes>
          <Route exact path="/" element={<Homepage />} />
          <Route exact path="/signin" element={<SignIn />} />
        </Routes>
      </div>
    </div>
  );
}

```

To ensure that special roles have access only to their own Routes, I have implemented Private Routes. It accesses the Redux store to know whether the user is logged in and also is accessing the role value, which are “Applicant” and “Employer” in my case. Once the application ensures that I’m a logged in user and knows my role it returns the children elements, which are the nested Routes, that will be rendered.

```

1  import { useSelector } from 'react-redux';
2  import { Navigate } from 'react-router-dom';
3
4  function PrivateRoute({ role, children }) {
5    const userRole = useSelector((state) => state.auth.user.role); // get the user's role
6    const isAuthenticated = useSelector((state) => state.auth.isLoggedIn);
7
8    if (isAuthenticated && userRole === role) {
9      return children;
10    } else {
11      return <Navigate to="/signin" replace />;
12    }
13  }
14
15  export default PrivateRoute;

```

**Redux.** Redux Toolkit is mainly used here to handle the authentication state, whether the user is logged in, logged out and stores the user’s data, like personal data and the auth token. First it starts from the authSlice.js, it uses the Redux Toolkit’s ‘createSlice’ to generate action creators and action types that are related to user authentication, it sets the initial state of the ‘auth’ slice of the state. The reducers field defines different cases for actions such as loginStart, loginSuccess, loginFailure, logout, and rehydrate. Finally, it exports the generated action creators and the reducer function.

```

1  import { configureStore } from '@reduxjs/toolkit';
2  import userReducer from './slices/userSlice';
3  import authReducer from './slices/authSlice';
4
5  const preloadedState = () => {
6    try {
7      const authToken = localStorage.getItem('authToken');
8      const user = JSON.parse(localStorage.getItem('user'));
9
10      if (authToken && user) {
11        return {
12          auth: {
13            isLoggedIn: true,
14            isLoggingIn: false,
15            loginError: null,
16            user,
17          }
18        }
19      } else {
20        return undefined;
21      }
22    } catch(err) {
23      return undefined;
24    }
25  }
26
27
src > redux > slices > authSlice.js > authSlice > reducers
1  import { createSlice } from '@reduxjs/toolkit';
2
3  const initialState = {
4    isLoggedIn: false,
5    isLoggingIn: false,
6    loginError: null,
7    user: {},
8  };
9
10 const authSlice = createSlice({
11   name: 'auth',
12   initialState,
13   reducers: {
14     loginStart: (state) => {
15       state.isLoggingIn = true;
16       state.loginError = null;
17     },
18     loginSuccess: (state, action) => {
19       state.isLoggedIn = true;
20       state.isLoggingIn = false;
21       state.user = action.payload;
22     },
23     loginFailure: (state, action) => {
24       state.isLoggedIn = false;
25     }
26   }
27 });

```

‘Axios’ library. Axios is an HTTP client library based on Promise that lets you send requests (Requests) to a given endpoint. This can be, for example, an external service API, or an internal Node.js server. Axios makes the process of handling requests easier and more efficient comparing to the native fetch.

```
src > api > api.js > api.interceptors.request.use() callback
  1 // api.js
  2 import axios from 'axios';
  3
  4 const api = axios.create({
  5   baseURL: 'http://localhost:7500',
  6 });
  7
  8 api.interceptors.request.use(
  9   (config) => [
10     const token = localStorage.getItem('authToken');
11     if (token) {
12       config.headers.Authorization = `Bearer ${token}`;
13     }
14     return config;
15   ],
16   (error) => {
17     return Promise.reject(error);
18   }
19 );
20
21 export default api;
```

Sign in and Sign up forms. Both are implemented as functional components, using a mix of React hooks and Material UI for styling and form controls. Sign In components is responsible for rendering a sign-in form and handling sign-in operations.

State variables are initialized using ‘useState’ to track the user role (either “Applicant” or “Employer”), password visibility, and Snackbar (to show alerts). ‘useForm’ from ‘react-hook-form’ is used to handle form validation and submission. ‘useDispatch’ from ‘redux’ is used to dispatch actions to Redux store. This component dispatches login actions (‘loginStart’, ‘loginSuccess’, ‘loginFailure’) to Redux store to manage the state of the authentication process. Upon successful login, the user is navigated to a page based on their role. The auth token is also stored in localStorage. If there’s an error during login, an error message is displayed using a Snackbar.

```
import { useDispatch } from 'react-redux';
import { loginStart, loginSuccess, loginFailure } from '../../../../../redux/slices/authSlice';

export default function SignIn() {
  const [role, setRole] = useState("Applicant");
  const [showPassword, setShowPassword] = useState(false);
  const togglePasswordVisibility = () => { setShowPassword(!showPassword); };
  const handleRoleChange = (event, newRole) => {
    if (newRole !== null) {
      setRole(newRole);
    }
  };
  const { register, handleSubmit, formState: { errors } } = useForm();
  const navigate = useNavigate();
  const [snackbar, setSnackbar] = useState({ open: false, message: '', severity: 'success' });
  const handleSnackbarClose = (event, reason) => {
    if (reason === 'clickaway') { return; }
    setSnackbar(prevSnackbar => ({ ...prevSnackbar, open: false }));
  };
  const dispatch = useDispatch();
```

**Sign up form.** This component is responsible for rendering a sign-up form and handling sign-up operations. Like Sign In, state variables are initialized using ‘useState’. The form includes fields for email, password, and either a full name (for “Applicant”) or company name (for

“Employer”). Password visibility can be toggled. Material UI components are used to design the form.

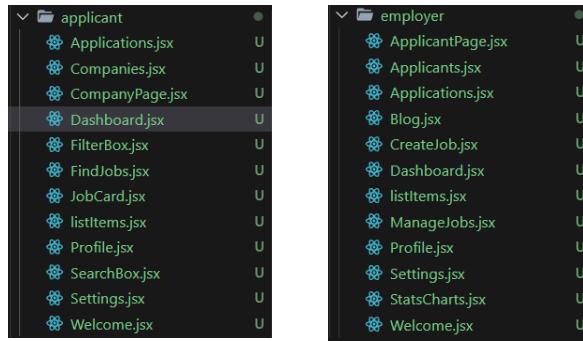
```
export default function SignUp() {
  const [role, setRole] = useState("Applicant");
  const [showPassword, setShowPassword] = useState(false);

  const togglePasswordVisibility = () => setShowPassword(!showPassword);
  const handleRoleChange = (event, newRole) => newRole !== null && setRole(newRole);
  const { register, handleSubmit, formState: { errors } } = useForm();

  const [snackbar, setSnackbar] = useState({ open: false, message: '', severity: 'success' });
  const handleSnackbarClose = (event, reason) => {
    if (reason === 'clickaway') { return; }
    setSnackbar({ prevSnackbar: { ...prevSnackbar, open: false } });
  };
  const navigate = useNavigate();

  const onSubmit = async (data) => {
    try {
      const response = await api.post(`/${role.toLowerCase()}/register`, data);
      if (response.status === 201) {
        setSnackbar({ open: true, message: 'Sign up successful', severity: 'success' });
        setTimeout(() => {
          navigate("/signin");
        }, 2500);
      }
    } catch (error) {
      setSnackbar({ open: true, message: 'Sign up failed', severity: 'error' });
    }
  };
}
```

Both of the roles have their own dashboards, where they can browse the pages and interact with an interface. Here you can see the applicant’s pages and employer’s pages:



Find jobs page. It fetches the jobs and display them and also here’s a pagination to display the specified number of job positions.

```
src > components > applicant > FindJobs.jsx > FindJobsPage
1 import React, { useState, useEffect } from 'react';
2 import { Grid, Typography, Container, Dialog, DialogTitle, DialogContent, DialogActions,
3 import JobCard from './JobCard';
4 import FilterBox from './FilterBox';
5 import SearchBox from './SearchBox';
6 import api from '../../../../../api/api';
7 import { useForm } from 'react-hook-form';
8 import { Snackbar, Alert } from '@mui/material';
9
10 export default function FindJobsPage() {
11   const [jobs, setJobs] = useState([]);
12   const [filteredJobs, setFilteredJobs] = useState([]);
13   const [search, setSearch] = useState('');
14   const [filterSettings, setFilterSettings] = useState({
15     type: '',
16     location: '',
17     category: '',
18     salaryRange: [0, 100000],
19   });
20
21   // pagination
22   const [page, setPage] = useState(0);
23   const [rowsPerPage, setRowsPerPage] = useState(5);
24
25   // fetch jobs
26   useEffect(() => {
27     const fetchJobs = async () => {
28       try {
29         const response = await api.get('/jobs');
30         setJobs(response.data);
31         setFilteredJobs(response.data);
32       } catch (error) {
33         console.error('Failed to fetch jobs:', error);
34       }
35     };
36     fetchJobs();
37   }, [page]);
38 }
```

I’ve described the main points and implementation of the frontend. I would have described all the details for each page, but it would be long and redundant, so I’ve described the main details.

## Backend:

For developing the backend side of the app I used the REST API architectural approach. The REST architectural style is the most common approach for API design. Also, the MVC (Model View Controller) pattern is used in my application. The main used frameworks are Node.js and Express.js. On the main features of the Node.js platform is non-blocking I/O, asynchronous approach, etc. (Casciaro & Mammino, 2020).

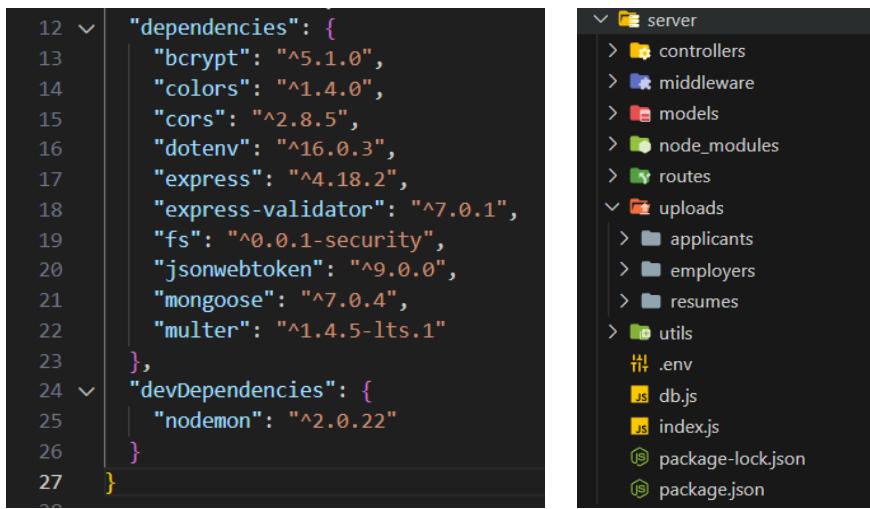
What is blocking I/O and why non-blocking I/O is better?

1. Blocking I/O. This is how classic web servers work, for example in Java. The concept is quite simple, there is a certain execution thread, which line by line executes some instructions, until the command is executed, we can not go to the next one, that is, the whole thread is blocked. If the instructions are simple, everything is fine, line by line we execute them, there are no difficulties. But not all instructions are simple, sometimes we need to write a file, get a file, work with the database or the network, and with a blocking model of behavior, the entire thread is blocked. In this case, the application is not able to handle other operations.
2. Non-blocking I/O. The server runs with only one Main Thread. With non-blocking I/O, system calls return control slowly without waiting for any data to be executed, read or written. Most operating systems have a more efficient mechanism for non-blocking operation and it is called the event demultiplexer.

At the core of NodeJS is Libuv, which deals just with input/output operations, at its core Libuv can manage threads, and their default number is 4, this number can be changed. The disadvantages of non-blocking I/O are that operations like reading a file or writing it to disk are very heavy, because they are done in a single thread, although with non-blocking I/O, so some parallelizing of these processes is still necessary (Libuv in Node.js, 2021).

Libuv is written in C, and the V8 engine, on which NodeJS is based, is written in C++. Both C and C++ can write some modules for NodeJS. And it tells us that some libraries can use threads.

Here's the packages and libraries used at the backend side and the folder structure. My project follows a standard, organize, and efficient structure, which ensures better maintainability, readability and scalability of the application. The backend is segmented into several distinct folders, each with its designated responsibility.



The image shows two side-by-side screenshots. On the left is a code editor displaying the 'dependencies' and 'devDependencies' sections of a package.json file. The 'dependencies' section lists various npm packages with their versions. The 'devDependencies' section includes 'nodemon'. On the right is a file explorer showing the directory structure of a project named 'server'. The structure includes 'controllers', 'middleware', 'models', 'routes', 'uploads' (which contains 'applicants', 'employers', and 'resumes'), 'utils', and several configuration and script files like '.env', 'db.js', 'index.js', 'package-lock.json', and 'package.json'.

```
12 "dependencies": {  
13   "bcrypt": "^5.1.0",  
14   "colors": "^1.4.0",  
15   "cors": "^2.8.5",  
16   "dotenv": "^16.0.3",  
17   "express": "^4.18.2",  
18   "express-validator": "^7.0.1",  
19   "fs": "^0.0.1-security",  
20   "jsonwebtoken": "^9.0.0",  
21   "mongoose": "^7.0.4",  
22   "multer": "^1.4.5-lts.1"  
23 },  
24 "devDependencies": {  
25   "nodemon": "^2.0.22"  
26 }  
27 }
```

I choose MongoDB for my project because of its adaptable, document-oriented data format, which makes it simple to store and retrieve a wide variety of complicated data types. This NoSQL database is suited for managing big volumes of data, which is essential for my project because it is scalable and delivers good speed. Additionally, MongoDB's robust querying and indexing capabilities and support for quick Agile development greatly increase productivity and efficiency.

Here's how I defined the models. Here I used the 'mongoose', it's the JS lib that establishes a connection between MongoDB and Node.js JS runtime environment.

```

Applicant.js
server > models > Applicant.js > EducationSchema
1 const {Schema, model} = require('mongoose');
2
3 const PositionSchema = new Schema({
4   title: { type: String },
5   company: { type: String },
6   location: { type: String },
7   startMonth: { type: String },
8   startYear: { type: Number },
9   endMonth: { type: String },
10  endYear: { type: Number },
11  description: { type: String },
12 }, { _id: true});
13
14 const EducationSchema = new Schema([
15   university: { type: String },
16   degree: { type: String },
17   startYear: { type: Number },
18   endYear: { type: Number },
19   description: { type: String },
20 ], { _id: true});
21
22 const CertificationSchema = new Schema({
23   name: { type: String },
24   authority: { type: String },
25   licenseNumber: { type: String },
26   description: { type: String }
27 }, { id: true});
28

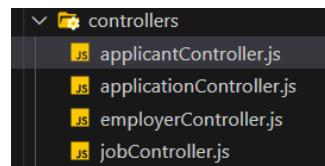
Application.js
server > models > Application.js > schema
1 const mongoose = require("mongoose");
2
3 let schema = new mongoose.Schema(
4   {
5     userId: {
6       type: mongoose.Schema.Types.ObjectId,
7       required: true,
8       ref: 'Applicant'
9     },
10    recruiterId: {
11      type: mongoose.Schema.Types.ObjectId,
12      required: true,
13      ref: 'Employer'
14    },
15    jobId: {
16      type: mongoose.Schema.Types.ObjectId,
17      required: true,
18      ref: 'Job'
19    },
20    status: {
21      type: String,
22      enum: [
23        "applied", // when an applicant has applied
24        "shortlisted", // when an applicant is shortlisted
25        "accepted", // when an applicant is accepted
26        "rejected", // when an applicant is rejected
27        "deleted" // when any job is deleted
28      ]
29    }
30  }
31);
32
33 module.exports = schema;

Employer.js
server > models > Employer.js > ...
1 const {Schema, model} = require('mongoose');
2
3 const EmployerSchema = new Schema({
4   companyName: { type: String, required: true },
5   email: { type: String, required: true, unique: true },
6   password: { type: String, required: true },
7   role: { type: String, default: 'Employer' },
8   avatar: { type: String },
9   phone: { type: String },
10  location: { type: String },
11  shortBio: { type: String }
12 });
13
14 module.exports = model('Employer', EmployerSchema);

Jobs.js
server > models > Jobs.js > JobSchema
1 const { Schema, model } = require('mongoose');
2
3 const JobSchema = new Schema({
4   userId: {
5     type: Schema.Types.ObjectId,
6     required: true,
7     ref: 'Employer'
8   },
9   companyName: { type: String, required: true },
10  title: { type: String, required: true },
11  type: { type: String, required: true },
12  location: { type: String, required: true },
13  employment: { type: String, required: true },
14  category: { type: String, required: true },
15  description: { type: String, required: true },
16  salary: { type: String, required: true },
17  currency: { type: String, required: true },
18  salaryFrequency: { type: String, required: true },
19  dateOfPosting: { type: Date, default: Date.now },
20  activeApplications: { type: Number, default: 0 },
21  acceptedCandidates: { type: Number, default: 0 }
22 });
23
24 module.exports = model('Job', JobSchema);

```

**Controllers.** They control the logic for the application's routes. It directs the request to the correct function in the Models and sends the response back to client.



**applicantController** controller uses the bcrypt, it's a password-hashing function widely used for secure password hashing and is essential part of my application's security middleware. Bcrypt is used for password hashing, salting, password verification.

The register function takes the request body, destructures it by taking, email, password, user name, checks if there's an existing user with the same email, if not it hashes the password, ans saves the new user, then it generates the token and sends the token as a response.

```

244 // auth
245 exports.register = async (req, res) => {
246   const errors = validationResult(req);
247   if (!errors.isEmpty()) {
248     return res.status(400).json({ errors: errors.array() });
249   }
250
251   try {
252     const { email, password, firstName, lastName } = req.body;
253
254     const existingUser = await checkEmailInBothCollections(email);
255     if (existingUser) {
256       return res.status(400).json({ error: 'Email already in use' });
257     }
258
259     const hashedPassword = await bcrypt.hash(password, 10);
260     const applicant = new Applicant({ email, password: hashedPassword, first
261     await applicant.save();
262
263     const token = generateToken(applicant, process.env.JWT_SECRET, 'id');
264     res.status(201).json({ token });
265   } catch (error) {
266     res.status(400).json({ error: error.message });
267   }
268 };

```

**Routes.** It's serving as the blueprint for the application's endpoint structure and the roadmap for the HTTP requests. For the authenticated requests it has to be used along with the authMiddleware, to prevent unauthorized access.

```

js jobRoutes.js ×
server > routes > js jobRoutes.js > ...
1  const router = require('express').Router();
2  const jobController = require('../controllers/jobController');
3  const authMiddleware = require('../middleware/authMiddleware');
4
5  router.get('/employer/jobs', authMiddleware, jobController.getEmployerJobs);
6  router.get('/jobs', jobController.getAllJobs);
7  router.post('/jobs', authMiddleware, jobController.createJob);
8  router.put('/jobs/:id', authMiddleware, jobController.updateJob);
9  router.delete('/jobs/:id', authMiddleware, jobController.deleteJob);
10
11 router.get('/employer/:id/stats', authMiddleware, jobController.getEmployer
12 |
13 module.exports = router;

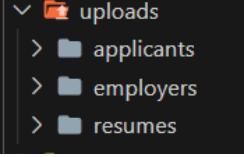
```

```

js employerRoutes.js ×
server > routes > js employerRoutes.js > ...
1  const router = require('express').Router();
2  const employersController = require('../controllers/employerController');
3  const { employerValidationRules } = require('../middleware/validationMiddlew
4  const { deleteAvatar, upload, getAllEmployers } = require('../controllers/e
5  const authMiddleware = require('../middleware/authMiddleware');
6
7  router.post('/employer/avatar', authMiddleware, upload.single('avatar'), (
8    if (!req.file) {
9      return res.status(400).send({ error: 'No file uploaded' });
10    }
11    next();
12  ), employersController.uploadAvatar);
13
14 router.get('/employers', employersController.getAllEmployers);
15 router.get('/employers/:id', employersController.getEmployerById);
16
17 router.delete('/employer/avatar', authMiddleware, deleteAvatar);
18 router.put('/employer/details', authMiddleware, employersController.updateE
19 router.post('/employer/register', employerValidationRules, employersController
20 router.post('/employer/login', employersController.login);
21
22 module.exports = router;

```

**Uploads.** For storing file the backend application is using ‘multer’ package. For example, the upload avatar function create the new directory in the uploads/applicants folder with the name of user id and stores the image inside.



```

exports.uploadAvatar = async (req, res) => {
  try {
    const user = await Applicant.findById(req.user.id);
    const newAvatarPath = req.file.filename;
    const userDir = path.join(__dirname, `..uploads/applicants/${req.user.id}`);
    const files = await fs.promises.readdir(userDir);

    for (const file of files) {
      const filePath = path.join(userDir, file);
      if (filePath !== path.join(userDir, newAvatarPath)) {
        await fs.promises.unlink(filePath);
      }
    }

    user.avatar = newAvatarPath;
    await user.save();
    res.json({ avatar: user.avatar });
  } catch (error) {
    console.log('Error in uploadAvatar:', error);
    res.status(500).json({ error: error.message });
  }
};

```

**Middleware.** Middleware is auxiliary code that runs before the main program, in this example the HTTP server in Node.JS. Most frequently, this is required to perform some further tuning or validate an incoming request. For instance, to retrieve cookies as an object or to convert data from a POST request that is a JSON string into a standard object.

```

const { verifyToken } = require('../utils/jwtHelpers');

const authMiddleware = (req, res, next) => {
  const authHeader = req.headers.authorization;

  if (!authHeader) {
    return res.status(401).json({ error: 'No token provided' });
  }

  const token = authHeader.split(' ')[1];

  try {
    const decoded = verifyToken(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    return res.status(401).json({ error: 'Invalid token' });
  }
};

module.exports = authMiddleware;

```

```

const { body } = require('express-validator');

const applicantValidationRules = [
  body('email').isEmail().withMessage('Please enter a valid email address'),
  body('password').isLength({ min: 8 }).withMessage('Password must be at least 8 characters long'),
  body('firstName').notEmpty().withMessage('First name is required'),
  body('lastName').notEmpty().withMessage('Last name is required'),
];

const employerValidationRules = [
  body('email').isEmail().withMessage('Please enter a valid email address'),
  body('password').isLength({ min: 8 }).withMessage('Password must be at least 8 characters long'),
  body('companyName').notEmpty().withMessage('Company name is required'),
];

module.exports = { applicantValidationRules, employerValidationRules };

```

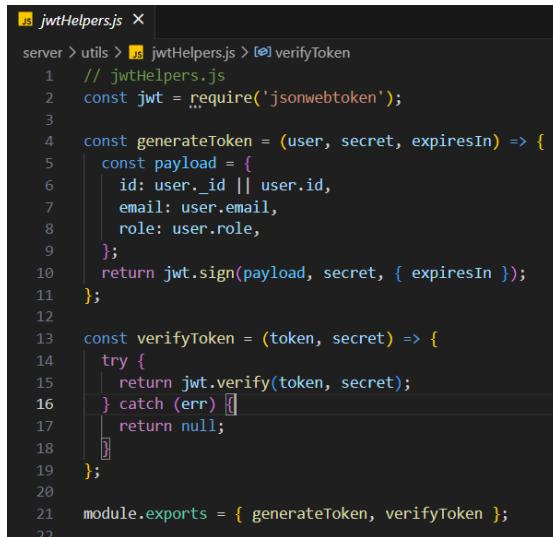
**authMiddleware.js** file defines a middleware function named ‘authMiddleware’. It imports a utility function named verifyToken from the jwtHelpers module. This function is used to verify JWT tokens. The middleware function retrieves the authorization header from the incoming HTTP request. This header should contain a JWT token. The token is verified using the verifyToken function. If the token is valid, the decoded information is added to the req.user object and the next middleware function is called.

## JWT tokens.

A JSON Web Token, or JWT, is a standardized, in some cases signed and/or encrypted data packaging format that is used to transmit information securely between two parties in client-server applications; tokens in turn are created on the server side, signed with a secret key, and sent to the client; the token is then verified for identity verification. Because it has been digitally signed, frequently using a secret key or a public/private key pair, this information may be checked and trusted. The server can validate users and allow access to secured routes and resources thanks to the widespread usage of JWTs for authentication and authorization (Poddar, 2022).

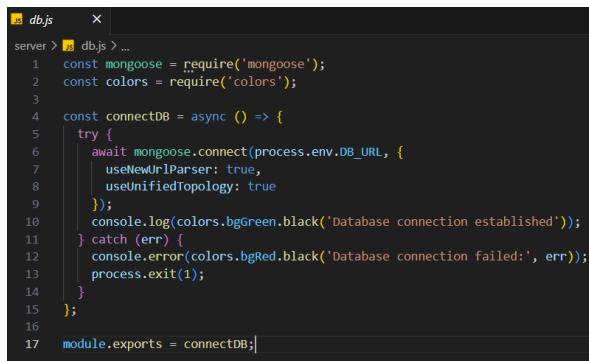
In my backend application the jwtHelpers.js file is responsible for generating and verifying tokens. `generateToken(user, secret, expiresIn)`: This function is used to create a new JWT. It takes three parameters: user, which is an object containing user data, secret, which is the secret key used to sign the token, and expiresIn, which sets the token's lifespan.

In the function, a payload object is constructed that includes the user's ID (`user._id` or `user.id`), email (`user.email`), and role (`user.role`). This payload is then signed using `jwt.sign()`, creating a new JWT. The secret is used to digitally sign the payload, and `expiresIn` sets the token's expiration time. The newly created JWT is then returned.



```
server > utils > jwtHelpers.js > verifyToken
1 // jwtHelpers.js
2 const jwt = require('jsonwebtoken');
3
4 const generateToken = (user, secret, expiresIn) => {
5   const payload = {
6     id: user._id || user.id,
7     email: user.email,
8     role: user.role,
9   };
10  return jwt.sign(payload, secret, { expiresIn });
11 }
12
13 const verifyToken = (token, secret) => {
14   try {
15     return jwt.verify(token, secret);
16   } catch (err) {
17     return null;
18   }
19 }
20
21 module.exports = { generateToken, verifyToken };
22
```

**Database connection.** This JavaScript code represents a connection setup with document-oriented MongoDB using the nodejs library mongoose. mongoose is the object data modeling library (ODM) for MongoDB and Node.js. This library is used to create relationships between data, provides data schema validation, and is used to translate between objects in code and the presentation of those objects in MongoDB.



```
server > db.js > ...
1 const mongoose = require('mongoose');
2 const colors = require('colors');
3
4 const connectDB = async () => {
5   try {
6     await mongoose.connect(process.env.DB_URL, {
7       useNewUrlParser: true,
8       useUnifiedTopology: true
9     });
10    console.log(colors.bgGreen.black('Database connection established'));
11  } catch (err) {
12    console.error(colors.bgRed.black(`Database connection failed: ${err}`));
13    process.exit(1);
14  }
15 }
16
17 module.exports = connectDB;
```

**AI Chat server.** First, various necessary modules are imported, including Express.js for creating the server, body-parser for parsing incoming request bodies, cors for enabling CORS, dotenv for loading environment variables, and the OpenAI API SDK. A new instance of the OpenAI API is created using the Configuration class. This requires an organization and apiKey, both of which are loaded from the environment variables. An Express.js application instance is created and stored in app. The application listens on the port specified in the process.env.PORT environment variable, or specified port number if it's not set. The application is configured to use the bodyParser.json() middleware, which parses incoming requests with JSON payloads, and cors(), which enables CORS. When a POST request is made to the server endpoint, the message from the request body is sent to the OpenAI API using the createChatCompletion method with the "gpt-3.5-turbo" model. In summary, this server allows for interfacing with the OpenAI API to create chat completions using the GPT-3.5-turbo model (Marion, 2023).

```
chat_api > js index.js > ...
1 import { Configuration, OpenAIApi } from "openai";
2 import express from "express";
3 import bodyParser from "body-parser";
4 import cors from "cors";
5 import dotenv from 'dotenv';
6
7 dotenv.config();
8
9 const configuration = new Configuration({
10   organization: process.env.ORGANIZATION,
11   apiKey: process.env.API_KEY,
12 });
13
14 const openai = new OpenAIApi(configuration);
15
16 const app = express();
17 const port = process.env.PORT || 4444;
18
19 app.use(bodyParser.json());
20 app.use(cors());
21
22 app.post("/", async (req, res) => {
23
24   const { message } = req.body;
25
26   const completion = await openai.createChatCompletion({
27     model: "gpt-3.5-turbo",
28     messages: [
29       {role: "user", content: `${message}`},
30     ],
31   });
32
33   res.json(completion);
34 });
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
948
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1098
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1186
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1286
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1386
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1990
1991
1992
1993
1994
1995
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
216
```

## 6. Conclusion

### 6.1 Benefits

#### a. Benefits to users :

1. **Convenience:** The system can provide a more convenient and user-friendly experience for both companies and job seekers, allowing them to easily post and search for job openings, apply for any jobs, and track the status of their applications.
2. **Efficiency:** The online recruitment system can improve and automate the recruitment processes, saving the time and resources for both companies and job seekers.
3. **Improved job filtering:** The online recruitment system provides a comprehensive job searching and filtering, allowing applicants to use various filters to find a right job for them.
4. **Data and analytics:** The system provides companies with data and insights on their recruitment efforts, such as the number of applications received and the success rate of hired candidates and their statuses.
5. **Enhanced security:** The system provides security measures to protect the privacy and personal information of both companies and job seekers

#### b. Benefits to me :

1. **Professional experience:** Building and launching an online recruitment system can provide valuable professional experience in software development, project management, and other related skills.
2. **Career advancement:** The project serves as a showcase of my skills and abilities, potentially leading to new job opportunities or career advancement, for example I can add this project in my portfolio.
3. **Professional networking:** The project offers the chances to network and establish connections with other experts in the field, including potential customers or collaborators.
4. **Educational benefits:** Since I developed this project, I gained valuable experience in web application development, which in turn strengthened the foundation for me to develop new applications, as I plan to apply my skills in the future.

**5. Improved CV:** This project can serve as a notable accomplishment to include on the my resume and may increase my chances of being considered for job opportunities.

**6. Opportunity for growth:** Building an online recruitment system can provide an opportunity for the creator to grow and expand their skills and knowledge.

## 6.2 Ethics

The development and deployment of an online recruitment system has a number of ethical implications that must be precisely considered to guarantee that the project is performed in a responsible and ethical manner.

One of the key ethical considerations is the potential impact of the platform on the employment opportunities and job market. The platform will have the ability to connect companies with job seekers, and could potentially have a substantial impact on the hiring process and the availability of jobs. It is important to make sure that the platform is designed and operated in a way that is fair and unbiased, and that it does not discriminate against any particular groups of job seekers. The problem is that women and ethnic minorities often experience worse labor market outcomes compared to men and the majority group in many countries (Kopp, Hangartner, & Siegenthaler, 2021).

Another ethical consideration is the handling of user data. Nowadays keeping customer information safe is a must for building a reliable communication and relation with customers. The platform will collect and store a large amount of personal information about both companies and job seekers, and it is very important to validate that the incoming information and user data is guarded responsibly and in conformity with good data protection regulations. This involves making sure that user data is protected, kept private, and only utilized for the purposes for which it was gathered.

A third ethical consideration is the potential impact of the platform on the environment. The platform will be hosted on a cloud-based server, and it is important to ensure that the server infrastructure is energy efficient and has a minimal impact on the environment. Also, these days it is important to reduce a carbon footprint (Usman & Radulescu, 2022) . I will also consider other environmental impacts of the platform, such as the use of paper and other resources, and work to minimize these impacts wherever possible.

Overall, it is necessary to confirm that the online recruitment system is built and operated responsibly and morally, keeping in mind how the platform can affect the job economy, user data, and the environment.

## **Why did I choose this project?**

I chose "Online Recruitment System" because this idea is very interesting. Recruitment products are in demand on the market, because nowadays more and more people are using online job search platforms, and companies are also looking for employees, and this web application will help bring employees and employers together.

I want to contribute solutions that can improve the process of finding and hiring top talent. Also, this project helped me develop my development skills such as web development and I also used the right methodologies and development principles so that the project structure is properly aligned to easily support the application in the future.

### **6.3 Future Works**

I am planning to add more features and deploy my project so that it can be visible by any users in the world. I would add recruitment specific algorithms for perfect job matching and recommendation system and make it more like a multifunctional platform, that will integrate some external API's to make the application more business oriented and be ready to serve large streams of users.

More features that are likely to be implemented in the future:

1. Artificial Intelligence Integration. It can be used to automate the processes, such as screening process, matching candidates' skills and experience with job requirements. This can significantly reduce the time spent on manual screening and increase the efficiency of the recruitment process.
2. Data Analytics. It can provide valuable insights for both companies and job seekers, so that the users can improve their strategies. This can help in making data-driven decisions and improving the overall recruitment process.
3. Integration with Other Systems. Integrating the online recruitment system with other HR systems (like onboarding and performance management systems) can streamline HR processes and improve the data consistency.

## 7. References

- Banks, A., & Porcello, E. (2020). *Learning React: Modern Patterns for Developing React Apps, 2nd Edition*. O'Reilly Media.
- Bradshaw, S., Brazil, E., & Chodorow, K. (2019). *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage, 3rd Edition*. O'Reilly Media.
- Casciaro, M., & Mammino, L. (2020). *Node.js Design Patterns: Design and implement production-grade Node.js applications using proven patterns and techniques, 3rd Edition*. Packt Publishing.
- Frye, L. (2019, August 16). *The Cost of a Bad Hire Can Be Astronomical*. Retrieved from shrm.org: <https://www.shrm.org/ResourcesAndTools/hr-topics/employee-relations/Pages/cost-of-bad-hires.aspx>
- Garreau, M., & Faurot, W. (2018). *Redux in Action*. Manning.
- Gurchiek, K. (2021, August 25). *Emerging Professionals: How to Attract and Keep Them*. Retrieved from shrm.org: <https://www.shrm.org/resourcesandtools/hr-topics/talent-acquisition/pages/emerging-professionals-how-to-attract-and-keep-them.aspx>
- Koch, T., Gerber, C., & J. De Klerk, J. (2018). The impact of social media on recruitment: are you LinkedIn? *SA Journal of Human Resource Management*, 1-14.
- Kopp, D., Hangartner, D., & Siegenthaler, M. (2021). Monitoring hiring discrimination through online recruitment platforms. *Nature*, 572-576.
- Libuv in Node.js*. (2021, September 3). Retrieved from geeksforgeeks.org: <https://www.geeksforgeeks.org/libuv-in-node-js/>
- Marion, S. (2023, May 31). *How to create an OpenAI API key*. Retrieved from gptforwork.com: <https://gptforwork.com/setup/how-to-create-openai-api-key>
- Poddar, R. (2022, March 24). *What is a JWT? Understanding JSON Web Tokens*. Retrieved from supertokens.com: <https://supertokens.com/blog/what-is-jwt>
- Usman, M., & Radulescu, M. (2022). Examining the role of nuclear and renewable energy in reducing carbon footprint: Does the role of technological innovation really create some difference? *Science of The Total Environment*.
- What is Responsive Design?* (2022, December 14). Retrieved from interaction-design.org: <https://www.interaction-design.org/literature/topics/responsive-design>