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

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

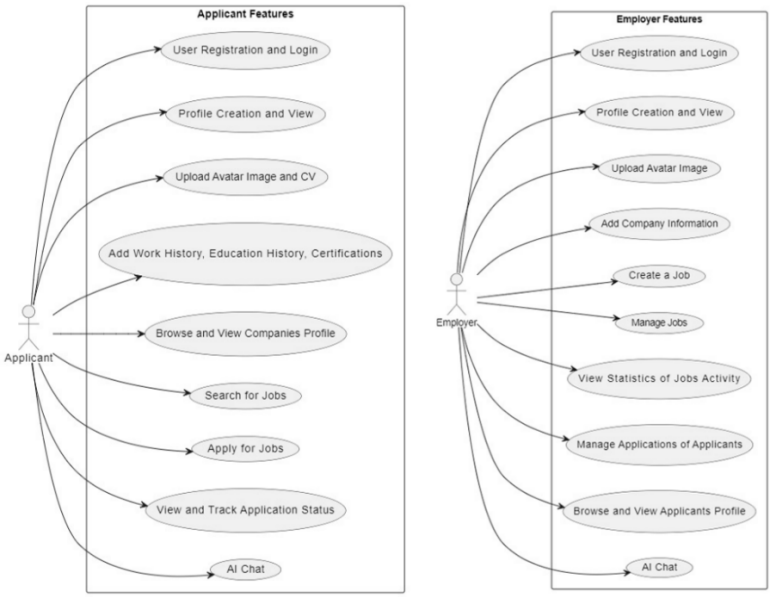
The goals of a project:

- 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 and LinkedIn and Indeed. These systems allow companies to post jobs and for applicant 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 bot 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 as 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:**

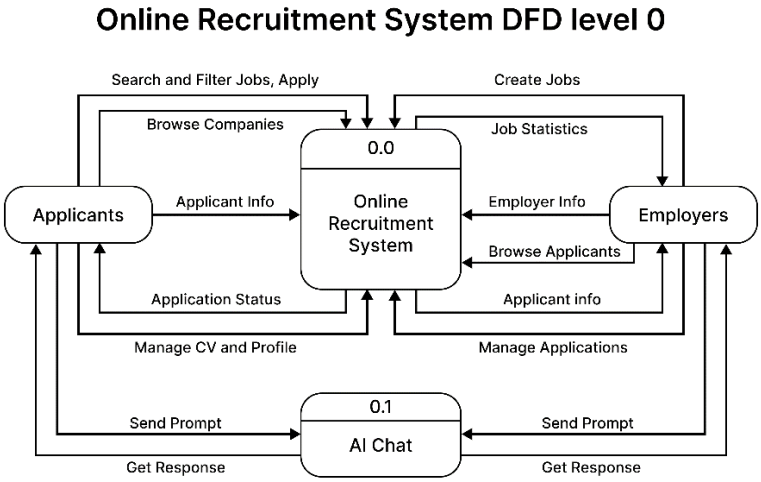
  Figma is a cloud-based design and prototyping tool that offers several benefits, including real-time collaboration, a wide range of design features, and the ability to easily share and collaborate with team members and stakeholders.

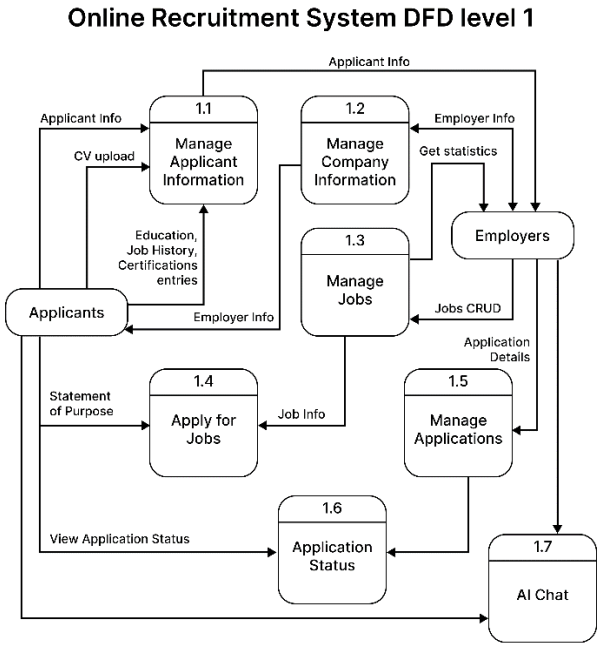
- **Git:**
  Git is a version control system that allows developers to track and manage changes to their code, and one of its main benefits is its ability to allow multiple developers to work on the same project concurrently while keeping track of all changes.
- **Postman:**
  Postman is a software service 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

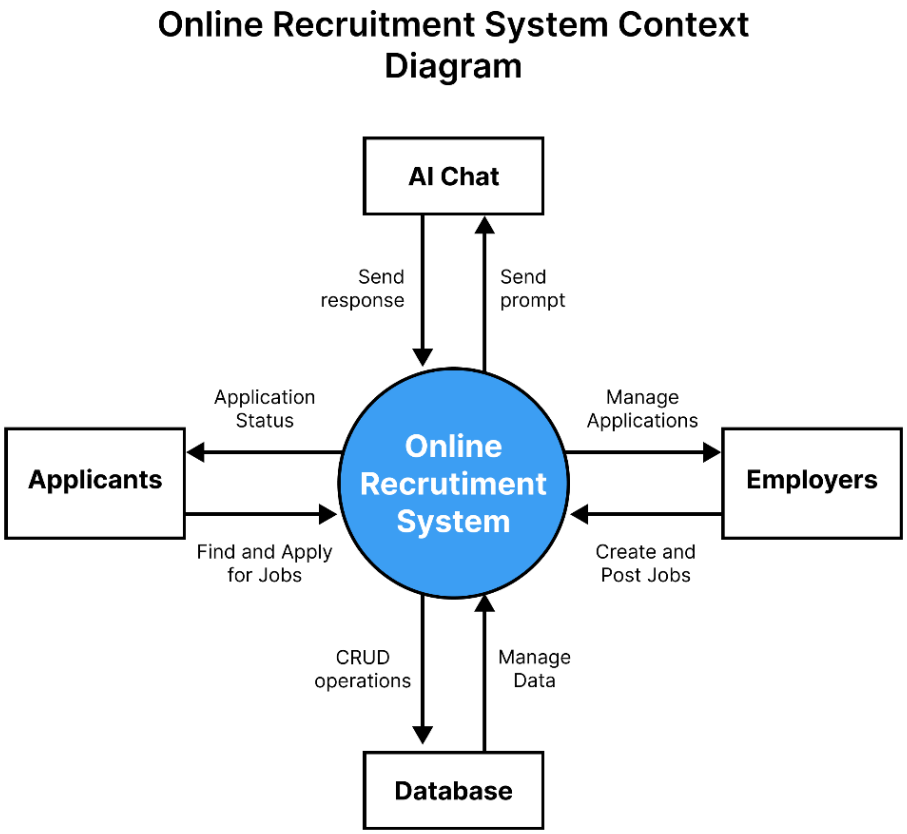
This is the application module that's represented in the DFD diagrams, you can see the applicants and employers. As it's seen here the main system is the Online Recruitment System itself along with the processes for both roles in my application. Also, there's a second application here which is AI Chat.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





4.2 Context Diagram

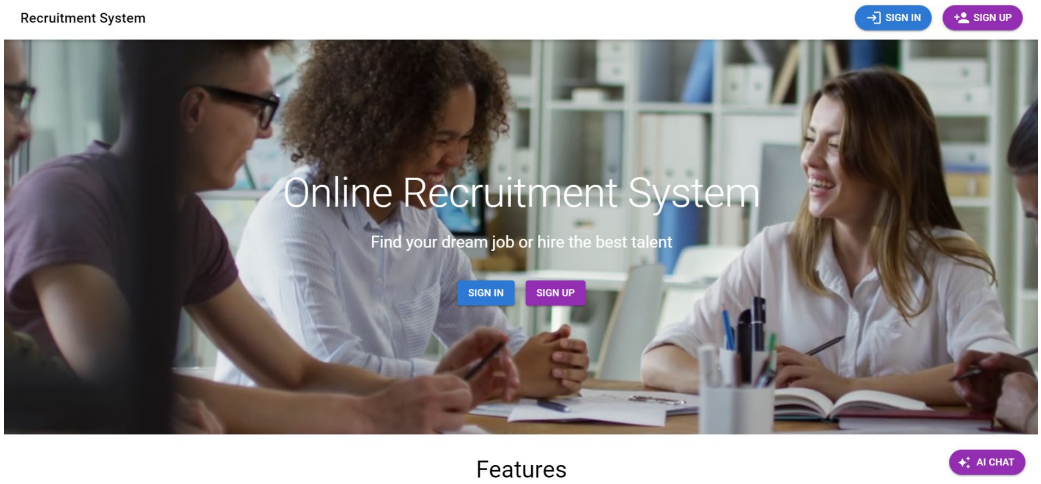
From this context diagram it is visible how the external entities can interact with the online recruitment system. This high-level abstraction makes it easy to display the main processes of an application.



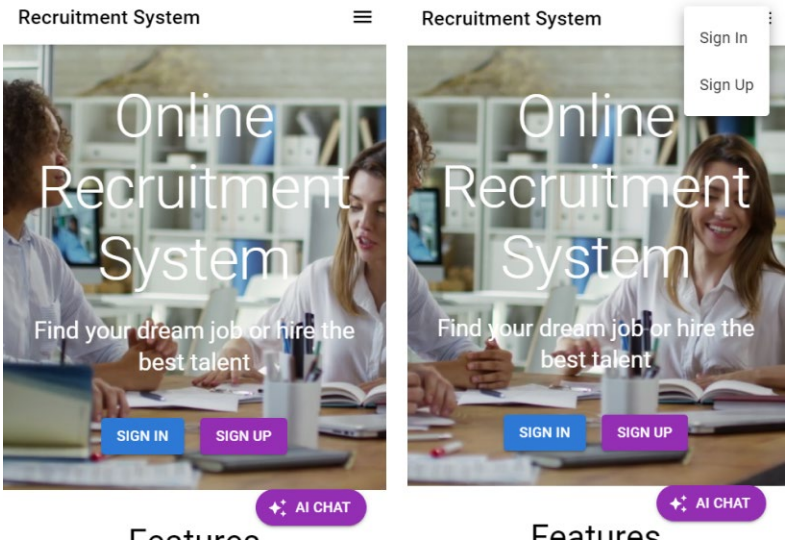
5. Methodology

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”.

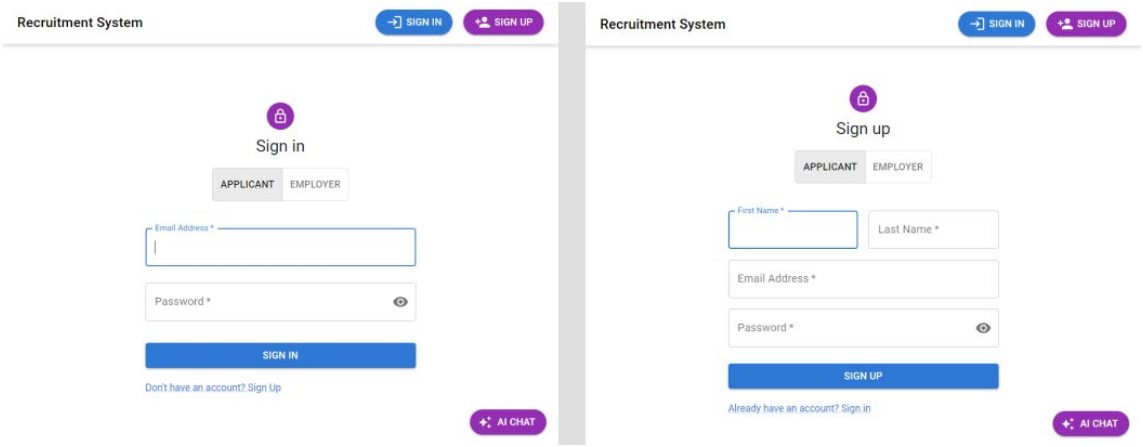


One of the main focuses for building a user-friendly interface is responsiveness, so that users can use the application from their smartphones and the layout will change accordingly, making the user experience more like using a mobile application (What is Responsive Design?, 2022).



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:

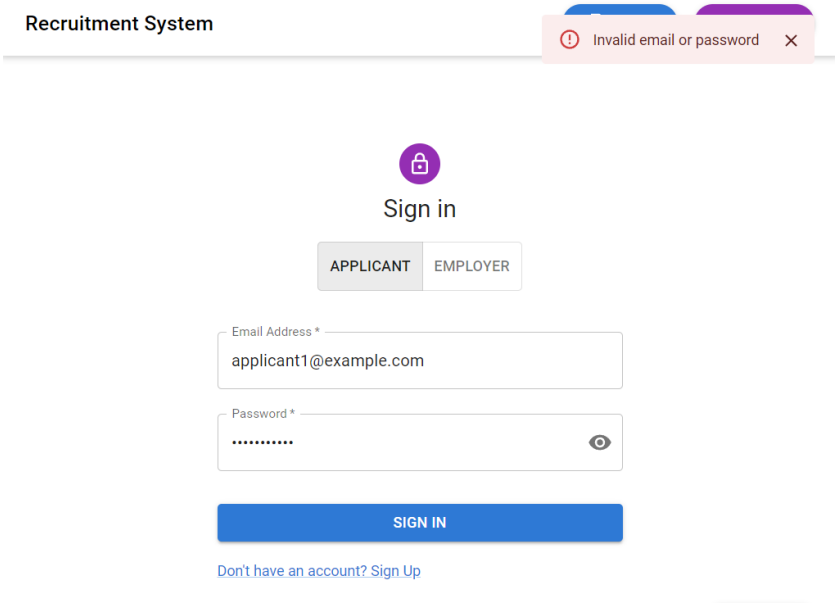


For the user input I implemented data validation both on the frontend and backend side to follow the best practices. Regular expressions are used to ensure that user entered the email in correct form, for the frontend validation I used the react-hook-form library.

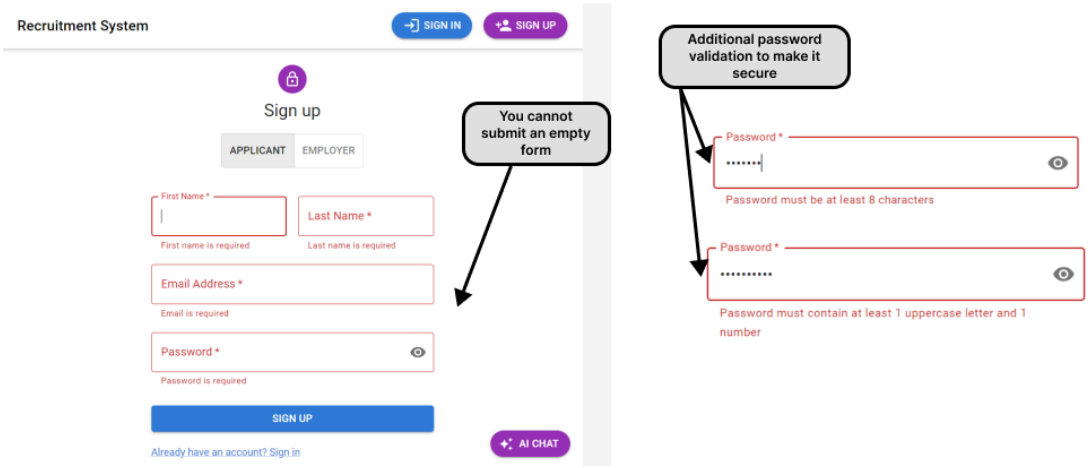
Below is the code snippet responsible for the frontend data validation:

```
<Box component="form" noValidate sx={{ mt: 1 }} onSubmit={handleSubmit(onSubmit)}>
  <TextField
    margin="normal"
    required
    fullWidth
    id="email"
    label="Email Address"
    name="email"
    autoComplete="email"
    autoFocus
    {...register("email", {
      required: 'Email is required',
      pattern: {
        value: /^[\w-]+(\.[\w-]+)*@([\w-]+\.)+[a-zA-Z]{2,7}$/,
        message: 'Wrong email format',
      },
    })}
    error={!!errors.email}
    helperText={errors.email && errors.email.message}
  />
  <TextField
    margin="normal"
    required
    fullWidth
    name="password"
    label="Password"
    type={showPassword ? 'text' : 'password'}
    id="password"
    autoComplete="current-password"
    {...register("password", {
      required: 'Password is required',
    })}
    error={!!errors.password}
    helperText={errors.password && errors.password.message}
    InputProps={{
      endAdornment: (
        <InputAdornment position="end">
          <IconButton
            onClick={togglePasswordVisibility}
            edge="end"
          >
            {showPassword ? <VisibilityOff /> : <Visibility />}
          </IconButton>
        </InputAdornment>
      ),
    }}
  />
```

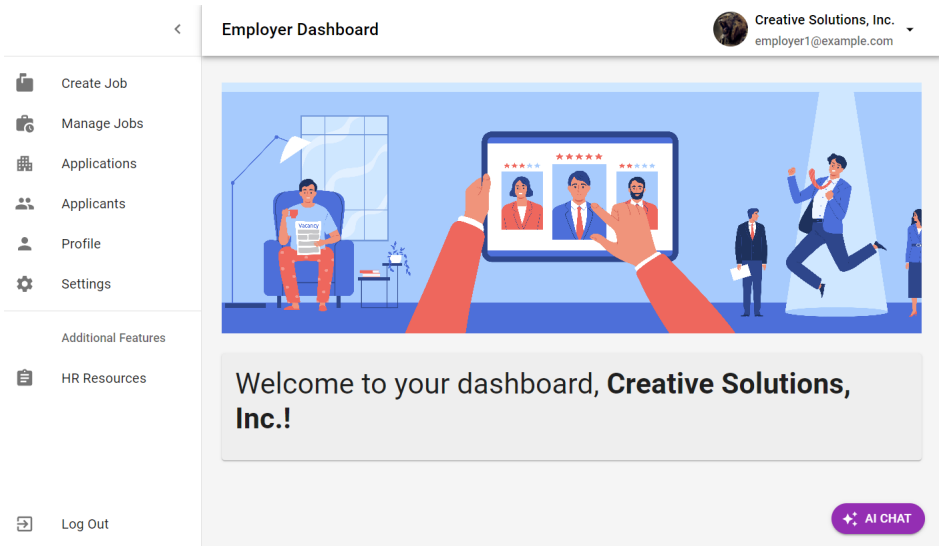
If user entered wrong credentials the app will notify the user.



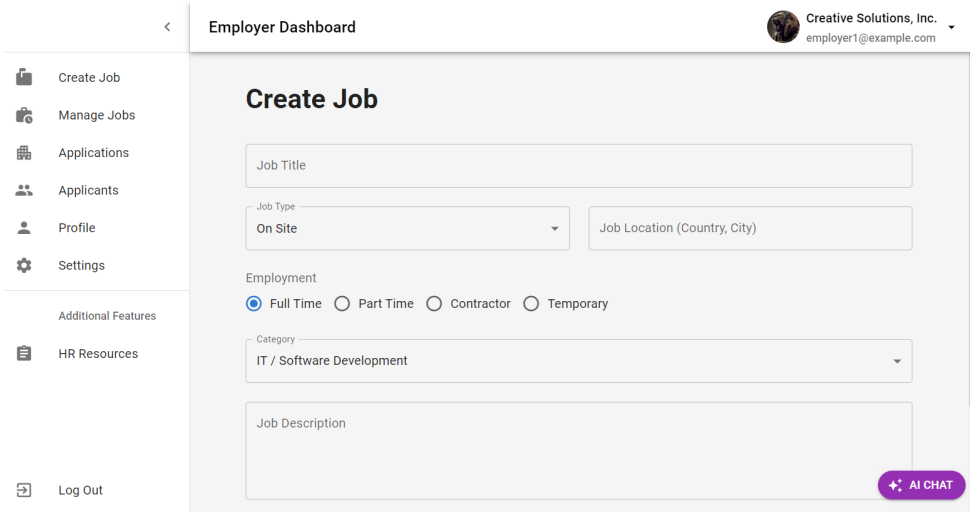
Here's the additional form validation:



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.

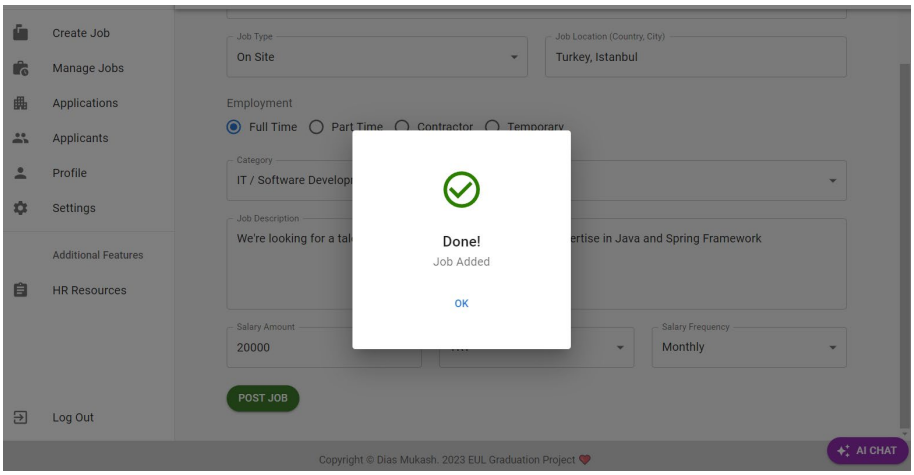


Create jobs page. Here employers can post jobs. There're many properties for making a job post, like "Job Title", "Job Type", "Job Location", "Employment", "Category", "Job Description", "Salary Amount", "Currency", "Salary Frequency" and "Post Job" button.

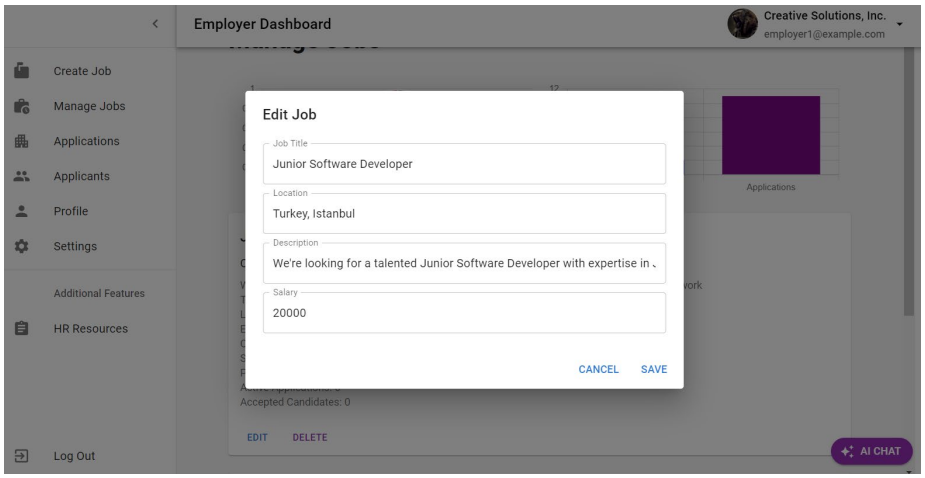


These fields are under the validation control as well, since employers cannot post an empty job posting, and for the "Salary Amount" field it strictly validates the value to be a number. All the fields are required, so when employer press the "Post Job" button, the new job posting is created.

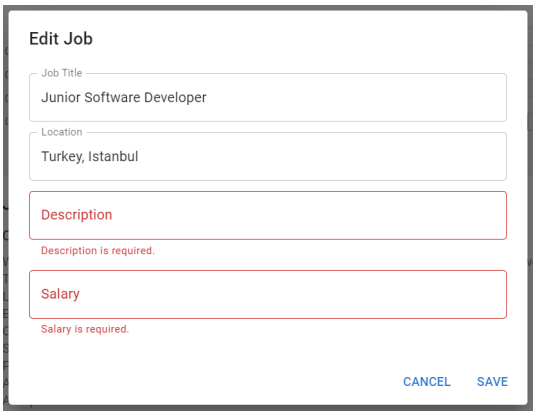
Once the job is posted, the user gets the confirmation success dialog box:



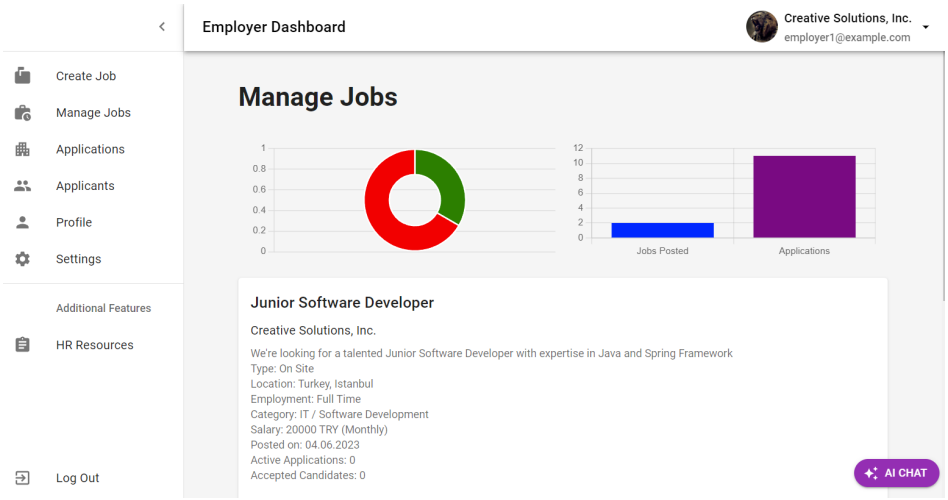
Manage jobs page. In this page employers can see and manage the jobs that they've created along with the job information. If employers click on edit button, they can edit the necessary information for the particular job position.



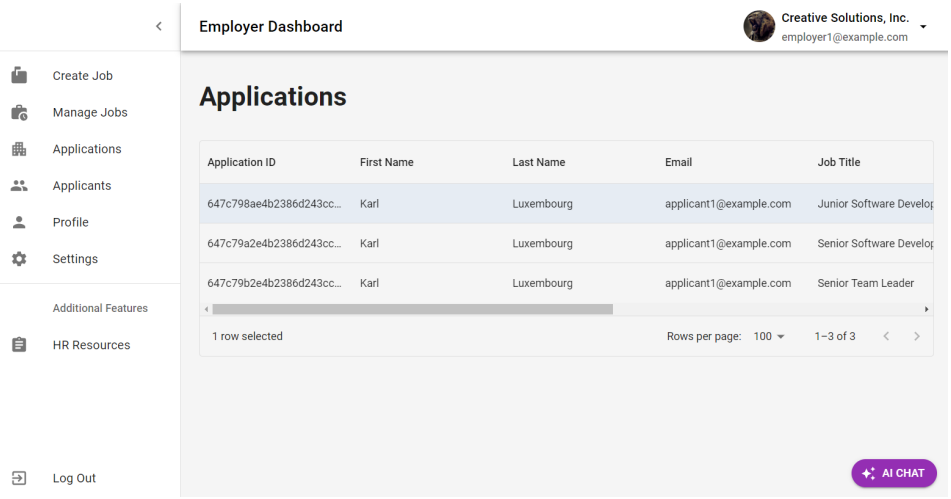
Edit job form cannot be empty if an employer wants to change the information, again it's the form validation which makes the application secure and reduces potential errors and bugs.



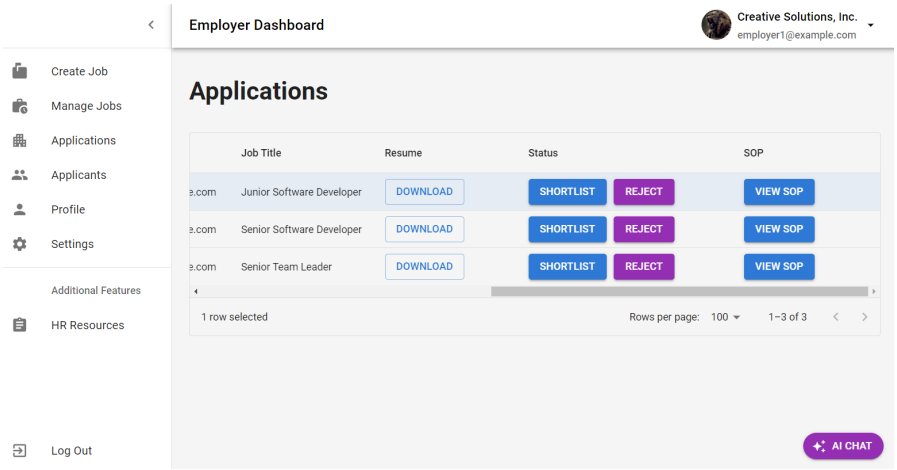
In the manage jobs page employers can see the statistics of the job activity, the number of accepted, shortlisted and rejected applicants, the number of jobs posted and the total applications count.



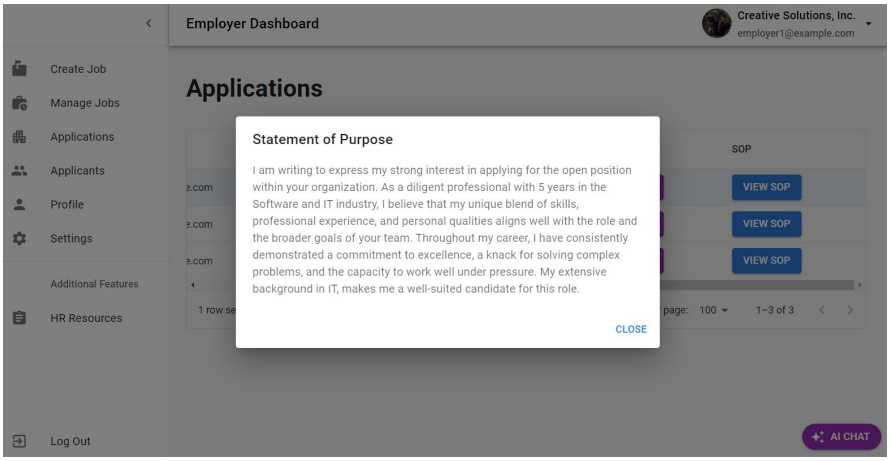
Applications tab. In this page employers can manage the applications. It's represented as the centralized table view, data grid for tracking applicants. When applicants successfully submit their applications, employers are getting their information and can decide whether to shortlist, accept of reject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



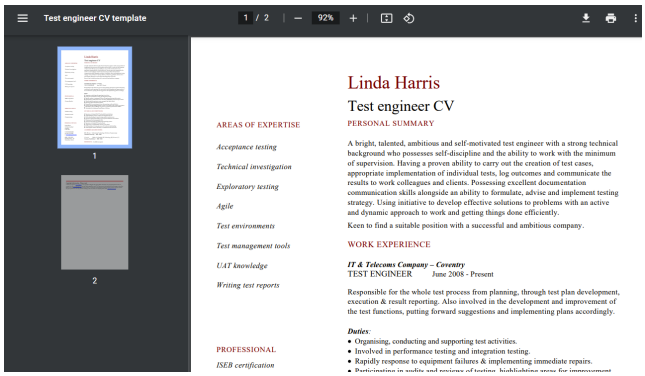
Here’s the continuation of the table, you can see the “Download”, “Shortlist”, “Reject” buttons and “View SOP” buttons.



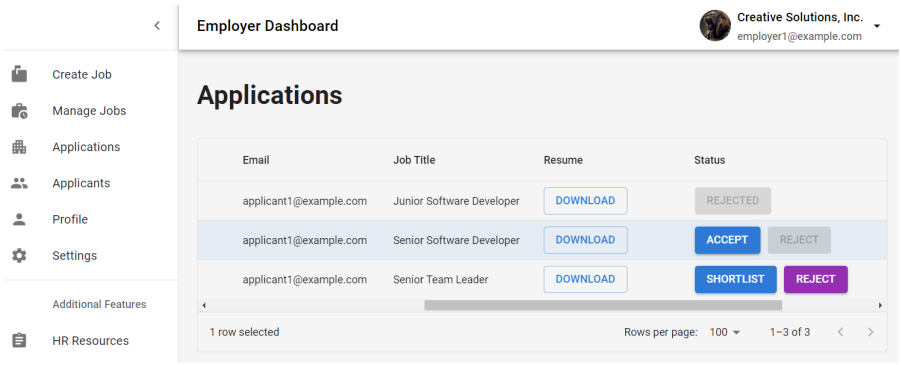
Here’s the continuation of the table, you can see the “Download”, “Shortlist”, “Reject” buttons and “View SOP” buttons. When employers click on “View SOP” button, they can see the statement of purpose sent by applicants, statement of purpose makes the application process clear for applicants and companies,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



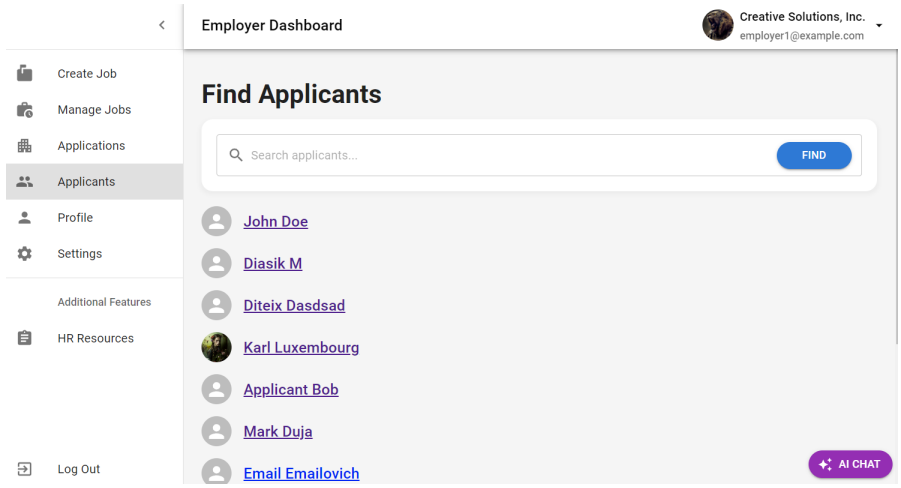
When employers click on the “Download” button in the “Resume” field, they can view and download the resume. These days companies allocate resources to have an access to a CV database, since it’s the most valuable asset in the recruitment systems. My application solves that problem by providing the Resume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



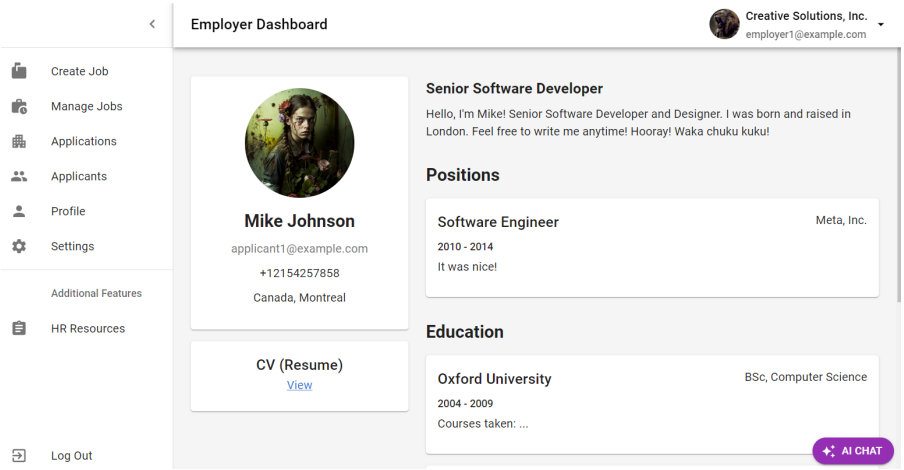
Employers can manage the application by choosing the shortlist, reject or accept options.



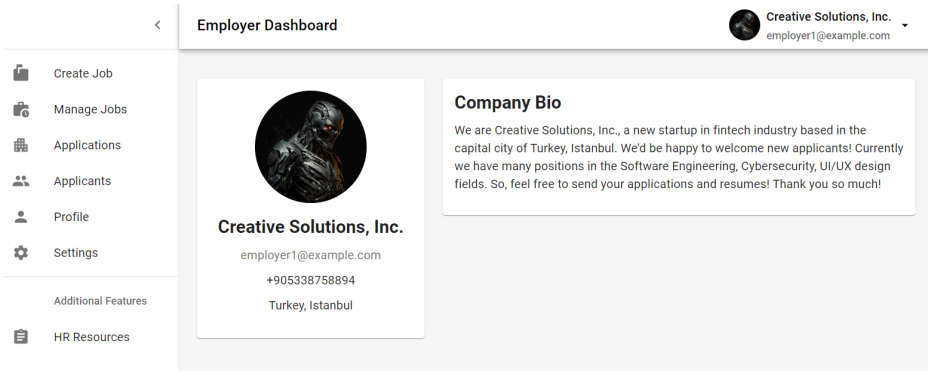
Applicants page. In the Applicants page employers can browse and search for applicants profiles. On the top of this page there's a search box.



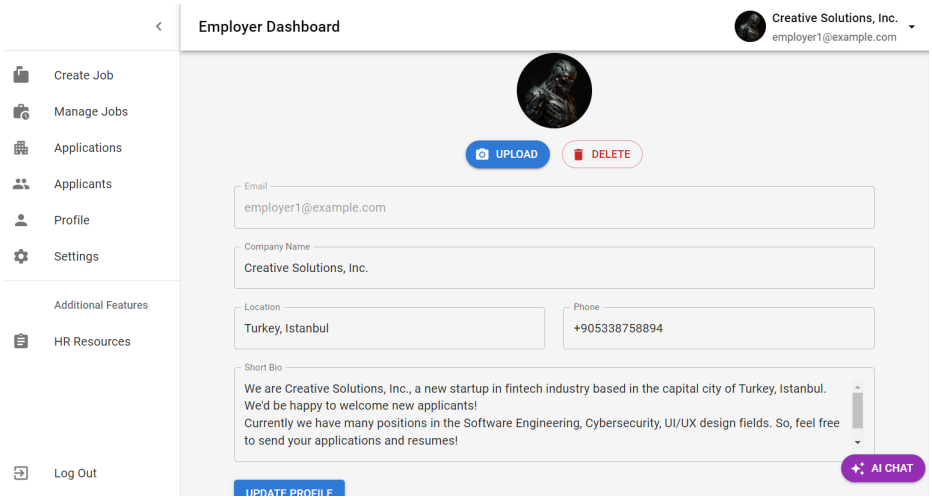
If employers click on the applicant's name, they can view their profile and from the profile page they can download applicant's CV and view all the necessary information about the applicant.



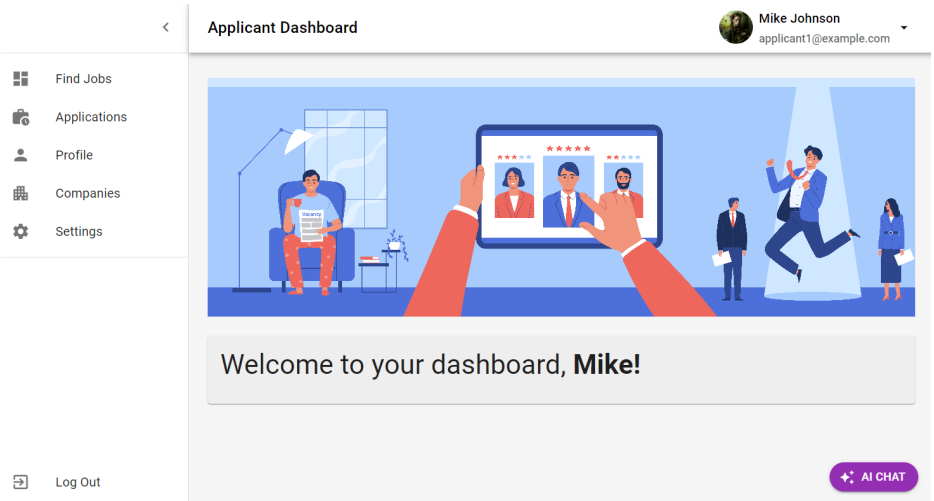
Profile page. In the profile page employers can view their profile and view how it is going to look like from the applicant's perspective.



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.

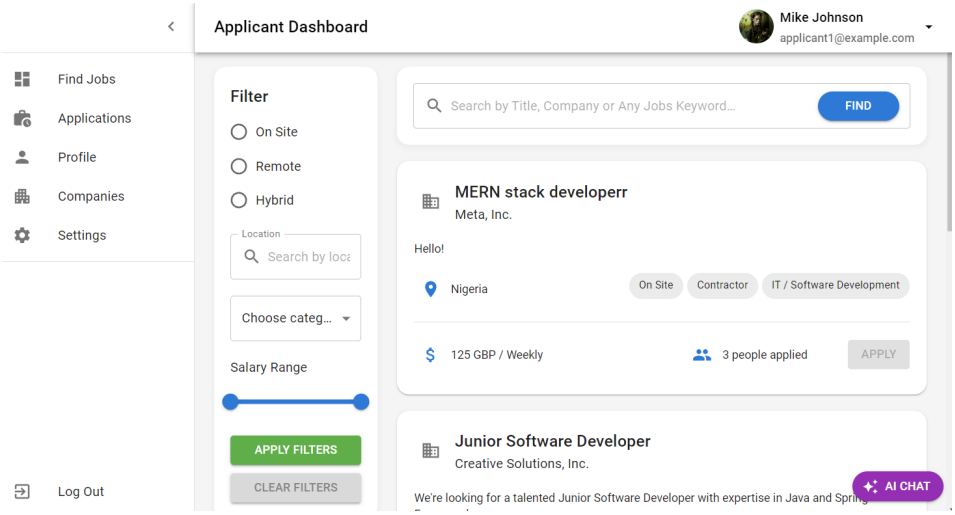


Applicant's side. When users are signing in, they can choose their roles as I mentioned before. If users choose an applicant's role, they're signing in as applicants.

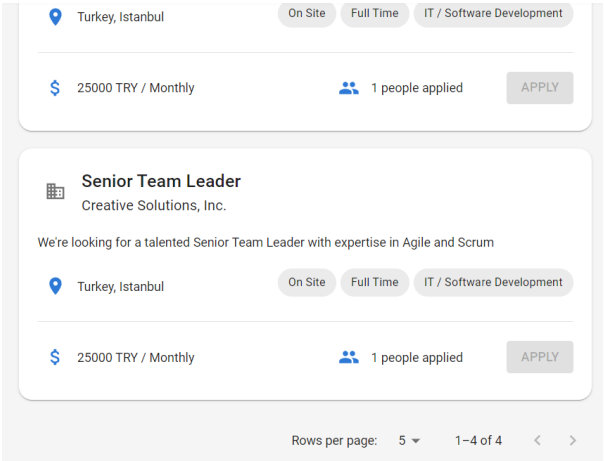


Find jobs page. In the “Find Jobs” page applicants can browse jobs by using the filter box on the left side of the page or the search field on the top. Filtering includes various parameters, which applicants can use to find the right job position for them, these parameters include: Filter by Job Type,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

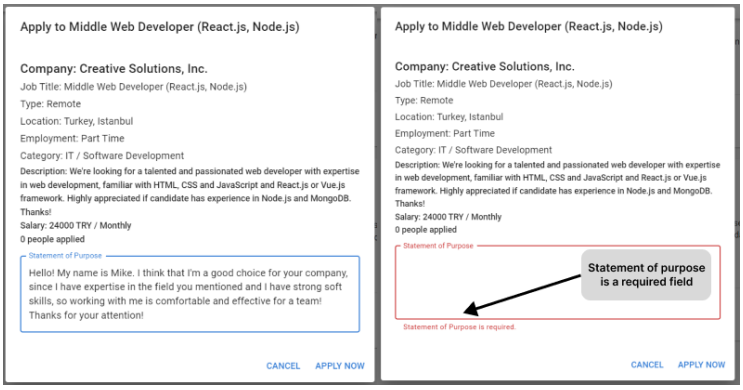
The job card’s layout is informative and well-designed so that the applicants can see the job details. It includes: Position Title, Company Name, Job Description, Job Type, Employment Type, Category, Location, Salary Amount, Salary Currency, Salary Frequency and the Number of Applicants applied and the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



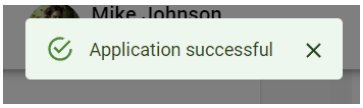
To display the jobs in a convenient format the table pagination is implemented to view the specified number of job postings on a single page. Here users can choose rows per page and navigate between pages.



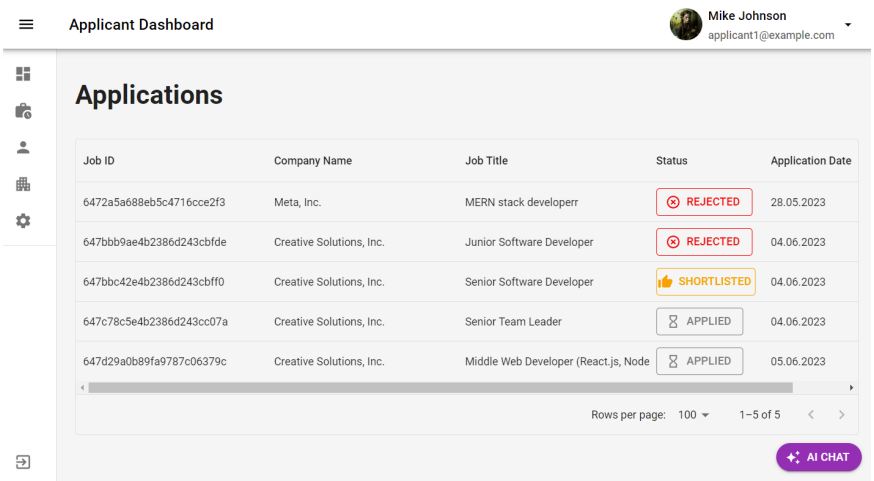
To apply for a job, applicants must send a statement of purpose (cover letter) to tell briefly about themselves and express why they want to apply for a particular job.



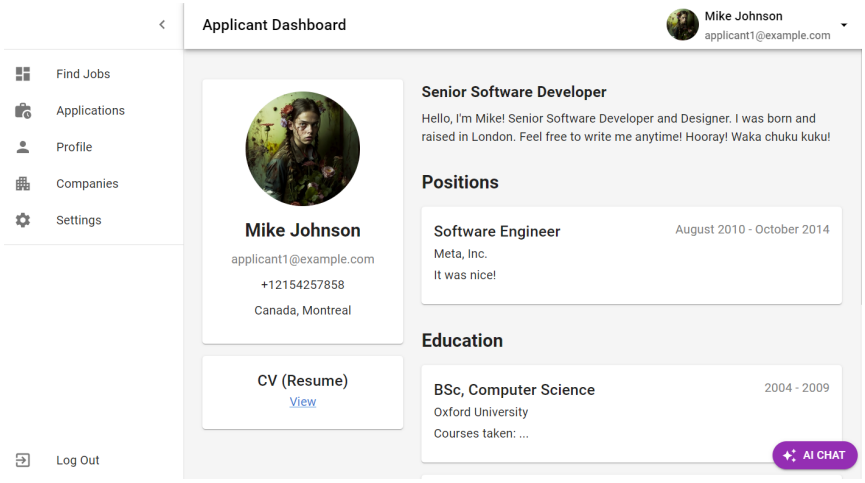
If the application is successful, users get the alert.



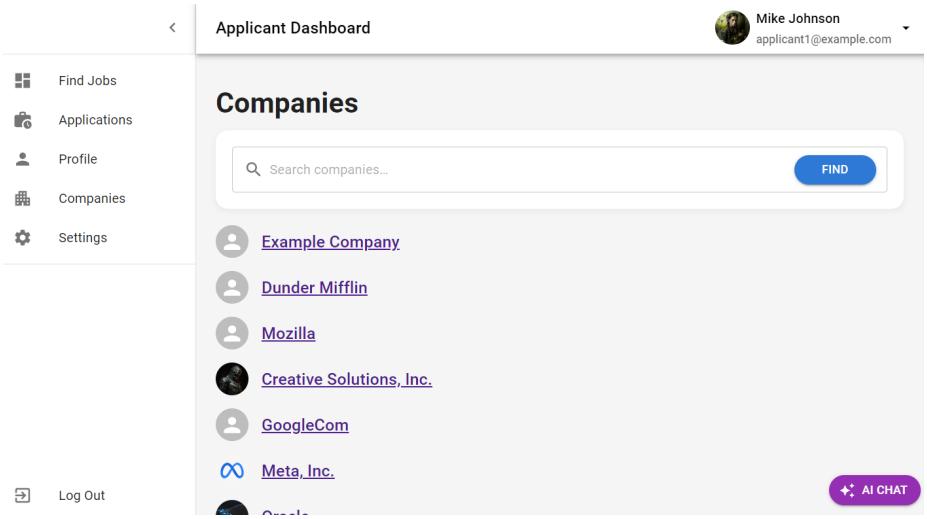
Applications page. This is one of the main pages for applicants, since it provides a centralized table view for the applications and their status.



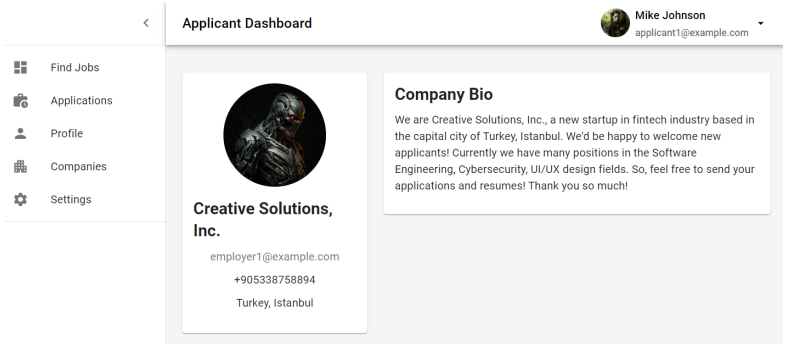
Profile page. Here applicants can see their profiles, with all the necessary information like profile image, personal data, CV, title, bio, job history, education and certifications fields.



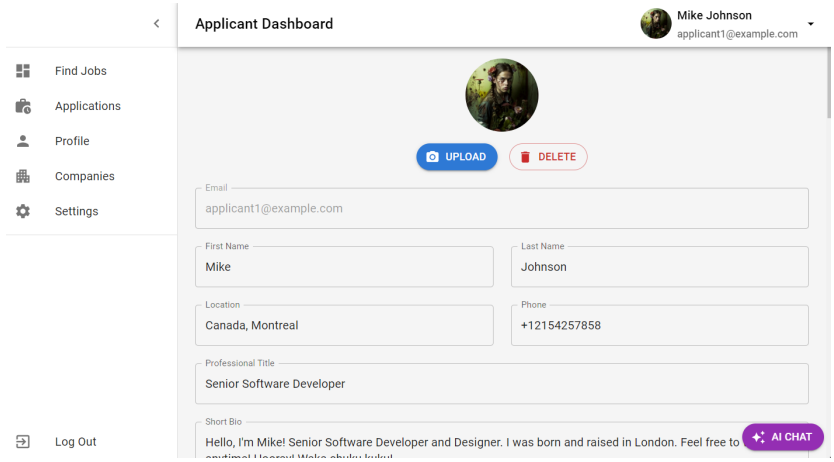
Companies page. The page where applicants can search companies. And to view the company's profile applicant can click on one of the company list elements.

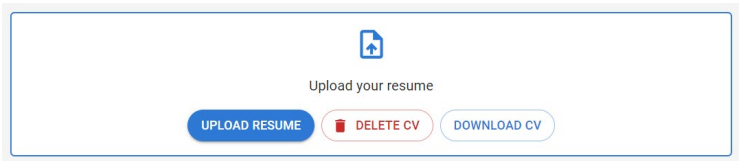


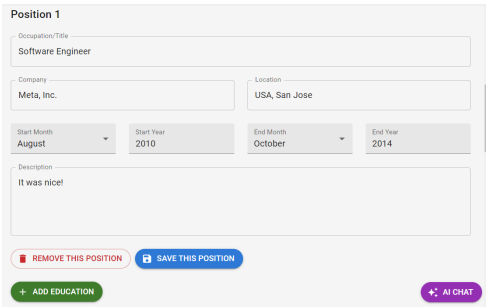
The company profile.

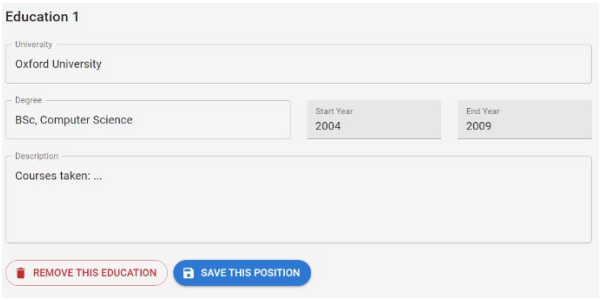


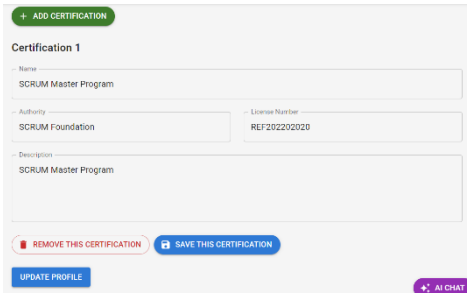
Settings page. The page where applicants can upload their profile pictures, change name if needed, add personal data, upload and store resume, add job history, education and certification entries.





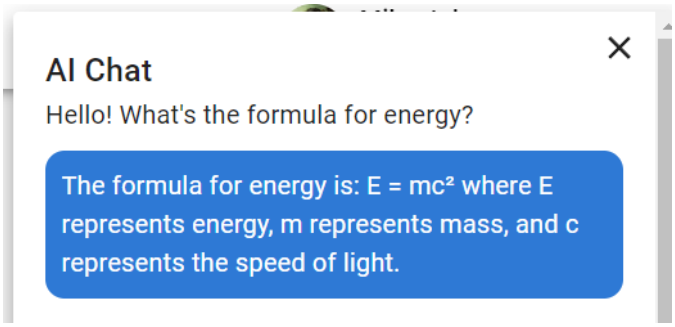






In the rapidly evolving landscape of technology, cutting-edge tools such as OpenAI's ChatGPT have become increasingly prevalent. OpenAI has established accessible APIs, thus allowing developers to incorporate their advanced products into their own applications. Consequently, I decided to integrate an 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.

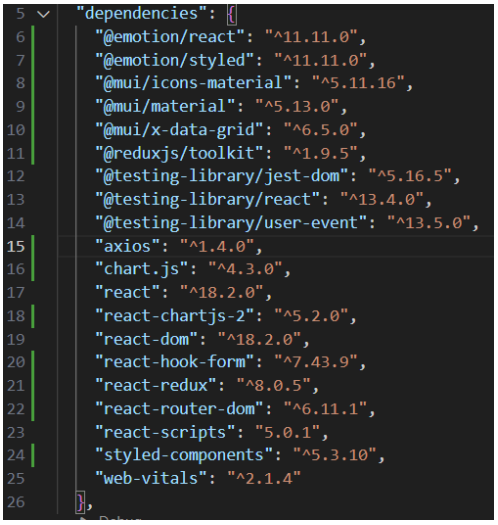


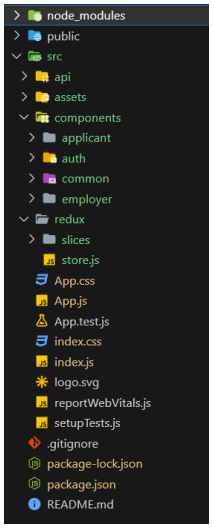
Frontend:

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.

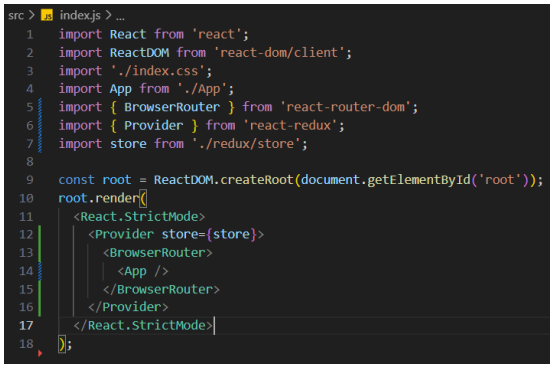
Material-UI is a popular React UI framework that provides a set of pre-built components, enabling developers to create visually appealing and responsive user interfaces with ease. It adheres to Google's Material Design guidelines, ensuring a modern and clean aesthetic that enhances user experience.

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





The 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.



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.

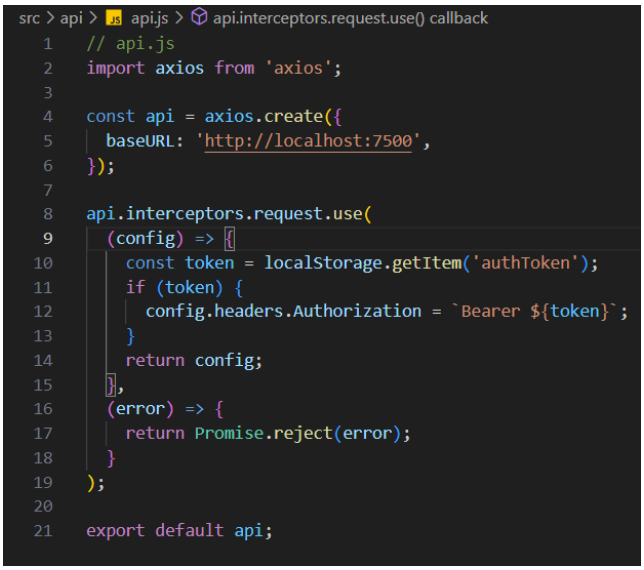
To ensure that special roles have access only to their own Routes, I have implemented Private Routes. It accesses the Redux store to know whether the user is logged in and also is accessing the role value, which are “Applicant” and “Employer” in my case. Once the application ensures that I’m a logged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

```jsx
import { useSelector } from 'react-redux';
import { Navigate } from 'react-router-dom';

function PrivateRoute({ role, children }) {
  const userRole = useSelector((state) => state.auth.user.role); // get t
  const isAuthenticated = useSelector((state) => state.auth.isLoggedIn);

  if (isAuthenticated && userRole === role) {
    return children;
  } else {
    return <Navigate to="/signin" replace />;
  }
}

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.

```jsx
import { configureStore } from '@reduxjs/toolkit';
import userReducer from './slices/userSlice';
import authReducer from './slices/authSlice';

const preloadedState = () => {
  try {
    const authToken = localStorage.getItem('authToken');
    const user = JSON.parse(localStorage.getItem('user'));

    if (authToken && user) {
      return {
        auth: {
          isLoggedIn: true,
          isLoggingIn: false,
          loginError: null,
          user,
        }
      }
    } else {
      return undefined;
    }
  } catch(err) {
    return undefined;
  }
}
```

‘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.



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.

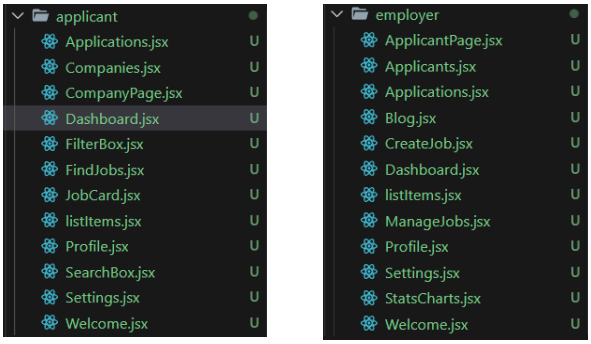
State variables are initialized using ‘useState’ to track the user role (either “Applicant” or “Employer”), password visibility, and Snackbar (to show alerts). ‘useForm’ from ‘react-hook-form’ is used to handle from validation and submission. ‘useDispatch’ from ‘redux’ is used to dispatch actions to 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.

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.

```jsx
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.

```jsx
src > components > applicant > FindJobs.jsx > FindJobsPage
import React, { useState, useEffect } from 'react';
import { Grid, Typography, Container, Dialog, DialogTitle, DialogContent, DialogActions,
import JobCard from './JobCard';
import FilterBox from './FilterBox';
import SearchBox from './SearchBox';
import api from '../../api/api';
import { useForm } from 'react-hook-form';
import { Snackbar, Alert } from '@mui/material';

export default function FindJobsPage() {
  const [jobs, setJobs] = useState([]);
  const [filteredJobs, setFilteredJobs] = useState([]);
  const [search, setSearch] = useState('');
  const [filterSettings, setFilterSettings] = useState({
    type: '',
    location: '',
    category: '',
    salaryRange: [0, 1000000],
  });

  // pagination
  const [page, setPage] = useState(0);
  const [rowsPerPage, setRowsPerPage] = useState(5);

  // fetch jobs
  useEffect(() => {
    const fetchJobs = async () => {
      try {
        const response = await api.get('/jobs');
        setJobs(response.data);
        setFilteredJobs(response.data);
      } catch (error) {
        console.error('Failed to fetch jobs:', error);
      }
    };
    fetchJobs();
```

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).

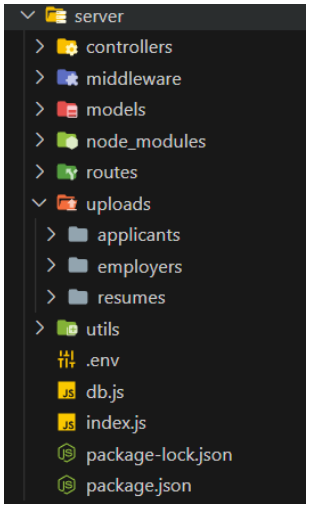
What is blocking I/O and why non-blocking I/O is better?

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 paralleling 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.



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
const {Schema, model} = require('mongoose');

const PositionSchema = new Schema({
  title: { type: String },
  company: { type: String },
  location: { type: String },
  startMonth: { type: String },
  startYear: { type: Number },
  endMonth: { type: String },
  endYear: { type: Number },
  description: { type: String },
}, {_id: true});

const EducationSchema = new Schema({
  university: { type: String },
  degree: { type: String },
  startYear: { type: Number },
  endYear: { type: Number },
  description: { type: String },
}, {_id: true});

const CertificationSchema = new Schema({
  name: { type: String },
  authority: { type: String },
  licenseNumber: { type: String },
  description: { type: String }
}, {_id: true});
```

Application.js

```
server > models > Application.js > schema
const mongoose = require("mongoose");

let schema = new mongoose.Schema(
  {
    userId: {
      type: mongoose.Schema.Types.ObjectId,
      required: true,
      ref: 'Applicant'
    },
    recruiterId: {
      type: mongoose.Schema.Types.ObjectId,
      required: true,
      ref: 'Employer'
    },
    jobId: {
      type: mongoose.Schema.Types.ObjectId,
      required: true,
      ref: 'Job'
    },
    status: {
      type: String,
      enum: [
        "applied", // when an applicant has applied
        "shortlisted", // when an applicant is shor
        "accepted", // when an applicant is accepte
        "rejected", // when an applicant is rejecte
        "deleted" // when any job is deleted
```

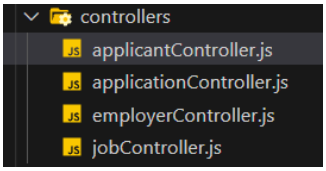
Employer.js

```
server > models > Employer.js > ...
const {Schema, model} = require('mongoose');

const EmployerSchema = new Schema({
  companyName: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  role: { type: String, default: 'Employer' },
  avatar: { type: String },
  phone: { type: String },
  location: { type: String },
  shortBio: { type: String }
})

module.exports = model('Employer', EmployerSchema);
```

Job.js

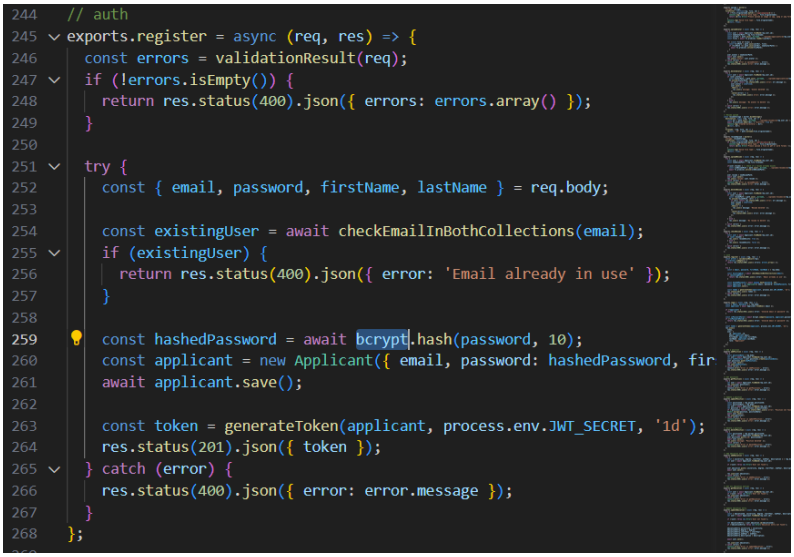
```
server > models > Job.js > JobSchema
const { Schema, model } = require('mongoose');

const JobSchema = new Schema({
  userId: {
    type: Schema.Types.ObjectId,
    required: true,
    ref: 'Employer'
  },
  companyName: { type: String, required: true },
  title: { type: String, required: true },
  type: { type: String, required: true },
  location: { type: String, required: true },
  employment: { type: String, required: true },
  category: { type: String, required: true },
  description: { type: String, required: true },
  salary: { type: String, required: true },
  currency: { type: String, required: true },
  salaryFrequency: { type: String, required: true },
  dateOfPosting: { type: Date, default: Date.now },
  activeApplications: { type: Number, default: 0 },
  acceptedCandidates: { type: Number, default: 0 },
});

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.

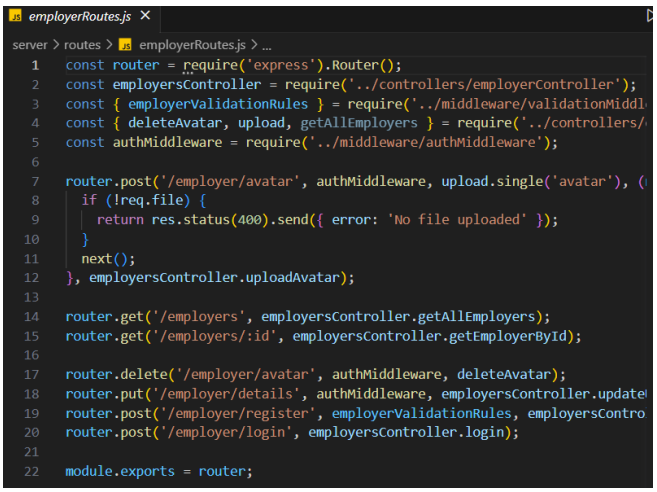
The register function takes the request body, destructures it by taking, email, password, user name, checks if there's an existing user with the same email, if not it hashes the password, ans saves the new user, then it generates the token and sends the token as a response.



```
// auth
exports.register = async (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  try {
    const { email, password, firstName, lastName } = req.body;

    const existingUser = await checkEmailInBothCollections(email);
    if (existingUser) {
      return res.status(400).json({ error: 'Email already in use' });
    }

    const hashedPassword = await bcrypt.hash(password, 10);
    const applicant = new Applicant({ email, password: hashedPassword, fir
    await applicant.save();

    const token = generateToken(applicant, process.env.JWT_SECRET, '1d');
    res.status(201).json({ token });
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
};
```

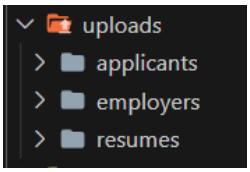
Routes. It's serving as the blueprint for the application's endpoint structure and the roadmap for the HTTP requests. For the authenticated requests it has to be used along with the authMiddleware, to prevent unauthorized access.

```
jobRoutes.js
server > routes > jobRoutes.js > ...
const router = require('express').Router();
const jobController = require('../controllers/jobController');
const authMiddleware = require('../middleware/authMiddleware');

router.get('/employer/jobs', authMiddleware, jobController.getEmployerJobs
router.get('/jobs', jobController.getAllJobs);
router.post('/jobs', authMiddleware, jobController.createJob);
router.put('/jobs/:id', authMiddleware, jobController.updateJob);
router.delete('/jobs/:id', authMiddleware, jobController.deleteJob);

router.get('/employer/:id/stats', authMiddleware, jobController.getEmploye

module.exports = router;
```



```
employerRoutes.js
server > routes > employerRoutes.js > ...
const router = require('express').Router();
const employersController = require('../controllers/employerController');
const { employerValidationRules } = require('../middleware/validationMiddl
const { deleteAvatar, upload, getAllEmployers } = require('../controllers/
const authMiddleware = require('../middleware/authMiddleware');

router.post('/employer/avatar', authMiddleware, upload.single('avatar'), (
  if (!req.file) {
    return res.status(400).send({ error: 'No file uploaded' });
  }
  next();
}, employersController.uploadAvatar);

router.get('/employers', employersController.getAllEmployers);
router.get('/employers/:id', employersController.getEmployerById);

router.delete('/employer/avatar', authMiddleware, deleteAvatar);
router.put('/employer/details', authMiddleware, employersController.update
router.post('/employer/register', employerValidationRules, employersContro
router.post('/employer/login', employersController.login);

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
    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
  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
  body('companyName').notEmpty().withMessage('Company name is required')
];

module.exports = { applicantValidationRules, employerValidationRules };
```

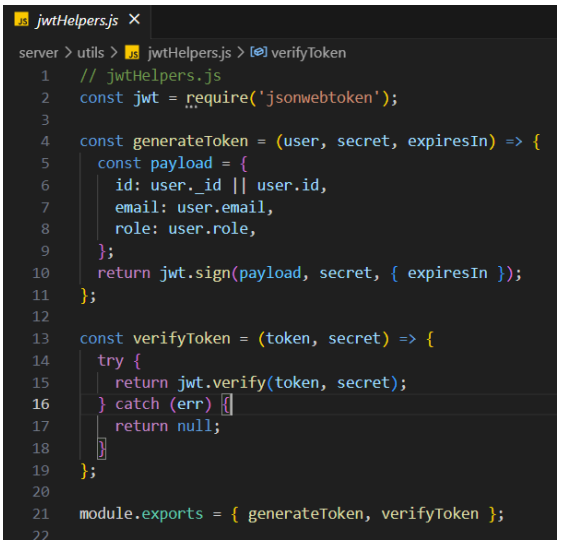
authMiddlware.js file defines a middleware function named 'authMiddleware'. It imports a utility function named verifyToken from the jwtHelpers module. This function is used to verify JWT tokens. The middleware function retrieves the authorization header from the incoming HTTP request. This header should 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.

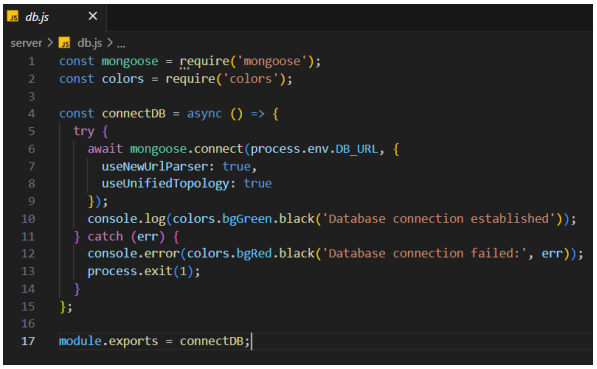
A JSON Web Token, or JWT, is a standardized, in some cases signed and/or encrypted data packaging format that is used to transmit information securely between two parties in client-server applications; tokens in turn are created on the server side, signed with a secret key, and sent to the client; the 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.



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.



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).

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 automated 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 increate 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