Expertise
and insight
for the future

Deepak Bhandari

Job Vacancy Application using MERN stack

Metropolia University of Applied Sciences

Bachelor's

ICT

Job Vacancy Application using MERN stack

07/11/2018

Author(s) Title Number of Pages Date	Deepak Bhandari Job Vacancy Application using MERN stack 36 pages + x appendices 07 November 2018
Degree	Bachelors of ICT
Degree Programme	Information and Communication Technology
Specialisation option	Mobile Application
Instructor(s)	Janne Salonen (Content Instructor) Sonja Holappa (Language Instructor)

This thesis focuses on the development of a full-stack job searching web application using Javascript technologies. The application is written using MERN stack (MongoDB, ExpressJS, ReactJS and NodeJS), ReactJS will serve the front-end while NodeJS and expressJS will keep backend stay connected to the database, MongoDB. The main objective of the project for the author was to learn full stack web development using ReactJS as well as to build a job searching application for his home city back in Nepal.

Each of the afore mentioned javascript libraries were thoroughly studied in-order to create the application. Various frameworks and their strengths were studied concluding to the most job savvy ReactJS backed by regular Mongo, Express and Node trio. React being one of the most popular front-end java library developed and being used by Facebook, provides multitudes of services in creating a single page web application.

This document seeks to explain the motivations in using the MERN stack and each of their brief explanation. This thesis also acts as a brief guidelines on developing a full-stack MERN web application. Further development process' will be explained on the latter half of the document. The application is a beta, so, real server deployments, user interface improvements and practical implementation will be carried out even after the thesis has been submitted. Security and deployment process' are further explained below.

Keywords	

Contents

Abbreviations

API	*Application Program Interface*
CSS	*Cascading Style Sheets*
DOM	*Document Object Model*
NPM	*Node Package Manager*
HTML	*Hypertext Markup Language*
JS	*JavaScript*
JSX	*JavaScript XML*
URL	*Uniform Resource Locator*
HTTP	*Hypertext Transfer Protocol*
TCP	*Transfer Control Protocol*
JSON	*JavaScript Object Notation*
XML	*Extensive Markup Language*
UI	*User Interface*
ODM	*Object Data Modeling*

1. Introduction

Technology has never been this interesting, specially web-technologies. Web developing using just HTML and CSS and linking many static pages together are the things of the past. Developers are continuously provided with so many options and services to tweak around the web-pages. Many frameworks and libraries are constantly being developed to create a web-page in a certain manner. Some provide efficiency, some agility, some promise on being light weight, while some boast on adaptability, however all of them focus on making the best looking and behaving web page.

Many choices results in confusion, similarly choosing a convenient set of frameworks and library or a Stack is an extremely hectic process. LAMP (Linux, Apache, MySql, PHP) was the first to popularise the word Stack, meaning a set of frameworks constituting a complete web application, i.e. front-end and back-end. On latter development of javascript frameworks, MEAN (Mongo DB, Angular JS, Express and Node), MERN replacing React with AngularJS was among the most used stacks. Choosing a stack is one of the most important aspects of web development. However, MERN being the title of this thesis already and a full stack component based platform, learning and implementing it could be fun as well as irritating and time consuming at times.

The web application is the author's idea towards an startup for popularising vacancy announcements via easily accessible web app back in cities of Nepal where unemployment rates are plunging for people not having a medium while economy is at the rise because of new government system. Nepal has recently thrown away the hundreds of years of autocratic monarchy and moved on to more democratic federal government. Businesses are more open ended and this ecstatic evolvement of web technologies certainly helps in blooming the reach and actual economic prosperity of specially small businesses. Since small businesses are the backbones of any developing economies. This publicly accessible application will help a lot of people find jobs and employees find their matching employer. A lot of people are not fit (either unqualified or overqualified) for their job and helping everyone secure their goals in the form of a vacancy application they just mailed because of this app would be a good social effort. While on further developments, author is motivated in creating a job pool app, where one can ask someone to do small jobs like delivery, car wash etc for calculated prices. Practical application will be implemented when the final project is delivered.

Furthermore, it is explained how MERN is one of the best full stack javascript framework and benefits of using React over other front-end frameworks and libraries.

2. Full Stack Web-development

Various web and native applications are developed using ‘stacks’ of various technologies. The word ‘stack’ was first referred to the LAMP stack, Linux as the OS, Apache as the http server, MySql the relational database and PHP as the programming language on which the application is developed. However, multipage applications are on the decent and single page applications are more popular because of their seamless user experience and lighter server calls which makes rendering the part of page easier without refreshing the page. So, the front-end frameworks or libraries those can produce single page applications are on demand. React being one of them.

Just like a good front-end, there needs to be a supporting database. This thesis is based on a NoSql database, MongoDB. NoSql databases are also called Not-Only-Sql because they also support SQL like query languages. MySql (SQL) stores data in rows and columns, NoSQL databases store their data in different structures: key-value pairs, wide columns, graphs, or documents. Simplicity, scalability, flexibility, availability and speed of NoSql, databases like MongoDB have gained fame.

To bind the front-end and database together and to output a logical result in the client screen, a strong backend is necessary. Popular languages like PHP, Ruby and Python are among the early server side languages. Since, Javascript has made it possible to create dynamic front end, its development has also had an impact on back-end, in the form of NodeJS and ExpressJS. As a contemporary website consist of client side graphical user interface (GUI), server side communication, server and the database to talk to, the complete package comprising of all these factors make up full stack web-development. And the current development of web technologies has further allowed to combine different libraries with various features, across different platforms to work on to a single stack. [6]

3. MERN Stack

MERN stack as briefly discussed above consists of 4 independent frameworks and libraries, Mongo DB, Express JS, React JS and Node, hence the abbreviation MERN. Below are described the individual aspects of MERN stack.

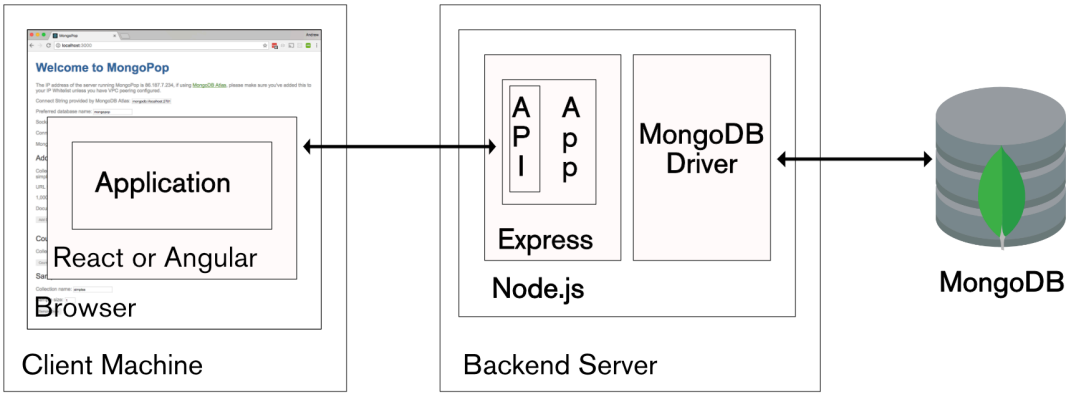

Fig 1. Full stack MERN architecture. Reprinted from The Modern Application stack [2].

3.1 Mongo DB

MongoDB is an open-source, cross-platform document-oriented NOSQL database. Initially released on Feb 11 2009 by MongoDB Inc, it is licensed under Server Side Public License (SSPL). MongoDB uses JSON like files with dynamic schemas. Data are mainly stored as documents and collections. Collections can have multiple documents and different schemas associated to them. Just like JSON objects, records on MongoDB are documents with data structures containing field and value pairs but with their own respective validations or schemas.

Unlike relational databases (RDB), MongoDB does not store data in tables. It assigns keys and values pairs. These records are stored and fetched using a *key item* that uniquely identifies the record, and is therefore used to quickly find the data within the database. Each document has its unique ID field as a primary or identifier key for query operations. Binary JSON (BSON) structured, schemas governing keys and the data stored in them, values are stored in respective fields in documents inside of collections. One collection has to be stored in one machine. Since JSON is compatible with JavaScript, queries in MongoDB are written using JavaScript. MongoDB's data modelling makes it quite easy and efficient to store the data and combine it with another data with any structure, without writing ambiguous validation rules, flexible data access and rich indexing functionality. Developers can dynamically perform schema modification without having a downtime, this feature is extremely convenient for developers making rapidly evolving applications.[2] Below is a basic representation of field and value pairs in MongoDB document.

```
{

name : “Deepak”,                    <- field: value

age : 25,                           <- field: value

hobbies: [“singing”, “reading”]     <- field: value

}
```

MongoDB comprises of rich query languages to support basic read and write CRUD operations while being able to easily text search and data aggregation. Among best benefits offered by MongoDB one of them is using dynamic schemas which eliminates the need to pre-define the structure, like fields or value types. Such models allows hierarchical relationships representation, array storage, and ability to change the records structure by simply adding or deleting fields.[3] Schemas are stored inside of Collections, within the documents. And each documents can have any amount of schemas and shapes inside of them. Because of which developers can store their data in the database without considering the database's structural design. MongoDB doesn't require data type validation therefore the database's keys and values can be updated when required.

Among other big MongoDB features, auto sharding data to horizontal scaling and replication of data among multiple servers for higher reach are the most developer friendly assets. All the data are stored as documents, therefore horizontal scaling allows data as documents to be stored at many different servers and locations, efficiency will be the optimum than having a single machine handle all the valuable user information. More machines means more space to support data growth and most importantly to maintain efficient read-write operations. Which also allows storing higher scaled information, when the start-up starts taking pace. Similarly, replication of the data prevents the chances of data-loss since the data is stored in multiple servers. When one server fails, the data still could be fetched from the replica. Highly efficient and higher scaled storable informations achieved via auto-sharding and server data replications keeps MongoDB edges away than other relational database counterparts like MySQL and Oracle databases.

Everything has drawbacks, of course MongoDB has few, one being the inability of processing multiple operation as one transaction like other relational databases. Therefore, if any operation causes an error the whole transaction fails. It also lacks joint operations features like MySQL which in turn makes it weak against data with multiple relationships among them. When making changes, multiple documents has to be updated. Changes in multiple data clusters in different documents should be tied to a single transaction to make sure all the collections are updated. Hence MongoDB being a non-transactional database, processes like such could be few of its weaknesses, it has hard times at such. However, the ability to store any kind of freeform data structure without any rule governing the relation can be a strength at times, applications can be built at any platforms and they can have any sort of data-structure they prefer. For the

handling of such ambiguous ungoverned data structure, Mongoose was created by MongoDB.

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides schemas to model application data which therefore cleans up the ambiguity of databases. Mongoose as a library contains built-in type casting, validation, query building, business logic hooks and more. Therefore it aids in handling the relationships between the data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB. Mongoose removes the need of writing difficult data validations, business logics and casting broiler-plates in the documents. Moreover, it provides additional sets of features on top of MongoDB, for instance it can help manage the connections to MongoDB database. It can be helpful just even to perform basic operations like read, write, and save data. By providing schema level validation rules, it also allows only valid data to be saved in MongoDB.

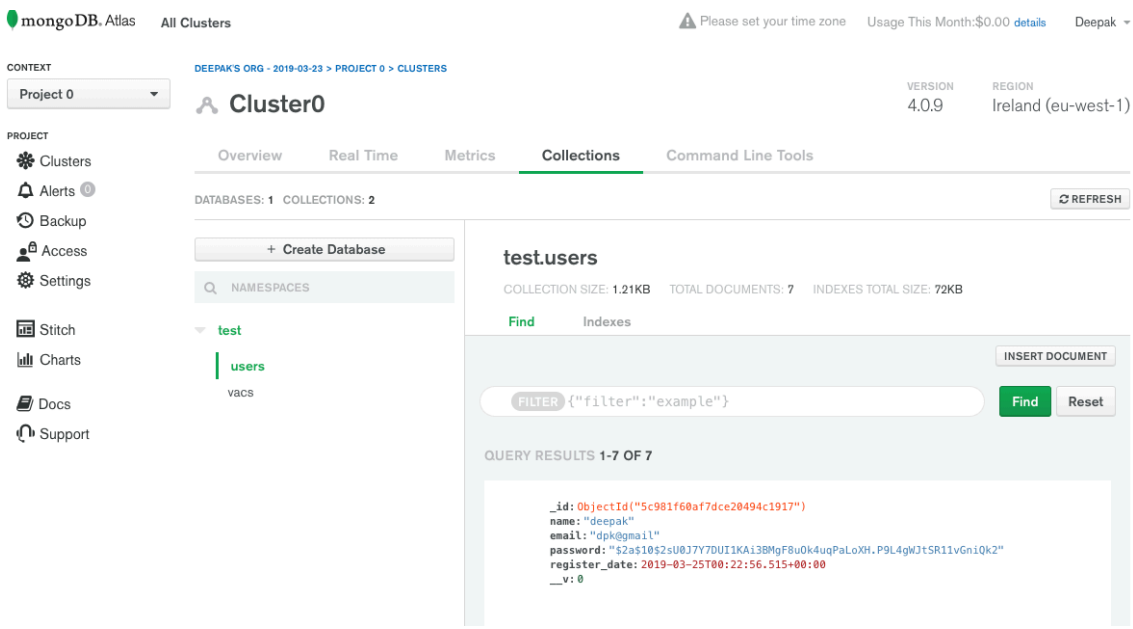

Fig 2. MongoDB Atlas screenshot.

Atlas provides easy web access to use MongoDB as a cloud service or it can be used through Mongo shell. Former is used as the method to access database for this

project. Atlas handles all the processes of deploying, managing and healing the deployments on the developers choice of cloud services (AWS, Azure and GCP). Mongo being an open source platform, free 0.5 GB of memory can be allocated to an individual project for education purposes.

3.2 Node.js

Node.js is the currently most popular open source web server environment developed by Ryan Dahl under Joyent Inc (currently owned by Samsung electronics). Being a runtime environment, Node.js helps in executing javascript codes outside of a browser. It uses Google chromes V8 engine to execute the javascript. V8 is originally Google's open-source runtime engine which was written using C++, similarly Mozilla uses SpiderMonkey and Microsoft uses Chakra as their respective runtimes. In layman's term, it helps translating javascript codes into machine level language, hence a machine is able to understand the javascript codes without the need of a browser. This factor is key towards developing full-stack applications. Node.js as a whole makes developing cross-platform, server-side and networking applications much easier.[7]

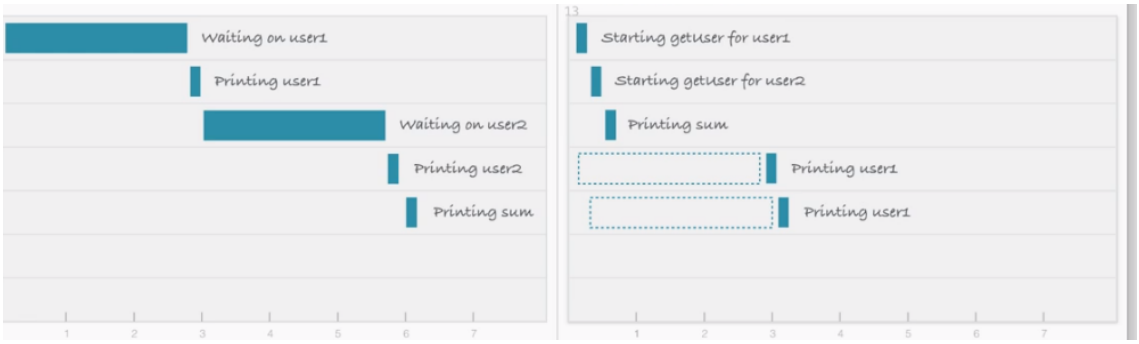

Fig 3. Blocking I/O (left) vs Non-Blocking I/O (right). Reprinted from What exactly is NodeJS [7]

V8 engine being the main component, Node also includes several modules like networking protocols such as HTTP and assists in installing other third party modules like MongoDB via npm (node package manager) tool. Node.js is an asynchronous, event driven engine, meaning that when an application makes a request, it continues working on that request to get a response while also continues working on other useful stuff. On the completions of requests, it informs the application about the response by a callback. It helps in processing a large amount of request in parallel, which amounts in making large scaling applications. MongoDB was also designed for being asynchronous, hence these two can really work together side by side.

Node.js makes use of event driven, non-blocking I/O model to execute requests.[7] Non-blocking I/O model makes node asynchronous.[7] The flow of the programs is defined by the events taking place. Javascript uses single threaded event-loop, multi threaded processes cant be executed. Hence, node has to execute multiple threads in parallel without waiting on every event. Below is an example of what blocking vs non-blocking I/O model looks like.

Node also provides tools and functions like NPM (node package manager) and require, which help managing third party libraries like Mongoose and Express to make development faster and efficient. NPM is the default package manager which comes bundled with the Node environment. These tools also help load inbuilt node modules into respective applications. Node modules are blocks of codes that can be used where needed but when not loaded do not impact the application codes. Developers can also build their own modules for personal use and for the community.

3.3 Express JS

Express is a very light, extremely minimal and flexible open-source Node.js web application framework designed for building web applications, mobile API applications and APIs. Express runs as a module or an element within the Node.js environment. Unlike popular frameworks like Koa.js and Happy.js, Express is much more minimal and provides advanced features in forms of plugins. Working as a layer above Node.js, Express can help designing single page, multi page and hybrid applications. Same back-

end applications could be written in plain Node, but Express provides with higher securities, much lesser lines of codes hence much faster development and also the easy accessibility of third-party plugins makes it the most popular Node.js framework among the developers.[11]

Express makes responding to requests with route support such that developers can write responses to specific URLs simpler. It also supports multiple template engines to simplify generating HTML. It can be easily installed using NPM and can be accessed as a require statement in the server files. Below is an instance of a basic HTTP server (server.js) that outputs 'Hello Express' built using Express.

```
const express = require('express');

var app = express();

app.get('/', (req, res) => {
    res.send('Hello Express')
});

app.listen(process.env.PORT || 3000)
```

Above written server.js file when ran using 'node server.js' in the terminal displays 'Hello Express' in browsers port 3000 (accessed using http://localhost:3000/ in the browser). One of the many benefits of express is the use of middlewares. Middleware are any numbers of functions invoked by the Express.js routing layer before handling the final request. Residing in the middle of raw request and final intended route, thus middleware. Callback on the third line is an example of an Express middleware.

Node.js and express do not necessarily come with a strict file and folder structure. Below are some of the components used along the express application.

3.3.1 Controllers

Controllers define the app routes and logic. This application has file server.js as the main route and auth.js, users.js and vacs.js as the secondary routes to access far corners of the application. Routes mainly deal with Get, Post, Update and Delete

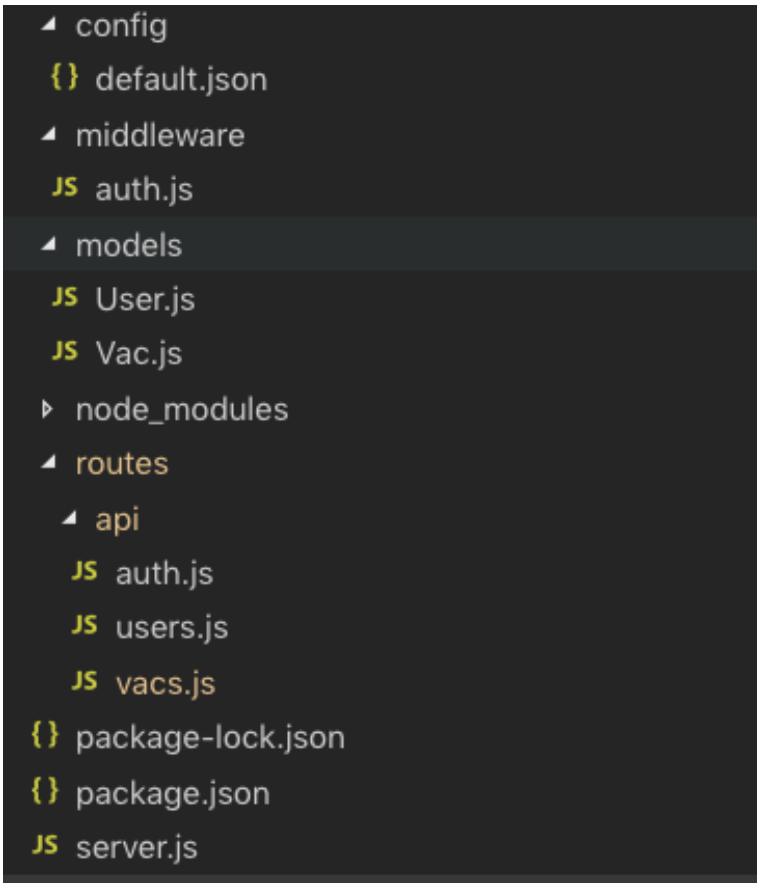

Fig 4. Back-end file structure.

3.3.2 Middlewares

Express middleware process the incoming requests before handling them down to the routes. Auth.js middleware is used to check for the user authentication before any end-user proceeds into the application. Below lines of code checks for the authentication token and proceeds accordingly.

```
const config =  require('config');
const jwt = require('jsonwebtoken');

function auth(req, res, next) {
    const token = req.header('x-auth-token');
    //check for token
```

```
    if(!token){
        return res.status(401).json({msg: "No token, au-
thorisation denied"})
    }
    try {
        //verify token
        const decoded = jwt.verify(token, config.get('jwt-
Secret'))
        //Add user from payload
        req.user = decoded;
        next();
    } catch (error) {
        res.status(400).json({msg:"Token is not valid"})
    }

}
module.exports = auth;
```

3.3.3 Models

These files represent the data structure of the application. It implements data logic and handles the storage to MongoDB. User.js and Vac.js are the basic data models or schemas used. Below is the data model for the entity User.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

// Create Schema
const UserSchema = new Schema({
  name: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true,
    unique: true
  },
```

```
  password: {
    type: String,
    required: true
  },
  register_date: {
    type: Date,
    default: Date.now
  }
});

module.exports = User = mongoose.model('user',
UserSchema);
```

3.3.4 Package.json

Package.json file is core to the Node.js ecosystem. It manifests all the tools, plugins and dependencies used along the development of the application.

3.4 ReactJs

React is one of the most widely used open source, component-based JavaScript library for building client side user interfaces naturally for single page applications. It was first developed by Facebook's engineer Jordan Walke and is constantly updated by Facebook and community of developers around the world. It was first deployed in Facebook's newsfeed in 2011.

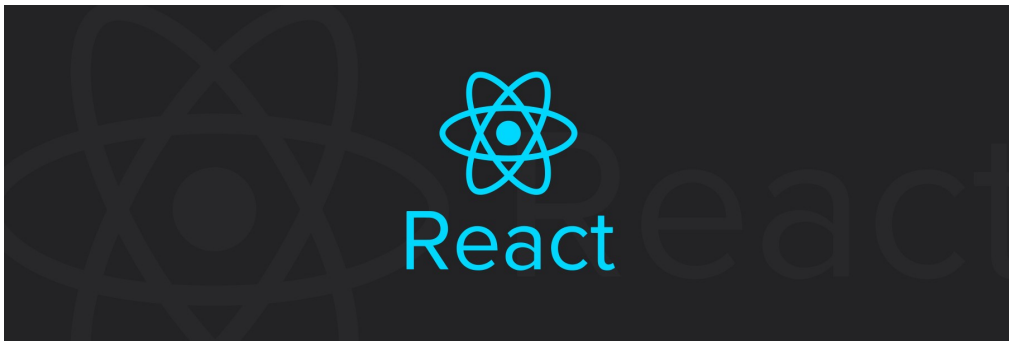

Fig 5. React logo. Reprinted from React webpage. [8]

React allows rendering different components with different events without necessarily having to re-render the web-page. As an UI interface library, react helps build fast, scalable, single-page, component based web-pages and applications. It most specifically targets the view layer of MVC pattern of web development. Unlike other front-end frameworks like Angular, React uses views that are declarative and makes code more predictable to write and easier to debug. It allows developers to declare what to render in the webpage instead of telling browser how to do it. React prepares simple views for every different state in the application, and will update and render just the right component when the data changes. Since React is just a library not a framework, it is in itself lightweight hence the applications are executed instantly. React is easily readable because it uses the basic javascript fundamentals. It also orders codes in the block of reusable components which can be updated every time a state is changed. [1]

Any react applications are made up of many different components styled around each other using bootstrap, react-strap in this project. Each component is responsible for rendering fairly a small, reusable piece of HTML code. Components can be nested within other ones, rendered on events to allow scalable applications to be built out of small reused blocks.. A component can also maintain internal state – for example, a

Cart component might contain state regarding current items inside of it, a TabList component can store a variable corresponding to the currently opened tab. In classical web-development, instant changes are made to the DOM (document object model), which is a slow process. But, React makes its own virtual DOM by copying the browser DOM, which is similar to the end-user DOM but it makes changes to the virtual DOM elements and only updates the browsers DOM if necessary by the comparison. Unlike browser DOM elements, react elements are plain objects and are more efficient to create.[3] The learning curve for React fundamentals is quite easy and the robust security makes it the most desired javascript framework among the developers.

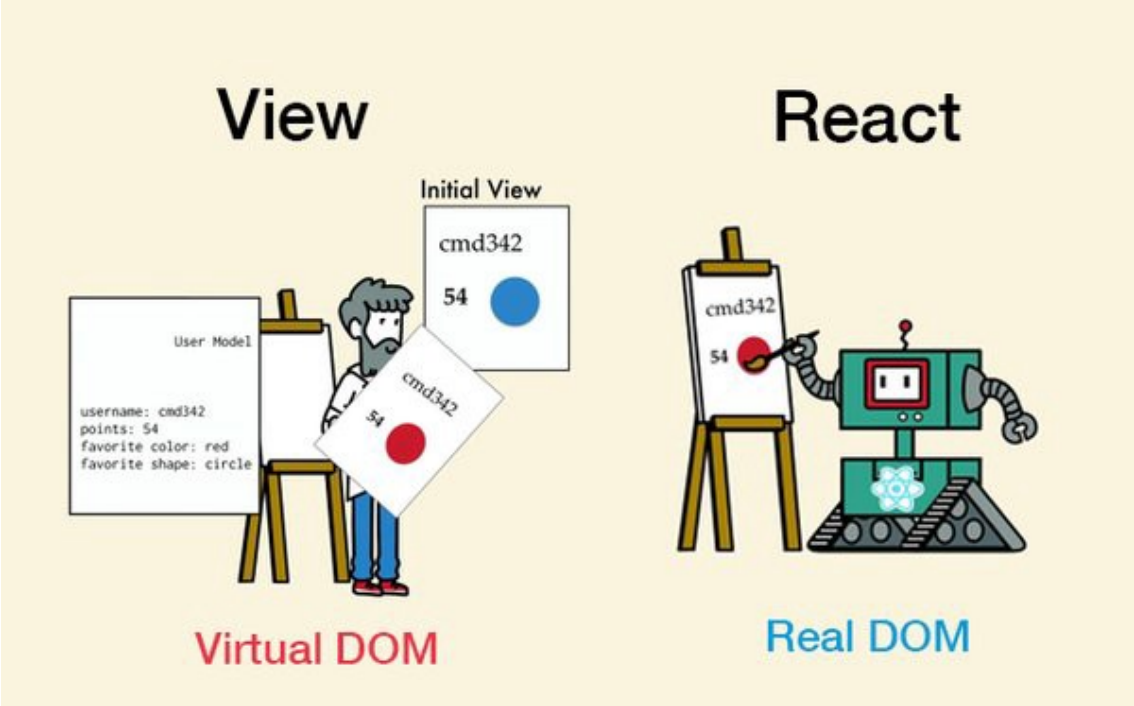

Fig 6. React virtual dom concept. Reprinted from What is React and why should we use it [2].

Instead of using regular javascript for templating, React uses JSX (Javascript and XML). Even though it's not mandatory, JSX allows developers to use markup and logic of the webpage in a single file usually called components. HTML syntaxes are processed into JavaScript calls.

Unidirectional data flow being one of the strength of React, properties are passed from parent to child components, which allows actions to be fired to the view, which in-turn updates the states. A state is the place where data comes to the components. One way binding is lesser error prone because developers have more control over the data. Knowing the flow of data, makes debugging extremely easier and efficient in react. Various lifecycle methods are used in react, render being one of the essential, component updates is much easier. [9]

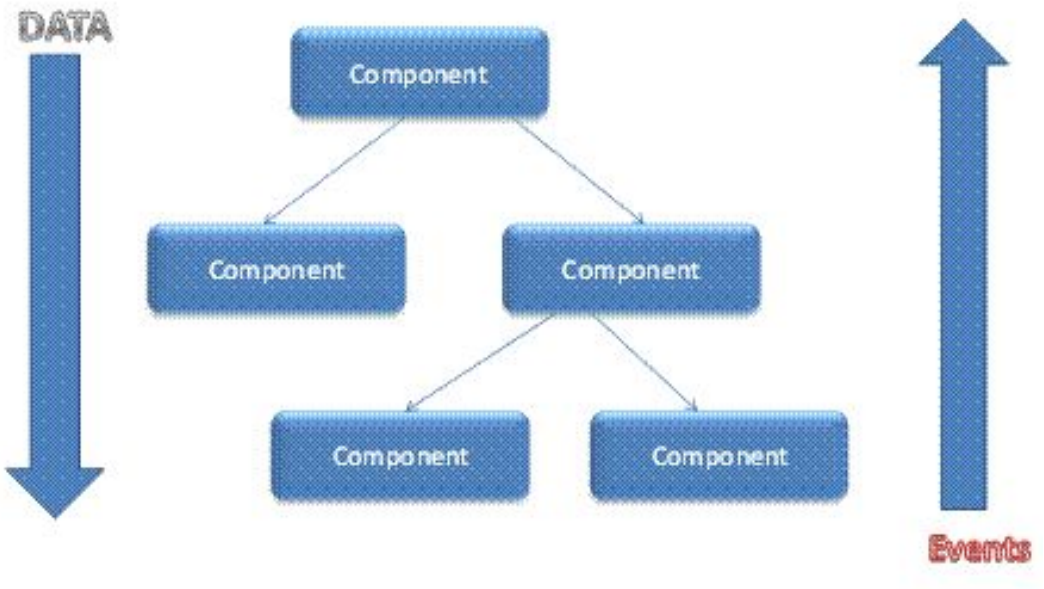

Fig 7. React components unidirectional data flow. Reprinted from What is React and why should we use it [2].

Components are the building blocks of any React app and they can be functional and class-based, with or without states. Functional components are basic javascript functions that usually do not hold any state in them while stateful components can hold states and pass on properties to the child-components. Class components can have lifecycle methods hence they can make changes to the UI and the data.

4. Tools, third party plugins and libraries

Multiple third party tools and plugins are used for the completion of the application. Even though MERN is a self sufficient ecosystem, third-party tools and plugins help a lot in providing the styling, feasibility and the writing of codes.

4.1 Visual Studio Code (VS Code)

VS Code is a popular source code editor by Microsoft. It can be used with many languages but since it is based on Electron, one of the Node.js frameworks, MERN development is easy and efficient. It is free for developers use and is open source under MIT license. Built-in terminal support makes it much easier for web-development.

4.2 JSON Web Tokens (JWT)

JWT is a JSON web object which helps creating a safe data communication between two parties which in application's context are the application's server and the end-user. The token is composed of a header, a payload and a signature. Header provides information how signature is to be computed. Payload is the data stored in the token, like the user information. The payload is encrypted or hashed to produce a signature.[13] It is used for the authentication of users in the application. Below is the way JWT attaches with the application to provide security and privacy to the contents.

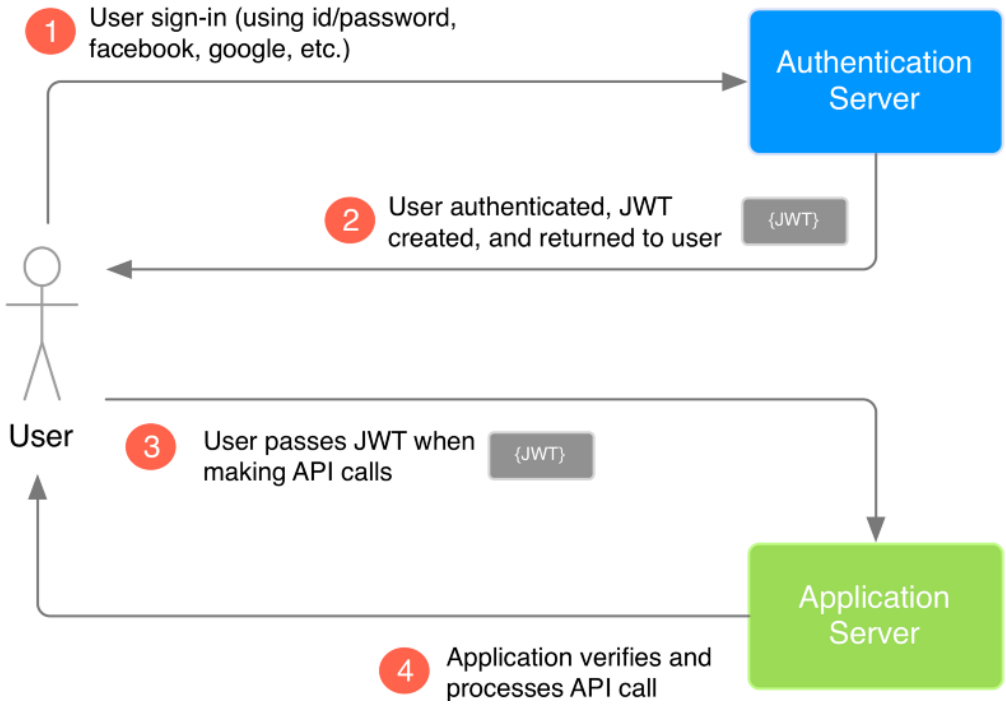

Fig 8. JWT to access the application server. Reprinted from Vandium Software [13].

4.3 Redux

Since, the data is stored as states among components in React, handling big scale application data becomes difficult. So to tackle that ambiguity, Redux was made. Redux manages the states in JavaScript applications. Even though React has brought their own state management tools in the form of hooks, Redux being old practise and stable one at that, this tool was chosen for this project. It can also be used for other javascript frameworks and libraries, like angular and vue. It is itself extremely lightweight at memory size 2KB with all the dependencies, so it doesn't quite increase the total applications asset size. It was inspired by Facebook's Flux and Elm to simplify the MVC

structure, specifically the Model and View relationship when the application size is scaled higher. It is modelled on functional programming concept.

Redux allows us to store all the states in one store and components can fetch them when required. Like other frameworks, React comes with internal state management system as well. It is not fully stable during the creating of this project as well as it is also good at only handling smaller scale application with smaller amount of states across smaller components. With larger applications state management is better with Redux.

When an app has components that share states or data, a separate handler or store for state to live in would be a great idea. Hence Redux implements that logic in providing states among components without linearly depending on the hierarchy. Generally, state has to live in the parent component in React for children to fetch, so ideally sibling components can't access the states among them. From parents to child it has to be passed on as props while Redux allows direct access to the central store, from where you can fetch all the current states.

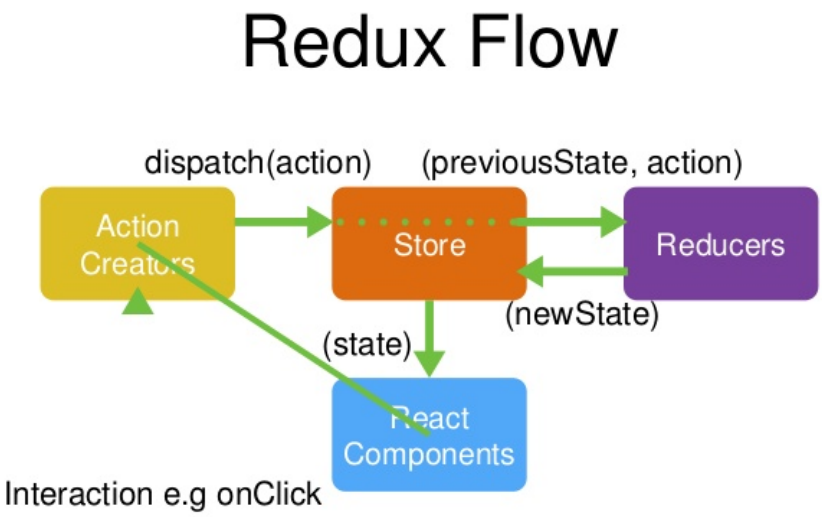

Fig 9. Redux state management. Reprinted from Vandium Software [13].

Redux has a store, reducers and actions which work together to provide states to the components. When a user creates an event, actions are dispatched to the central store as the only information that store receives from the user, that makes reducers fire a method on the previous state depending upon the action received from the store that decides what will happen to the old state. Below are brief descriptions of each Actions, Reducers and the Store.

4.3.1 Actions

Actions are events like user interactions, API calls and form submission that send data to the store. These are the only source of communication to the store but actions only define what is gonna be the new state/object, not how. Actions are plain JavaScript objects sent using store.dispatch() Actions must contain a type variable and a payload variable. Type property that defines the type of action that is to be carried out and a payload that is the actual information that is to be passed to the store. [5] Actions are created using an action creator. Below is an example of getVacs action with the action type GET_VACS imported from ./types.

```
import {
    GET_VACS,
    ADD_VAC,
    DELETE_VAC,
    VACS_LOADING
  } from '../actions/types';

  export const getVacs = () => dispatch => {
    dispatch(setVacsLoading());
    axios
      .get('/api/vacs')
      .then(res =>
        dispatch({
          type: GET_VACS,
          payload: res.data
        })
      ).catch(err =>
        dispatch(returnErrors(err.response.data, err.response.status))
      );

  };
```

Fig 10. Screenshot of the application action.

4.3.2 Reducers in Redux

Reducers govern how the state of an application is changed upon an action is dispatched containing a payload. All the application state is stored in Redux as a single object. Reducers takes an action and decide what method is to be fired depending upon the action that was dispatched. Hence a new state is produced every time.

Just like array.reduce in javascript, reducers take one object and one call back and return one object or array at the end, hence its called reducer. Same object or state passed through reducers callback yields the same new state every time because no additional callback is applied since they are usually passed as action conditions. Instead of mutating the old state a new state is created by the reducers plain functions.

The following figure is one of the reducers (vacReducer.js) used in the vacancy announcement application. It takes a state, applies action and returns a new state. Important action types are imported again from ../actions/types and many action cases are applied to produce different result for different actions.

```
import {
    GET_VACS,
    ADD_VAC,
    DELETE_VAC,
    VACS_LOADING
  } from '../actions/types';
const initialState = {
    vacs: [],
    loading: false
}



  export default function(state = initialState, action) {
    switch (action.type) {
      case GET_VACS:
        return {
          ...state,
          vacs: action.payload,
          loading: false
        };
      case DELETE_VAC:
      return {
          ...state,
          vacs: state.vacs.filter(vacs => vacs._id !== action.payload)
      };
      case ADD_VAC:
      return {
          ...state,
          vacs: [action.payload, ...state.vacs ]
      };
      case VACS_LOADING:
      return {
        ...state,
        loading: true
      };
      default:
        return state;
    }
  }
```

Fig 11. Screenshot of the application reducer.

4.3.3 Redux Store

Store in redux is the central container that holds the application state. It also behaves as the actions and reducers gateway.[5] One react application always has one store but might have multiple actions and reducers depending upon the number of components and events available. Developers can access and store state, and register or unregister event listeners with the use of helper methods. Reducers produce new state which is stored in a separate file named store which makes it available all across the application. Following is the Store implemented for the Vacancy Announcement application using syntax createStore.

```
import { createStore, applyMiddleware, compose } from 'redux';
import thunk from 'redux-thunk';
import rootReducer from './reducers';

const initialState = {};

const middleWare = [thunk];

const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;
const store = createStore(
    rootReducer,
    initialState,
    composeEnhancers(applyMiddleware(...middleWare))
  );

  export default store;
```

Fig 12. Screenshot of the application store.

4.4 React-strap

React-strap is a javascript library containing structural React Bootstrap components that handle application's composition and control.[10] It works mostly with the styling of the front-end also providing various advanced UI features. React-strap along with React-transition-group was used in the application to create headers, navigation bars and for generally giving the media-query adaptability and styling. Following elements were imported and used in the nav-bar component of the application.

```
import {
  Collapse,
  Navbar,
  NavbarToggler,
  NavbarBrand,
  Nav,
  NavItem,

  Container
} from 'reactstrap';
```

4.5 Axios

Axios is a famous JS library that allows performing HTTP requests in both computers via NodeJS and directly on browsers. It is supported by majority of browsers and is a better alternative for reacts own Fetch API. Axios is promise based library which enables performing asynchronous XHR requests.

Axios over native Fetch API comes with a lot of perks, some are listed below

1. Axios supports older browsers without the need of polyfill unlike fetch.

2. It has a way to abort a request as well set response timeout

3. Has a in-built CSRF protection and includes upload progress

4. Allows JSON data transformation and most importantly works with node

5. Vacancy Announcement Application (VacApp)

VacApp is a web application project designed using MERN stack. It targets mostly the small businesses in helping them find the right employee whilst helps unemployed users score a job-interview. VacApp is designed to benefit both sides of the employer and employee spectrum. For everyday user, searching for jobs, VacApp can be an easy medium to fetch the most suitable. The usability is efficiently designed to manage any vacancy posts for employers as well. Below is a general flow diagram on how the application works.

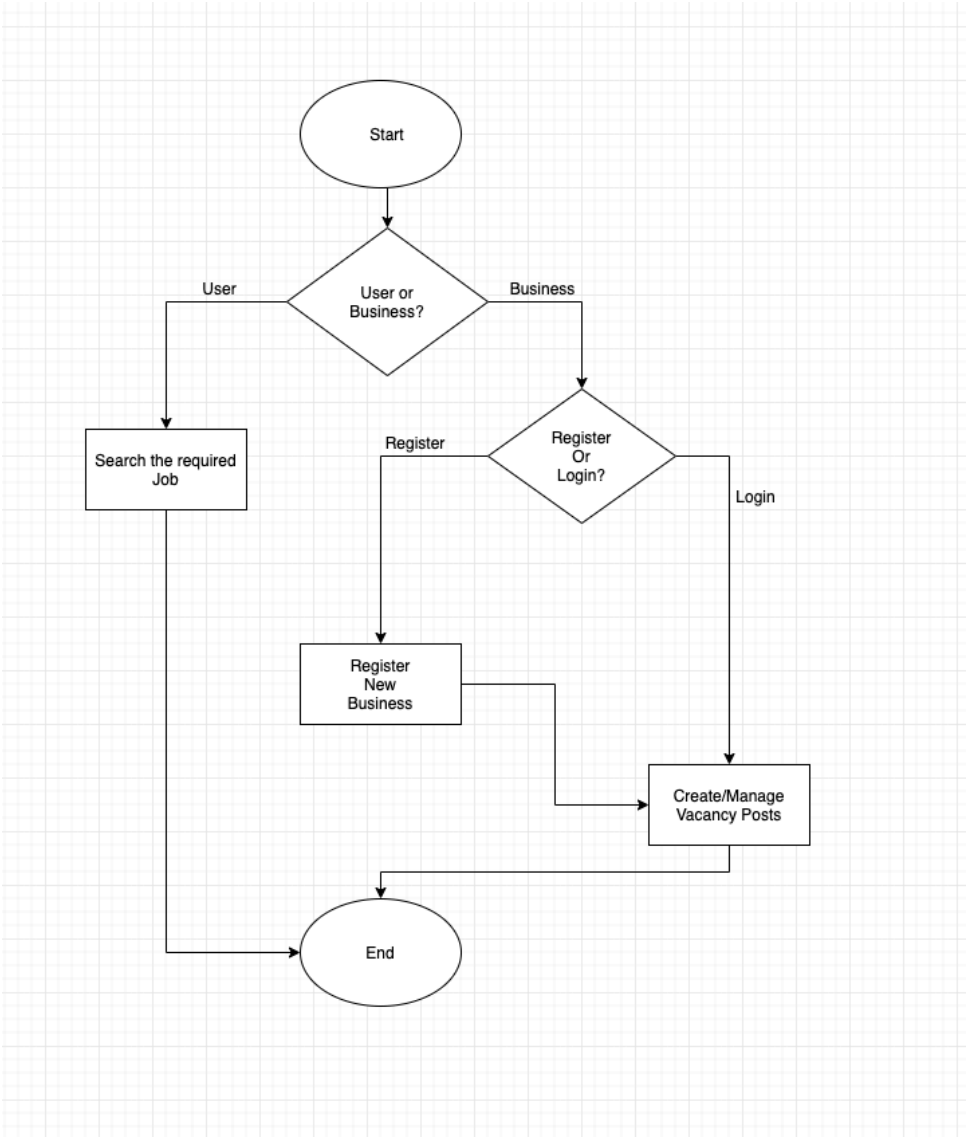

Fig 13. Flow diagram of the VacApp application.

5.1 User Interface

VacApp, under production mode, starts with the fairly simple home page. Home page allows users searching jobs to hit the job name and city to find the right job closest to them. Home page also features navigation bar which holds the login or register buttons based on the current status of business users. All the components in the application are designed in-order them to fit in the size of the end-user screen. Hence, the navigation bar is retractable on the phone screens while fully featured when accessed through a computer. Although all the contents are mostly rendered conditionally from a single main component VacancyApp.js, in application separate pages are redirected depending on the authentication status. Below is the figure of the applications home-screen with and without the search criteria filled and a snippet of code that renders it respectively.

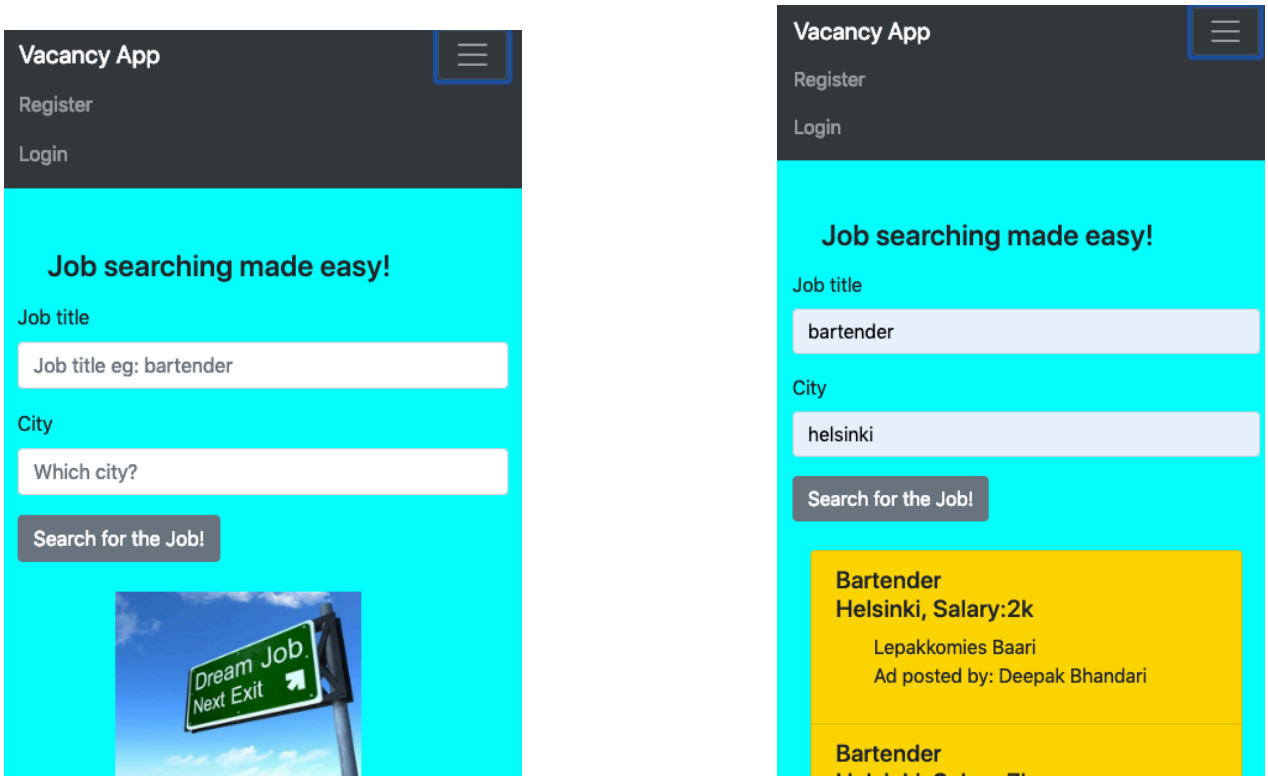

Fig 14. Screenshots of the application's home screen before and after search values entered.

```
<Form onSubmit={this.sermit}>
          <FormGroup>
            <Label for="title">Job title</Label>
            <Input
              type="text"
              onChange={this.oncser}
              name="serTitl"
              className="inp"
              placeholder="Job title eg: bartender"
            />
          </FormGroup>
          <FormGroup>
            <Label for="city">City</Label>
            <Input
              type="text"
              onChange={this.oncser}
              name="serCit"
              className="inp"
              placeholder="Which city?"
            />
          </FormGroup>
          <Button>Search for the Job!</Button>
        </Form>
        <br />


        {this.state.isSubmitted && renderSe}
```

General job searching users can get all their work done on the main screen. Since the application is designed to be extremely simple and easy to work with, general users do not necessarily have to login. They can just fetch their necessities and close the application, nobody is searching for a job all the time.

On the other hand, primary consumers of the application, employers, generally businesses can register new business or login to the user they have already created from the navigation bar. Separate react-strap modal elements are provided for register and login and they float above the home-screen. Below is the figure depicting login modal and a snippet of the modal's code.

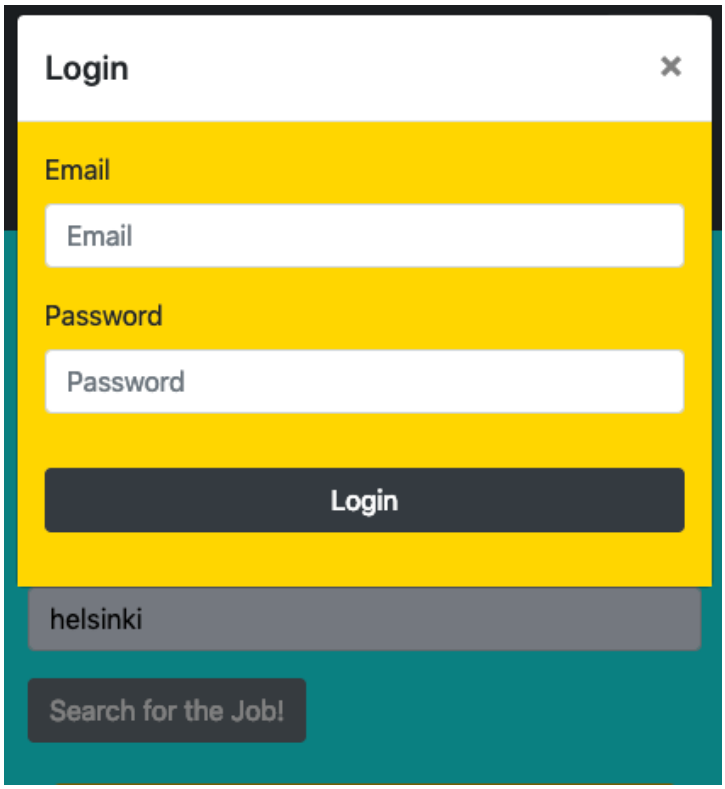

Fig 15. Screenshot of the application's login modal.

```
toggle()=> {
    // Clear errors
    this.props.clearErrors();
```

```
    this.setState({
      modal: !this.state.modal
    });
  };

  onChange = e => {
    this.setState({ [e.target.name]: e.target.value });
  };

  onSubmit = e => {
    e.preventDefault();

    const { email, password } = this.state;

    const user = {
      email,
      password
    };

    // Attempt to login
    this.props.login(user);
  };

  render() {
    return (
      <div>
        <NavLink onClick={this.toggle} href='#'>
          Login
        </NavLink>

        <Modal isOpen={this.state.modal} toggle={this.tog-
gle}>
          <ModalHeader toggle={this.toggle}>Login</Modal-
Header>
          <ModalBody className="lis">
            {this.state.msg ? (
              <Alert color='danger'>{this.state.msg}</
Alert>
            ) : null}
```

```
            <Form onSubmit={this.onSubmit}>
              <FormGroup>
                <Label for='email'>Email</Label>
                <Input
                  type='email'
                  name='email'
                  id='email'
                  placeholder='Email'
                  className='mb-3'
                  onChange={this.onChange}
                />

                <Label for='password'>Password</Label>
                <Input
                  type='password'
                  name='password'
                  id='password'
                  placeholder='Password'
                  className='mb-3'
                  onChange={this.onChange}
                />
                <Button color='dark' style={{ marginTop:
'2rem' }} block>
                  Login
                </Button>
              </FormGroup>
            </Form>
          </ModalBody>
        </Modal>

      </div>
    );
  }
```

When the business user registers and logs in using the created credentials, user-feed is redirected. The feed is empty for new registration while it will contain all the vacancy

posts the user will ever create using the app. User-feed is where the users can create, update and delete the existing vacancy posts.

Vacancy posts are the blocks of information provided by the business user which can be searched by the job seeking consumers. Vacancy posts are unique and private to the business user in the sense only they can update or delete the information. Vacancy posts are listed in the home screen if the user search criteria meets the post's information. Modals are again executed while creating the vacancy posts. Business user clicks on the vibrant Add Vacancy button provided on the top-left corner of the application to create a post. A modal is launched to enter all the vacancy information for example title, city and the businesses contact information. Below is the figure of the app's User-feed.

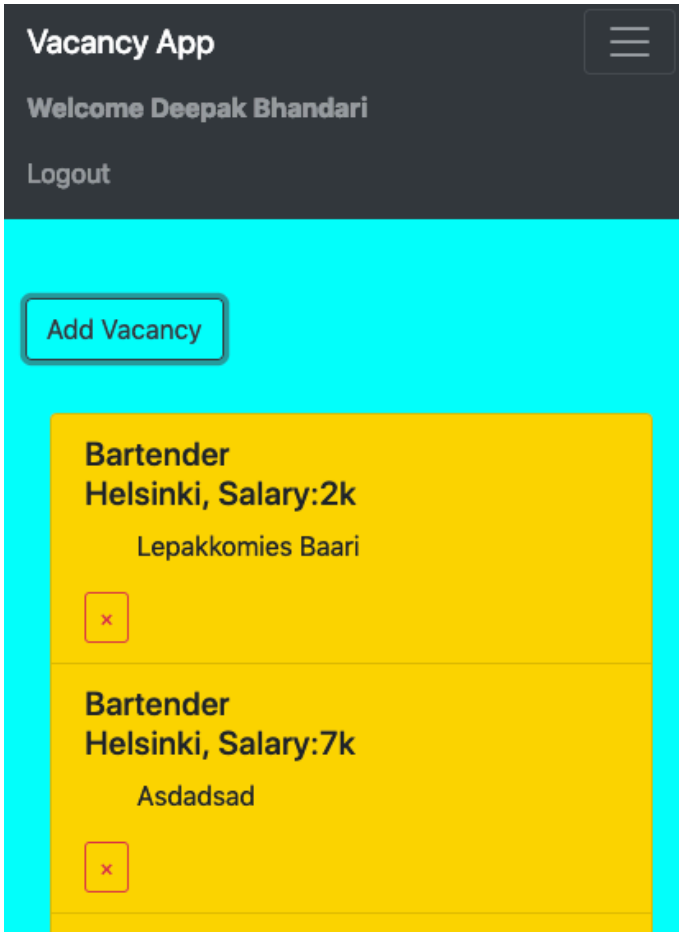

Fig 16. Screenshot of the application's User-feed.

User feed is rendered conditionally depending on the authentication status. Option to delete any post is only provided if the user is authenticated, posts on home-screen do not have that luxury.

```
if (this.props.isAuthenticated ) {
      return (
        <Container>
      <Fragment>
          <ListGroup>
            <TransitionGroup className="vac-app">
              {vacs.map(({ _id, title, city, sal, desc,
uid }) => {
                return uid === user.name ? (
                  <CSSTransition key={_id} timeout={500}
classNames="fade">
                    <ListGroupItem className="lis">
                      <ListGroupItemHeading>
                        {title} <br /> {city + ","}
{"Salary:"}
                        {sal}{" "}
                      </ListGroupItemHeading>
                <ListGroupItemText className="list-text">
                        {" "}
                        {desc}{" "}
                      </ListGroupItemText>
                      {this.props.isAuthenticated ? (
                        <Button
                          className="remove-btn"
                          outline
                          color="danger"
                          size="sm"
                          onClick={this.onDeleteClick-
.bind(this, _id)}
                        >
                          &times;
                        </Button>
                      ) : null}
                    </ListGroupItem>
                  </CSSTransition>
```

```
            ) : null;
          })}
        </TransitionGroup>
      </ListGroup>
    </Fragment>
  </Container>
);
```

Add Vacancy fires a react-strap modal which has text fields to enter the post's information. Add vacancy modal asks business users to hit the accurate information regarding the job since seeking costumers are also watching the same vacancy posts. Below is a screenshot of the add-vacancy modal and a block of code to render it.

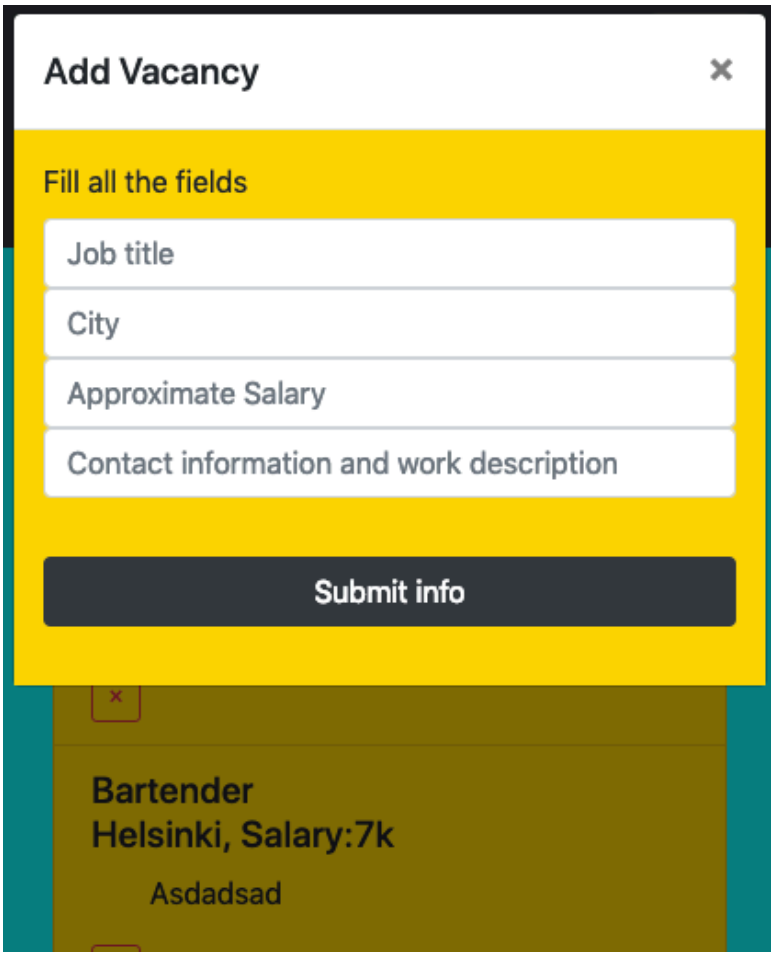

Fig 17. Screenshot of the VacApp's after-authentication Add Vacancy React-strap modal.

```
render() {

    return(
      <div>

        {this.props.isAuthenticated ? (
        <Button
        color='dark'
        outline
        style={{ marginBottom: '2rem' }}
        onClick={this.toggle}>

        Add Vacancy</Button>
        ) : (
          <h4 className='mb-3 ml-4'>Job searching made
easy!</h4>
        )}

        <Modal isOpen={this.state.modal} toggle={this.tog-
gle}>
          <ModalHeader toggle={this.toggle}>Add Vacancy</
ModalHeader>
          <ModalBody className="lis">
            <Form onSubmit={this.onSubmit}>
              <FormGroup>
                <Label for='vac' key="id">Fill all the
fields</Label>
                <Input
                  type='text'
                  name='title'
                  placeholder='Job title'
                  onChange={this.onChange}
                />
                <Input
                  type='text'
                  name= 'city'
                  placeholder='City'
                  onChange={this.onChange}
                />
```

```
                <Input
                  type='text'
                  name= ‘sal'
placeholder='Approximate Salary'
                  onChange={this.onChange}
                />
                <Input
                  type='text'
                  name= 'desc'
                  placeholder='Contact information and
work description'
                  onChange={this.onChange}
                />
                <Button color='dark' style={{ marginTop:
'2rem' }} block>
                  Submit info
                </Button>
```

5.2 Testing and deployment

Application’s API end-points are tested using postman. Postman is an API development environment that works around complete software development cycle, designing and mock, testing, documentation and monitoring. It enables to test the same request against different environments with environment specific variables. Register and login endpoints were tested and verified before starting the user interface development. Login endpoints required authentication and register endpoints were tested without authenticating, just by asserting GET/POST requests to /api/users/login and /api/users/register respectively.

The application was finally deployed to heroku using heroku login in terminal.

The heroku deployed application can be accessed by browsing

www.tinyurl.com/DeepVac

6. Discussion

This thesis study was carried out to produce a suitable vacancy application however walking through the learning curve of Mongo, Express, React and Node was the bigger challenge. Web-development is not in itself a vast topic to learn, simpler web-pages can be created by merely touching the depth of javascript. But to be able to find just the right libraries and adapt efficient functionalities is the tougher path to walk. Javascript runtime to load the server and javascript enriched user-interface makes it easier for developers to make changes around the code while keeping the usability of code blocks at maximum.

It took around 3 months to complete the beta working VacApp application. It took a confusing yet fruitful 2 months just to learn the basics of the MERN. This project is not at its peak. More ideas like google or facebook login, photos and shareable posts, and actual business confirmation could still be stitch into it. There are already bigger more popular already existing same purpose apps, but they are usually hefty and process taking. VacApp on the other hand would be 3 clicks go app, if a user is searching for a job. If a business, then they would login to their accounts and create a simple vacancy announcement post. Consumers crave for simplicity, big social medias are improving accordingly.

The application still can produce a meaningful vacancy post and could be launched in my city. More improvements will be made and earlier mentioned features will be added soon. This thesis could be a great start for someone trying to learn the MERN stack. It provides the list of detailed tools and ideas to create a one-paged full-frame react web-page.

React being the user-interface, rendering becomes extremely fast. It is made by Face-book and also provides mobile solutions with react native, so it is extremely appre-ciable for the attraction it has created among big companies. The ability to change parts of a webpage with components is simply amazing. Rendering just a part reduces the amount of refreshing the page and deletes the compulsion of jumping over hyper-linked pages.

7. Conclusion

The goal of this thesis was to create a suitable vacancy announcement application using the widely popular MERN stack. Hence, this thesis discusses the application and contents of MERN stack and ideas on using it to create a suitable web-app project. Concepts of Redux and other plugins to make a web-page fast yet attractive are discussed all around the thesis.

In the light of JavaScript development and right when they made it work outside of browsers, JavaScript front end development had to take its pace, developers were given with frameworks after frameworks, Angular, VueJs, React etc. With the ability of working front and back-end solely with JavaScript, MERN stack was authors choice of stack. So the thesis is a documentation of all the MERN development. It opens up opportunities for anyone wondering about the stack. It explains on creating APIs using Express on NodeJS framework. Briefly defines the attachment of MongoDB and its benefits.

The application is completed as much the project proposal had described. With further development, author plans on launching the application in his city with the co-operation of the district government. Application will help a great deed in people either searching for job or businesses searching for the right employee.

References

1. What is ReactJS and Why should we use it? [Internet]. c-sharpcorner.com 2018 [cited 14 November 2018]. Available from: https://www.c-sharpcorner.com/article/what-and-why-reactjs/
2. The modern application stack [Internet] Mongodb Official Website 2017 [cited 26-January 2017] Available at: https://www.mongodb.com/blog/post/the-modern-application-stack-part-1-introducing-the-mean-stack
3. React.js: a better introduction to the most powerful UI library ever created. [Internet] hackernoon.com 2018 [cited 3 September 2018]. Available from: https://hackernoon.com/react-js-a-better-introduction-to-the-most-powerful-ui-library-ever-created-ecd96e8f4621
4. A beginners guide to Redux [Internet] gistia software engineering 2017 [cited 4 September 2017] Available from: https://www.gistia.com/beginners-guide-redux/
5. “Hello World” in React-Redux [Internet] medium.com 2018 [cited 4 April 2018] Available at: https://medium.com/@lavitr01051977/easy-redux-b29391b499cb
6. Build a Login/Auth App with the MERN stack [Internet] Bits and Pieces 2018 [cited 21 November 2018] URL: https://blog.bitsrc.io/build-a-login-auth-app-with-mern-stack-part-1-c405048e3669
7. What exactly is Node.js? [Internet] freeCodeCamp 2018 [cited 18 April 2018] URL: https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5
8. Components and Props [Internet] reactjs.org Official Website URL: https://reactjs.org/docs/components-and-props.html
9. MERN Stack Front To Back: Full Stack React, Redux & Node.js [Internet] Udemy Brad Traversy 2019 [last updated April 2019] Available at: https://www.udemy.com/mern-stack-front-to-back/
10. React-strap [Internet] Official Website Available at: https://reactstrap.github.io/components/listgroup/#
11. Basic routing [Internet] Express JS Official Website 2018 URL: https://expressjs.com/en/starter/basic-routing.html
12. ReactJS-JSX [Internet] TutorialsPoint 2019 URL: https://www.tutorialspoint.com/reactjs/reactjs_jsx.htm
13. 5 Easy Steps to Understanding JSON Web Tokens (JWT) [Internet] Vandium Software [cited 16 May 2016] URL: https://medium.com/vandium-software/5-easy-steps-to-understanding-json-web-tokens-jwt-1164c0adfcec