**1. INTRODUCTION**

**1.1 Necessity of a Technology-based Learning System**

**Online learning** refers to an internet-based learning environment that can connect students of diverse backgrounds who boast different perspectives. A higher education institution will use a learning management system, or LMS, to facilitate online learning. A learning management system is a software application for the administration, documentation, tracking, reporting, automation, and delivery of educational courses, training programs, or learning and development programs.

In today’s technological era, there has been an exponential increase in demand for well-trained employees in the software and technology industries, which have roots in almost all other industries of the modern times. However, in our country, there has been a decrease in the quality of the employees that graduate college every year, as there has been a proportionate increase in the quantity of said employees, mainly due to the neglect of the importance of a foundation of essential knowledge during the days of education. A study notes that while 60% of the candidates cannot write code that compiles, only 1.4% can write functionally correct and efficient code.

This project aims to tackle this problem by encouraging its users, be it students or graduates alike, to go from having no knowledge in any domains, to full-fledged Software Developers having knowledge of various domains of today’s IT industry. This overall contributes to reducing the Employability gap in the country

Fig. 1.1: Graph of Engineering Fresher’s Coding Knowledge

**1.2 Advantages of a Learning System Website**

* **Organizes eLearning content in one location**

Instead of having learning content spread out over different devices and websites, users can store all of their learning sources in one location. This reduces the risk of losing essential data and makes it easier to learn more efficiently.

* **Easily tracks learner progress and performance**

A Learning System website gives its users the ability to keep track of their learning progress with relevant statistics and helps them ensure that they are meeting their performance milestones. Most Learning Management Systems feature reporting and analytical tools that allows its learners to pinpoint areas of their learning course that may be lacking, as well as where they excel.

* **Reduces Learning and Development costs**

A Learning System website gives one the power to completely do away with instructor travel costs, online training site rentals, and printed eLearning materials. An online learner can carry out all their training online, which means they can save a sizeable sum on their learning as well as the development budget.

* **Reduces Learning and Development time**

A Learning System website can also reduce online training time, owing to the fact that it gives online learners only the information they need in a direct and organized manner. Instead of having to sit through a lengthy half-hour online training course, online learners can simply click on the online modules they need and absorb the knowledge in a fraction of the time. They can also assess their understanding by taking online tests.

**1.3 Target Users for a Technology-based Learning System**

* **Students:** As students are still in the phase of education, the website encourages such students to inculcate good learning practices and essential basic foundational topics early in their learning process, as it is a beginner friendly approach
* **Graduates/Software Employees:** The website is also useful for people who completed their education but do not have the required knowledge and wish to begin learning new domains from scratch.

**2. LITERATURE SURVEY**

The base knowledge on how to use MERN stack to develop e-commerce websites was gathered. We utilized the information on the advantages and disadvantages of e-commerce. Next we referenced the overall architecture of a basic e-commerce website. We also studied the distinction of the Frontend and Backend, and what parts of the MERN stack was involved in each stage of the development of the website. [1]

In order to decide whether to make the website a full-fledged e-commerce website or not, and to understand the various types of e-commerce and m-commerce websites, the distinction between the two, and to understand the advantages, limitations and security, we studied the International Journal of Advanced Research in Computer and Communication Engineering. This journal paper mainly shed some light on the limitations and security issues of such e-commerce applications, and we weighed these limitations with the advantages of such a web application, and came to a compromise of making the website a semi e-commerce website for now, where a majority of the portal’s content is accessible for free, while only certain content such as the Consultancy will be gated with paid membership, which is for future enhancement. [2]

To make some further analysis on the decision between regular and e-commerce websites, we studied the trend of the future of e-commerce in India, and took note of how the trend has been growing in our country and how it would continue to grow in the coming years, and whether the profit would be worth the security risks that come with e-commerce websites or not. [3]

We then studied the basic components of MERN Stack technology such as: MongoDB, ExpressJS framework, ReactJS library, and NodeJS platform. Discussing the basic functions of an e-commerce web application such as sign up, sign in, showing dashboards, displaying store categories and products Using MERN Stack technology in conjunction with Braintree to build a web application to search for product stores and payment gateway.[4]

Next, we learned how to develop a full MERN stack single-page web application, the detailed process, installation requirements, and methodologies involved in making a proper web application using the MERN stack. [5]

We also looked at documentations and resources online from websites for the development of a MERN stack application, and also referenced the best practices involved. [10]

Finally, we did some case studies on similar website portals and noted the common design and implementation concepts used for the development of these similar website portals in order to use in our project. [12]

**3. SYSTEM DESIGN**

**3.1 UML Diagrams**

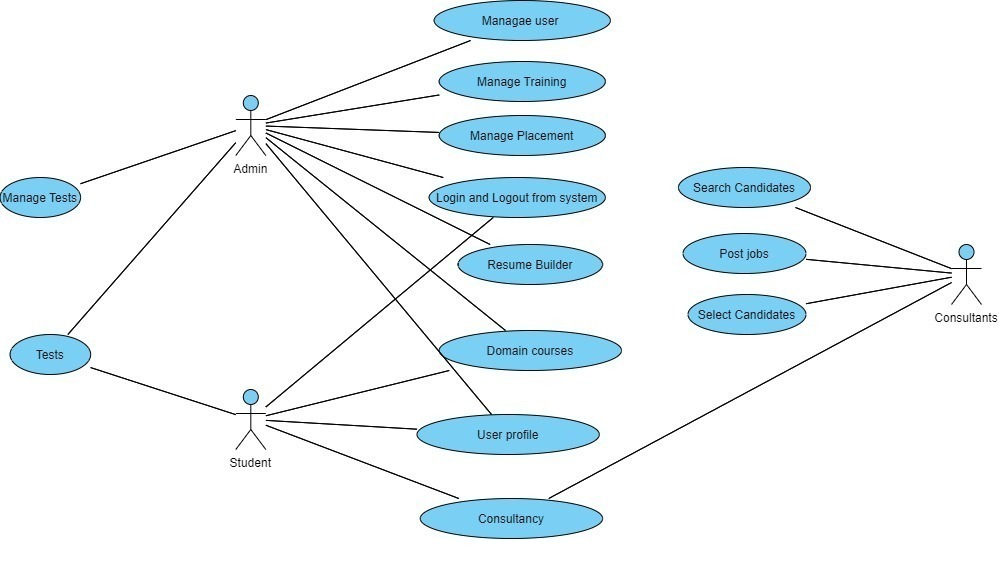
**UML** (Unified Modeling Language) is a standard language for specifying, visualizing, constructing, and documenting the artefacts of software systems.

**3.1.1 Use Case Diagram**

**Usecase diagrams** are usually referred to as behaviour diagramsusedto describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors).

This project includes the following Actors and Use Cases as illustrated in Fig. 3.1:

* Admin: A special user who can create and manage the website content
* Student: Regular user who can interact with website, but not edit
* Consultants: Skilled experts who can coach Students and provide counselling

 Fig. 3.1: Use Case diagram for Datai Web Application

**3.1.2 Architecture Diagram**

An architecture diagram is a graphical representation of a set of concepts, that are part of an architecture, including their principles, elements and components.

This project contains a special type of architecture called Model-View-Controller architecture. Each of these components are built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects. Following are the components of MVC as illustrated in Fig. 3.2:

* **Model:** The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. The MongoDB and Node frameworks in this project handles the Model
* **View:** The View component is used for all the UI logic of the application. For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with. The React framework in this project handles the View
* **Controller:** Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. For example, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The Express framework in this project handles the Model

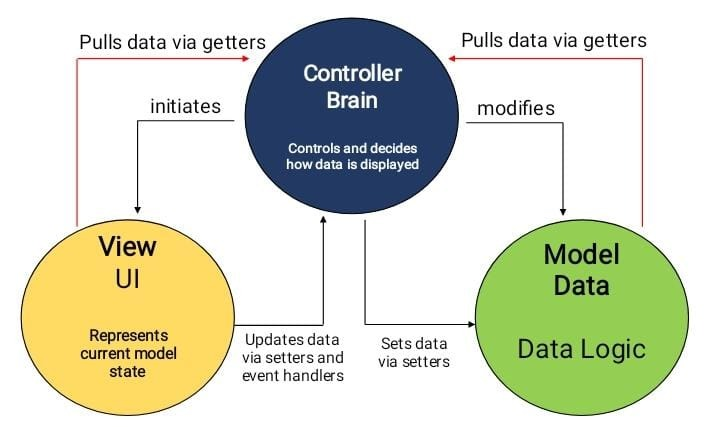


Fig. 3.2: Architecture Diagram for Datai Web Application

**3.2 Requirments**

**3.2.1 Hardware Requirements**

Processor : 32/62 Bit

Speed : 2.0 GHZ (Minimum)

Primary Memory : 8 GB RAM

Hard Disk : 1 GB (Minimum)

Chrome Browser : Latest Version

Internet Connection : 15 Mbps (Minimum)

**3.2.2 Software Requirements**

Languages Used : JavaScript, TypeScript

Platform : Windows XP/7/8/10

Tools Used : Google Chrome, Visual Studio Code

Additional Tools : Ionic, Postman API, MongoDB Atlas

**3.3 Project Planning**

Project planning is the main part of project which decides whether the project will be successful or not. Due to a limited deadline of 1 month, we split up our days into two phases. The first phase, lasting one week, started with the team learning the basics of MERN stack through online courses, in order to fully understand how to develop a web application using this particular stack, which served as a basic foundation for the entire development part of the project.

Despite its face-level complexity in architecture, the clean split up of the different modules made the planning and preparation for phase two easier. We assigned various modules to different members of the team, and began collecting additional information more specifically for the implementation of each domain, namely the Dashboard (with graphs), the Profile Page, the Assessments feature, the Resume Builder and the Leaderboard.

**What is a Learning Management System?**

A **learning management system** (**LMS**) is a software application for the administration, documentation, tracking, reporting, automation, and delivery of educational courses, training programs, or learning and development programs.The learning management system concept emerged directly from e-Learning. Although the first LMS appeared in 1924 in the higher education sector,the majority of the LMSs today focus on the corporate market. Learning management systems make up the largest segment of the learning system market. The first introduction of the LMS was in the late 1990s.Learning management systems have faced a massive growth in usage due to the emphasis on remote learning during the COVID-19 pandemic.

Learning management systems were designed to identify training and learning gaps, using analytical data and reporting. LMSs are focused on online learning delivery but support a range of uses, acting as a platform for online content, including courses, both asynchronous based and synchronous based, as illustrated in Fig 3.3. In the higher education space, an LMS may offer classroom management for instructor-led training or a flipped classroom. Modern LMSs include intelligent algorithms to make automated recommendations for courses based on a user's skill profile as well as extract metadata from learning materials to make such recommendations even more accurate.

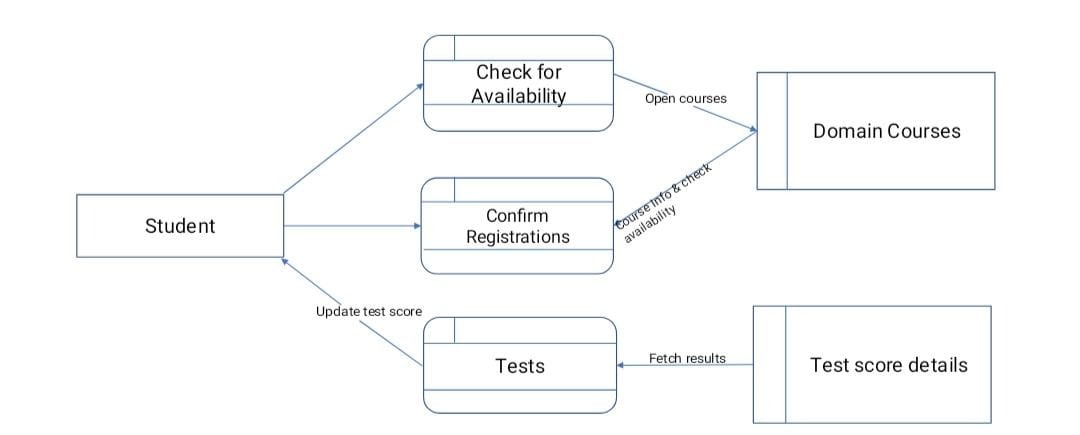


Fig. 3.3: Process Flow of a Learning Management System

**Learning Management System Development with MERN Stack**

JavaScript is a flexible scripting language that comes with a wide variety of built-in frameworks along with Node.js in order to aid development of various types of web applications.

In this project, we utilized three main frameworks, namely:

* MongoDB: Database framework that uses NoSQL to store JSON based web objects in the backend.
* Express.js: A backend web application framework that is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js
* React.js: A JavaScript library for building user interfaces based on UI components, which is used as a base in the development of single-page or mobile applications

These three libraries/frameworks served as a foundation for the web application to be built, and greatly enhanced the speed as well as the quality of the project completion

**3.4 Feasibility Study**

Feasibility studies aim to objectively and rationally uncover the strengths and weaknesses of the existing system or proposed venture. In its simplest term, the two criteria to judge feasibility are cost required and value to be attained. As such, a well-designed feasibility study should provide historical background of the project. Generally, feasibility studies precede technical development and project implementation. The assessment of feasibility study is based on the following factors:

1. Technical Feasibility
2. Operational Feasibility

**3.4.1.** **Technical Feasibility**

Generally, feasibility studies precede technical development and project implementation. The assessment is based on a system requirement in terms of Input, Processes, Output, Fields, Programs, and Procedure. This can be quantified in terms of volumes of data, trends, frequency of updating, etc., in order to estimate whether the new system will perform adequately or not.Technological feasibility is carried out to determine the capability, in terms of software, hardware, personnel and expertise, to handle the completion of the project.

**3.4.2. Operational Feasibility**

Operational feasibility is a measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development. The operational feasibility of the system can be checked as it solves the problems and reduces the complications occurring in the paper-pencil test.

**4. IMPLEMENTATION**

**INTRODUCTION**

Here, the problem formulation, the related fundamentals and method of simulation would be discussed. The main choice of Integrated Development Environment for the development and testing of the web application is Visual Studio Code (VSCode). Visual Studio Code is a source code editor made by Microsoft for Windows, Linux and macOS. Features include:

* Support for Debugging
* Syntax highlighting and intelligent code completion
* Code refactoring
* Support for various language syntax highlighting, and downloadable library of extensions.
* Version control integration: unified user interface for Git

Other alternatives for Visual Studio Code also include popular text editors such as Atom, Brackets, Notepad++, etc.

The platform for the testing and implementation of the server API was Postman. Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so that one can create better APIs at a faster pace.

Features of Postman include:

* Postman API which allows users to programmatically access data stored in a Postman account and perform all the basic CRUD operations on collections, environments, mocks and more.
* Postman Echo, which is a serve that allows the testing of REST clients and making sample API calls.
* Provides a programmable way to visually represent requests and responses using HTML, CSS and JavaScript
* Suite of external libraries available to use in the pre-request and test script tabs.
* Workflow Control
* Collaboration feature which allows commenting, forking, branching, pull requests, tagging, etc.
* Monitors for Regression Testing

**4.1 JavaScript**

JavaScript, often abbreviated as JS, is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS. Over 97% of websites use JavaScript on the client side for web page behaviour, often incorporating third-party libraries. All major web browsers have a dedicated JavaScript engine to execute JavaScript on users' devices.

JavaScript is a high-level, often just-in-time compiled language that conforms to the ECMAScript standard. It has dynamic typing, prototype-based object-orientation, and first-class functions. It is multi-paradigm, supporting event-driven, functional, and imperative programming styles. It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM).

In this project, the various libraries used are explained clearly. The JavaScript version used is ES2015, which is the first major upgrade to JavaScript since 1997. The libraries used are Node.js, React, Express and MongoDB, which altogether constitute the MERN stack, which is widely used for developing single-page and mobile applications.

**4.2 Node.js**

Node.js is an open-source, cross-platform back-end JavaScript runtime environment that runs on the B8 engine and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting- running scripts on the server-side to produce dynamic web page content before the page is sent to the user’s web browser. Consequently, Node.js represents a “JavaScript everywhere” paradigm, unifying web-application development around a single programming language, rather than different languages for server-side and client-side scripts.

Node.js has an event driven architecture capable of asynchronous I/O. These design choices aim to optimize throughput and scalability in web applications with many I/O operations, as well as real-time web applications.

**4.3 React.js**

React is a free and open source front—end JavaScript library for building user interfaces based on UI components. It is maintained by Meta (formerly Facebook) and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications. However, React is only concerned with state management and rendering that state to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

**4.4 Express.js**

Express is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications. It facilitates the rapid development of Node based Web applications. Following are some of the core features of Express framework −

* Allows to set up middleware to respond to HTTP Requests.
* Defines a routing table which is used to perform different actions based on HTTP Method and URL.
* Allows to dynamically render HTML Pages based on passing arguments to templates.

**Installation of Express.js**

The following line of code can be downloaded using either the Node terminal or the command line client:

$ npm install express –save

The above command saves the installation locally in the **node\_modules** directory and creates a directory express inside node\_modules. You should install the following important modules along with express –

* **body-parser** − This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data.
* **cookie-parser** − Parse Cookie header and populate req.cookies with an object keyed by the cookie names.

$ npm install body-parser --save

$ npm install cookie-parser --save

$ npm install multer –save

**4.5 MongoDB**

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. Its main features include:

* Ad-hoc queries
* Indexing
* Replication
* Load balancing
* File storage
* Aggregation
* Server-side JavaScript Execution
* Capped collections
* Transactions

**4.6 JSON**

JavaScript Object Notation (JSON, pronounced /ˈdʒeɪsən/; also /ˈdʒeɪˌsɒn/[note 1]) is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value). It is a very common data format, with a diverse range of applications, such as serving as replacement for XML in AJAX systems

JSON is a language-independent data format. It was derived from JavaScript, but many modern programming languages include code to generate and parse JSON-format data. The official Internet media type for JSON is application/json. JSON filenames use the extension .json.

JSON's basic data types are:

* Number: a signed decimal number that may contain a fractional part and may use exponential E notation, but cannot include non-numbers such as NaN. The format makes no distinction between integer and floating-point. JavaScript uses a double-precision floating-point format for all its numeric values, but other languages implementing JSON may encode numbers differently.
* String: a sequence of zero or more Unicode characters. Strings are delimited with double-quotation marks and support a backslash escaping syntax.
* Boolean: either of the values true or false
* Array: an ordered list of zero or more values, each of which may be of any type. Arrays use square bracket notation with comma-separated elements.
* Object: an unordered collection of name–value pairs where the names (also called keys) are strings. Objects are intended to represent associative arrays, where each key is unique within an object. Objects are delimited with curly brackets and use commas to separate each pair, while within each pair the colon ':' character separates the key or name from its value.
* null: An empty value, using the word null

Whitespace is allowed and ignored around or between syntactic elements (values and punctuation, but not within a string value). Four specific characters are considered whitespace for this purpose: space, horizontal tab, line feed, and carriage return. In particular, the byte order mark must not be generated by a conforming implementation (though it may be accepted when parsing JSON). JSON does not provide syntax for comments.

Early versions of JSON (such as specified by RFC 4627) required that a valid JSON text must consist of only an object or an array type, which could contain other types within them.

Example showing a possible JSON representation describing a person :

"firstName": "John",

"lastName": "Smith",

"isAlive": true,

"age": 27,

"address": {

"streetAddress": "21 2nd Street",

"city": "New York",

"state": "NY",

"postalCode": "10021-3100"

},

"phoneNumbers": [

{

"type": "home",

"number": "212 555-1234"

},

{

"type": "office",

"number": "646 555-4567"

},

{

"type": "mobile",

"number": "123 456-7890"

}

"children": [],

"spouse": null

**4.7 Git**

Git is a software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity and support for distributed, non-linear workflows.

**4.8 Modules**

Following are the modules used in this project:

**4.8.1 Login/Registration**

The Registration and Login pages are the first pages that the user interacts with upon first visiting the website. The phases of development for the Login and Registration pages are:

* UI development
* Creating a User Schema and connecting to the Database
* Creation of API links, and adding Authentication actions and reducers
* URL Authentication protection and Navigation
* Form Validation
* Login and Logout functionality

**4.8.2 Dashboard**

The Dashboard is the main page for the user after logging in. If the user is already previously logged in, and the session is still active, then the user is directly taken to the Dashboard after loading the website. The phases of development for the Dashboard page are:

* UI development
* Creating a Profile Schema and connecting to the database
* Setting up API and creating actions and reducers to bring in user specific information to the Dashboard
* Creating a Domain Schema and connecting to the database
* Setting up API and creating actions and reducers to display domain information on the Dashboard.
* Creating a User-Test Schema and connecting to the database
* Setting up API and creating User-Test actions and reducers to store user specific test scores
* Score statistics graph UI on the dashboard, and fetching User-Test data to display in graphs.
* Navigation routes to and from Dashboard

**4.8.3 Profile Page**

The Profile page consists of the user’s basic information, as well as additional information regarding selected domains, skills, prior education and experience. The user can add more domain, skill, education and experience fields at any time. Following are the features that were implemented on the Profile page:

* UI development
* Using Profile schema actions to GET and POST Profile data from the Profile Page, and display correct profile information
* Create Profile feature after initial registration, after which the feature is changed to View Profile
* Options to Add and Edit the Domains, Skills, Education and Experience fields of the profile.
* Navigation routes to and from Profile page.

**4.8.4 Assessments**

The Assessments module consists of Assessments or Tests that belong to a specific domain. Each domain can consist of multiple assessments that can be taken at any stage by the user. The scores from each assessment are calculated upon submission and then stored in the User-Test Schema. Following are the features that were implemented on the Assessments page:

* UI development
* Passing of test information (Test Name and Domain ID) in the URL and fetching the correct test using this information and the Domains API and actions.
* Proper display of fetched Assessment questions as well as answer keys.
* Assessment score calculation upon submit, and creation of User-Test object
* Passing User-Test object to User-Test Schema action and storing the information to the database.
* Navigation routes to and from Assessment Page.

**4.8.5 Resume Builder**

The Resume Builder is a feature that compiles the information entered in the profile page, formats the information using Xelatex, and converting this to a PDF using a specific Python script. The PDF File is then downloaded to the user’s local system via the browser download option.

**4.8.6 Leaderboard**

The Leaderboard Page consists of a list of the top 5 users and their average scores. The purpose of this page is to encourage healthy competition among the users by providing the users a comparison of their performance with other users’ performance on the platform.

**4.9 Source Code**

**4.9.1 Authentication Module**

import axios from 'axios'

import{

REGISTER\_SUCCESS,

REGISTER\_FAIL,

USER\_LOADED,

ADMIN\_LOADED,

AUTH\_ERROR,

LOGIN\_SUCCESS,

LOGIN\_FAIL,

LOGOUT,

CLEAR\_PROFILE

} from './types'

import setAuthToken from '../utils/setAuthToken'

import { setAlert } from './alert'

const jwt = require('jsonwebtoken');

//LOAD USER

export const loadUser = () => async dispatch => {

if(localStorage.token){

setAuthToken(localStorage.token);

}

try {

const res = await axios.get('/api/auth');

dispatch({

type: USER\_LOADED,

payload: res.data

});

} catch (err) {

dispatch({

type: AUTH\_ERROR

});

}

}

//REGISTER

export const register = (name,email,password) => async dispatch => {

const config = {

headers: {

'Content-Type': 'application/json',

}

}

const body = JSON.stringify({name,email,password});

try{

const res = await axios.post('http://localhost:5000/api/users',body,config);

dispatch({

type: REGISTER\_SUCCESS,

payload: res.data

})

dispatch(loadUser());

}

catch(err)

{

const errors = err.response.data.errors;

if(errors){

errors.forEach(error=>dispatch(setAlert(error.msg,'danger')));

}

dispatch({

type:REGISTER\_FAIL

});

}

}

//LOGIN

export const login = (email,password) => async dispatch => {

const config = {

headers: {

'Content-Type': 'application/json',

}

}

const body = JSON.stringify({email,password});

try{

const res = await axios.post('http://localhost:5000/api/auth',body,config);

console.log("LOGIN!!!")

const x = jwt.decode(res.data.token);

console.log(x);

console.log(res);

dispatch({

type: LOGIN\_SUCCESS,

payload: res.data

})

dispatch(loadUser());

}

catch(err)

{

const errors = err.response.data.errors;

if(errors){

errors.forEach(error=>dispatch(setAlert(error.msg,'danger')));

}

dispatch({

type:LOGIN\_FAIL

});

}

};

//LOGOUT

export const logout = () => dispatch => {

dispatch({type: CLEAR\_PROFILE})

dispatch({type: LOGOUT})

};

**4.9.2 Domains Module**

import axios from "axios";

import { GET\_DOMAIN, GET\_DOMAINS, DOMAIN\_ERROR, GET\_TEST, TEST\_ERROR } from "./types";

//get all domains

export const getDomains = () => async dispatch => {

try {

const res = await axios.get('/api/domains');

dispatch({

type: GET\_DOMAINS,

payload: res.data

});

} catch (err) {

dispatch({

type: DOMAIN\_ERROR,

payload: { msg: err.response.statusText, status:err.response.status }

});

}

};

//get domain by ID

export const getDomainById = (domainId) => async (dispatch) => {

try {

const res = await axios.get(`/api/domains/${domainId}`);

dispatch({

type: GET\_DOMAIN,

payload: res.data

});

} catch (err) {

dispatch({

type: DOMAIN\_ERROR,

payload: { msg: err.response.statusText, status: err.response.status }

})

}

};

**4.9.3 Profile Module**

import axios from "axios";

import { setAlert } from "./alert";

import { GET\_PROFILE,PROFILE\_ERROR } from "./types";

//Get current users profile

export const getCurrentProfile = () => async dispatch => {

try {

const res = await axios.get('api/profile/me');

dispatch({

type: GET\_PROFILE,

payload: res.data

})

} catch (err) {

dispatch({

type: PROFILE\_ERROR,

payload: {msg: err.response.statusText,status: err.response}

})

}

}

//Create/Update Profile

export const createProfile = (formData) => async dispatch => {

try {

const config = {

headers: {

'Content-Type' : 'application/json'

}

}

const res = await axios.post('/api/profile',formData,config)

dispatch({

type: GET\_PROFILE,

payload: res.data

});

window.location.replace(window.location.href.split('/').slice(0,-1).join('/')+'/Dashboard')

dispatch(setAlert('Profile Created'));

} catch (err) {

const errors = err.response.data.errors;

if(errors){

errors.forEach(error=>dispatch(setAlert(error.msg,'danger')));

}

dispatch({

type: PROFILE\_ERROR,

payload: {msg: err.response.statusText,status: err.response}

})

}

}

export const updateEdu = (formData) => async dispatch => {

try {

const config = {

headers: {

'Content-Type' : 'application/json'

}

}

const res = await axios.put('/api/profile/education',formData,config)

dispatch({

type: GET\_PROFILE,

payload: res.data

});

window.location.replace(window.location.href.split('/').slice(0,-1).join('/')+'/Dashboard')

// dispatch(setAlert('Profile Created'));

} catch (err) {

const errors = err.response.data.errors;

if(errors){

errors.forEach(error=>dispatch(setAlert(error.msg,'danger')));

}

dispatch({

type: PROFILE\_ERROR,

payload: {msg: err.response.statusText,status: err.response}

})

}

}

export const updateDomain = (formData) => async dispatch => {

// console.log("Outside")

try {

// console.log("Inside")

const config = {

headers: {

'Content-Type' : 'application/json'

}

}

const res = await axios.put('/api/profile/domain',formData,config)

dispatch({

type: GET\_PROFILE,

payload: res.data

});

window.location.replace(window.location.href.split('/').slice(0,-1).join('/')+'/Dashboard')

// dispatch(setAlert('Profile Created'));

} catch (err) {

const errors = err.response.data.errors;

if(errors){

errors.forEach(error=>dispatch(setAlert(error.msg,'danger')));

}

dispatch({

type: PROFILE\_ERROR,

payload: {msg: err.response.statusText,status: err.response}

})

}

}

export const updateExp = (formData) => async dispatch => {

try {

const config = {

headers: {

'Content-Type' : 'application/json'

}

}

const res = await axios.put('/api/profile/experience',formData,config)

dispatch({

type: GET\_PROFILE,

payload: res.data

});

window.location.replace(window.location.href.split('/').slice(0,-1).join('/')+'/Dashboard')

// dispatch(setAlert('Profile Created'));

} catch (err) {

const errors = err.response.data.errors;

if(errors){

errors.forEach(error=>dispatch(setAlert(error.msg,'danger')));

}

dispatch({

type: PROFILE\_ERROR,

payload: {msg: err.response.statusText,status: err.response}

})

}

}

**4.9.4 Assessments Module**

import axios from "axios";

import { applyMiddleware } from "redux";

import { setAlert } from "./alert";

import { GET\_TEST, GET\_TESTS, CLEAR\_TEST, TEST\_ERROR, REGISTER\_SUCCESS } from "./types";

//get all tests

export const getTests = () => async dispatch => {

dispatch({ type: CLEAR\_TEST });

try {

const res = await axios.get('/api/tests');

dispatch({

type: GET\_TESTS,

payload: res.data

});

} catch (err) {

dispatch({

type: TEST\_ERROR,

payload: { msg: err.response.statusText, status: err.response.status }

});

}

};

//get test by ID

export const getTestById = (testId) => async (dispatch) =>

{

try {

const res = await axios.get(`/api/tests/${testId}`);

dispatch({

type: GET\_TEST,

payload: res.data

});

} catch (err) {

dispatch({

type: TEST\_ERROR,

payload: { mag: err.response.statusText, status: err.response.status }

});

}

};

//Add Test

export const add\_test = (name,testName,activeStatus) => async dispatch => {

const config = {

headers: {

'Content-Type': 'application/json',

}

}

const body = JSON.stringify({testName,name,activeStatus});

try{

const res = await axios.post('http://localhost:5000/api/domains',body,config);

console.log("RETURNING NOW");

window.location.replace(window.location.href.split('/').slice(0,-1).join('/')+'/AddTest?'.concat(`testName=${testName}&name=${name}`))

}

catch(err)

{

// console.log(err)

dispatch(

setAlert(err.response.data.msg,'danger')

)

}

}

export const add\_test\_to\_domain = (formData,testName,name) => async dispatch => {

const config = {

headers: {

'Content-Type': 'application/json',

}

}

// formData = JSON.stringify({formData})

const body = JSON.stringify({formData,testName,name});

try{

const res = await axios.put('http://localhost:5000/api/domains/addQues',body,config);

console.log(res);

window.location.replace(window.location.href.split('/').slice(0,-1).join('/')+'/AdminPage')

}

catch(err)

{

// console.log(err)

dispatch(

setAlert(err.response.data.msg,'danger')

)

}

}

**4.9.5 User’s Tests Module**

import axios from 'axios';

import { setAlert } from './alert';

import {

GET\_USER\_TESTS,

USER\_TEST\_ERROR,

GET\_ALL\_USER\_TESTS,

ALL\_USER\_TEST\_ERROR

} from './types';

//get current user's tests

export const getCurrentUserTests = () => async dispatch => {

try {

const res = await axios.get('/api/user\_test/me');

dispatch({

type: GET\_USER\_TESTS,

payload: res.data

});

} catch (err) {

dispatch({

type: USER\_TEST\_ERROR,

payload: { msg: err.response.statusText, status: err.response.status }

})

}

}

export const getAllUserTests = () => async dispatch => {

try {

const res = await axios.get('/api/user\_test/all');

dispatch({

type: GET\_ALL\_USER\_TESTS,

payload: res.data

});

} catch (err) {

dispatch({

type: ALL\_USER\_TEST\_ERROR,

payload: { msg: err.response.statusText, status: err.response.status }

})

}

}

//update user\_test by test Name

export const createCurrentUserTest = (testName, testData, edit = false) => async dispatch => {

try {

const config = {

headers: {

'Content-Type': 'application/json'

}

}

const res = await axios.post(`/api/user\_test/${testName}`, testData, config);

dispatch({

type: GET\_USER\_TESTS,

payload: res.data

});

dispatch(setAlert(edit ? 'Test Updated' : 'Test Created'));

} catch (err) {

dispatch({

type: USER\_TEST\_ERROR,

payload: { msg: err.response.statusText, status: err.response.status }

})

}

}

**4.9.6 Resume Builder Module**

import axios from "axios";

import { PDF\_FAIL, PDF\_LOADED } from "./types";

import FileSaver from 'file-saver';

//Send Resume

export const loadRes = (req,res) => async dispatch => {

try {

// const res\_down = await axios.get('api/send\_res/downloadPDF');

console.log("Inside LOADRES")

await axios.get('api/send\_res/downloadPDF',{responseType: 'blob'}).then((res)=>{

FileSaver.saveAs(res.data, 'User\_Resume.pdf');

})

dispatch({

type: PDF\_LOADED,

payload: true

})

} catch (err) {

console.log("ERROR OCCURED WHILE GETTING RESUME");

dispatch({

type: PDF\_FAIL,

payload: {msg: err.response.statusText,status: err.response}

})

}

}

//Build Resume

export const py\_scr = (profile) => async dispatch => {

const config = {

headers: {

'Content-Type': 'application/json',

}

}

const body = JSON.stringify(profile);

try {

const res = await axios.post('api/res\_build',body,config);

console.log(res);

if (res.data=="success\r\n")

{

console.log("Inside PY\_SCR")

dispatch(loadRes());

// const res\_builder = axios.get('api/send\_res');

}

else

{

console.log("Need the other success");

}

} catch (err) {

console.log("ERROR OCCURED");

}

};

**5. PROJECT TESTING**

**5.1. Introduction**

In general, testing is finding out how well something works. In terms of human beings, testing tells what level of knowledge or skill has been acquired. In computer hardware and software development, testing is used at key checkpoints in the overall process to determine whether objectives are being met.

**5.2. Types of Testing**

**5.2.1. Code Testing**

This examines the logic of the program. For example, the logic for updating various sample data and with the sample files and directories were tested and verified. In this project, the inbuilt debugging feature of Visual Studio Code IDE was utilized to perform initial Code Testing.

**5.2.2 System Testing**

Executing this specification starting what the program should do and how it should perform under various conditions. Test cases for various situation and combination of conditions in all the modules are tested.

Some of the test cases include:

* Form Validation cases for Registration/Login
* Form Validation cases for Edit Profile
* Correct statistics display of tests for different scores and test status
* Proper display of scores on Dashboard and Leaderboard
* React based Page Routing, and secure URL Authentication verification

**5.2.3 Unit Testing**

In the unit testing we test each module individually and integrate with the overall system. Testing focuses verification efforts on the smallest unit of software design in the module. This is also known as module testing. The module of the system is tested separately. This testing is carried out during programming stage itself. In the testing step each module is found to work satisfactorily as regard to expected output from the module. There are some validation checks for fields also. It is very easy to find error debut the system. Each Module can be tested using the following two Strategies:

1. White Box Testing
2. Black Box Testing

**5.2.4. White Box Testing**

White box testing is a testing case design method that uses the control structure of the procedure design to derive test cases. All independents path in a module are exercised at least once, all logical decisions are exercised at once, execute all loops at boundaries and within their operational bounds exercise internal data structure to ensure their validity.

The following are the paths that were tested during the White Box testing phase:

* Regular flow:
  + The user first creates an account on the Registration Page. They are then automatically logged in and sent to the Dashboard.
  + The user is then prompted to create their profile in the Profile Page, where they are asked to enter basic information as well as their domains of interest. They can also download their resume via Resume Builder.
  + The user then navigates back to the Dashboard where they can view basic links and information regarding their selected domains.
  + Next, the user can choose to take Assessments, and upon pressing Submit, the code is saved to the database.
  + The user can come back to the Dashboard and view their updated score statistics on the Dashboard and Leaderboard pages.
* Not Authenticated:
  + The user, upon entering information on the Registration page, are notified of incorrect information entered, and are asked to re-enter correct information. The same path is also explored for Login page.
  + If the user tries to access the Dashboard URL before logging in, they will be redirected back to the Landing Page.
* Admin Authenticated:
  + If the user has admin permission, they are directed to the Admin Dashboard upon login. They can then add new Tests and change content.

**5.2.5. Black Box Testing**

Black Box Testing attempts to find errors in following areas or categories, incorrect or missing functions, interface error, errors in data structures, performance error and initialization and termination error. Here all the input data must match the data type to become a valid entry.

Following are the features that were tested during Black Box testing:

* Profile and User data being correctly fetched from the Database to the Dashboard and Profile pages, according to the specific authenticated user.
* User Test scores being correctly fetched to the Dashboard, and segregated as per their respective domains on the Dashboard.
* Proper User login and logout, session management, and URL authentication.
* Score calculation and storage upon completion of Assessments. Subsequent display of scores in Graphs on the Dashboard.
* Proper formatting of PDF Resume based on Profile information.

**6. SCREENSHOTS**

The Registration page is the first page that the user visits for the first time, wherein the user can register onto the platform and create an account. The Registration page is shown in Fig 6.1

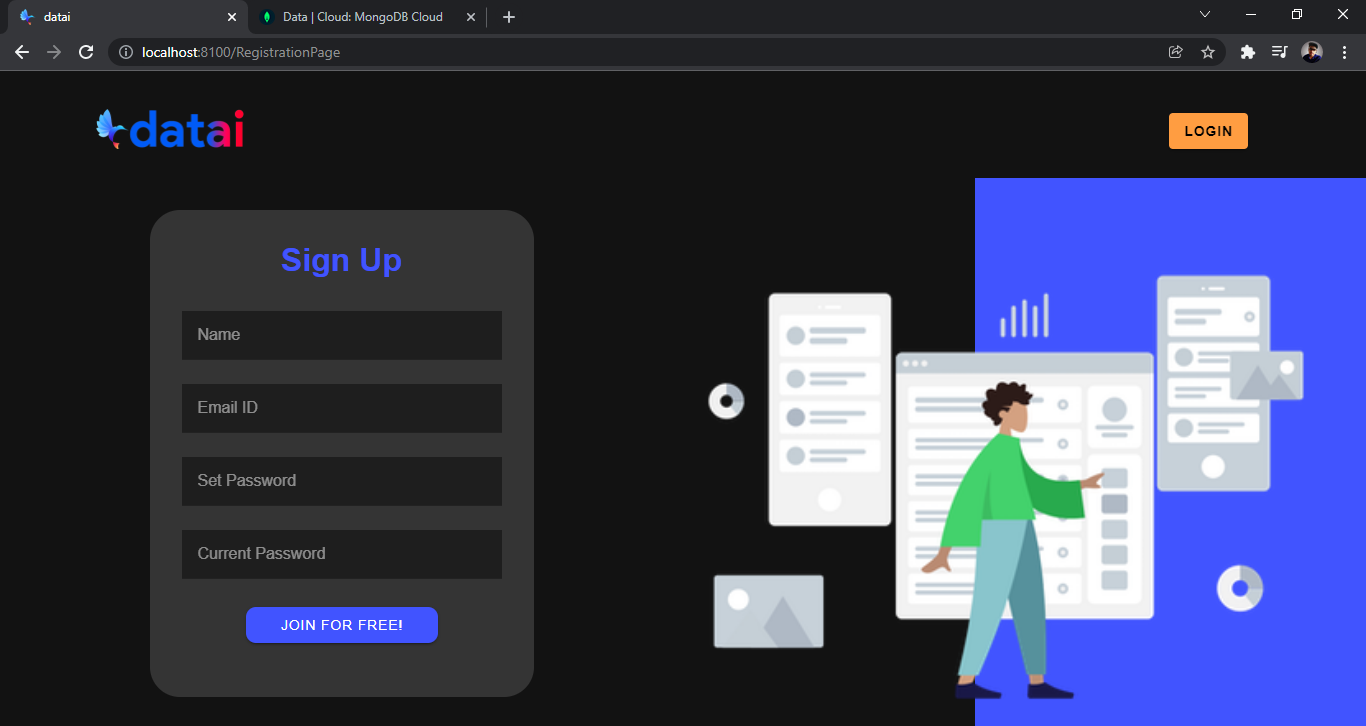
****

Fig. 6.1: Registration Page

Similarly, when the user returns after registration, they are required to log into the website using their email ID and password which was entered during the Registration, as shown in Fig 6.2

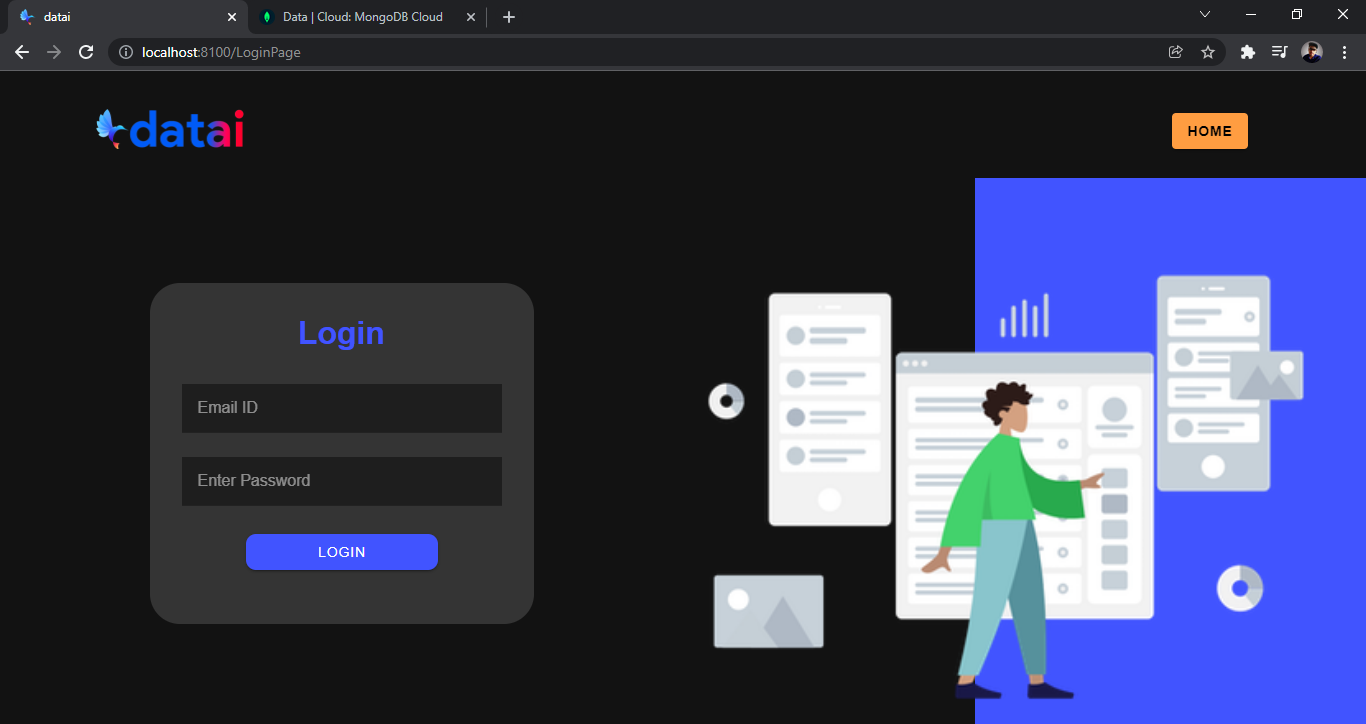


Fig. 6.2: Login Page

Next, the user is taken to the Dashboard after logging in, where they can view basic profile information, and relevant links and statistical graphs which are separated according to each of the domains they have selected. From this page they can then take the tests from the Tests Available, and view their score history in the Test Statistics, as shown in Fig 6.3 The user can also view the scores of other users in the Leaderboard shown in Fig 6.4

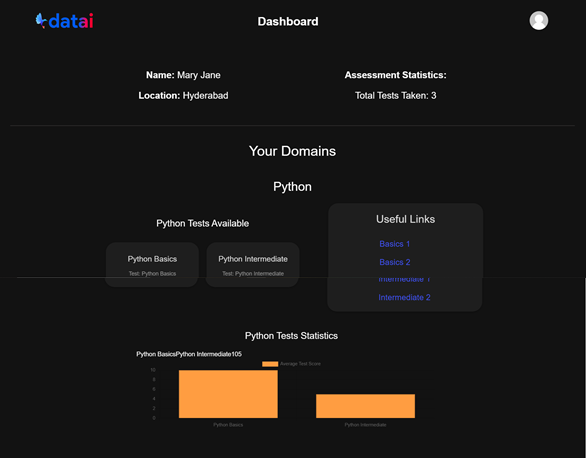


Fig. 6.3: Dashboard

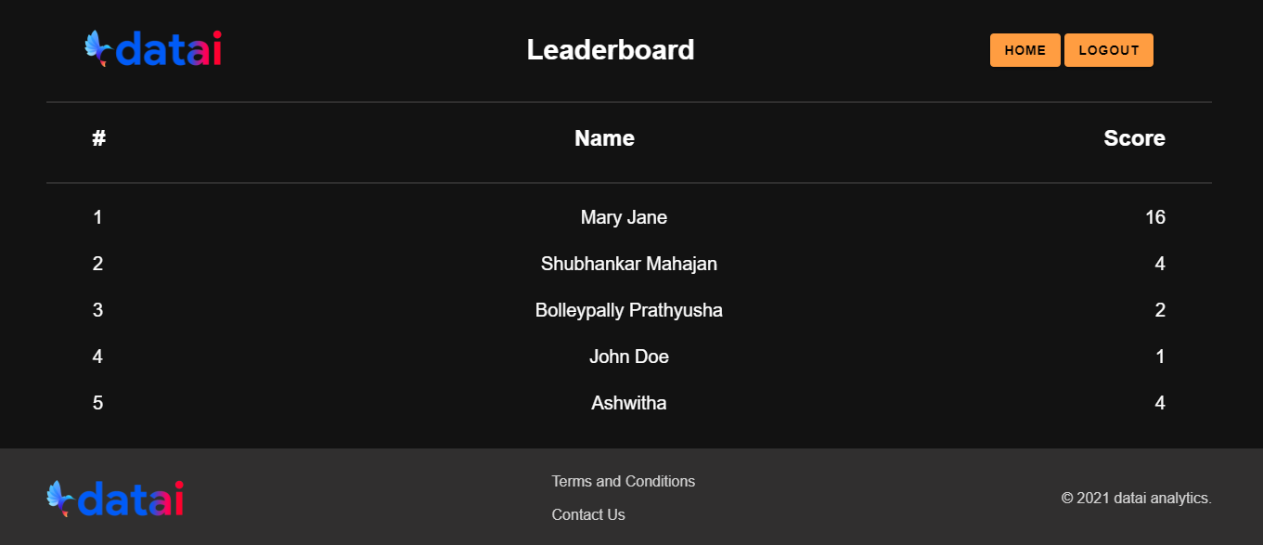


Fig. 6.4: Leaderboard

**7. RESULTS**

The Resume Builder outputs the following file in the form of a PDF file as shown in Fig. 6.5. which can then be downloaded onto the browser

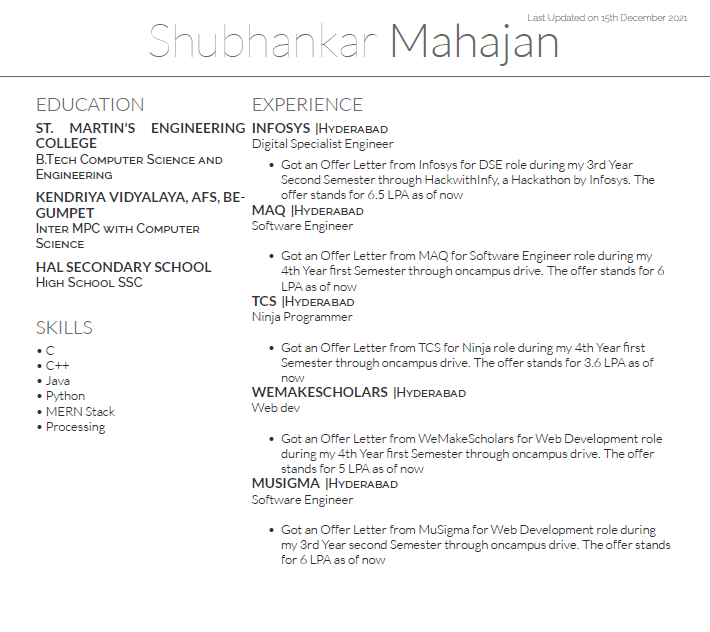


Fig. 6.5: Downloadable Resume Builder PDF

**8. CONCLUSION**

This project and its results enable its users to both self-learn concepts as well as test their knowledge with the provided tests, and assess their performance in these tests with related graphs. They can also view their comparative progress with other users in the leaderboard and use it to encourage healthy peer-level competition, and download their own resume in an appropriate PDF format.

**FUTURE SCOPE**

For future enhancements, one can consider the following ways:

* Usage of Firebase, a Backend as a Service in place of the manual Mongoose-Express set up, in order to access the benefits of cloud functions as well as a more well-built backend setup.
* Addition of a payment portal for certain courses and services that can be accessed after payment of a certain fee.
* Comments and Posts on tests and domains in order to enable user interaction which could enhance the learning experience.
* Enhanced and in depth analysis of test scores for self-improvement

**9. REFERENCES**

1. Naidu, N. D., Adarsh, P., Reddy, S., Raju, G., Kiran, U. S., Sharma, V., ... & Sharma, V. (2021). E-Commerce web Application by using MERN Technology. *International Journal for Modern Trends in Science and Technology*, *7*, 1-5
2. Niranjanamurthy, M., Kavyashree, N., Jagannath, S., & Chahar, D. (2013). Analysis of e-commerce and m-commerce: advantages, limitations and security issues. *International Journal of Advanced Research in Computer and Communication Engineering*, *2*(6), 2360-2370
3. Chanana, N., & Goele, S. (2012). Future of e-commerce in India. *International Journal of Computing & Business Research*, *8*.
4. Mai, N. (2020). E-commerce Application using MERN stack.
5. Subramanian, V. (2017). *Pro MERN Stack*. Apress.
6. Ullah, S. E., Alauddin, T., & Zaman, H. U. (2016, January). Developing an E-commerce website. In *2016 International Conference on Microelectronics, Computing and Communications (MicroCom)* (pp. 1-4). IEEE.
7. King, D. N., & King, D. N. (2004). *Introduction to e-commerce*. Prentice Hall.
8. Nemat, R. (2011). Taking a look at different types of e-commerce. *World Applied Programming*, *1*(2), 100-104.
9. Hoque, S. (2020). *Full-Stack React Projects: Learn MERN stack development by building modern web apps using MongoDB, Express, React, and Node. js*. Packt Publishing Ltd.

[10] https://www.udemy.com/course/ionic-react/

[11] https://www.javatpoint.com/reactjs-tutorial

[12] ccbp.in NxtWave | CCBP 4.0 Certification Programs (2021)

[13] https://app.joinsuperset.com/#/s/jobprofiles