Job Dating Web Application with Modern Web Technologies

Dilip Poudel

Bachelor’s Thesis
Business Information Technology
2021

Author(s)
Dilip Poudel

Degree programme
Business Information Technology (Bite2017)

Report/thesis title
Job Dating Web Application with modern technologies

Number of pages
41

The main goal of this research is to identify the modern stacks and technologies which are most commonly used in the present competitive market and implement those technologies in a job dating full stack web application. The paper will provide a description and implementation of different technologies which are implemented in building the product. It will go through the full stack technology and provide a general idea of the recent technologies that are essential in modern web applications.

This is a product-based research, resulting in a creation of a full stack web application. The main objective is to present fundamental concepts of the variant web elements and stacks, implement those in practice, and handover an MVP (Minimum Viable Product) to the client.

The idea behind this research is to find a solution to some of the problems job seekers may encounter in the present job market. The current job market is competitive and full of flaws, especially relating to the recruitment process. Many highly skilled job applicants struggle to find jobs. Therefore, we would like to present a fairer job application process by considering the perspective of the job applicant and providing a solution via our product. In consequence, the project is created to aid the combat against discrimination in the job market.

The core functionality of this web application is to find and match relevant applicants to companies, giving them a chance to prove themselves. The content will consist mainly in two categories: The theoretical part and the practical part. The theoretical part will describe the technologies - why they are important to modern web application and what are their use cases and roles in current web technology. The practical part will consist of the project development.

Keywords
Full Stack, Frontend, Backend, Modern Web Technologies

Table of Contents

1 Introduction

The evolution in web technologies has been changing rapidly over the past decades, which is simultaneously exciting news and a challenge for web developers. Web development used to be nothing more than the use of HTML, CSS and JavaScript on the client side and PHP or some other language on the server side to build software that can be run in web browsers via the internet. Because of the new stacks and technologies, the structure, business logic and scopes have been broadening dramatically in web applications. Due to the vast changes in the field, such as the emergence of new programming languages, web frameworks, and libraries, most of the companies focus on using updated technologies to make highly reliable, scalable and maintainable software. Therefore, it is important for a developer to have a deep understanding of those stacks.

In the present market, it is pivotal for a company to choose the right technologies before starting the software development process. There are many reasons behind this. To mention a few, the selection of appropriate technology and stack allows significant improvements such as rapid prototyping, code reusability, constant iteration, DRY (Don't Repeat Yourself) concept in code logic, high efficiency, and robustness. Another benefit of using modern technology is their capacity for a developer to provide clean code to other developers who can maintain it easily in the future which will reduce maintenance costs. Another example of minimizing costs is to use a prototype from an existing clean and maintainable project's code instead of building an application from scratch. (Web Dev Zone, Nov 04, 2020)

This thesis is carried out as a research project on the topic *A job dating web application with modern technologies*. The main objective of the paper is to study the different components of the most popular stacks and tools to build a fast and interactive web application. The research is also suggesting the best design tools from a user experience point of view.

The thesis provides a clear concept in both the traditional approach and the modern approach of web application. It consists of mainly five chapters, which employ research approaches whose findings have been implemented in practice. This chapter started with the introduction section which is followed by chapters that build upon each other. The second chapter will provide a brief history of the traditional web technologies and the third chapter of the modern web technologies. Likewise, the fourth chapter is successively describing the various popular technologies which are applied in the project and it will include some snapshots of implementation of those technologies. The fifth chapter will show the implementation of various stacks and the development process from the initiation to end step. The conclusion of the report is presented in chapter 6.

The application that is going to be built as a team project work is a Full Stack Web Application called **Tindev**. It is a dating application type of a product but especially made for job hunting purposes. The concept of the application is to make the recruitment process more anonymous by hiding some personal information of the applicants, such as their age, nationality and gender, and concentrate on their skills and capacities which will be matched with the requirements of the companies by using a match algorithm. When the match is found, the application provides an option for a conversation between the job hunter and the company via the application chat feature, which can help the job seeker to get a job interview and land a job.

1.1 Company Introduction and the Solution

Integrify Oy is a digital consulting company and a teaching institute which helps immigrants to develop their technical skills by training them in knowledge of up-to-date industry skills and trying to find them job opportunities in the job market. The company was founded in 2016 during the refugee crisis and it has become a source of hope for students who are highly skilled but not getting jobs. They join the training program to polish and add new skills to their portfolio and co-operate with the company.

Integrify follows the B2B business model with the following competences: Web Development, Native Application Development, Backend Development, and Infrastructure. The main training subjects of the program are: Accessibility in Web, React JS, Node JS, TypeScript, AWS, Git, and GitHub. During the 4 - 8 months training period, the trainee will grow as a full stack developer and may possibly get an opportunity to work on a project at Integrify or one of the companies that Integrify collaborates with. In the best-case-scenario, the project work will lead to a permanent job position. Similar to Integrify, which tries to lower the threshold of finding a job for people with immigrant backgrounds, Tindev is created for the purpose of making the job seeking process less discriminating.

1.2 Objectives and Goals

The purpose of this thesis is to deploy the first version of web application with a Job Dating Web Application system that gives the possibility to create a profile for both the job seeker and the job employer and find the job that is matched to each other based on employer's requirements and the job seeker's skills. The thesis will go through the theory regarding the different stacks of modern web application through front end to backend including the design tools and collaboration tools. Firstly, the short theory of the stacks will be introduced to spread the basic idea of today's tools and stack to the audience and it will go through the major steps to build the product. The thesis goes through the theoretical concepts of most popular stacks with some figure of syntax and relevant stacks or may be the diagram.

The thesis will also introduce the used stacks of the project which are in high demand in the market in modern web development and the constructive steps to configure a real web application which is going to build. It will demonstrate the development of defined stacks in the built application and the basic introduction. The emphasis is placed on the building process of the web application that follows the competitive technologies. The various techs which are involved in building web applications are the keys of the thesis's contents. On the other hand, the report serves as a guideline for researching different stacks.

The report will contain the basic work introduction, workflow, and the main application of stack in the project. The audience will get the idea about modern web applications and how they are built up nowadays. In addition, the thesis will provide some theoretical concepts as well as code snapshots to demonstrate how the technological improvements make developing web apps easier. It will show that the code is more human readable in modern web development than what it was previously. Thus, providing the basic theoretical concepts on how to use modern web frameworks and technologies via implementation of a real web application is the main goal of the thesis.

1.1 Out of Scope/Limitations

The thesis has some limitations. Firstly, the thesis will not cover the research of the customer as it is only the creation. It provides some practical implementations with different figures to substantiate the instruction of building a web application with updated stacks. It supports the implementation to figure out the scope of different stacks in modern web development but will not cover the advanced features of those stacks. Only the needed amount of concept for developing the web application will be described. A full presentation of how to implement each used technology is not possible in the context of this report, but the basic idea about how, where and in which cases the technologies are implemented in modern web development will be presented.

In overall, the thesis will not be focused on which features are available in the application. Instead, the report will provide a general flow of the development process of the application. Furthermore, the application will not be deployed to the Google Cloud; It will still be in the development phase during the end of the thesis. However, to make sense of the deployment implementation on a basic level, Heroku and Netlify Cloud Deployment will be implemented.

2 Classical Approach and the History of Web Pages

The English scientist **Tim Berners-Lee** invented the World Wide Web, also known as WWW (CERN, 1991). He was successful in writing the first web browser in 1990, which happened 20 years after the invention of the internet. By that period, millions of computers could be connected via the internet. In 1990, Berners-Lee specified three different fundamental technologies called HTML, URL, and HTTP, in which the web was developed.

The web pages developed at that time were static instead of displaying dynamic contents. They were serverless and they did not have a database. HTML was the main technology for creating web pages. Neither the scripting language JavaScript nor PHP existed yet and all the contents to be displayed in the pages were hard coded. HTML itself was also not very powerful and it was lacking a lot of features in comparison to the present HTML5 technology.

In 1993, when Mosaic browser was released, it became a fundamental source of the websites and started to change the technologies. The concepts of various features, such as bookmarking and images along with text, were introduced with the browser. In 1995, JavaScript was launched and it improved web pages with various interactive elements including vector animation (Oleg Uryutin, 2018). After the launch of the scripting language, the revolution began with the introduction of game changers such as PHP, AJAX, CSS (Cascading Style Sheet), and cookies. In 2005, the concepts of responsive web design, web app development, and asynchronous web apps came into practice. These technologies started to bring vast changes to web pages and contribute a lot of features to make beautiful and more functional websites. In that period, video streaming, flash video, and audio were also able to be maintained as the content of web pages. When a new programming language such as PHP was launched, the pages started to get new functionality and they became more dynamic. The mentioned figure illustrates the beginning year of the evolution in web development. (Alex McPeak, 2018)

3 Modern Web Application Development

Modern web applications are dynamic because the contents of the website change by using the database concepts. It means that developers use the database to store the data and somehow, they fetch the data into the client-side application. Because of this concept, developers do not need to touch the code to be able to display different contents on web pages. For instance, there is a hotel web page and the CEO of the hotel is *Jack John*, his name displayed on the web page. Now the CEO is changed and the new CEO is *John Albert*. To display the name of the new CEO, it is only necessary to update the database, not the code of the web pages. This concept brings huge benefits in a technical sense.

The web application architecture is one of the most important characteristics of today's applications. Although it may take a lot of time to build a sound architecture, it is very essential in building a long lasting application. A careful planning of the application architecture obviously helps in the future in different scenarios, for instance, in cases where the application requires more features, or the use of the application is overloaded and there is a need to switch to another server.

Security is also a significant part of the development process of a modern web application. Nowadays, most of the web applications are tested starting from the first stage, design, and architecture, to include all the security aspects in order to avoid any financial and reputational costs associated with data losses, downtime and vulnerability discoveries. Likewise, it is good practice to keep the changes as constant. It is a never-completable process to build modern web applications which need continual updates in order to adapt to the security trends and consider other changing trends as well. The code structure also needs to be clear and more human readable for the future developer who will maintain the project later. (Joe Stangarone, 2019)

Another important aspect that needs to be considered during the development of a web application is what kind of tools web developers need. Tools are the web developer's best friend. For instance, IDEs like Visual Studio Code speed up the development process and Git, GitHub and GitHub Actions are useful project management tools. Libraries and frameworks are also vital tools for raising the coding speed.

The expectations towards web applications have been growing dramatically because of the revolution in various stacks and technologies. Applications need to be well designed from a user experience point of view. It is a very interesting part to analyse what the user wants from the application and how the application could be made easier to use. Another expectation is the responsive design and the availability of the application at any time from anywhere in the world with high speed in performance.

A quality application is one that is developed considering the present as well as the future. The application is developed to solve the user's long-term needs, taking into account the changing technology, the possibility to scale the application as the business grows, as well as the maintenance in the future.

4 Theoretical Framework and used tools in development process

This chapter will describe the theoretical concepts of modern tools and technologies which are serialized as the top hierarchy in order. The chapter will discuss the environment in which the project is built up and tested and goes through the theoretical knowledge of each technology.

4.1 Node

Node JS was released in 2009 by Ryan Dahl. It is built on top of Google Chrome's V8 engines, which execute the JavaScript code inside *and outside* of the browser. After the launch of Node JS, JavaScript has been updated regularly with modern and powerful syntax. Node JS is one of the major components of modern web technologies. It plays a vital role in making more dynamic and consistent web applications. The main purpose of Node JS is to build scalable network applications.

Node JS is mostly useful for server-side programming, for instance, for creating servers or writing APIs. It runs as an asynchronous operation and is known as an open-source *event-driven* **JavaScript runtime environment.** Event-driven is a programming paradigm which means that the flow of programming is determined by events such as user actions (mouse click, form submit, key press, et cetera.). It builds and runs the application without threads. Asynchronous means non-blocking input output. Because of this feature, multitasking can be performed at once without multithreading.

The core functionality of an event loop is to achieve multitasking. It will have a sequence or a queue of tasks which need to be completed one by one. Node JS performs this event loop with the help of callbacks. A callback is nothing other than a function which is passed as an argument into another function (NodeJS in Action, October 2013).

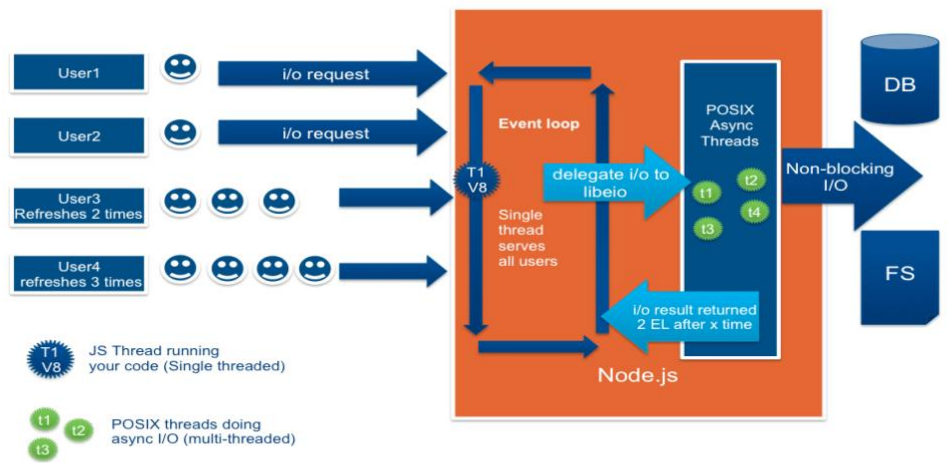
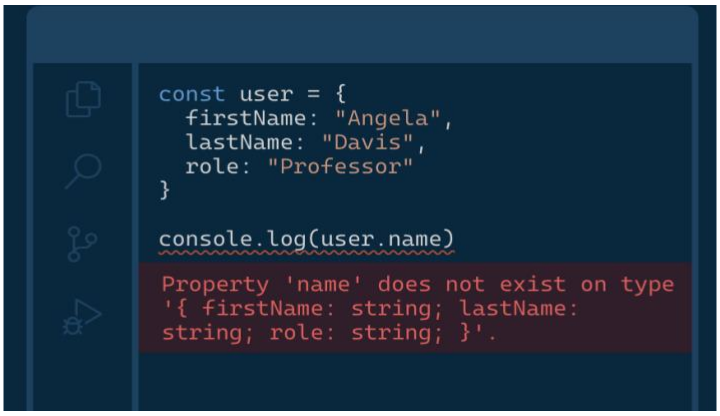


Fig 1: Work-flow of Node JS platform (Node.JS in Action, 2013)

4.2 TypeScript

TypeScript is an open-source language which makes JavaScript advance one level up. JavaScript is a loosely typed language. It means that, for instance, when storing a value into a variable, JavaScript does not look for the type of the variable. Because of this issue, a type coercion may occur. Type coercion is the process of converting value from one type to another type. It may break the application or raise another issue in the future. To prevent this from happening, using TypeScript is the only way for JavaScript to strictly define the type of the data used in a program. A file written in TypeScript will convert to JavaScript during compilation time by JavaScript compiler or Babel. (TypeScript Document)

Defining types provides a better solution for a developer to describe the shape of an object and provide better documentation as well as allow TypeScript to validate that the code is working correctly. Instead of defining types in a JavaScript code, a developer may instead decide to use interfaces which allows to get a lot of power without writing additional code. TypeScript is very useful for developers who have a good understanding in JavaScript because it saves time by catching errors and providing fixes before the code is run. (TypeScript Document)



```
const user = {
  firstName: "Angela",
  lastName: "Davis",
  role: "Professor"
}

console.log(user.name)
Property 'name' does not exist on type
'{ firstName: string; lastName:
string; role: string; }'.
```

Fig 2: Capture of Code in TypeScript (TypeScript Document)

4.3 Husky

Husky is an NPM library which is registered as a Node package. The main implementation of this dependency in the project is to prevent unnecessary commits. If the project is set up with Husky, the developer cannot commit and push code to GitHub if there is a compilation error.
Husky is a dev dependency, meaning that it is applicable during the development of the project. When the project is deployed to the cloud, it is not necessary to run the application. The main features of this library are: Zero dependencies, lightweight, fast usage, support for macOS, Linux and Windows.
To bind the project with Husky, execute the following command in the relevant terminal of the project and set appropriate configuration in the package.json file of the project. (Husky Document)

```
npm install husky@next --save-dev (if we are using NPM)
yarn add husky@next –dev (if we are using yarn)
```

4.4 Backend

Today’s web applications are faster and more maintainable than what they used to be. The credit of this revolution goes to the backend, also known as the server-side application. The core concept of the backend is to store data received from the client and provide it, for instance, in a user profile when the user visits the client-side application. Because of this concept, the application does not need to load all the data at once.

In fact, a server-side application is in itself enough to be considered a web application, if the client does not need to interact with GUI. However, today’s sophisticated web application cannot run without both the front-end and backend. The client-side application is specified to communicate with the server while having a nice looking GUI for the sake of a good user experience.

The backend application serves as a REST API. REST is the acronym of Representational State Transfer and API stands for Application Programming Interface. This application is stateless, meaning that all the data of the application is stored in a database and it serves the data to the client based on HTTP Methods. The server sends the response (data) as the requirement of the user (request).

The client-side application communicates with the server side application using different protocols. Protocols are nothing more than a set of rules for exchanging data in a network. Protocols vary with different tasks and layers. HTTP Protocol, TCP/IP, FTP, SMTP, SOAP, and REST are the most common protocols in use. The general steps of communication between client-side and server-side are: 1) The client-side establishes the connection by the request using HTTP, 2) The server sends back the response using the same protocol. Figure 3 illustrates the working flow in between the client and the server in terms of http protocol. (MDN Documents)

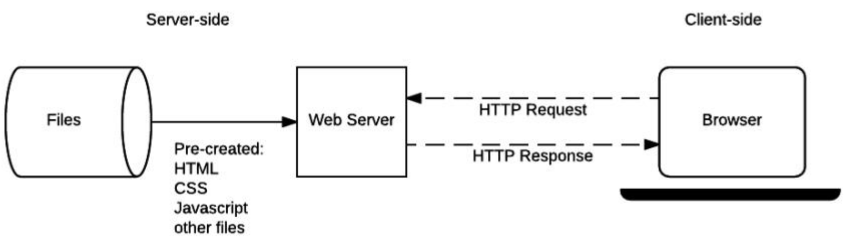


Fig 3: Representation of the flow of communication between client and server

The client side loads and sends the HTTP request to get some data. In the backend, the request is being listened to by the web server and it will verify which pages, file, or data the client is asking for. It searches the data either from the database or from pre-created other pages. When the server succeeds in finding the requested data or pages, it sends the response to the client. Thus, the server is always in the state of listening to the requests of the client and serves the response according to the user's interactivity on the browser.

Node JS is one of the most popular JavaScript based frameworks to be handling client requests on the backend and it is used in the thesis project as well. One significant benefit of using Node JS is the lack of needing to learn a new programming language if the developer is already familiar with JavaScript. Some other popular programming languages for the backend are PHP, Ruby, Python, and Java. (Marcelo Pastorino).

4.4.1 Node Package Manager (NPM)

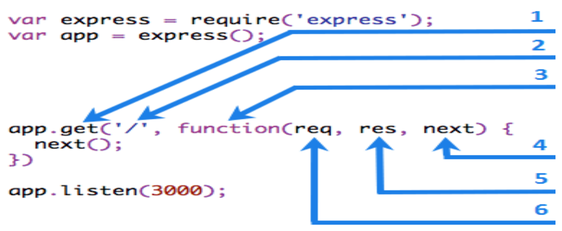
NPM is the acronym of Node Package Manager. It is the main source of finding different packages and dependencies for Node JS and MERN applications. NPM is the world's largest software registry and publicly available at www.npmjs.com. There are thousands of Node.js packages developed by the open-source community and by Node.js team, making it easier to write code. For instance, the Express library, Mongoose to work with MongoDB databases and React packages are very commonly in use. NPM packages make the code clean and more human readable and NPM provides regular updates for the packages automatically. In addition, the packages are optimizable. They can be installed or uninstalled via the NPM command line interface. (Node Documentation, 2019)

4.4.2 Express

According to the official documentation site of Express JS, *"Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications".* The definition is elaborating that Express is a framework for Node.js, which works on the top of the core modules without hiding any of the features of Node.js and provides a very clean and readable function and syntax to create the server. The Express.js module is much easier to understand and more maintainable in the application than the native Node modules. Because of these features, the largest companies are also adopting Express. Express makes the module integration easy to handle with clear syntax and it maintains the sound structure of the application. Express can be installed by using the NPM command: **npm install express**.

The Express server is made up of three main building blocks known as **router**, **routes** and **middleware**. The core functionality of the server depends on its well-designed routing methods. They are normally known as controller functions. The controller decides what the server needs to do in the specific route that it is matched with. As a workflow, the client sends the request to the server to get some resources and the server serves the resources as a response based on the defined controller or routing method.

Express makes this tedious job easier by allowing developers to create the routes as an easy structure. The figure 4 illustrates the syntax for creating an Express server. (Express Documentation, 2020)



```
var express = require('express');
var app = express();

app.get('/', function(req, res, next) {
  next();
})

app.listen(3000);
```

Fig 4: Structure of an HTTP server in Express (Code World, 2020)

The first arrow represents the HTTP method for which the middleware function applies. The most used HTTP methods are GET, POST, PUT, DELETE and PATCH. The GET method is for pulling the resources from the server. The POST method is for creating new data in the server. For example, when the user signs up, the post method is invoked. The PUT method is for updating existing resources in the server, and the DELETE method is to remove data from the server. The PATCH method is for making partial changes to an existing resource.

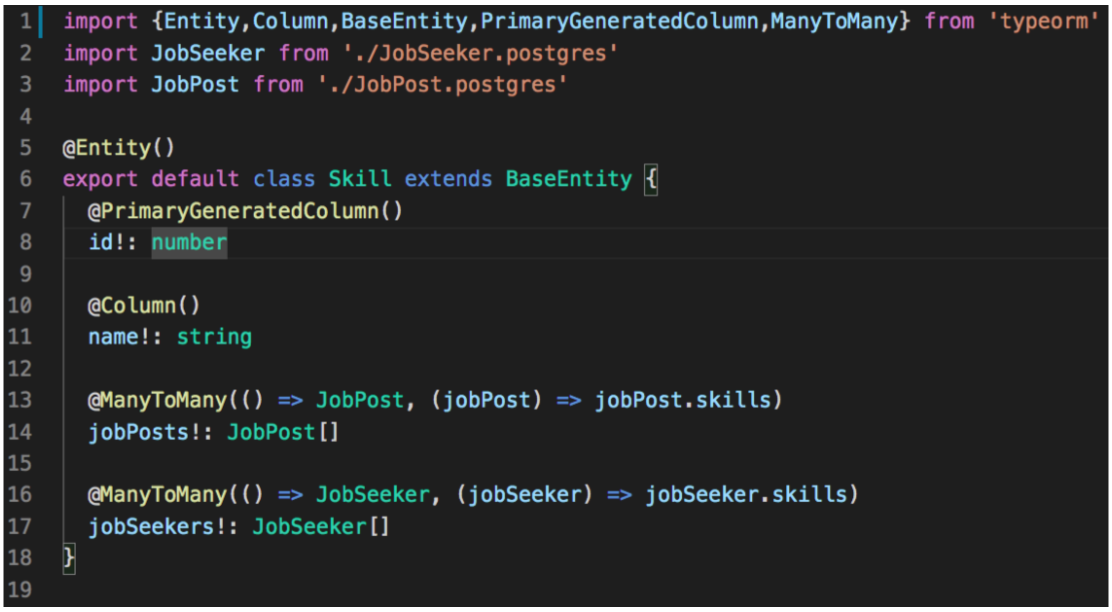
The second arrow in figure 4 represents the path (route) for which the middleware function applies. The third arrow is the middleware function. The fourth one indicates the callback argument to the middleware function, which is called “next” by convention. On line 5 is the HTTP response argument to the middleware function and on line 6 the HTTP request object. (Code World, 2020)

4.4.3 Postgres

Postgres is a powerful, open-source object-relational database system which extends SQL (Sequential Query Language System) with many features that help storing and scaling the most complicated data workloads in the form of a Relational Database Management System. Nowadays, some companies want to use MongoDB as an NOSQL (Not Only Structure Query Language) database instead of using a relational database. Some other companies which are using SQL, are switching to Postgres. Another latest good news for a Postgres developer is that when the version 13 was released in Sept 2020, Postgres conforms to at least 170 of the 179 mandatory features for SQL:2016 core conformance. The main features of Postgres are Data Integrity, Concurrency Performance, Reliability, Disaster Recovery, Security (Authentication: GSSAPI, SSPI, LDAP, SCRAM-SHA-256, Certificate, and more), Extensibility, Internationalisation, and Text Search. (Postgresql Documentation, 2020)

4.4.4 TypeORM

In the field of Computer Science, ORM is the acronym for Object-relational mapping which represents a programming technique for converting data between incompatible type systems using object-oriented programming languages. The term *TypeORM* is an ORM that is able to run an internet-based application written in either JavaScript or TypeScript in Node JS and Electron platforms. The main purpose of TypeORM is to support the latest JavaScript features and provide additional features which will help to develop any kind of application that uses databases. It works based on entities and columns, entity managers, and clean object relational models. Some of the other features of TypeORM are cascades, indices, migration and automatic migration generations, working with multiple database types, schema declaration in models, connection configuration in JSON/xml/yaml/env formats, comfort in working in NodeJS/Browser/ionic and CLI (Command Line Interface), et cetera. (TypeORM Documentation, 2020)



```
1  import {Entity,Column,BaseEntity,PrimaryGeneratedColumn,ManyToMany} from 'typeorm'
2  import JobSeeker from './JobSeeker.postgres'
3  import JobPost from './JobPost.postgres'
4
5  @Entity()
6  export default class Skill extends BaseEntity {
7    @PrimaryGeneratedColumn()
8    id!: number
9
10   @Column()
11   name!: string
12
13   @ManyToMany(() => JobPost, (jobPost) => jobPost.skills)
14   jobPosts!: JobPost[]
15
16   @ManyToMany(() => JobSeeker, (jobSeeker) => jobSeeker.skills)
17   jobSeekers!: JobSeeker[]
18 }
19
```

Fig: 5 Structure of Entity with TypeORM (Sources: Thesis Project)

In figure 5, the first line imports the required modules from TypeORM. Lines 2 and 3 are for importing the other entities as a module base. Lines from 5 to 18 represent an entity called **Skill** which has the relation with the imported entities (line 2 & line 3) **JobSeeker** and **JobPost**. The Skill entity has a many-to-many relation with both of the imported entities and on lines 13-14 and 16-17 are the syntaxes for binding the relations. Line 7 is for the id, generated by the database automatically for each row, and the type of the id is represented on line 8. The column called “name” has the type of string which is mentioned on lines 10-11. The many-to-many relations in between skill <=> job post and skill <=> job seeker have the type of an array which is denoted on lines 14 and 17.

4.4.5 PassportJS

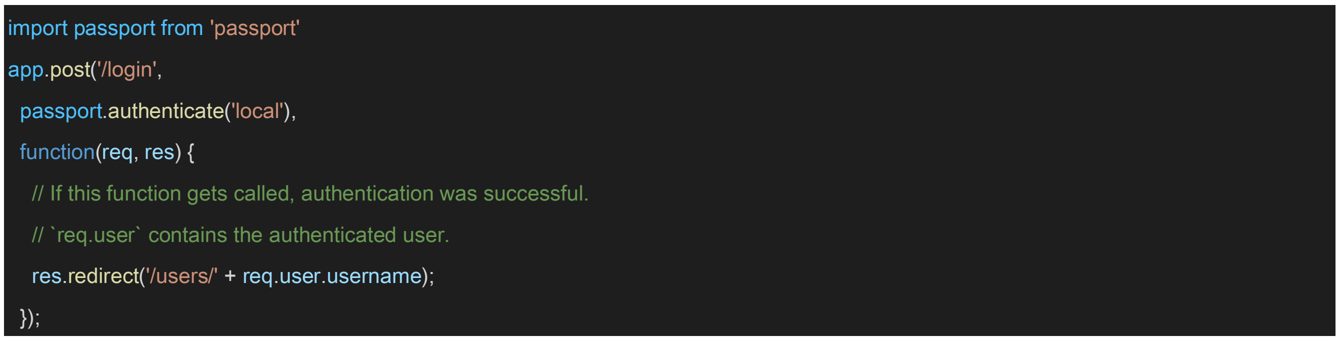
In the modern trend of web application, the easiest way for accomplishing user authentication when building Node JS applications is to use an open-source NPM library known as PassportJS. It is a very flexible library, easy to apply, and it keeps the code clean and maintainable. This technology has also been implemented in the thesis application in the scope of login and resources usage.

Traditionally, web applications were limited to using the username and password system to authenticate the user in the application, but modern applications are switching to third party services like the OAuth providers such as Facebook, Twitter, Google, and GitHub.

Nowadays, the applications are dynamic and often role based. For instance, different users may have different content on the pages based on their profile. As a result, security and privacy are the main things that need to be taken into consideration when building a web application.

There are various strategies for user authentication. For the purposes of privacy and security, PassportJS is one of the best approaches for protecting the web application. It is trusted by developers and users around the globe. PassportJS is customizable and secure, the code is lightweight, and there are 300+ authentication strategies to choose from (PassportJS Library, 2017). These features make it reliable and secure for user authentication. (Passport NPM Document, 2020).

It depends on different scenarios what kind of authentication is needed for the application. PassportJS provides token based, OAuth based, and OpenID based options for authentication. In the thesis project, Passport Local Strategy and JWT (JSON Web Token) strategy are implemented. In Node JS applications, PassportJS is used as a middleware function. In the JWT strategy, when the user's email and password are matched, the user will get a token and based on that token, they can use the resource further.



```
import passport from 'passport'
app.post('/login',
  passport.authenticate('local'),
  function(req, res) {
    // If this function gets called, authentication was successful.
    // `req.user` contains the authenticated user.
    res.redirect('/users/' + req.user.username);
  });
```

Fig 6: Syntax for PassportJS (Source: PassportJS Document 2020)

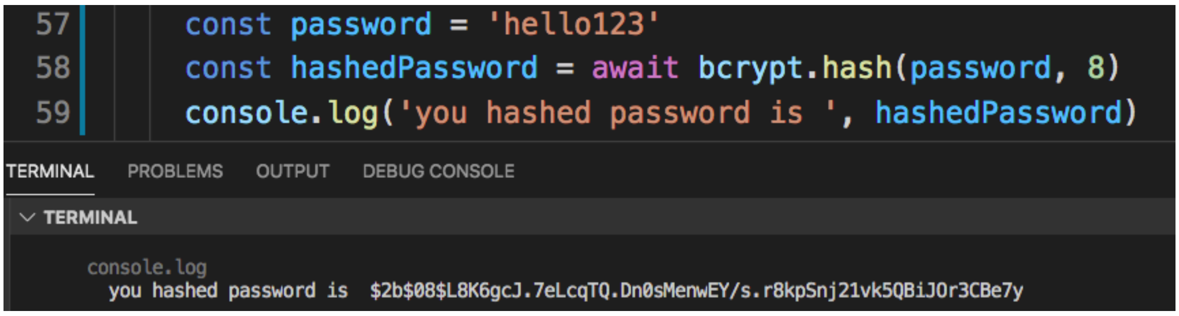
4.4.6 BcryptJS

Saving the plain password into the database is not a good idea in building the web application or any software. So, the hash concept comes into account in this scenario. Hash is only the solution even

though the database administrator also cannot see the users' password. From the security point of view, it is crucial to implement this concept. In modern web design, BcryptJS is the solution to hash and make it secure.
BcryptJS is an important and very popular NPM library which is mostly applied in Node JS applications. It is used for protecting the user password by hashing it before saving the password to the database. The main workflow is:

step 1: User signs up with email and password in the client side
step 2: Server-side application hashes the password
step 3: Saving the hashed password into database

Plain passwords are easy to hack and crack. They are vulnerable against brute force attacks and rainbow attacks. BcryptJS is one of the solutions to protect against attacks by using the OpenBSD algorithm. OpenBSD algorithm is an open source, security-oriented, UNIX-like operating system-based algorithm. The hashing algorithm when using BcryptJS is a one-way algorithm. It is not possible to get back the plain password. (BcryptJS 2017).



```
const password = 'hello123'
const hashedPassword = await bcrypt.hash(password, 8)
console.log('you hashed password is ', hashedPassword)
```

```
console.log
  you hashed password is  $2b$08$L8K6gcJ.7eLcqTQ.Dn0sMenwEY/s.r8kpSnj21vk5QBiJOr3CBe7y
```

Fig: 7 Syntax for hashing password. (Source: Project Work)

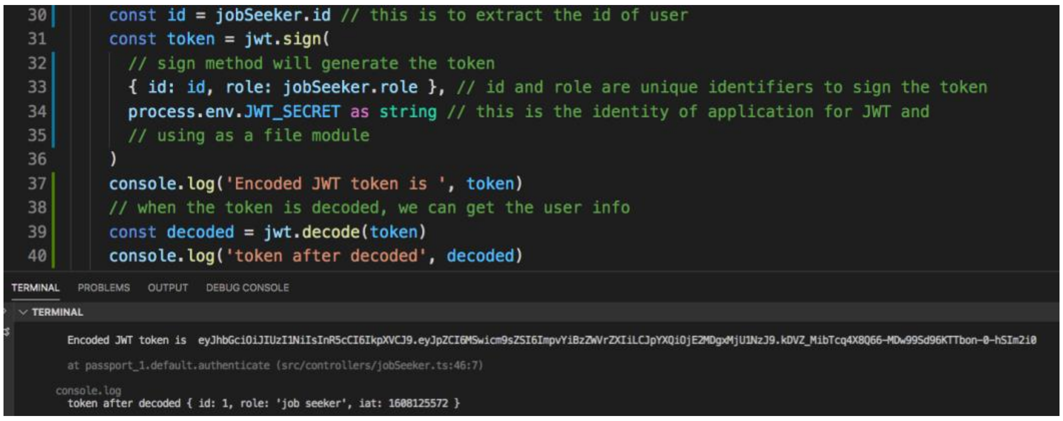
Bcrypt has two common methods called **hash** and **compare**. The hash method takes two arguments as shown above in figure 7 and the second argument is the number of rounds. Higher rounding values hash the password strongly. As a result of hashing a password, the plain password is converted to a long string. The compare method also takes two arguments: the given password and the hashed password.

4.4.7 JWT

JSON Web Token is a digitally signed and open industry standard (RFC 7519) method for representing claims securely between two parties. It allows developers to decode, verify and generate the signed token.

The main implementation of the JWT token is used in authorization because it is secure and simple to use. When a user requests to login with their credential and the credential is matched programmatically (for example, by using PassportJS), JWT generates a token and signs it with the user id (or some other unique identification of the user) as well as with a JWT secret. The JWT secret is a set of strings/characters made for the identification of the JWS. Once the token is matched, it sends a BaseURL64 encoded payload, which contains the user information as requested (JWT Documentation 2020).

When the request is successful, the user gets the token and sends it to the server frequently to further use the resources until they logout of the system. The logout functionality is simply a mechanism of deleting a token. Next time, when the user tries to login and wants to use the resource, the same workflow will be repeated.



```
    const id = jobSeeker.id // this is to extract the id of user
    const token = jwt.sign(
      // sign method will generate the token
      { id: id, role: jobSeeker.role }, // id and role are unique identifiers to sign the token
      process.env.JWT_SECRET as string // this is the identity of application for JWT and
      // using as a file module
    )
    console.log('Encoded JWT token is ', token)
    // when the token is decoded, we can get the user info
    const decoded = jwt.decode(token)
    console.log('token after decoded', decoded)
```

```
Encoded JWT token is  eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwicm9sZSI6ImpvYiBzZWVrZXIiLCJpYXQiOjE2MDgxMjU1NzJ9.kDVZ_MibTcq4X8Q66-MDw99Sd96KTTbon-0-hSIm2i0

at passport_1.default.authenticate (src/controllers/jobSeeker.ts:46:7)

console.log
 token after decoded { id: 1, role: 'job seeker', iat: 1608125572 }
```

Fig: 8 Code illustration of JWT token. (Source: Project work)

4.4.8 Files Uploading in AWS S3

The project is integrated with AWS (Amazon Web Services). A service called S3 (Simple Storage Service) allows uploading files into buckets in AWS. A bucket is a container where data can be saved as objects. It allows storing and retrieving any amount of data at any time from anywhere. AWS provides services for creating, deleting, and updating buckets, making a bucket private or public, moving one bucket into another, et cetera. (AWS Documentation 2020)

The S3 technology has been used in the project to store user profile pictures into the cloud. Before connecting the app with AWS cloud-buckets, it is necessary to set up AWS accounts, bucket names, policies, and credentials at the AWS. The diagram below illustrates the workflow in between the full stack application and AWS S3. (AWS Documentation 2020)

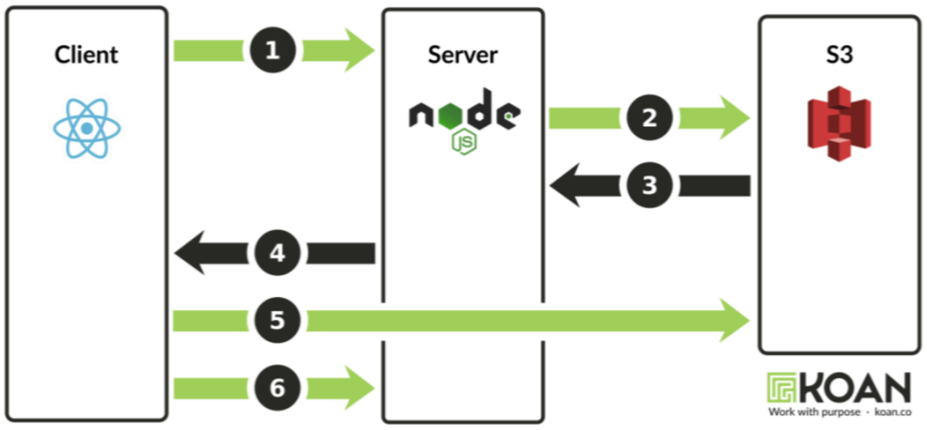


Fig: 9 High Level workflow for uploading file into AWS S3 (Ashwin Bat, Jun 20, 2020)

1. The client navigates to the user profile page which includes a component for uploading images. At this point, the client requests a pre-signed upload URL from the server.
2. The server relays this request to S3.
3. S3 returns a new pre-signed URL to the server.
4. The backend/server returns the pre-signed URL to the client.
5. The frontend displays an interface for the user allowing them to select a file to upload. The user selects a file from their computer and the client uploads the selected file to the pre-signed URL. This time, the request goes directly to S3 where the server has reserved the URL for upcoming images to upload.
6. The client sends a request to the server again, letting it know that the upload has completed. To keep persisted state down to minimum, the server do not store the URL it generates from step 1 in this request. The server writes the pre-signed URL to **imageURL** field in the User model, persisting it to the database (Ashwin Bhat, 2020).

To run all the steps up above, there are some NPM package libraries available which help to write clean and maintainable code. Among them, **aws-sdk** is one of the most important NPM packages that comes into consideration in the server/backend. Similarly, on the client side, the NPM package used in the project is called **react-images-upload** which has a nice built-in UI (user interface) for uploading images.

4.5 Frontend

In the field of software engineering, frontend is known as the presentational layer of a full stack application, compliant with the separation of concerns principle. It is also called a client-side application. The main purpose of the frontend is to display the functionality of an application in a nice presentational way using Graphical User Interface (GUI), which helps the user interact with the application easily.

Frontend Developers need to be experts in many technologies in the constantly changing field of web development. The design of a frontend application has switched to a component-based approach design, which is a very cost effective and maintainable way of creating applications. (Max Pekarsky, Feb 3, 2020). Some of the competitive technologies which popularized this revolution in web technology are described below.

4.5.1 React JS

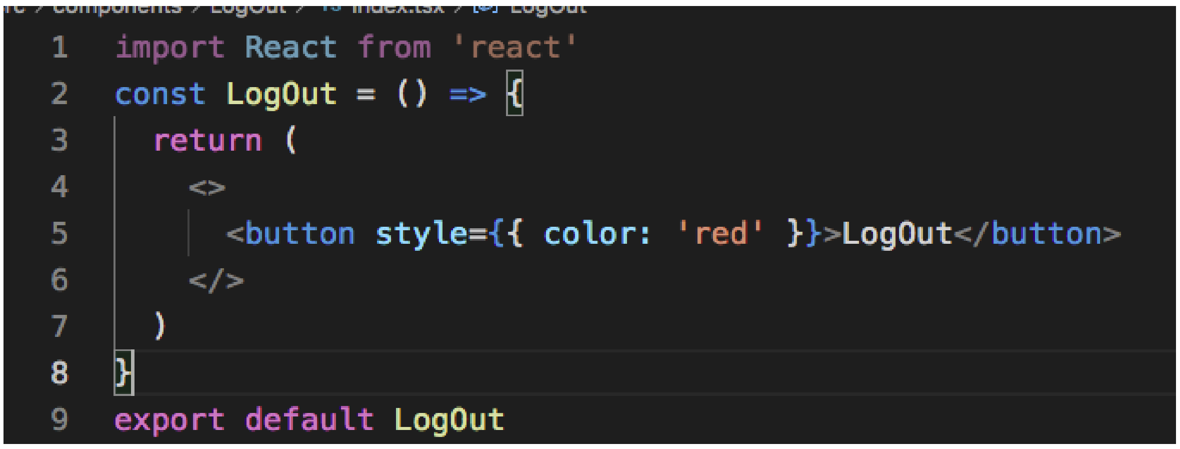
React JS is a popular, open-source front-end JavaScript library used in modern web applications, which makes creating interactive UIs easier. The ReactJS works as a declarative view which makes the code more predictable and easier to debug for the developer. It is maintained by Facebook and a community of individual developers and companies.

The main purpose of React JS is to implement the code logic inside the view that the user interacts with. With React JS, developers can design simple components - for instance, a search bar or a button - and render a new view if the state of that component has changed when the user interacts with the app. The working mechanism or the loading process of the web pages is completely different compared to how applications worked previously. It renders only the necessary components of the application and this is why React applications are called component-based web applications. As a result, the product reacts faster to user interaction, it is easier and faster to design, the components are reusable, and the app is high performance. Using React JS, building small components and encapsulated components which manage their own state, a developer can build complex UIs.

As a real life example of the benefits that React JS provides is how the “Like” button works in Facebook. In the past, when the user clicked the button, the entire page reloaded and then showed the new amount of likes, but nowadays the reload is no longer necessary.

React JS is regularly updated and integrated with modern features and syntax of JavaScript. It supports all the latest syntax of JavaScript but in case the browser does not support the features, it may fail to run the application. Some browsers do not support features introduced in ES5 or newer ECMAScript versions. To avoid the issue and to be able to run the program in every browser, React JS has been integrated with Babel.js and Webpack when setting up the project by using a tool called *create-react-app*. The application can be initialized with the command **npx create-react-app projectName**.

Babel is a transpiler which converts the modern JavaScript syntax to a browser-compatible JavaScript. Webpack allows developers to write modular code and it bundles it together into small packages to optimize the load time of the pages. (React Document 2020)



```
rc > components > LogOut > index.tsx > LogOut
1  import React from 'react'
2  const LogOut = () => {
3    return (
4      <>
5        <button style={{ color: 'red' }}>LogOut</button>
6      </>
7    )
8  }
9  export default LogOut
```

Fig 10: Demo of ReactJS Component (Source: Project Work)

In the figure above, line 1 allows the developer to write the component in React JS. Without importing the React library, the compiler will not understand JavaScript XML (JSX) and will complain about an error. From line 3 to 9 is the definition of the functional component which renders the logout button. The code from line 5 to 7 is written in JSX which is a syntax extension to JavaScript. The name of the rendered element is ‘button’ with an attribute called ‘style’ to which the property colour is passed as an object (key-value pair). The style attribute changes the color of the text on the button to red. The text **LogOut** between the ankle brackets is rendered to the screen**.** On line 9, the
module is exported, ready to be used in other components in the application. This is the basic syntax and concept of React JS.

4.5.2 Redux

Redux is a state management library, which can be used in JavaScript applications for managing and updating the application's state using events called actions. When the application grows, managing the state of each component becomes difficult. Redux provides a centralized architecture, meaning that the state of the application is kept inside one store (an object), which can be used across the entire application. This will ensure that the state can be updated in a predictable fashion. In other words, Redux works as a pipeline to provide the necessary state to a particular part of the application and listens to events when the state needs to change in the component according to the triggered event. (Redux Document 2020)

React-Redux is a small standalone JS library and commonly used with several packages. It can be installed with the command: **npm install react-redux**.

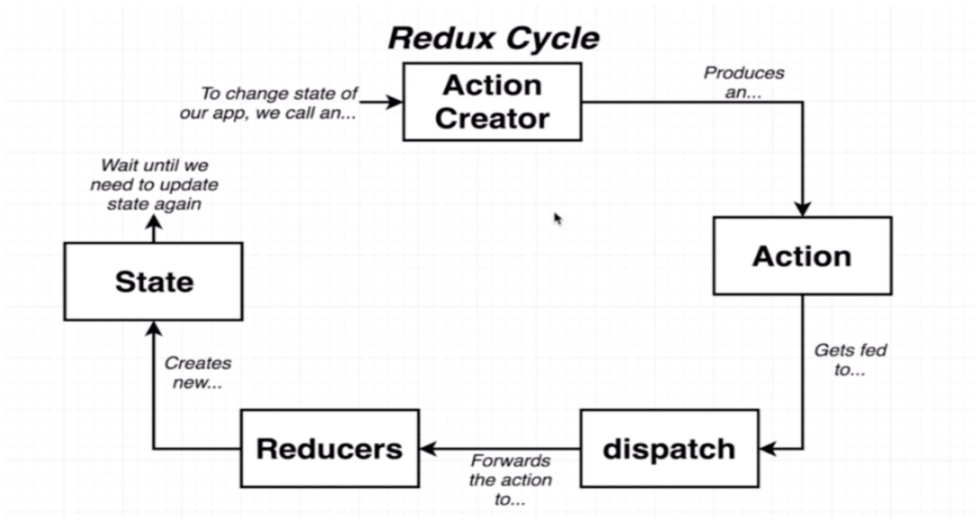

Redux Cycle
To change state of our app, we call an...
Action Creator
Produces an...
Action
Gets fed to...
dispatch
Forwards the action to...
Reducers
Creates new...
State
Wait until we need to update state again

Fig: 11 High Level Workflow of Redux (React-Redux flow, 2019)

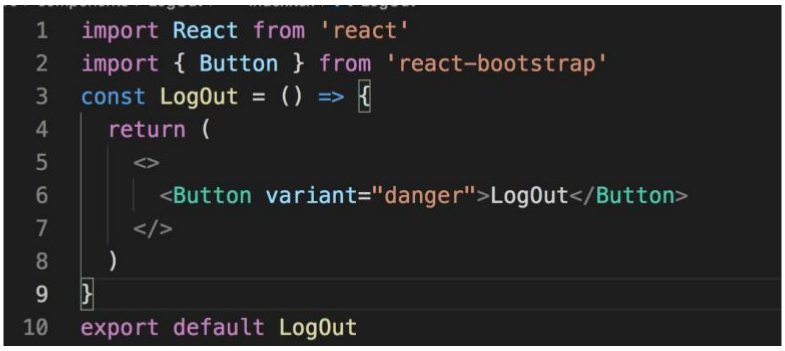
In figure 19, **action creator** is a function which creates an **action**. The action is invoked when the state needs to change in the application. The action gets fed to **dispatch**, which is another function that makes copies of the action and sends them out to the **reducer**. The reducers are functions as well, and they do all the work, taking the action and some existing data (when provided) and changing the data according to the type and payload of the action, and finally updating the state of the data in the **store**. The store includes all the reducers in the application and serves as a central repository of the data that they have.

4.5.3 Bootstrap

UX-design and the style of web pages are eventually the elements that either attract people to use the application or repel them from using it. For instance, colors, graphics, animations, and responsive design are some of the key factors in making the website more attractive to the user. In this project, the Bootstrap library has been chosen for styling purposes, not only because it makes the development process faster and easier, but also because it provides a nice, clean and professional look and responsive features. As such, the application is suitable to visit on every device from mobile phone to computer.
According to **Jef Raskin**, “The way that you accomplish tasks with a product - what you do and how it responds - that’s the interface”(The Human Interface, 2000. page 2). Interface is not limited only in buttons and menus. It covers the main interactions between the user and the browser on every device. User interface is not only how it looks, it also describes how it works. So, to resolve the issue of this chapter, Bootstrap is becoming a fast and modern technology that fulfills the users’ expectation in design by providing a nice interaction. It is built in CSS technology available as component based. So, BootStrap helps to make a nice design and view of the web in a small time. The color, texture, contrast,

layout, shape and size, graphics, animation, responsive design are the solutions that can be resolved from Bootstrap.

Bootstrap is one of the world's most popular frameworks for building responsive, mobile-first sites with JS delivery and ready-made templates. It is easy to use, saves time and the amount of code that needs to be written. It is cost effective and makes websites look good. Bootstrap provides features such as Button, Navbar, Grid Layout, and styled page templates. Bootstrap can be used either as a vanilla JavaScript Bootstrap or React Bootstrap in React apps. The command **npm install react-bootstrap** will install the React Bootstrap library and all the components can be used as module base components by importing them from the library. (Bootstrap Document, 2020).



```
 1  import React from 'react'
 2  import { Button } from 'react-bootstrap'
 3  const LogOut = () => {
 4    return (
 5      <>
 6        <Button variant="danger">LogOut</Button>
 7      </>
 8    )
 9  }
10  export default LogOut
```

Fig: 12 Using Bootstrap with React (Source: Project Work)

In figure 12, line 2 imports the React Bootstrap and the React component is returning the 'Button' element in JSX on line 6. The Bootstrap component can also be customized by passing props in it. For instance, the rendered Button has the property variant with the value *danger* and as a result the button's background color is red.

4.6 Project Management

Leading a project successfully can be helped with proper project management tools. Nothing can replace a good project leader, but in the context of this report the emphasis is placed on the technology developed for managing web application projects. Project management binds the work with costs, time, and manpower effectively so that there is a high chance to finish the project in time within the mentioned budget and resources. Basically, project management is about recording and distributing who will do what and when.

The thesis project has been managed with tools and technologies that are freely available for everyone. To track the work and contribution of the team, we have been using Git, GitHub, GitHub Project, and GitHub Action.

4.6.1 Git

Git is one of the most essential technologies for a full stack developer in the present market. According to the latest Stack Overflow Developer Survey, more than 70% of the developers use Git. It is a free and open source version control tool, designed to handle everything from small to very large scale projects with high speed and efficiency. Git allows and supports developers to create multiple local branches which makes it easier to work on different features in the application. It allows a frictionless switch between different contexts. It means that one can create a branch, experiment with a new feature in the new branch and after commiting changes, switch back to the original branch to continue working on it or merge the changes from the other branch to it.

Git is usable for Role-Based Codelines. For instance, the master branch is usually used for committing code for production, whereas the develop branch is meant for day-to-day work or testing. Another practical feature of Git is its Feature Based Workflow. In a project, it is good practice to create a new branch for each new feature. (Git Document, 2020).

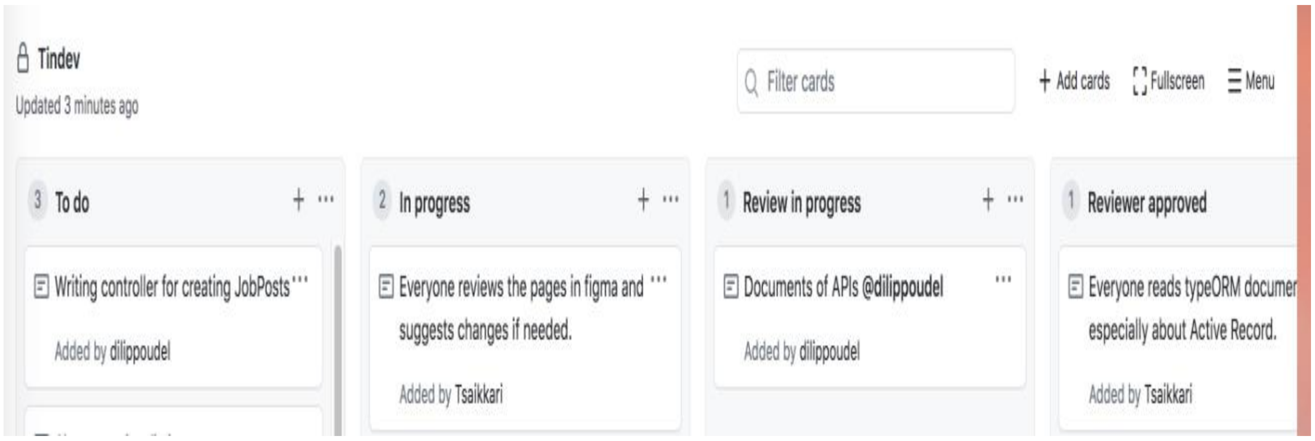
4.6.2 GitHub

GitHub is a provider for internet hosting and it is commonly used in software development as a version control and a collaboration tool. It manages the distributed source code using Git and offers multiple version control features. The difference between the Git and GitHub is that Git is related only to the local machine, whereas GitHub is accessible from everywhere. GitHub creates a backup file with all the committed stages of the project and becomes a distributor of the source code. In addition, it provides access control and several collaboration features such as bug tracking, feature requests, continuous integration and wikis. It also provides an easy tool for project management in software development. (GitHub Document 2020)

The amount of people using GitHub is increasing dramatically. As of January 2020, GitHub has over 40 million users and more than 190 million repositories are available in GitHub. Most of the services that are provided in GitHub are free of charge while more advanced professional and enterprise services are commercial. (Stack Overflow Developer Survey 2020).

GitHub is the main tool that helps to work and contribute in a team. The project boards on GitHub are one of the most useful features to help developers and teams organize and prioritize their work. It provides project boards for specific feature work, comprehensive roadmaps and even release checklists. It is a flexible feature for creating customized workflows and it is made up of issues, pull requests and notes that are categorized in columns. One can drag and drop a note from one column to another, depending on the progress in that particular task. Tasks can be assigned to different team members which helps in keeping track of the progress. (GitHub Document 2020)

GitHub provides templates for a quick set-up of a new project board including instructions on how to use the board. (Project management, GitHub 2020)
The figure below shows our team project board in the beginning of the backend project.


Tindev
Updated 3 minutes ago

Filter cards
+ Add cards
Fullscreen
Menu

3 To do
Writing controller for creating JobPosts
Added by dilippoudel

2 In progress
Everyone reviews the pages in figma and suggests changes if needed.
Added by Tsaikkari

1 Review in progress
Documents of APIs @dilippoudel
Added by dilippoudel

1 Reviewer approved
Everyone reads typeORM documen
especially about Active Record.
Added by Tsaikkari

Fig :13 The project boards of team collaboration ()

4.6.3 GitHub Action

GitHub Actions is a free service provided for public repositories. It is an automation process for software development which assists developers by debugging the code on behalf of them. CI/CD (Continuous Integration/Development) offered by GitHub allows developers to build, test and deploy code from GitHub repositories. It runs the CI/CD pipeline directly on GitHub along with the code repository. The built-in CI/CD service is provided by the biggest code hosting provider. For the security point of view, it is highly secure because the code stays in the repository. GitHub actions are integrated with GitHub events like PR (pair review), push, and pull request.

In GitHub Actions, some of the main concepts are Workflow, Job, Step, Action, Runner and Event. **Workflow** is an automated process that can be set-up to build, test, release, and configure the code, among other things. Workflow needs to be defined in a YAML file and pushed to .github/workflows folder in the repository. **Step** represents a specific task that runs in a **job**. All steps of the same job run sequentially in the same virtual machine but in different processes. A step can run a command (such as bash) or an action. **Action** is a combination of individual tasks that run as a single unit of work. An action must be used as a step in a job. **Runner** is a virtual environment (such as virtual machine or Docker container) that runs the job. Each runner runs one job at a time, and it can be GitHub-hosted or self-hosted. **Event** is a specific activity that triggers the workflow run. Some of the instances of events are: Creating PR (Pair Review), commit, pull request, and pushing code to the branch. (GitHub Action Documentation, 2020)

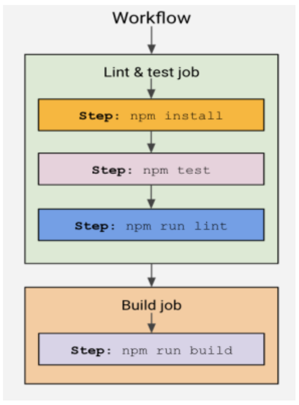


Fig: 14 Basic workflows of GitHub Actions

4.7 Apps on Cloud / Deployment

An essential part of the development process of a web application is to deploy the application to the cloud where it can be viewed by people. During the development, the product can be run, tested and manipulated several times. When everything is ready, or the app is proved as bugless, then it goes to the production. It means that the app is ready to be used not only from the local machine. It should be accessible from everywhere from the web. For this purpose, the backend part of the thesis project has been deployed to the Heroku platform.

Heroku is a very popular container-based cloud platform as a service (PaaS) which allows users to deploy, manage and scale web applications. The platform is elegant, flexible and easy to use, and it provides an URL to the deployed application which one can showcase in the market. Heroku is fully managed, meaning that it takes care of the hardware infrastructure for the server on behalf of the developer. (Deploying NodeJs to Heroku, 28 Jan, 2021)

5 Development Process

This chapter describes the general steps of the development process in the thesis project. It reports the major flow of tasks and the steps to be taken that are highly important for an entire product. This chapter and subchapter will discuss how the defined technologies are bound together with the architecture. When the project idea is born, the first step is to discuss and elaborate it with the team members. Understanding and analysing the idea is a very crucial and initial part of the development process of the application. Rather than starting to code directly without carefully thinking about the idea and analysing it from different angles, it is of great importance to study the subject identifying the problems and finding solutions to them. The team members must understand the workflow of the application. In the implementation of stacks in the project, the project has been initiated with the backend/Server side set up. First, the backend is completed including an automation testing process and then the project jumps to the client-side architecture, design and implementation. Project has been set up in both frontend and backend with the technologies mentioned above.

5.1 Database Design and Services

Designing the database is the first step to start up the project in the development mode. The backend of the application needs to know what kind of resources should be available for the client-side user. The thesis project's database has been designed according to the necessary entities/tables for the application, which have been visualized in a diagram showing all the relations between the entities (Figure 17). For example, *jobseeker*, *employer*, and *skill* are the main entities in the database and the relations between them are defined by following the BCNF rules to avoid data redundancy. The design of the database followed the strict rules of the Relational Database Management System to make the database reliable for the application.

A jobseeker can sign up and login as a user to the application and add their profile information. Those are the main services designed for the jobseeker. The jobseeker may add one or more skills to their skill list on their profile page and they may also match to one or more job posts. The match is made according to the required skills in the job post and the skills of the job seeker. If the jobseeker has at least 3 of the required skills in a job post, there is a match. The relation in between a job post and a jobseeker is a many-to-many relation and the relation in between the jobseeker and skill is also a many-to-many relation.

Services for the employer are similar to the jobseeker's ones with the addition of the opportunity to create, edit and delete job posts.

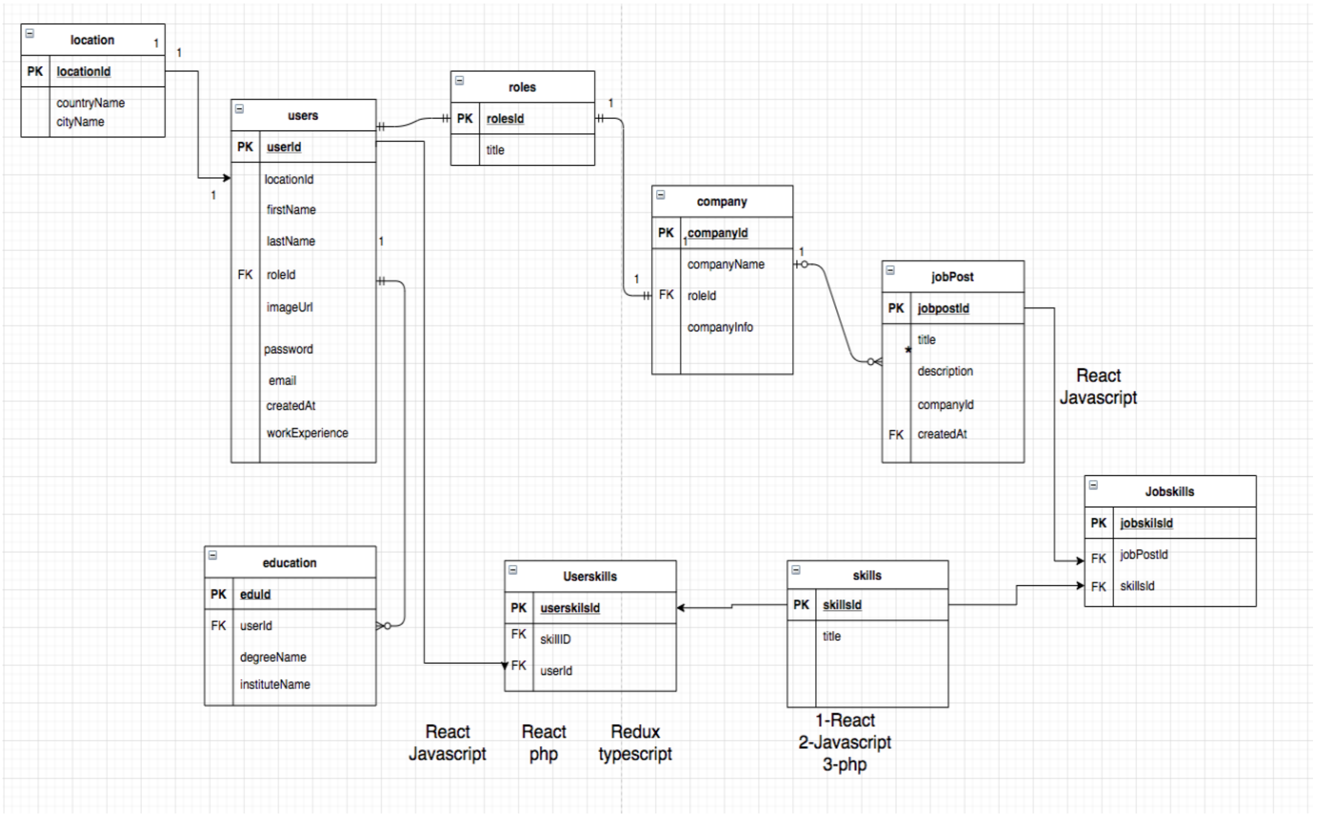
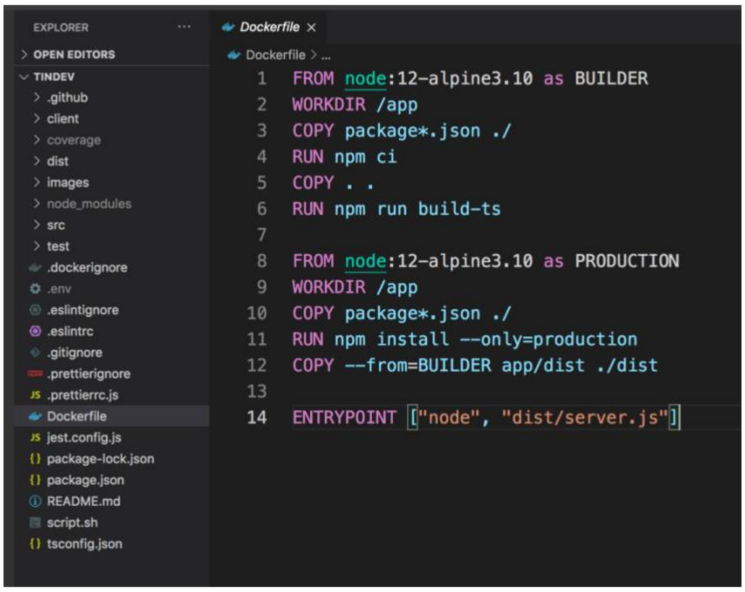


Fig: 15: ER (Entity Relationship) diagram of database (Source: Project work)

5.2 Project Setup for Backend

The backend section of the project is integrated with many technologies such as Node JS, Express, Postgres, and TypeORM, which are discussed in the previous chapter. The project repository has been initiated by using the command **git init** in a directory named Tindev. The command to initialize the project, **npm init**, is provided by Node to create a *package.json* file which includes the dependencies the project needs. After that, the project set up has been integrated with the typescript configuration. In the directory, typescript has initialized and set up. The rest of the files and folders have been created to manage the code and create the structure. Because of TypeScript, all the files have an extension of dot ts (.ts). The *Dockerfile* is created to run and manipulate the established project in the local machines of the team members. After writing the Docker file, a minor configuration has been set up in the package.json file which keeps the record of all installed dependency. A *gitignore* file is created to avoid pushing sensitive information (for example, environment variables such as database secret and JWT secret key) and unnecessary files (for example node modules) to GitHub. The Dockerfile has its own *dockerignore* file for the same purposes. The file *script.sh* is for running the docker container when the project is started up with command "**npm run db**". When we hit this command in the terminal of the relevant directory, the script.sh file will run to initialize the image of Postgres database and Node into your local machine so that the developers don't need to download separately the Postgres database. Here is the image of the docker container file.



```
FROM node:12-alpine3.10 as BUILDER
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build-ts

FROM node:12-alpine3.10 as PRODUCTION
WORKDIR /app
COPY package*.json ./
RUN npm install --only=production
COPY --from=BUILDER app/dist ./dist

ENTRYPOINT ["node", "dist/server.js"]
```

Fig: 16 Screen shot of Docker file

The project has been bound with the NPM package of Husky to prevent unnecessary commits with error. Before pushing to GitHub, the Husky package will run and try to detect errors before accepting the commit. If there is any error, the commit will fail. The code for this functionality is set up in the package.json file which records all the installed NPM packages. The *README.md* file contains the instructions and guidelines on how to clone the project from GitHub and set it up on the local machine, and how to contribute to the project if one should choose to do so.

5.4 Files and Folder Structure in server side and client side

The project includes multiple folders, such as **src**, **test**, **dist** and **client**. Each folder stands for a different purpose. The folders are structured based on the files that they consist of. The src folder contains the main code spread in different subfolders and files according to which module they belong to. It contains the subfolders for controllers, entities, helpers, middlewares, routers, typings, and utils, as well as the *app.ts* and *server.ts* files. The app.ts and server.ts are in the top hierarchy of the file structure and they are meant for running the project.

The test folder of the project contains the testing files to test the services and APIs. The dist folder contains all the related files that execute when the project runs. The files which are produced inside the dist have a dot js extension and they contain vanilla JavaScript code.

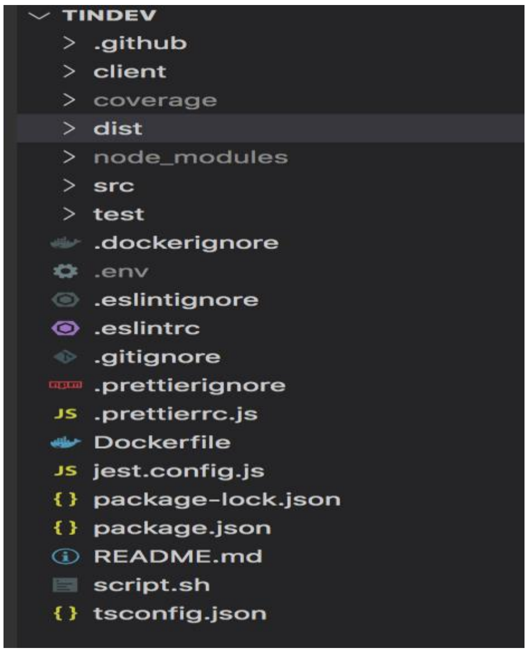

```
⌄ TINDEV
  > .github
  > client
  > coverage
  > dist
  > node_modules
  > src
  > test
    .dockerignore
    .env
    .eslintignore
    .eslintrc
    .gitignore
    .prettierignore
 JS .prettierrc.js
    Dockerfile
 JS jest.config.js
 {} package-lock.json
 {} package.json
    README.md
    script.sh
 {} tsconfig.json
```

Fig: 16 File and folder structure of the server side.

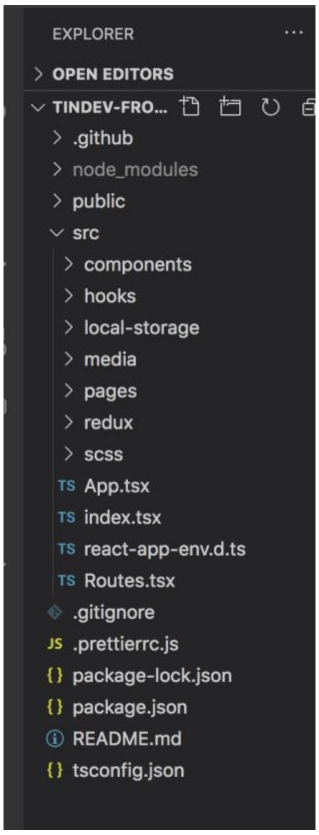
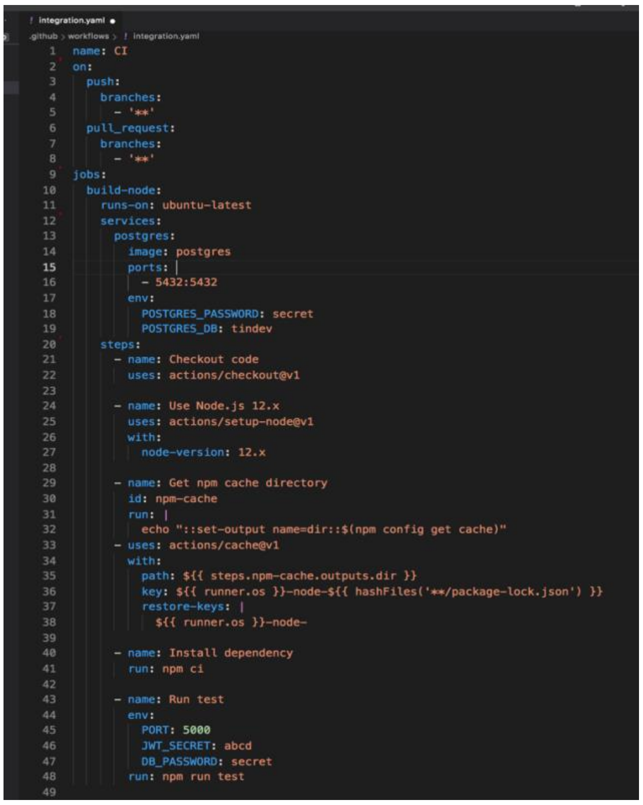

```
EXPLORER                ...
> OPEN EDITORS
⌄ TINDEV-FRO...
  > .github
  > node_modules
  > public
  ⌄ src
    > components
    > hooks
    > local-storage
    > media
    > pages
    > redux
    > scss
    TS App.tsx
    TS index.tsx
    TS react-app-env.d.ts
    TS Routes.tsx
    .gitignore
 JS .prettierrc.js
 {} package-lock.json
 {} package.json
    README.md
 {} tsconfig.json
```

Fig 17: File and Folder structure of the client side.

The figure 17 illustrates the folder structure of client side react application which is also integrated with typescripts. The folder dot github is about the automation to CI/CD (Continuous Integration/Delivery) the project when it is pushed and merged into the master branch. This folder has a file named integration.yaml. Here is the screenshot of a yaml file that helps to check, test code, build and run the application into GitHub when it is either pushed or merged into any branches. The main idea behind this is that it ensures the developer that the pushed code or changed code has bug or not while running into the server. If there is something wrong while building the application, a warning message is displayed. The ubuntu-latest has been in use to run the build node application.

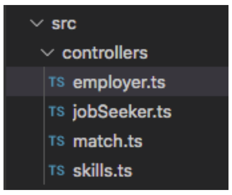


```yaml
name: CI
on:
  push:
    branches:
      - '**'
  pull_request:
    branches:
      - '**'
jobs:
  build-node:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres
        ports:
          - 5432:5432
        env:
          POSTGRES_PASSWORD: secret
          POSTGRES_DB: tindev
    steps:
      - name: Checkout code
        uses: actions/checkout@v1

      - name: Use Node.js 12.x
        uses: actions/setup-node@v1
        with:
          node-version: 12.x

      - name: Get npm cache directory
        id: npm-cache
        run: |
          echo "::set-output name=dir::$(npm config get cache)"
      - uses: actions/cache@v1
        with:
          path: ${{ steps.npm-cache.outputs.dir }}
          key: ${{ runner.os }}-node-${{ hashFiles('**/package-lock.json') }}
          restore-keys: |
            ${{ runner.os }}-node-

      - name: Install dependency
        run: npm ci

      - name: Run test
        env:
          PORT: 5000
          JWT_SECRET: abcd
          DB_PASSWORD: secret
        run: npm run test

```

Fig: 16 Screen shot of yaml file in server side

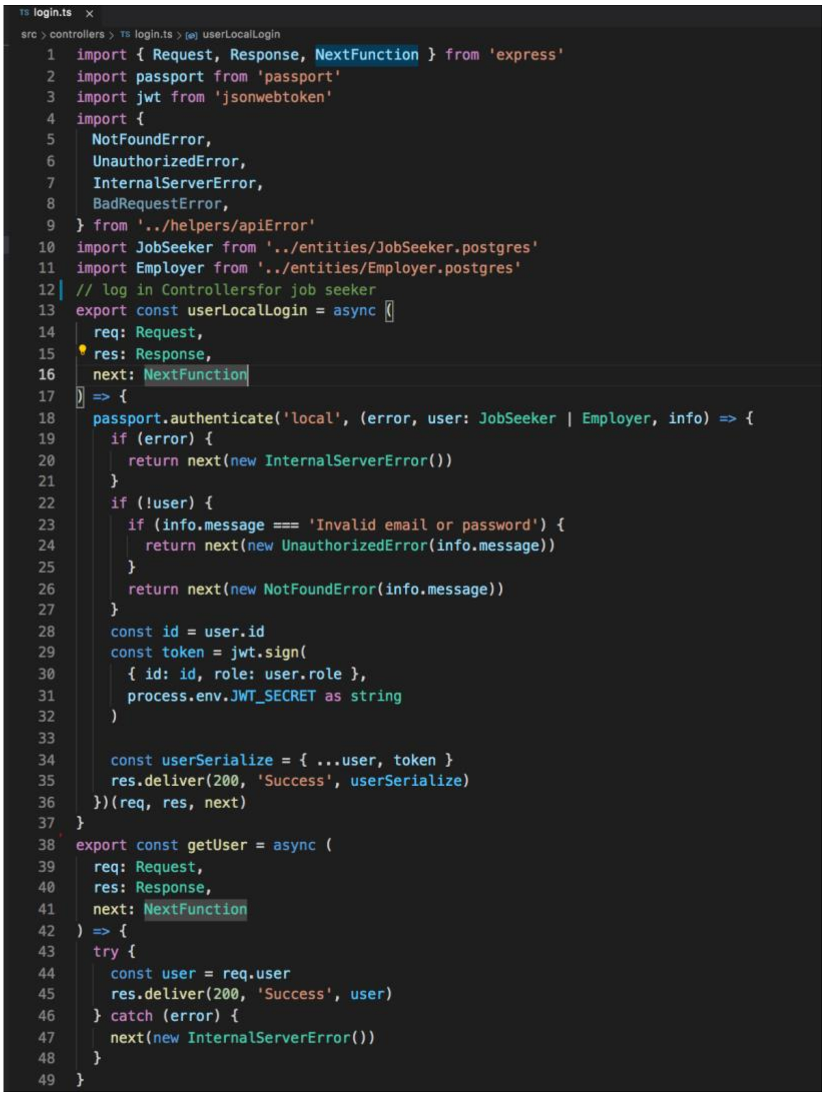
5.3 Writing Controller, Service API and Testing

This is the third major step of the project development process in which writing the actual code comes into practice. In this step, the src (source) folder will include all the necessary files and related subfolders for creating the actual functionality of the backend. Controllers are collections of different files and each file is written as a module, meaning that the files are importable and exportable to use in other files. Each file is related to its entity base. For instance, the Employer entity relates to the employer controller file (employer.ts) in which the signing-up, updating, creating job posts, deleting job posts, and adding skills as requirements to the job post services are declared. Each file of the controller folder consists of the services for the entity they relate to.



```
src
  controllers
    TS employer.ts
    TS jobSeeker.ts
    TS match.ts
    TS skills.ts
```

Fig:17 A capture of files in the controllers folder

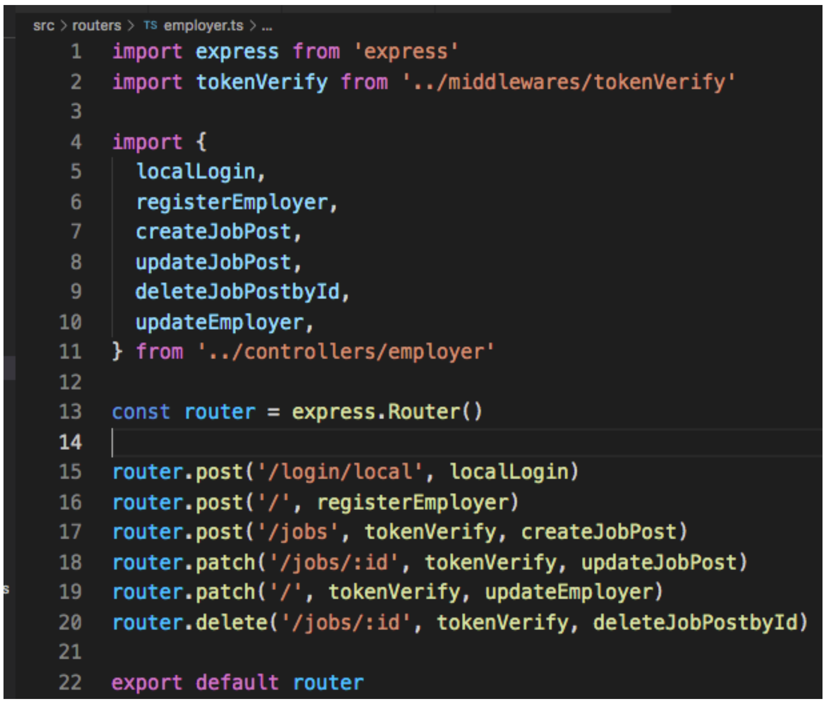

login.ts ×

src > controllers > login.ts > userLocalLogin

```typescript
import { Request, Response, NextFunction } from 'express'
import passport from 'passport'
import jwt from 'jsonwebtoken'
import {
  NotFoundError,
  UnauthorizedError,
  InternalServerError,
  BadRequestError,
} from '../helpers/apiError'
import JobSeeker from '../entities/JobSeeker.postgres'
import Employer from '../entities/Employer.postgres'
// log in Controllersfor job seeker
export const userLocalLogin = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  passport.authenticate('local', (error, user: JobSeeker | Employer, info) => {
    if (error) {
      return next(new InternalServerError())
    }
    if (!user) {
      if (info.message === 'Invalid email or password') {
        return next(new UnauthorizedError(info.message))
      }
      return next(new NotFoundError(info.message))
    }
    const id = user.id
    const token = jwt.sign(
      { id: id, role: user.role },
      process.env.JWT_SECRET as string
    )

    const userSerialize = { ...user, token }
    res.deliver(200, 'Success', userSerialize)
  })(req, res, next)
}
export const getUser = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    const user = req.user
    res.deliver(200, 'Success', user)
  } catch (error) {
    next(new InternalServerError())
  }
}
```

Fig: 18 Snapshot of Login services/Controllers (Source: Thesis's project)

The above figure illustrates two controllers in a single file: **userLocalLogin** and **getUser**. When the user/client tries to access the application, the **userLocalLogin** controller verifies if the credentials are matched. The token is generated if the credentials are matched and the user is verified as an authenticated user. The second job of the controller is to respond to the data of the authenticated user when the user tries to use the application resource. Thus, the main concept of the controller is to manipulate the request which comes from the client. Each controller has its own job. Controllers are simply functions that handle the coming request in the server side application. Controllers can be defined as module based and therefore can be exported and imported into other files.



```
 1  import express from 'express'
 2  import tokenVerify from '../middlewares/tokenVerify'
 3
 4  import {
 5    localLogin,
 6    registerEmployer,
 7    createJobPost,
 8    updateJobPost,
 9    deleteJobPostbyId,
10    updateEmployer,
11  } from '../controllers/employer'
12
13  const router = express.Router()
14
15  router.post('/login/local', localLogin)
16  router.post('/', registerEmployer)
17  router.post('/jobs', tokenVerify, createJobPost)
18  router.patch('/jobs/:id', tokenVerify, updateJobPost)
19  router.patch('/', tokenVerify, updateEmployer)
20  router.delete('/jobs/:id', tokenVerify, deleteJobPostbyId)
21
22  export default router
```

Fig: 19 Routes in the employer router file

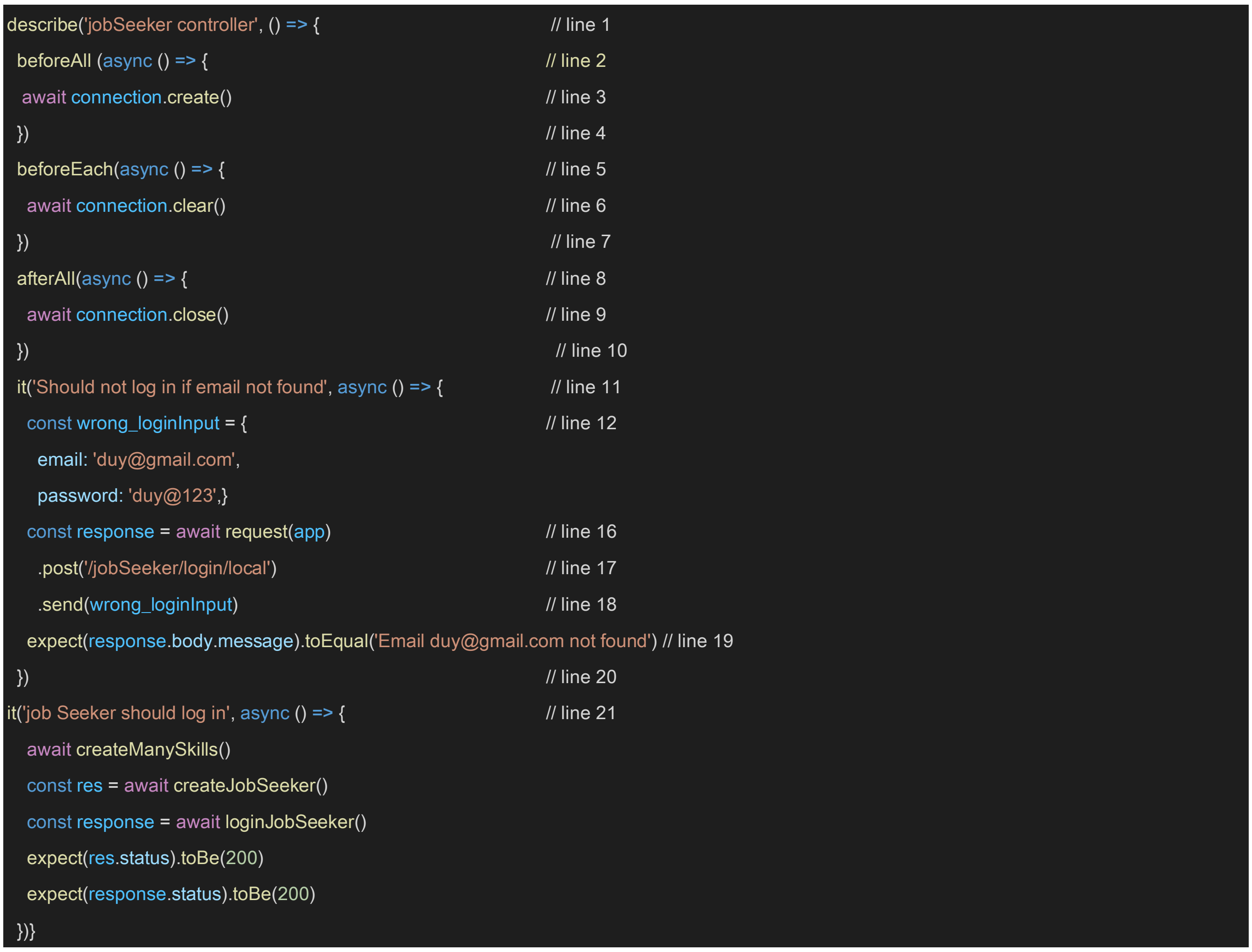
In figure 21, lines 4-11 is importing all the services defined in the employer's controller. Lines 15-20 define the routes to use the imported services. For instance, on line 15, when a user attempts to login, the request goes to the '/login/local' and the method is POST. **'/login/local'** is the portion of URL's path that will be used to send the request from the client in the client-side application. The second argument is **localLogin** which is imported from the controller. The email and password go to the controller as a request object and the controller works to authenticate the user based on that provided credential. If the credential is matched, then the user will be successfully logged in and able to use other resources too.

Likewise, on line number 17, the user can create a job post in the '**/jobs**'. Before executing the controller function **createJobPost**, there is another function passed as an argument called **tokenVerify**. This is a middleware and the task of this middleware is to identify the logged in user. If the user is not logged in, the user gets a response as an error message in the login page. This is the basic workflow how the controller, APIs and services are defined.
Here is the screenshot of all controllers on the server side.

5.4 Testing

After writing the controllers and APIs (Application Programming Interface), the step goes to testing to make sure that the defined route and the controllers' functions are perfectly working or not. That's why it is very crucial to bring testing into practice. Testing is an essential part of the software development process. It ensures the quality and assurance of the application. The Test-Driven Development (TDD) approach is a commonly used procedure in web development. TDD means that all the software requirements are being converted to test cases before the application is fully developed. The main benefits are code quality and maintained performance. There are various tools and libraries to test JavaScript code, such as Jest library, Mocha, and Chai. They are applicable for the automation testing.

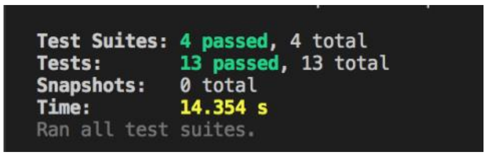
The **Jest** library along with **Postman** are used in the thesis project for testing the backend during the development mode. Postman is a GUI application for checking the http request and response status of the application. Jest is a delightful JavaScript automated testing framework for unit-testing with a focus on simplicity. This library is applicable to testing the code base of projects which use Babel, Typescript, Node, React and more. The main functionality of Jest testing is that the test is written once after which the test suite is run every time. It means, that while developing the project, if there is a side effect in some file, it notifies about it immediately during the test run. (Jest NPM Documentation, 2020).
Below is a demonstration of a test written for a backend using Jest.



```
describe('jobSeeker controller', () => {                          // line 1
 beforeAll (async () => {                                        // line 2
  await connection.create()                                      // line 3
 })                                                              // line 4
 beforeEach(async () => {                                        // line 5
   await connection.clear()                                      // line 6
 })                                                               // line 7
 afterAll(async () => {                                          // line 8
   await connection.close()                                      // line 9
 })                                                                // line 10
 it('Should not log in if email not found', async () => {         // line 11
   const wrong_loginInput = {                                    // line 12
     email: 'duy@gmail.com',
     password: 'duy@123',}
   const response = await request(app)                           // line 16
     .post('/jobSeeker/login/local')                             // line 17
     .send(wrong_loginInput)                                     // line 18
   expect(response.body.message).toEqual('Email duy@gmail.com not found') // line 19
 })                                                              // line 20
it('job Seeker should log in', async () => {                     // line 21
   await createManySkills()
   const res = await createJobSeeker()
   const response = await loginJobSeeker()
   expect(res.status).toBe(200)
   expect(response.status).toBe(200)
 })}
```

Fig 20: Code snapshot of test file (Source: Project Work)

Figure 20 is extracted from the thesis project's test suite file for the jobseeker controller. The first line describes the testing of related files and lines 2-10 represents an asynchronous function which runs every time before and after the tests run. beforeAll(), beforeEach, afterAll() are for creating, clearing, and closing the connection with the database for every test respectively. The test is represented on lines 11-20. The test suite will pass if the expected value, which is returned on line 16, is equal to the parameter of .toEqual() method.
A test file may have any number of tests. The figure above has one test suite and two tests: line 11 and line 21. When the tests file is run with the command, **npm run test,** the tests are all passing, as shown in figure 14:

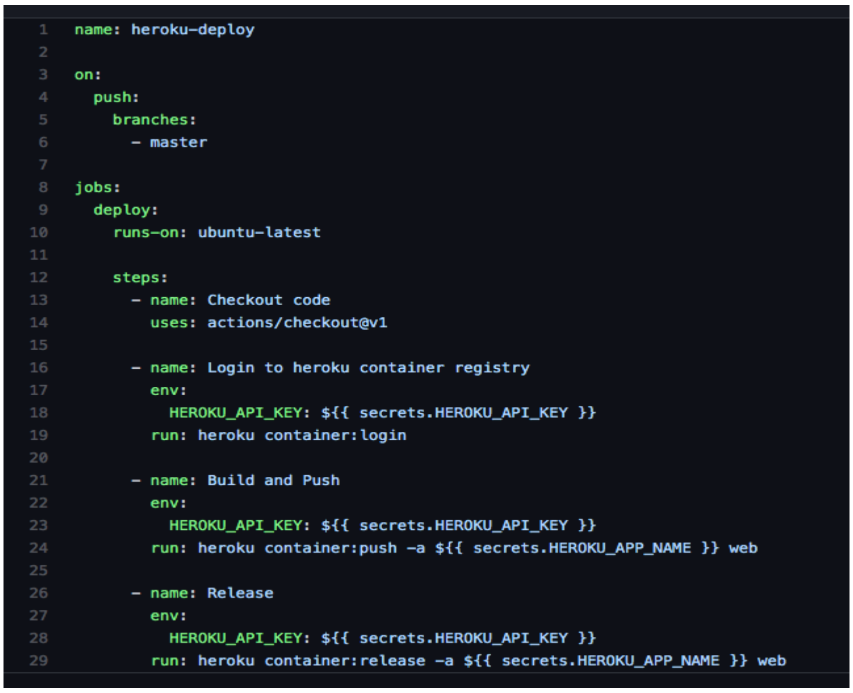


```
Test Suites: 4 passed, 4 total
Tests:       13 passed, 13 total
Snapshots:   0 total
Time:        14.354 s
Ran all test suites.
```

Fig 21: A result of a test run (Source: Project Work)

5.5 Backend on Cloud

The project uses GitHub actions and CI/CD (Continuous Integration/Development) for deployment automation, which is fully integrated with the Heroku service provider. For this, the platform needs the credentials to sign in and run the workflow which is defined in the YAML file in .github/workflows directory.



```yaml
name: heroku-deploy

on:
  push:
    branches:
      - master

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v1

      - name: Login to heroku container registry
        env:
          HEROKU_API_KEY: ${{ secrets.HEROKU_API_KEY }}
        run: heroku container:login

      - name: Build and Push
        env:
          HEROKU_API_KEY: ${{ secrets.HEROKU_API_KEY }}
        run: heroku container:push -a ${{ secrets.HEROKU_APP_NAME }} web

      - name: Release
        env:
          HEROKU_API_KEY: ${{ secrets.HEROKU_API_KEY }}
        run: heroku container:release -a ${{ secrets.HEROKU_APP_NAME }} web
```

Fig: 22 YAML file for deploying to Heroku (Source: Project Work)

Line 1 in figure 22 represents the name of the entire workflow. Line 3 is to specify the event and branches that will trigger the workflow. The name of the event is on line 4, and when the event occurs in the master branch, the workflow runs to deploy to Heroku. Form line 8 forwards are declared the jobs to be done, which in this case is only the deployment. The name of the job is declared on line 9. Line 10 declares the virtual machine on which the workflow will run. To accomplish this job, there are various steps that need to pass. Each step has the name property. The first step is to check out the code which uses the actions on line 14. The second step is to login into the Heroku container with the Heroku API key as a secret key. It stands as a credential and runs the Heroku login container. In the third step, Build and Push are for building a docker image from the Dockerfile file and pushing the image to the Heroku container registry. The final step is to release this image to the web.

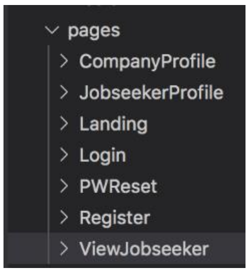
5.6 Client-side Architecture and Design Pages Using Figma

The early stage in the creation phase of the project is very important in terms of being able to stand out with the product in the competitive market filled with all kinds of web applications. The design of the application and the UX (user experience) need to be carefully done, usually by a team dedicated to this part of the process. In the thesis project, the first discussion topics in the team were about the pages that should be included in the application and which of them needed user authentication.

The client-side application architecture has been designed as follows:

a. Landing page: The first page that the user sees, consisting options to either go to the sign-up page or the login page.
b. Login page: The page is designed for the jobseeker and the employer to login.
c. Sign-up: On this page, the user creates their credentials and chooses their role - jobseeker or employer - after which they will be directed to the login page.
d. Profile page: The page is different for both users. On this page, the user will create their profile by filling a form. They can also navigate to the page displaying a list of matched job posts, if they are signed in as a jobseeker, or a list of matched job seekers, if they are signed in as an employer. They will also see a link to the chat page. On the employer profile page, there is a link to another page where they can add a new job post. The employer can also check out all of their job posts and edit or delete them.

In addition to the above mentioned pages, the application includes a page for resetting the password and a job seeker view page for the company to take a closer look at the jobseeker they are matched with.



```
⌄ pages
  > CompanyProfile
  > JobseekerProfile
  > Landing
  > Login
  > PWReset
  > Register
  > ViewJobseeker
```

Fig: 23 The structure of the application pages (Source: Project Work)

Each page is designed using a versatile design tool called **Figma**. It is a collaborative interface design tool which can be implemented in many ways, such as in prototyping. Figma is very flexible and fast to work with when creating animated prototypes that look like real web pages.
(Figma Document, 2020).
Here are some images extracted from the implementation of Figma in project.

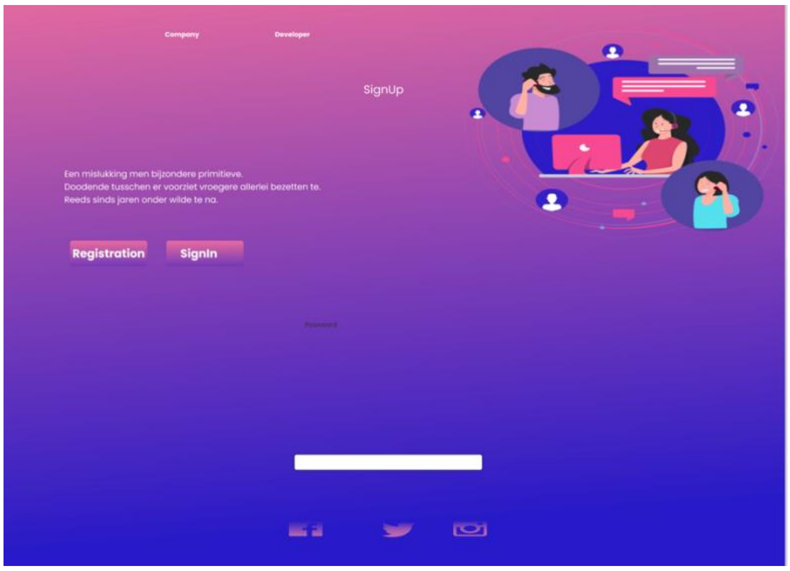

Company
Developer
SignUp
Een mislukking men bijzondere primitieve.
Doodende tusschen er voorziet vroegere allerlei bezetten te.
Reeds sinds jaren onder wilde te na.
Registration
SignIn
Password

Fig: 24 Design of Landing page in Figma. (Source: Project Work)

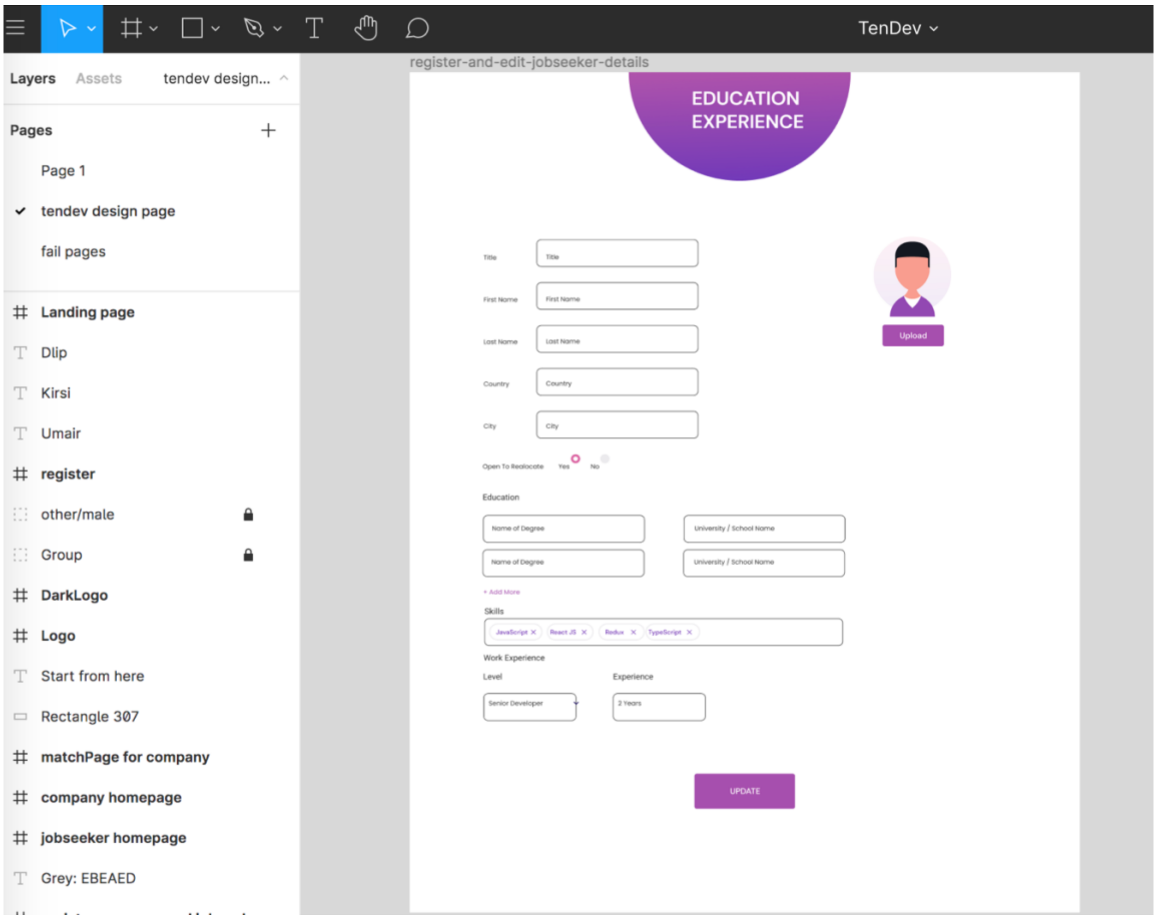
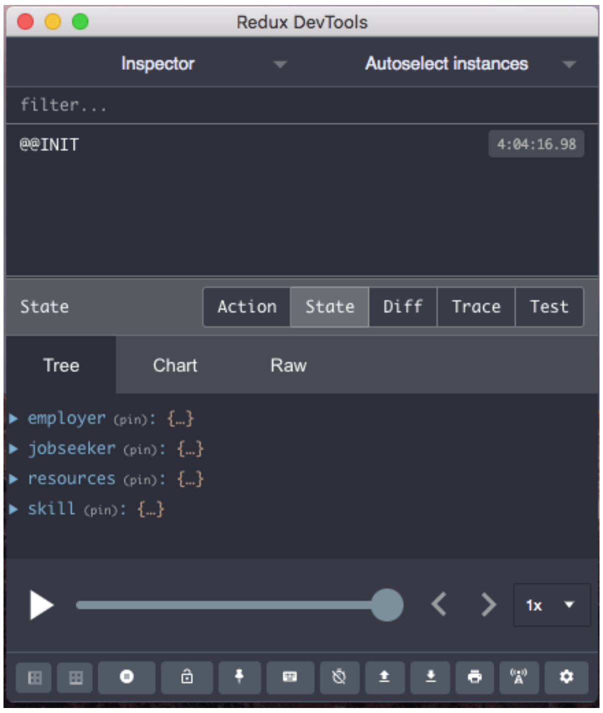


Fig: 25 A form page designed in Figma. (Source: Project Work)

Before writing any code, a prototype of a user experience design is needed for viewing and analyzing the structure and design of the pages that are going to be built. This page above is the jobseeker profile form page where the jobseeker fills in their profile information. Based on the designed prototype, developers can decide early on which features are not feasible or necessary to keep and which parts need to be changed or improved. It saves time from writing unnecessary code.

5.7 Project Setup and Integration with Redux (State Management Library)

Some of the dependencies in the package.json file are meant to support the development mode only, TypeScript being the main dev dependency in the project. As discussed in the theoretical chapter, Husky is also a dev dependency, used for avoiding unnecessary commits. The third dev dependency is the **Chrome Redux DevTools Extension**. Its purpose is to monitor the state in the application. When an action is dispatched, it monitors the changes in the state: What those changes are and what data each action has. It is a very handy tool to see the changes of the state in Redux-integrated applications. When the dependency is installed and the Redux store file configured, the tools can be accessed via the browser. The devtools can be opened either in a separate window/panel (figure 25) or in a new tab using the inspection option of chrome tools.

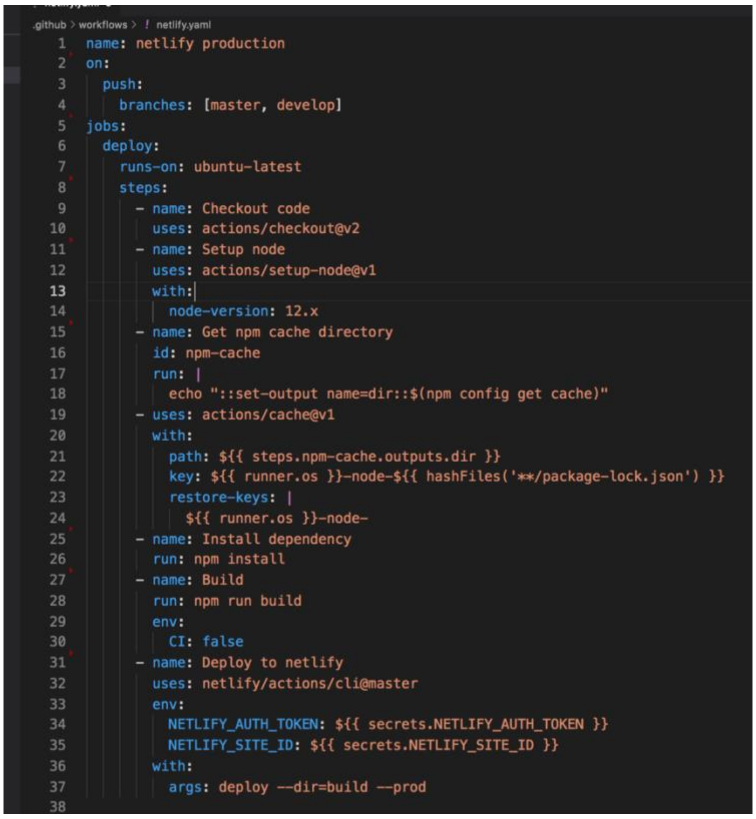


```
@@INIT                                  4:04:16.98
▸ employer (pin): {…}
▸ jobseeker (pin): {…}
▸ resources (pin): {…}
▸ skill (pin): {…}
```

Fig: 26 The Redux devtools window for monitoring the state (Source: Project Work)

5.8 Client-side app on Cloud

The project's backend application has been deployed into Heroku Cloud. But our client-side application has been deployed into the Netlify account. As like in Heroku, the account has registered in Netlify and set up all the environment variables such as AWS secret key and SSH keys. The source code from GitHub has been linked with the Netlify account. The deployment has an automation process defined and registered all the jobs in a yaml file. When the branch is merged into the master or develop, the file created inside dot github/workflow monitor the changes and deploy to Netlify cloud. All the jobs are run in the sequence defined in netlify.yml file. The following figure illustrates all the jobs and steps to deploy and run the application into Netlify cloud platform.



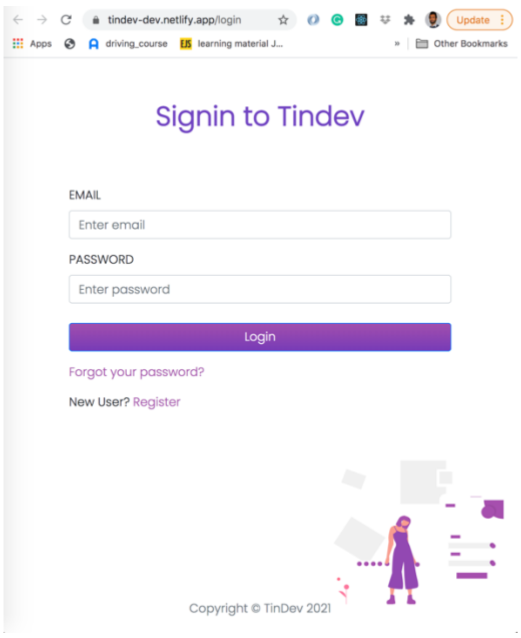
```
.github > workflows > ! netlify.yaml
name: netlify production
on:
  push:
    branches: [master, develop]
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2
      - name: Setup node
        uses: actions/setup-node@v1
        with:
          node-version: 12.x
      - name: Get npm cache directory
        id: npm-cache
        run: |
          echo "::set-output name=dir::$(npm config get cache)"
      - uses: actions/cache@v1
        with:
          path: ${{ steps.npm-cache.outputs.dir }}
          key: ${{ runner.os }}-node-${{ hashFiles('**/package-lock.json') }}
          restore-keys: |
            ${{ runner.os }}-node-
      - name: Install dependency
        run: npm install
      - name: Build
        run: npm run build
        env:
          CI: false
      - name: Deploy to netlify
        uses: netlify/actions/cli@master
        env:
          NETLIFY_AUTH_TOKEN: ${{ secrets.NETLIFY_AUTH_TOKEN }}
          NETLIFY_SITE_ID: ${{ secrets.NETLIFY_SITE_ID }}
        with:
          args: deploy --dir=build --prod
```

Fig: 27 Yaml file to integrate automation to deploy client side into Netlify Cloud Platform.

The solution is not permanent but applied for the purpose of providing an overview of deployment to Netlify and to be able to visit the website on the internet.

The url of the client side application is: https://tindev-dev.netlify.app/ .

The backend url of the application is: https://tindev-dev-deploy.herokuapp.com.



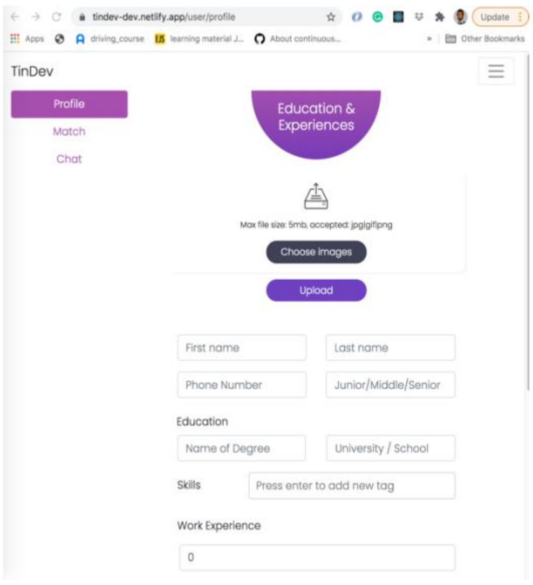


Fig: 27 Examples of pages in the application in Heroku deployment (Source: project Work)

6 Conclusion

The main intention of the thesis was to build the initial version of a web application using the latest tools and technologies. Now, the application has been successfully built with today's trend and deployed into Heroku and Netlify cloud platform. In summary, the trends in web development and technology are constantly changing and the appearance of modern websites as well as the constructive way of building web components are far superior and out-reaching than before. The scope of web applications has broadened because of the competitive market and users' demand. Only secured, fast and reliable web applications meet users' wants and needs. In consequence, the tools and techniques are rapidly advancing to meet them.

Looking from a developer's perspective, it is crucial to follow updated technologies to be able to fulfill the expectations of the end user and avoid becoming outdated in the field of web development. This thesis emphasizes the importance of staying on top of the game by introducing some of the latest and popular web development trends and tools and how to implement them in a real web application.

This project-based research has worked through the development process of the full stack application, introducing and implementing modern stacks from Postgres database to Node JS in the backend and React JS to Bootstrap in the frontend. In between, many highly useful and popular features and tools, such as PassportJS and Git have been presented.

Writing the thesis has been an opportunity to reinforce and accumulate more knowledge about the modern stacks. The report provides insights into selecting better technology according to the needs of the web application. Hopefully, the thesis will support the developers who are trying to improve their chances in the current job market by providing them a list of essential stacks in modern web development and the basic understanding of them.
Because of the latest tools and stacks, it was a nice journey in the development of secured and maintainable application. The main benefits of the following stacks are that it is very handy and easy to understand the source code and effortless to develop the application in the next version. All the environment is set up already and the required platform will be installed into the local machine with the help of docker technologies. Hence, as a result, any developer can pull the source code from GitHub and use it in their own way.
Now any Job Seeker can register only with his email and password and can create own profile to hunt the job. On the other hand, the employer can publish the job post with the requirements and own basic information. Thus, the built web applications are providing these services and has handover to the commissioner.

6.1 Personal Conclusion

As I experienced in building this web application, it was really the golden chance for me to improve my skills set from the learning point of view and the opportunity made my skills wide. This thesis and projects worked pushed my skills further. When I had begun the project, it was very tough and later it forced me to read and follow the different article that also helped me a lot to build the project and work in a team. So, the main achievement in this project is not only the thesis writing, it also teaches me how to work in team collaboration, how to follow the documentation and made me familiar with different stacks that are vital in modern web design. Besides these, from my personal perspective, depending on the internet blogs, articles, colleagues, I developed my confidence to take the project into the next level.

6.2 Further Development

The first version of the application is providing the basic features which is the main aims of thesis. There are some shortcomings in the application. For example, the chat service has not been implemented so far. I am thinking to make it into the next level. Images are not perfectly managed into the AWS (Amazon Web Service) s3 bucket. The recovery of password features has not been integrated now which is not good in production mode. So, if I could implement those features, the application would go to the next level and raise the value of application itself.

7 References

CERN 1991, Web History Documentation. URL : https://home.cern/science/computing/birth-web/short-history-web. Accessed: 02 Dec 2020

Uryutin, O. 2018. *A brief history of web app*. URL: https://medium.com/@aplextor/a-brief-history-of-web-app-50d188f30d. Accessed: 02 Dec 2020

Joe Stangarone, 2019. Essential Principles of Modern Web Developments. URL: https://www.mrc-productivity.com/blog/2019/09/7-principles-of-modern-web-application-development-2/. Accessed: 04 Dec 2020

Marcelo Pastirino, FrontEnd vs Backend. URL: https://www.pluralsight.com/blog/software-development/front-end-vs-back-end. Accessed: 04 December 2020

Web & Mobile Application Design & Development Company in NYC | NJ. (2019). *Modern Web Application Development: Top 6 Highly Preferred Principles*. URL:

https://www.technosip.com/modern-web-application-development-top-6-highly-preferred-principles/. Accessed 7 Dec. 2020

Code World (2020). How to create an http server with Express in Node.JS. URL: https://ourcodeworld.com/articles/read/261/how-to-create-an-http-server-with-express-in-node-js. Accessed: 15 December 2020.

AWS S3 (2020). The S3 service in AWS. URL: https://aws.amazon.com/s3/. Accessed: 18 December 2020.

Project Management with GitHub (2020). Features of GitHub Project management tools. URL: https://github.com/features/project-management/. Accessed: 19 December 2020.

Passport Strategy, 2020. Passport Strategy Documentation. URL: http://www.passportjs.org/docs/. Accessed: 17 Dec 2020.

React Documentation (2019) URL: https://reactjs.org/. Accessed: 15 December 2020

JavaScript Documentation (2019) URL:https://developer.mozilla.org/enUS/docs/Web/javascript Accessed: 16 December 2020

Redux Documentation. State Management Library. URL: https://redux.js.org/. Accessed: 28 November 2020

Jest Documentation. JavaScript Testing Framework. Url: https://jestjs.io/. Accessed: 12 Dec 2020

Files Handling Documentation. How to deliver files from client to AWS bucket? URL: https://medium.com/developing-koan/uploading-images-to-s3-from-a-react-single-page-application-45a4d24af09f. Accessed: 12 January 2021

Stack Overflow Developer Survey (2020). URL: https://insights.stackoverflow.com/survey/2020. Accessed: Dec 20, 2020

Node.js Documentation(2019). *URL*: https://nodejs.org/en/. Accessed: 18 December 2020

Deploying Node.js Apps on Heroku (2021). URL: https://devcenter.heroku.com/articles/deploying-nodejs. Accessed: 15 Jan 2021.

Docker Documentation (2018).*URL*: https://www.docker.com/. Accessed: 20 December 2020

PostgreSQL Documentation. URL: https://www.postgresql.org/docs/. Accessed: 20 December 2020

TypeORM Documentation URL: https://typeorm.io/#/. Accessed: 22 December 2020

React-Bootstrap Documentation. URL: https://react-bootstrap.github.io/. Accessed: 24 December 2020

Figma Documentation. URL: https://www.figma.com/developers. Accessed: 20 December 2020

TypeScript Documentation. URL: https://www.typescriptlang.org/docs/. Accessed 30 November 2020

GitHub Actions Documentation. URL: https://docs.github.com/en/freeproteam@latest/actions. Accessed 30 November 2020

A Front-End Framework, Max Pekarsky (Feb 3, 2020). Does your web app need a front-end framework? URL: https://stackoverflow.blog/2020/02/03/is-it-time-for-a-front-end-framework/. Accessed: 11 January 2021