**WEEKLY REPORT**

WEEK-1(From Dt 15-05-2023 to Dt 20-05-2023)

**TOPIC:** HTML5 & CSS

**ACTIVITY LOG FOR THE FIRST WEEK**

|  |  |  |  |
| --- | --- | --- | --- |
| DAY & DATE | BRIEF DESCRIPTION OF DAILY ACTIVITY | LEARNING OUTCOME | INCHARGE SIGNATURE |
| DAY-1 | * Introduction to Web Development * Introduction to HTML5 * Basic Page Structure | Technologies and Tools used for web Design. Learned about Text editors like VS Code and sublime text |  |
| DAY-2 | * Formatting Page Content * Creating Lists * Structuring Content | Created div elements, unordered lists and ordered lists. Understanding content models. |  |
| DAY-3 | * Creating Links * Controlling Styling * Basic Scripting | Anchor element, linking to external pages, styling an element and adding colours. |  |
| DAY-4 | * Getting Started with CSS3 * CSS Core * Typography | Inline, internal and external CSS. Learned about type, class and ID selector. |  |
| DAY-5 | * Layouts | Designed box model, floats, display and positioning. Horizontal navbar with display and float property. |  |
| DAY-6 | * Advanced Layout | Designs with a multi-column and grid layouts. Intro to flexbox and CSS flex box. |  |

**OBJECTIVES:**

* Formatting Page Content using HTML
* Structuring Content using HTML
* Typography and Layouts using CSS

**DETAILED REPORT:**

**Day 1:**

**Introduction to Web Development:**

**Web development**refers to the creating, building, and maintaining of websites. It includes aspects such as web design, web publishing, web programming, and database management. It is the creation of an application that works over the internet i.e., websites.

The word Web Development is made up of two words, that is:

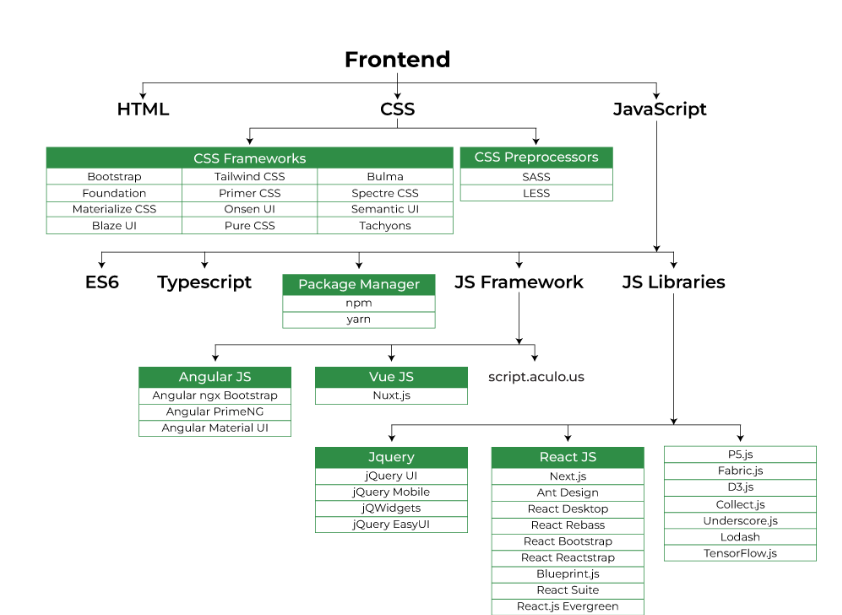
* **Web:** It refers to websites, web pages or anything that works over the internet.
* **Development:** It refers to building the application from scratch.

**Web Development can be classified into two ways:**

1. **Frontend Development**
2. **Backend Development**

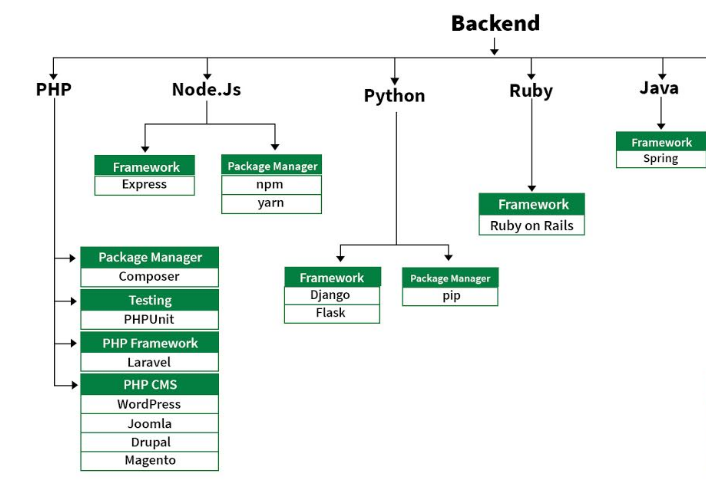
## **Frontend Development:**

The part of a website where the user interacts directly is termed as front end. It is also referred to as the ‘client side’ of the application.



## **Backend Development:**

Backend is the server side of a website. It is part of the website that users cannot see and interact with. It is the portion of software that does not come in direct contact with the users. It is used to store and arrange data.



**Day 2:**

**Introduction to HTML5:**

HTML (Hypertext Markup Language) is the standard markup language used for creating and structuring web pages. It provides a set of tags that define the structure and content of a web document.

**Some Basic Tags of Html are discussed:**

<html>: The <html> tag is the root element of an HTML document. It defines the beginning and end of the HTML content.

<head>: The <head> tag is used to define the head section of the HTML document. It contains meta-information about the document, such as the title of the page, links to external stylesheets or scripts, character encoding declarations, and more.

<body>: The <body> tag represents the main content of the HTML document. It encapsulates all the visible content that appears on the web page, including text, images, links, headings, paragraphs, and other elements.

<h1> to <h6>: These tags are used to define headings of different levels, where <h1> represents the highest level (most important) heading, and <h6> represents the lowest level (least important) heading.

<p>: The <p> tag is used to define a paragraph. It is commonly used to structure and represent blocks of text in HTML documents.

<a>: The <a> tag creates a hyperlink and is used to define an anchor or a link to another web page, document, or a specific location within the same page.

<img>: The <img> tag is used to insert an image into an HTML document. It requires a src attribute that specifies the source (URL or file path) of the image.

<ul> and <li>: These tags are used to create an unordered list. The <ul> tag defines the start and end of the list, while <li> tags define each list item within the unordered list.

<div>: The <div> tag is a generic container that is used to group and style other HTML elements. It does not carry any semantic meaning on its own but serves as a way to organize and structure content.

**Day 4:**

**Getting Started with CSS:**

CSS (Cascading Style Sheets) is used to style and format HTML elements, providing control over the visual presentation of web pages.

**Here is the basic syntax of CSS:**

Selector: The selector specifies which HTML elements you want to style. It can be an element name, class, ID, or a combination of these. For example:

Element Selector: h1 targets all <h1> elements.

Class Selector: .my-class targets elements with the class "my-class".

ID Selector: #my-id targets the element with the ID "my-id".

Declaration Block: A declaration block is enclosed in curly braces {} and contains one or more declarations. Each declaration consists of a property and a value, separated by a colon. For example

selector {

property: value;

}

Property: A property defines the aspect of the element you want to style, such as color, font-size, background-color, etc.

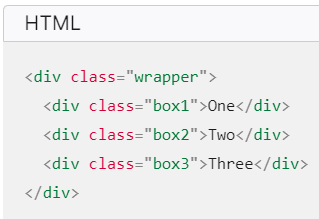
Value: The value specifies the specific setting for the property. For example, for the color property, the value can be red, #FF0000, or rgb(255, 0, 0).

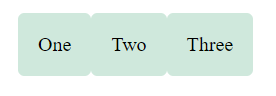
**Day 5:**

**CSS Layouts:**

**Flexbox:**

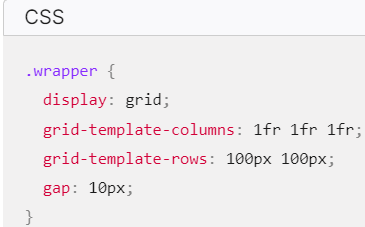
Flexbox is the short name for the [Flexible Box Layout](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_flexible_box_layout) CSS module, designed to make it easy for us to lay things out in one dimension — either as a row or as a column. To use flexbox, you apply display: flex to the parent element of the elements you want to lay out; all its direct children then become flex items. if we add display: flex to the parent, the other items now arrange themselves into columns.

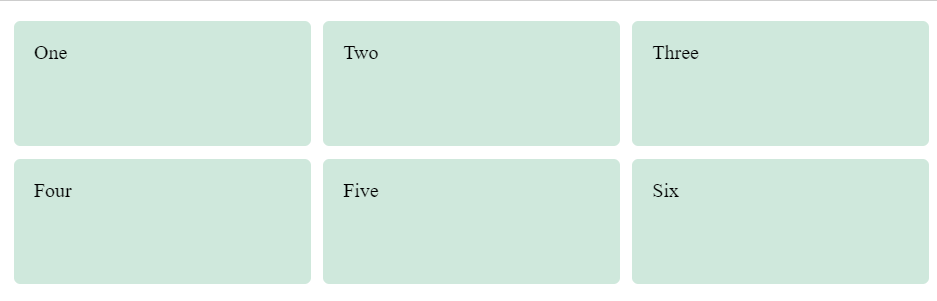
 



## [Grid Layout](https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Introduction#grid_layout)

While flexbox is designed for one-dimensional layout, Grid Layout is designed for two dimensions — lining things up in rows and columns. In addition to using display: grid, we also define some row and column tracks for the parent using the [grid-template-rows](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-template-rows) and [grid-template-columns](https://developer.mozilla.org/en-US/docs/Web/CSS/grid-template-columns) properties respectively.



**Floats**

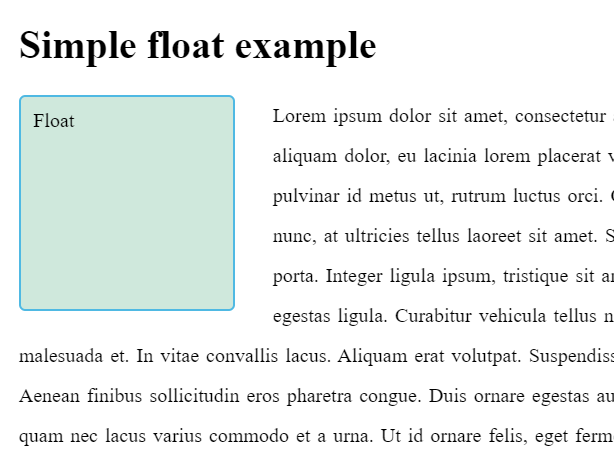
Floating an element changes the behavior of that element and the block level elements that follow it in normal flow. The floated element is moved to the left or right and removed from normal flow, and the surrounding content floats around it.

The [float](https://developer.mozilla.org/en-US/docs/Web/CSS/float) property has four possible values:

* left — Floats the element to the left.
* right — Floats the element to the right.
* none — Specifies no floating at all. This is the default value.
* inherit — Specifies that the value of the float property should be inherited from the element's parent element.

** **

**Output:**

****

**CSS Positioning:**

CSS positioning includes the position property, which allows you to precisely position elements on a page. The position property values include static, relative, absolute, fixed, and sticky.

**Day 6:**

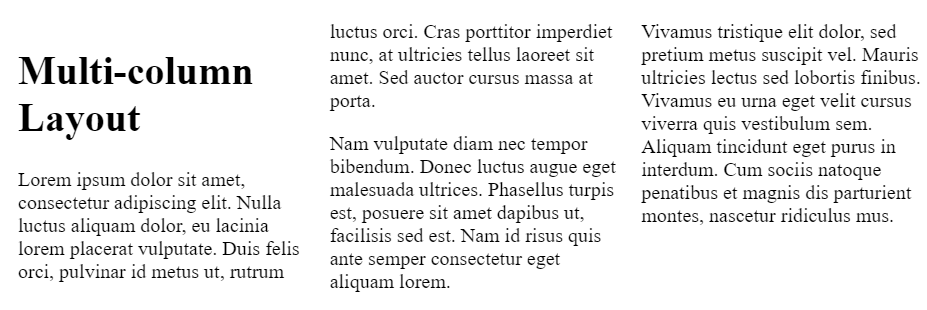
**Advanced Layouts:**

## **Multi-Column Layout:**

The multi-column layout CSS module provides us a way to lay out content in columns, similar to how text flows in a newspaper. To turn a block into a multi-column container, we use either the [column-count](https://developer.mozilla.org/en-US/docs/Web/CSS/column-count) property, which tells the browser how many columns we would like to have, or the [column-width](https://developer.mozilla.org/en-US/docs/Web/CSS/column-width) property, which tells the browser to fill the container with as many columns as possible of a specified width.

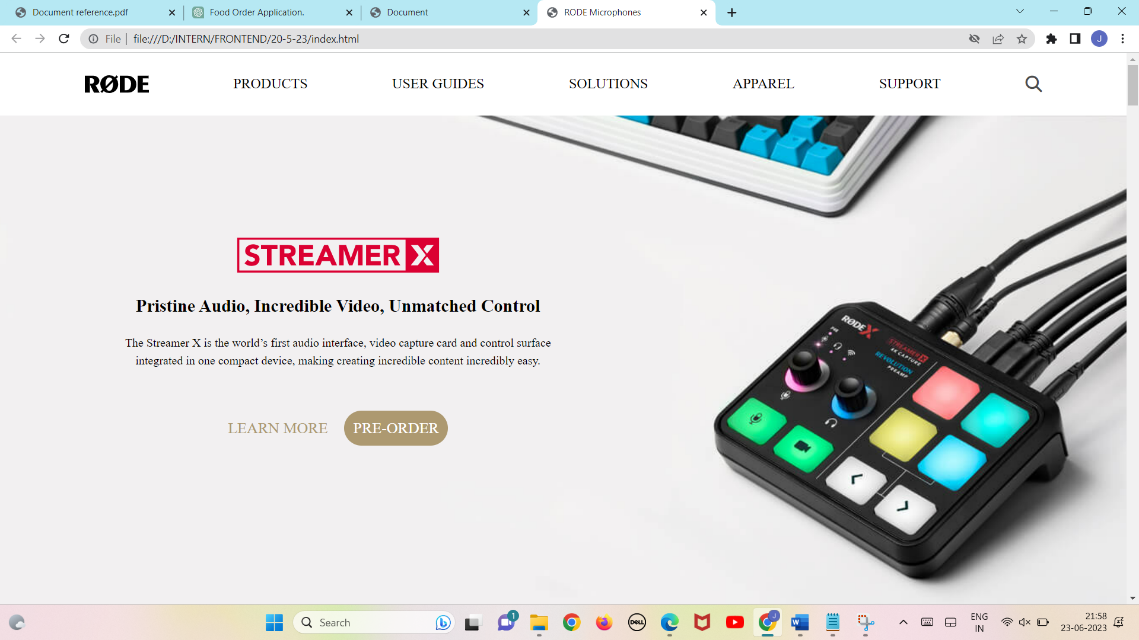
 

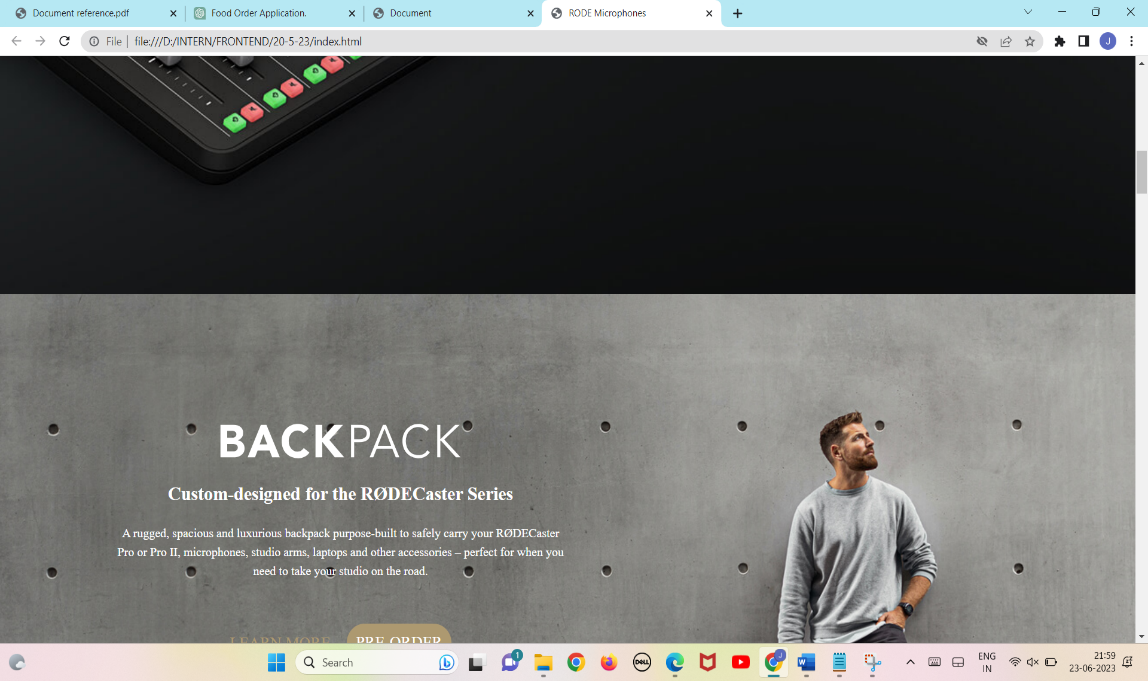
**Output:**

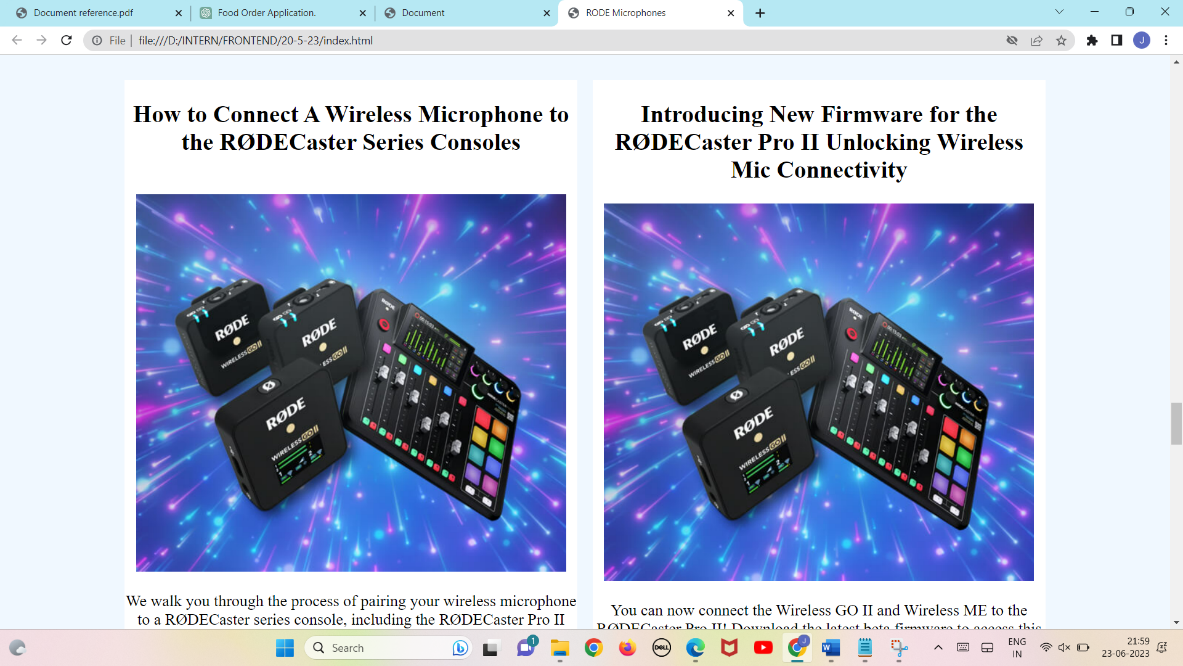


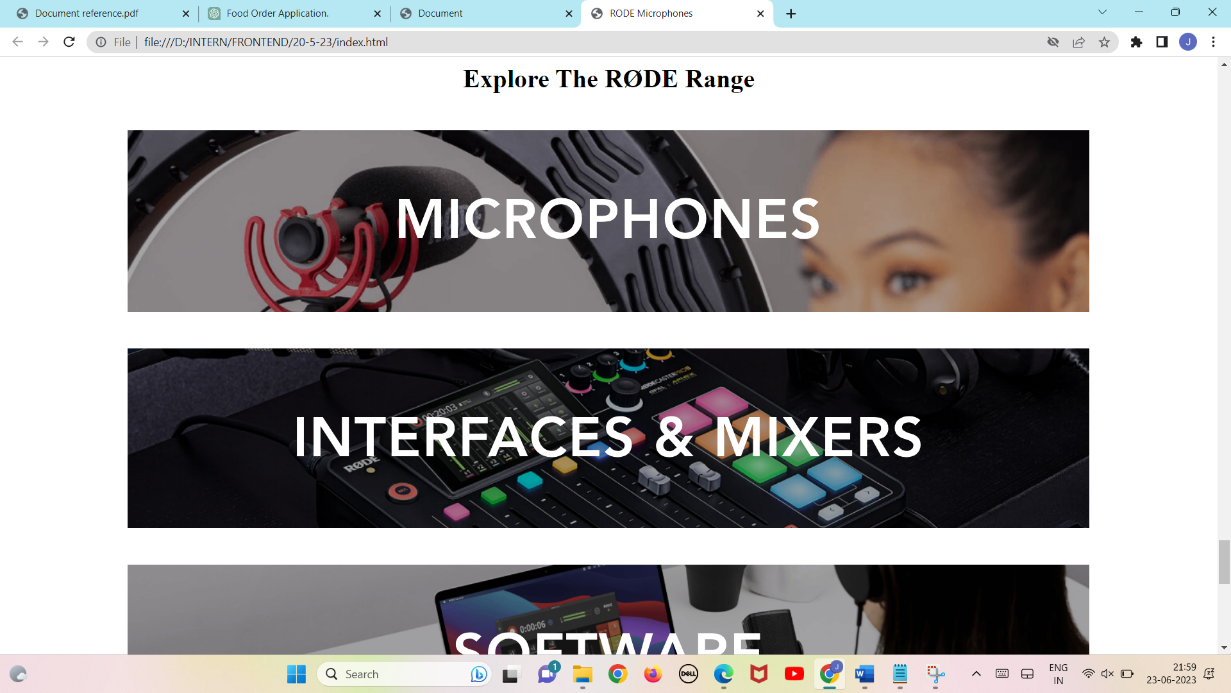
**TASK: Creating a simple website**

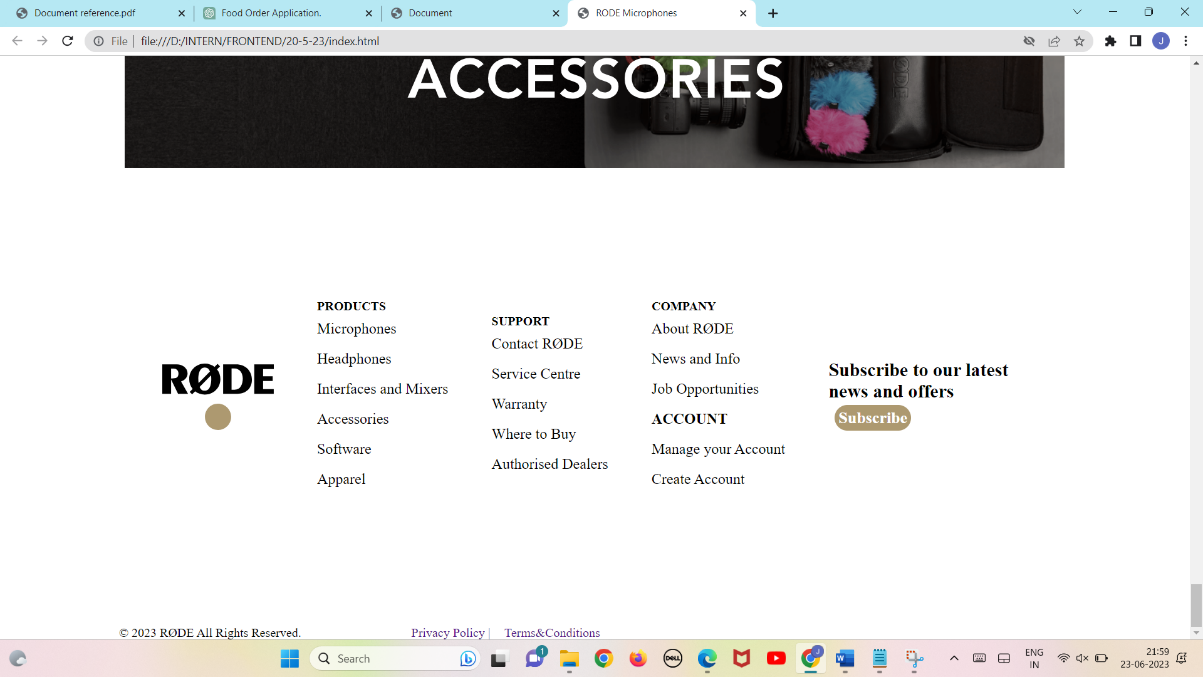
We created a sample website using simple html, CSS and Bootstrap. Bootstrap is a popular front-end framework that provides a collection of pre-built CSS and JavaScript components, along with a grid system and responsive design utilities



****

****

****

**`**

**WEEKLY REPORT**

WEEK-2 (From Dt 22-05-2023 to Dt 27-05-2023)

**TOPIC:** JavaScript

**ACTIVITY LOG FOR THE SECOND WEEK**

|  |  |  |  |
| --- | --- | --- | --- |
| DAY & DATE | BRIEF DESCRIPTION OF DAILY ACTIVITY | LEARNING OUTCOME | INCHARGE SIGNATURE |
| DAY-1 | * Introduction to JavaScript * Basics | Tools for JavaScript development. Adding inline JavaScript to an HTML element. |  |
| DAY-2 | * Writing JavaScript * Control Flow * Arrays | Basics syntax & comments. Operators, Data Structures and types. Conditional statements and logic. |  |
| DAY-3 | * Loops and Iteration * Function, | For, while loops. Defining and calling functions. Learned about arrow functions. |  |
| DAY-4 | * Essential JavaScript Built-in Methods * Writing JavaScript-Advanced | Text formatting-string literals. Practiced Regular expressions. |  |
| DAY-5 | * Objects * Keyed Collections | Creating Objects, Objects and properties, defining Methods. |  |
| DAY-6 | * JavaScript Modules * JavaScript and the DOM |  |  |

**OBJECTIVES:**

* Introduction to JavaScript
* Control Flow, Arrays and Functions
* JavaScript Modules
* JavaScript and DOM

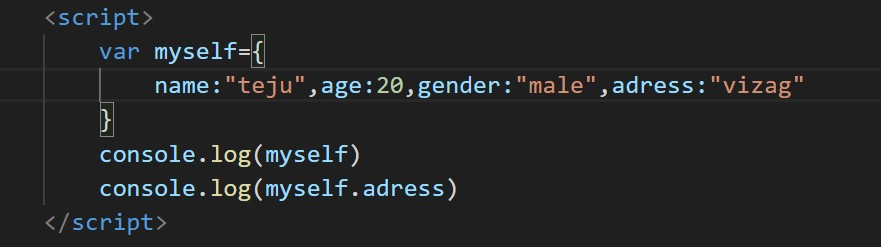
**DETAILED REPORT:**

**JavaScript:**

JavaScript is a versatile and widely-used programming language that enables dynamic interactivity and behavior on web pages. It is primarily used in web development to enhance the functionality and user experience of websites and web applications.

**Script tag:**

The <script> tag is an HTML element used to embed or reference JavaScript code within an HTML document. It allows you to include JavaScript code directly in the HTML file or link to an external JavaScript file.



The console would print the name “teju” and address “vizag”.

**Operators:**

compound assignment operators:

combining the arithmetic and assignment operators

* == compares only the data it doesn’t consider the datatype

3=="3" is true

* === is used to compare the data and datatype

3==="3" is false

logical operators:

* && (and)
* || (or)
* ! (not)

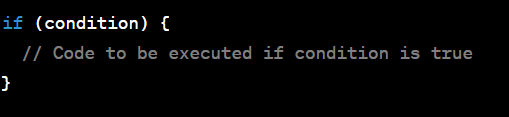
relational operators:

* <,<=
* >,>=
* ==,!=

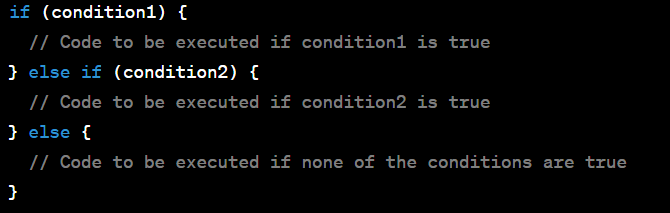
**Conditional statements:**

JavaScript provides several control flow statements that allow you to make decisions, repeat actions, and handle different scenarios in your code.

* If



* Else if



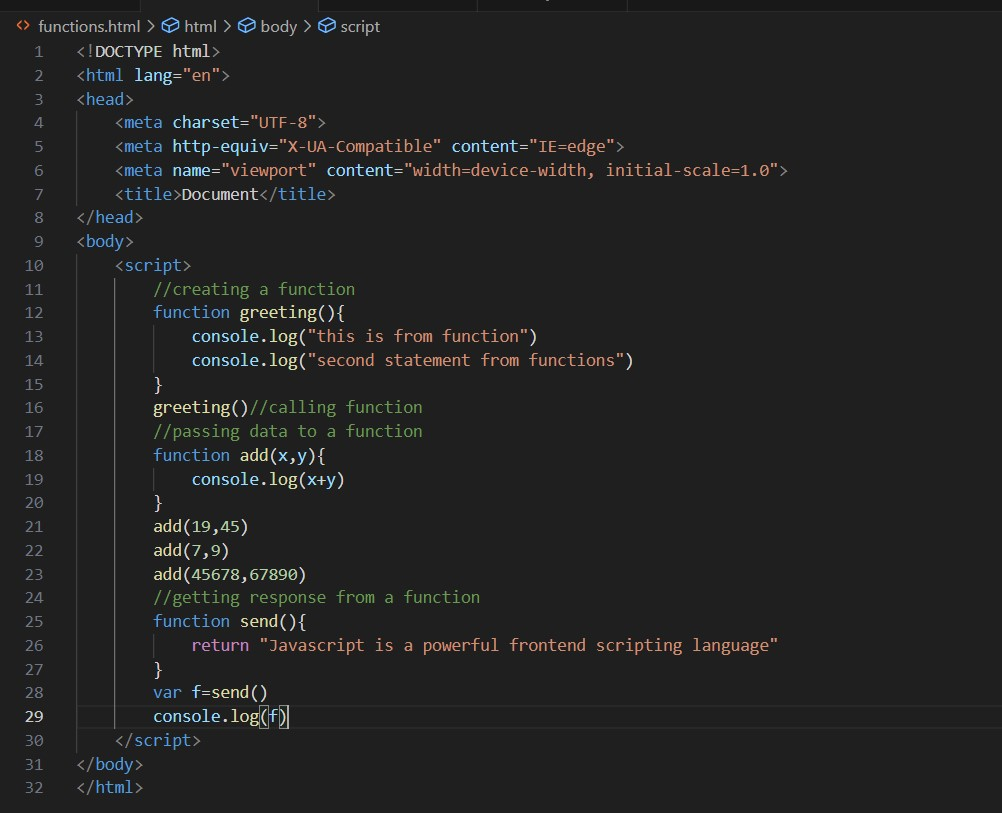
* Switch



**Functions:**

Functions are reusable blocks of code that can be defined once and executed multiple times.

1. Function Declaration
2. Function Expression, parameters and return statements
3. Function Invocation

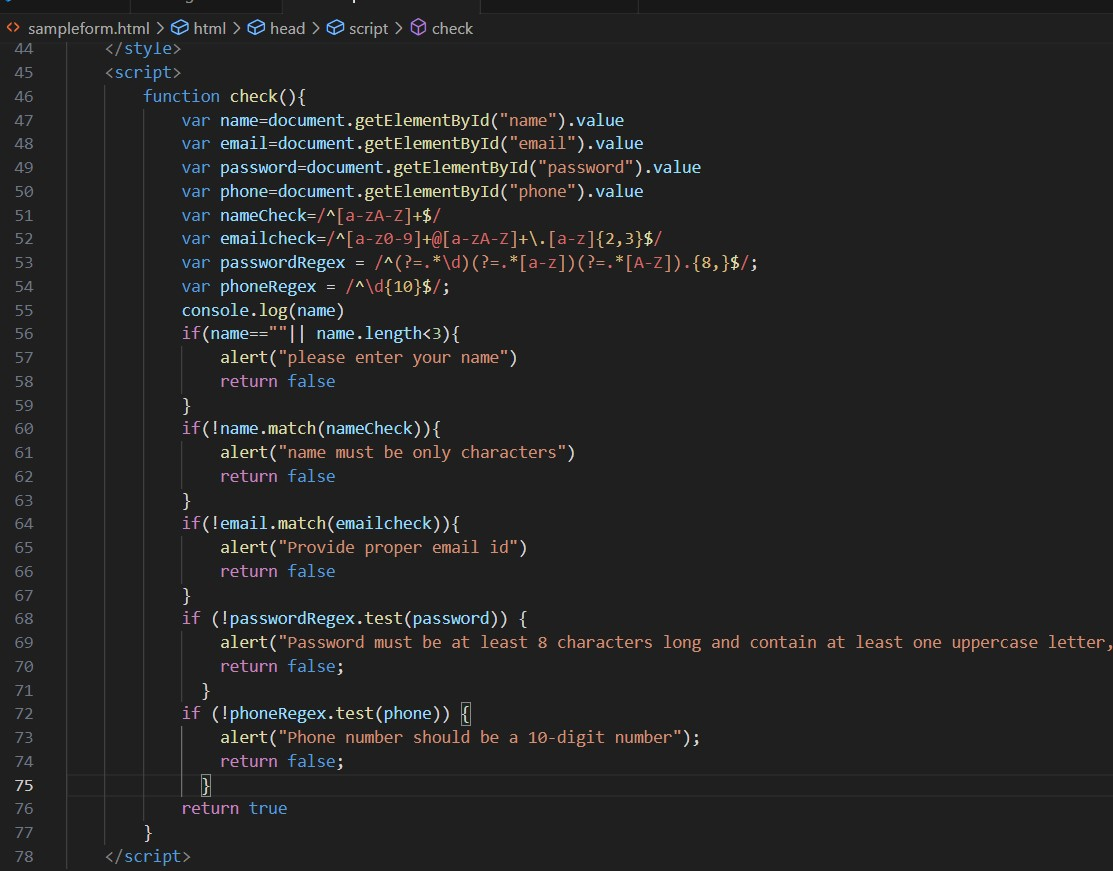


BOM (Browser object model):

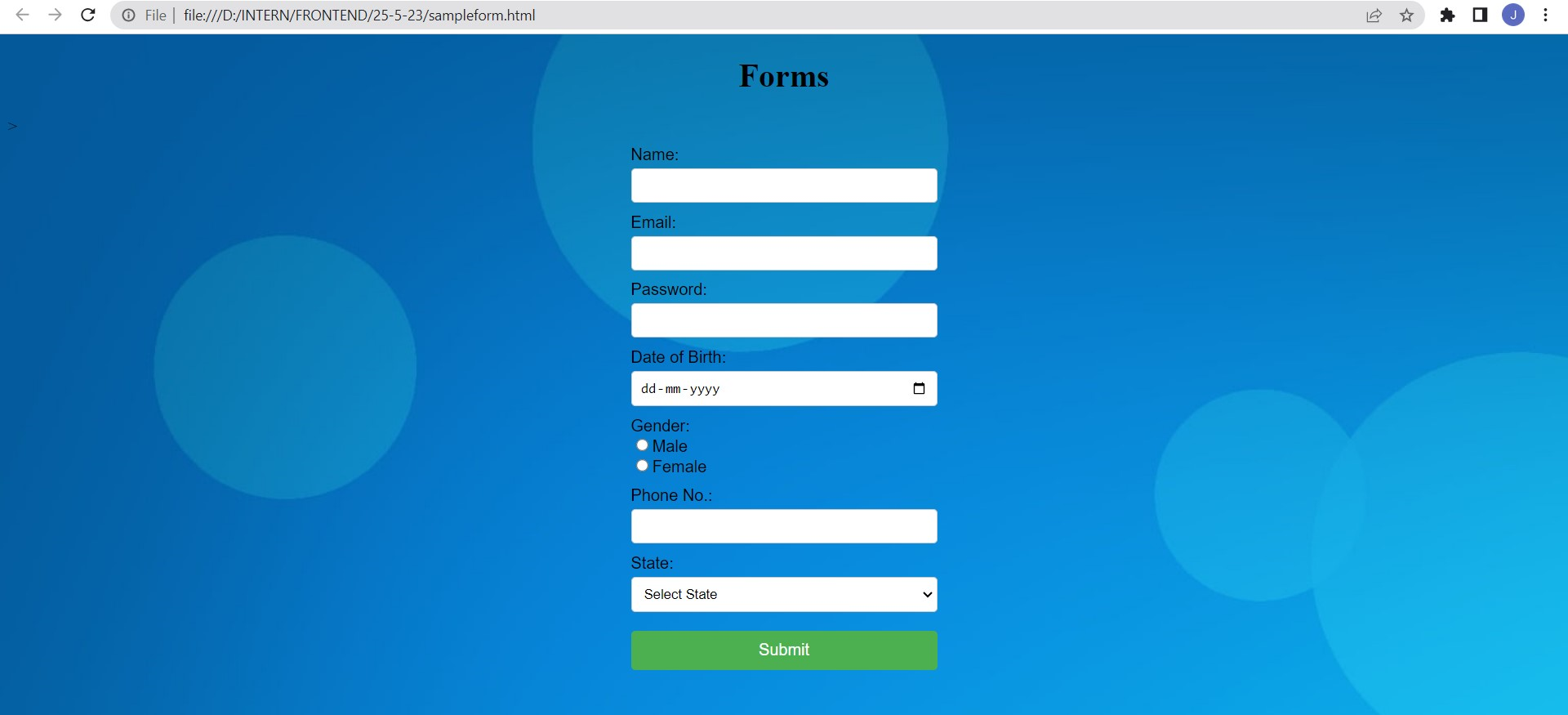
* alert ()
* prompt ()
* confirm ()

**Task: Creating a Sample Form**

We used the following javascript regular expressions and validations for validating the details of the user like email, password, phone number etc.



**The form looked like this:**



**WEEKLY REPORT**

WEEK-3(From Dt 29-05-2023 to Dt 03-06-2023)

**TOPIC:** ReactJS

**ACTIVITY LOG FOR THIRD WEEK**

| DAY & DATE | BRIEF DESCRIPTION OF DAILY ACTIVITY | LEARNING OUTCOME | PERSON INCHARGE SIGNATURE |
| --- | --- | --- | --- |
| DAY-1 | * Introduction to ReactJS * ReactJS Setup |  |  |
| DAY-2 | * ReactJS Components * ReactJS State |  |  |
| DAY-3 | * ReactJS Props * ReactJS Component LifeCycle |  |  |
| DAY-4 | * ReactJS Forms, Events * ReactJS Refs, Fragments |  |  |
| DAY-5 | * ReactJS Lists, Keys, Maps * ReactJS Router |  |  |
| DAY-6 | * ReactJS Flux * ReactJS Redux |  |  |

**OBJECTIVES:**

* ReactJS Setup
* ReactJS State and props
* ReactJS Forms, Events
* ReactJS Router
* ReactJS Redux

**DETAILED REPORT:**

**ReactJS:**

React, sometimes referred to as a frontend JavaScript framework, is a JavaScript library created by Facebook.

React is a tool for building UI components.

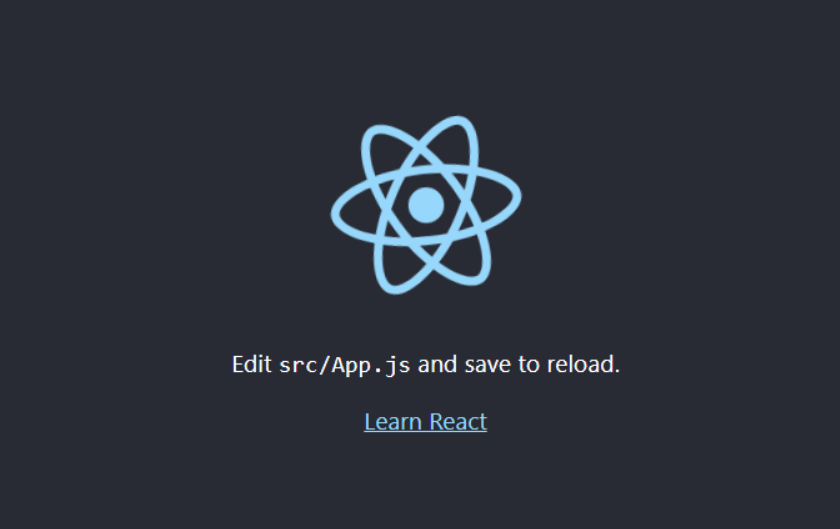
Run this command to create a React application named my-react-app:



Run this command to run the React application my-react-app:



A new browser window will pop up with your newly created React App!



**React Components:**

Components are like functions that return HTML elements. Components come in two types, Class components and Function components.

### **Class Component**

A class component must include the extends React.Component statement. This statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.

The component also requires a render () method, this method returns HTML.

**Function Component**

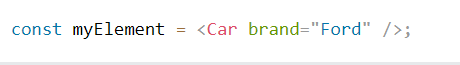
A Function component also returns HTML, and behaves much the same way as a Class component, but Function components can be written using much less code, are easier to understand.

## **Props**

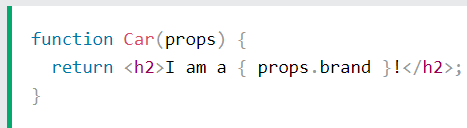
Components can be passed as props, which stands for properties.

Props are like function arguments, and you send them into the component as attributes.

Example:



The component receives the argument as a props object:

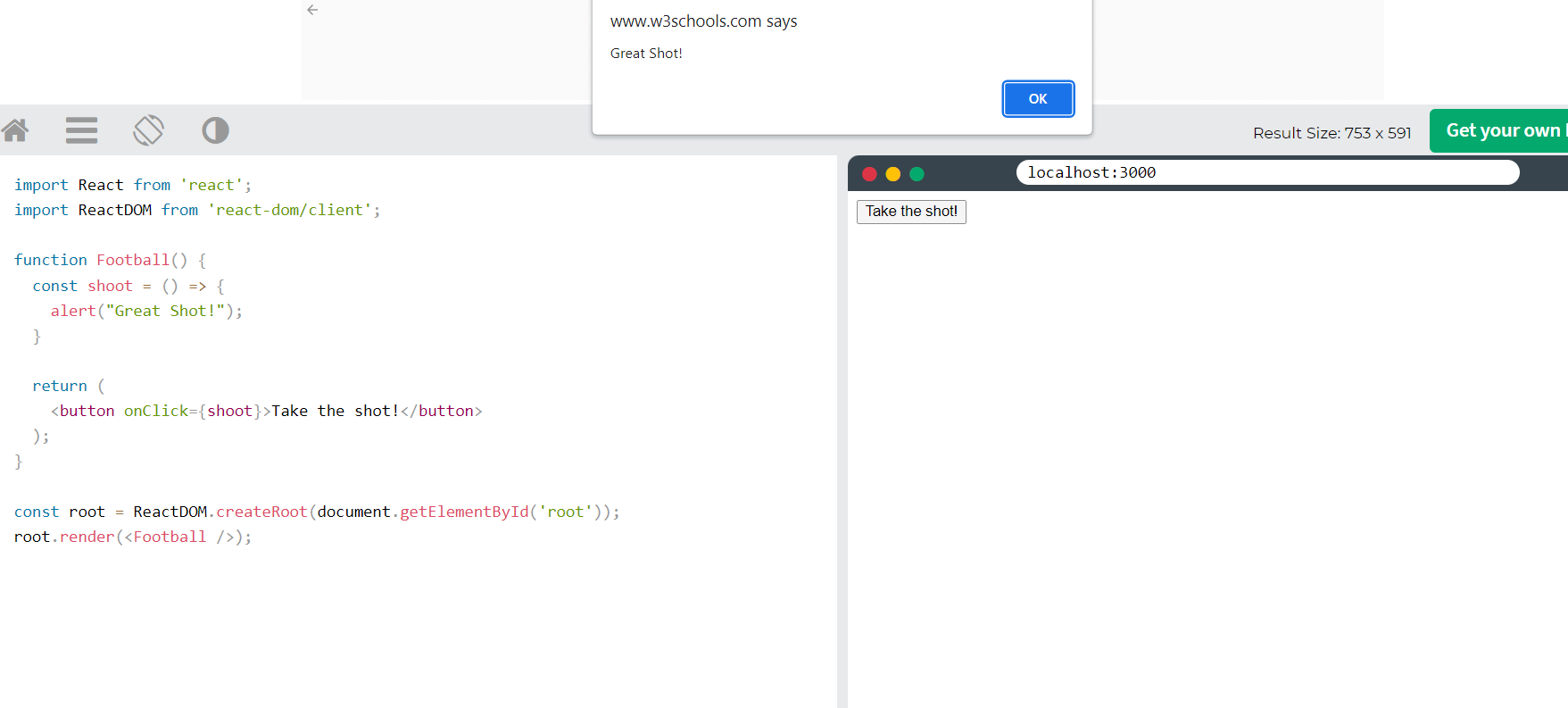


We will get the message “I am a Ford” in the webpage

Events:

Just like HTML DOM events, React can perform actions based on user events.

React has the same events as HTML: click, change, mouseover etc.



Clicking the "Take the shot!" button will trigger the shoot function and display an alert with the message "Great Shot!".

**ROUTER:**

React Router is a popular library in the React ecosystem that provides routing capabilities to create single-page applications with multiple views. It allows you to handle navigation, define routes, and render different components based on the current URL.

To use React Router, you need to install it as a dependency in your project:



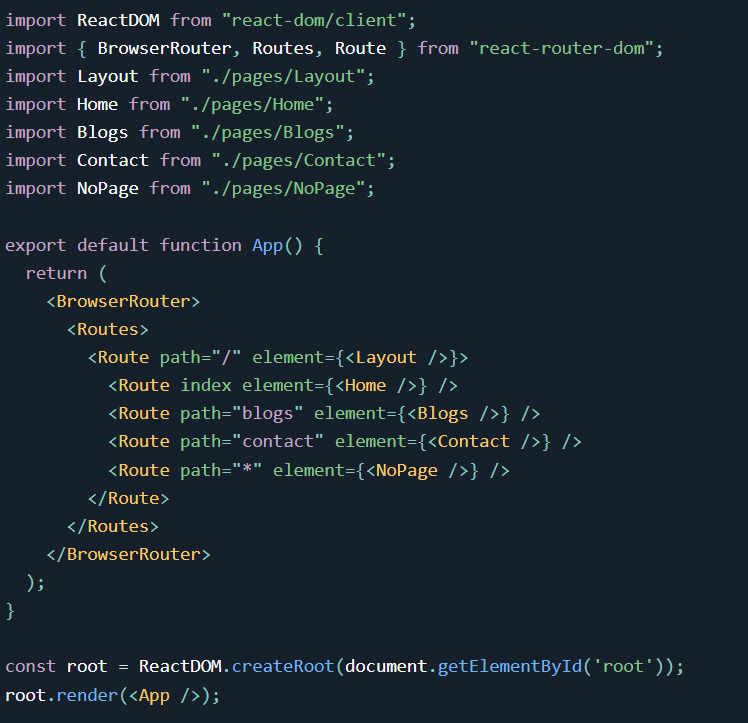
To create an application with multiple page routes, let's first start with the file structure.

Within the src folder, we'll create a folder named pages with several files:

src\pages\:

* Layout.js
* Home.js
* Blogs.js
* Contact.js
* NoPage.js

Each file will contain a very basic React component.



We wrap our content first with <BrowserRouter>.

Then we define our <Routes>. An application can have multiple <Routes>. Our basic example only uses one.

<Route>s can be nested. The first <Route> has a path of / and renders the Layout component.

The nested <Route>s inherit and add to the parent route. So the blogs path is combined with the parent and becomes /blogs.

The Home component route does not have a path but has an index attribute. That specifies this route as the default route for the parent route, which is /.

Setting the path to \* will act as a catch-all for any undefined URLs. This is great for a 404 error page.

The Layout component has <Outlet> and <Link> elements.

The <Outlet> renders the current route selected.

<Link> is used to set the URL and keep track of browsing history.

Anytime we link to an internal path, we will use <Link> instead of <a href="">.



**REDUX:**

Redux:

* Action which is going to specify the basic operation type and payload
* Reducer which carries logic or functionality of the action
* Store works like a database
* Dispatch works as a mediator between actions and reducers

redux:

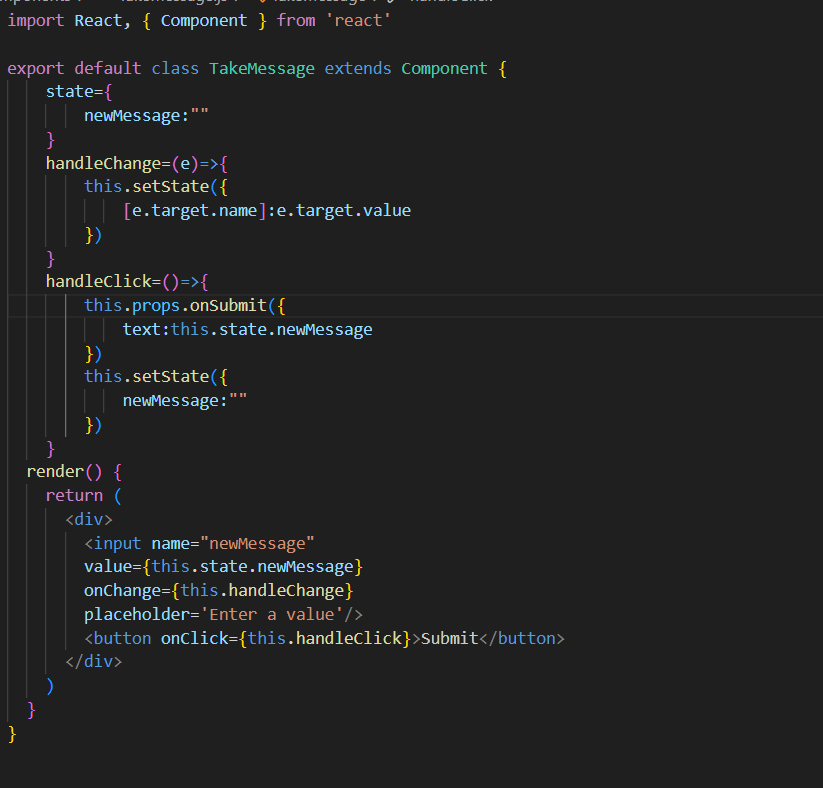
* + combineReducers
  + createStore

react-redux:

* + Provider
  + useDispatch
  + useSelector

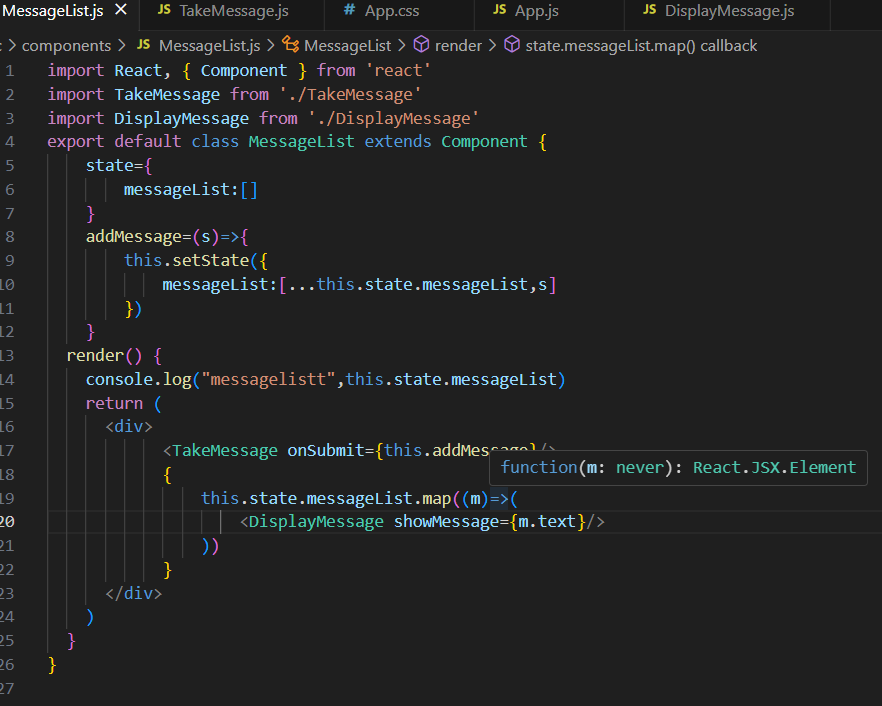
**Task:**

Create a react app with TakeMessage.js, Messagelist.js, DisplayList.js files. The website takes the values from the user and on clicking the submit button, the values will be displayed on the screen and the messages will be stored in the message list.



The handleSubmit method is passed as the onSubmit prop. The handleSubmit method receives the submitted message object and can be used to process or display the message.

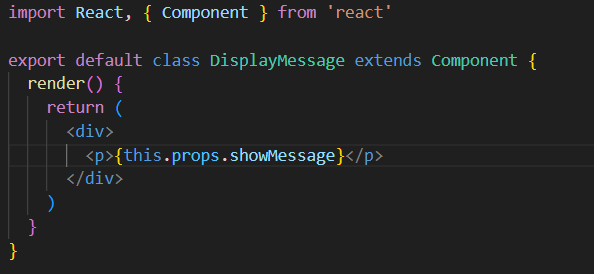
When a user enters a message and clicks the submit button, the handleSubmit method will be called, logging the message object to the console.



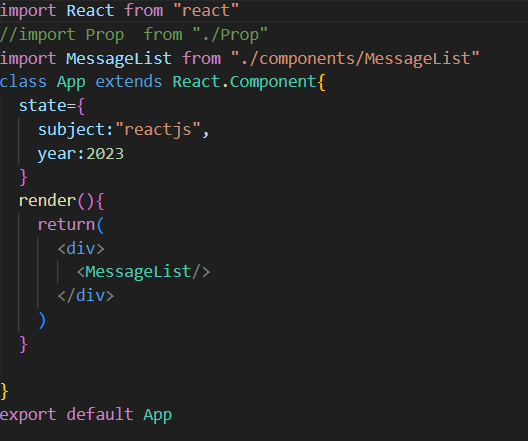
In the component's state, there is an array called messageList that represents the list of messages. The initial state is an empty array.

The addMessage method is used to add a new message to the messageList array. It takes a message object as an argument and updates the state by spreading the existing messageList array and appending the new message.

In the render method, the TakeMessage component is rendered, passing the addMessage method as the onSubmit prop. This allows the user to enter messages and submit them.

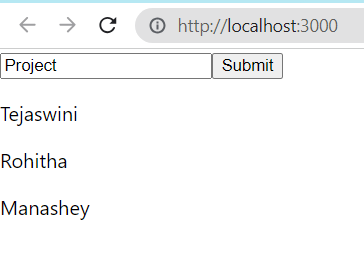
****

**App.js**

****

It imports React from the "react" module and the MessageList component from the "./components/MessageList" file.

**Output:**

****

**WEEKLY REPORT**

WEEK-4(From Dt 05-06-2023 to Dt 10-06-2023)

**TOPIC:** ExpressJS

**ACTIVITY LOG FOR FOURTH WEEK**

| DAY & DATE | BRIEF DESCRIPTION OF DAILY ACTIVITY | LEARNING OUTCOME | PERSON INCHARGE SIGNATURE |
| --- | --- | --- | --- |
| DAY-1 | * Introduction to ExpressJS * Installation of ExpressJS |  |  |
| DAY-2 | * Request & response Operations in ExpressJS |  |  |
| DAY-3 | * ExpressJS CRUD Operations and Routing |  |  |
| DAY-4 | * ExpressJS Middleware |  |  |
| DAY-5 | * ExpressJS Modules |  |  |
| DAY-6 | * Nodemon in ExpressJS |  |  |

**OBJECTIVES:**

* Introduction to ExpressJS
* ExpressJS Request & Response
* CRUD Operations and Routing
* Nodemon in ExpressJS

**DETAILED REPORT:**

**API:**

An API (Application Programming Interface) is a set of rules and protocols that allows different software applications to communicate with each other. It defines how requests for certain functionalities or data should be made and how responses will be formatted.

When you make a request to an API, the response you receive is typically formatted in JSON (JavaScript Object Notation). JSON data is structured as key-value pairs, similar to objects in JavaScript. The key is a string that represents a property name, and the value can be of various types, including strings, numbers, booleans, arrays, or even nested objects.

{

"key":value;

"key":value;

"key":value

}

RESTful APIs (Application Programming Interfaces) expose resources and allow clients to perform CRUD (Create, Read, Update, Delete) operations on those resources using HTTP methods.

Here's an overview of RESTful HTTP methods commonly used in REST API calls:

1. **GET:** The GET method is used to retrieve a representation of a resource or a collection of resources. It should not modify any data on the server. For example:

Retrieve a list of all users: GET /api/users

1. **POST:** The POST method is used to create a new resource on the server. It typically sends data in the request body, which will be used to create the resource. For example:

Create a new user: POST /api/users

Create a new blog post: POST /api/posts

1. **PUT:** The PUT method is used to update or replace an existing resource with the provided representation. It requires sending the entire updated representation of the resource. For example:

Update a user's profile: PUT /api/users/{id}

Replace an entire blog post: PUT /api/posts/{id}

1. **PATCH:** The PATCH method is used to partially update an existing resource with the provided changes. It allows sending only the modified fields without replacing the entire representation. For example:

Update a user's email address: PATCH /api/users/{id}

Update specific fields of a blog post: PATCH /api/posts/{id}

1. **DELETE:** The DELETE method is used to delete an existing resource. It removes the specified resource from the server. For example:

Delete a user: DELETE /api/user

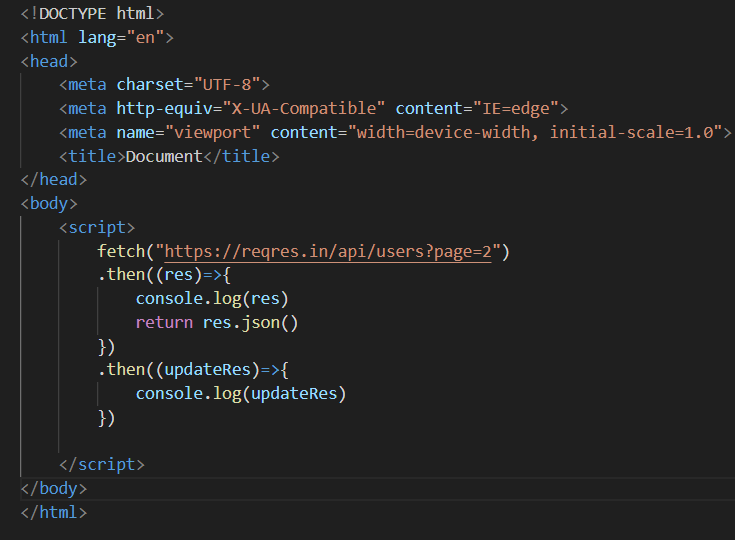
The server-side implementation of the RESTful API should handle these requests, validate input, perform necessary actions on the resources, and return appropriate responses, such as the requested data, success status codes, or error messages.

**Fetch and Promise:**

The fetch function and Promise are both JavaScript features used for making asynchronous requests and handling asynchronous operations.

1. **fetch**

The fetch function is a built-in JavaScript function used to make HTTP requests. It provides a simple and modern way to fetch resources, such as JSON data, from a server. The fetch function returns a Promise that resolves to the response object representing the server's response.

****

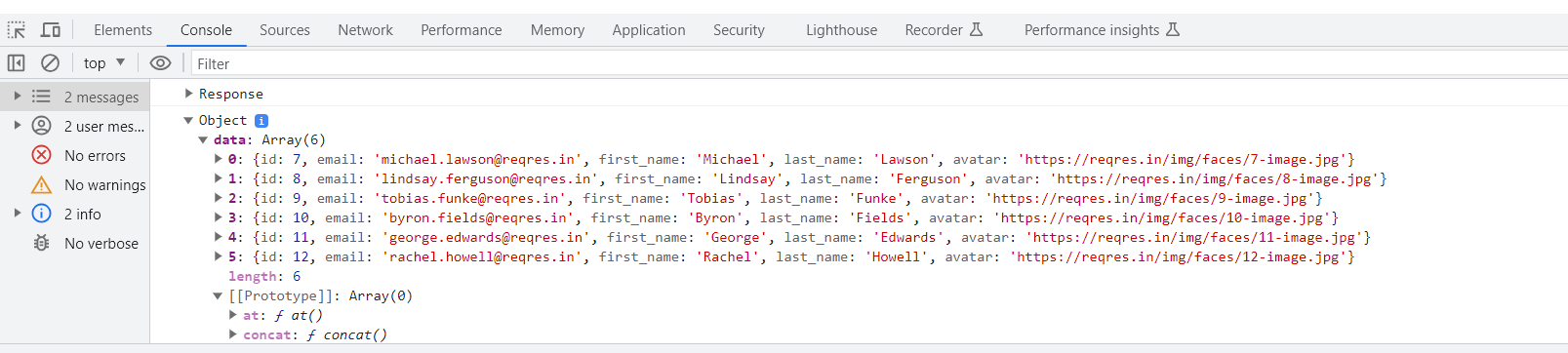
The fetch function is used to make an HTTP GET request to the specified URL, "https://reqres.in/api/users?page=2".

The fetch function returns a Promise, which is chained with a .then() method to handle the response. The first .then() block receives the response object as a parameter and logs it to the console using console.log().

Inside the first .then() block, another .then() block is chained to parse the response body as JSON using the .json() method. This returns another Promise that resolves to the parsed JSON data.

Finally, the second .then() block receives the parsed JSON data as a parameter and logs it to the console using console.log().

When you open the fetch.html file in a web browser and inspect the browser's console, you should see the logged response object and the parsed JSON data from the API request.

****

**Promise:**

A Promise is an object that represents the eventual completion (or failure) of an asynchronous operation and its resulting value. Promises provide a way to work with asynchronous code in a more readable and manageable manner.

Promises have three states:

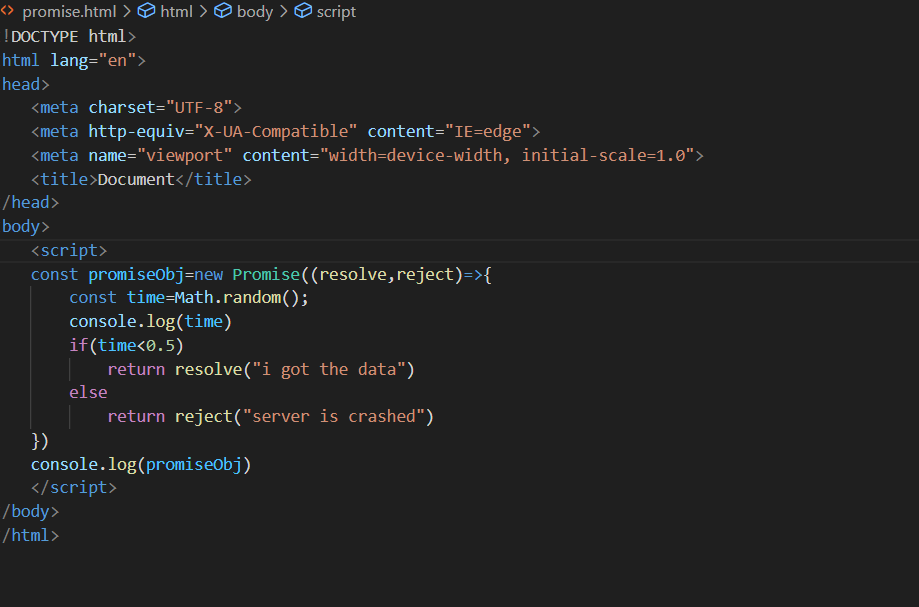
Pending: The initial state, representing an ongoing asynchronous operation.

Fulfilled: The state when the asynchronous operation is completed successfully, and the promise is resolved with a value.

Rejected: The state when the asynchronous operation encounters an error or failure, and the promise is rejected with a reason or error.

Promises are typically used with .then() and .catch() methods to handle the resolved or rejected state of the promise.

**Example:**



a Promise object named promiseObj is created using the Promise constructor. The Promise constructor takes a function (often referred to as the executor function) that is called immediately and receives two parameters: resolve and reject. These parameters are functions used to indicate the fulfillment or rejection of the promise.

Inside the executor function, a random number time is generated using Math.random(). The value of time is logged to the console using console.log ().

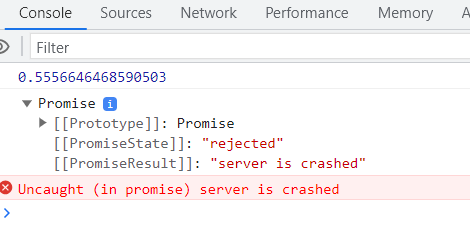
An if statement is used to check the value of time. If time is less than 0.5, the promise is fulfilled by calling resolve ("i got the data"). Otherwise, if time is greater than or equal to 0.5, the promise is rejected by calling reject ("server is crashed").

Finally, the promiseObj is logged to the console using console.log ().

When you open the HTML file in a web browser and inspect the browser's console, you will see the logged value of time and the promiseObj. The promiseObj represents the state of the Promise and can be in one of three states: pending, fulfilled, or rejected.

Depending on the randomly generated value of time, the promiseObj will either be fulfilled with the message "i got the data" or rejected with the message "server is crashed".

The example demonstrates the basic concept of a Promise and how it can be used to handle asynchronous operations that may succeed or fail.

****

The value obtained is 0.555 and is greater than 0.5 so the message “server is crashed” is obtained.

**WEEKLY REPORT**

WEEK-5 (From Dt 12-06-2023 to Dt 17-06-2023)

**TOPIC:** NodeJS

**ACTIVITY LOG FOR FIFTH WEEK**

| DAY & DATE | BRIEF DESCRIPTION OF DAILY ACTIVITY | LEARNING OUTCOME | PERSON INCHARGE SIGNATURE |
| --- | --- | --- | --- |
| DAY-1 | * Introduction to NodeJS * Installation of NodeJS |  |  |
| DAY-2 | * NodeJS Modules * Application of NodeJS |  |  |
| DAY-3 | * Request & response Operations in NodeJS |  |  |
| DAY-4 | * Response Status Codes |  |  |
| DAY-5 | * Packages in NodeJS |  |  |
| DAY-6 | * Difference Between Node and Express |  |  |

**OBJECTIVES:**

* Introduction to NodeJS
* NodeJS Modules
* Request and Response operations in NodeJS
* fetch and axios packages

**DETAILED REPORT:**

**What is NodeJS?**

* Node.js is an open-source server environment
* Node.js is free
* Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
* Node.js uses JavaScript on the server

## **Why Node.js?**

Node.js uses asynchronous programming!

**request and response operations:**

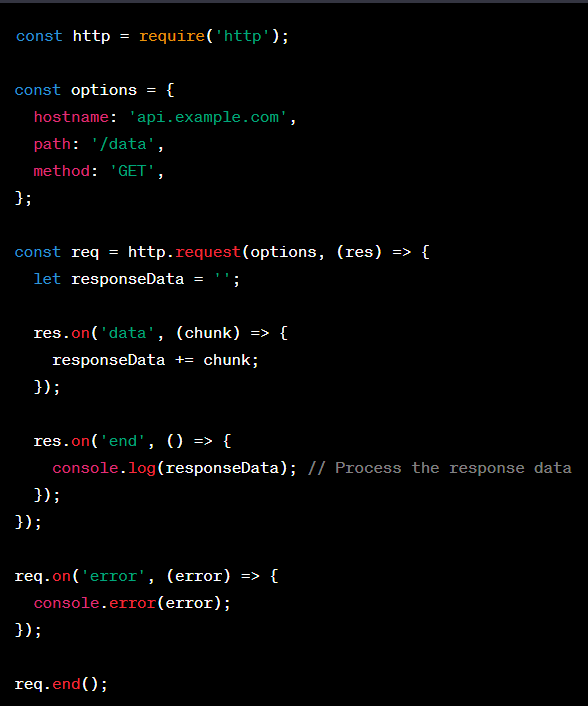
In Node.js, you can perform request and response operations using the built-in http module or third-party libraries such as axios or node-fetch. Let's explore the basic usage of both approaches:

**Using the http module (built-in):**

The http module provides functionality to create an HTTP server, send HTTP requests, and handle HTTP responses.

Here's an example of making an HTTP GET request using the http module:

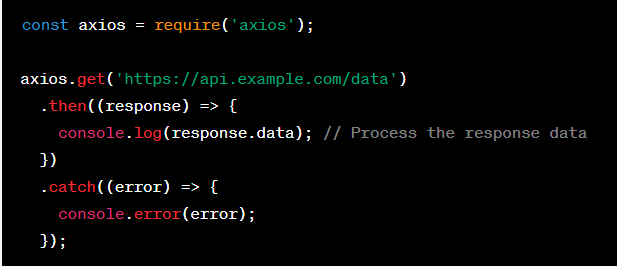
In this example, an HTTP GET request is made to 'api.example.com/data'. The response data is received in chunks using the 'data' event and concatenated to responseData. When the response ends ('end' event), the accumulated data is logged.



**Using third-party libraries (axios or node-fetch):**

Third-party libraries such as axios or node-fetch provide a more convenient and flexible way to make HTTP requests in Node.js, handling request and response operations with simpler APIs and additional features.

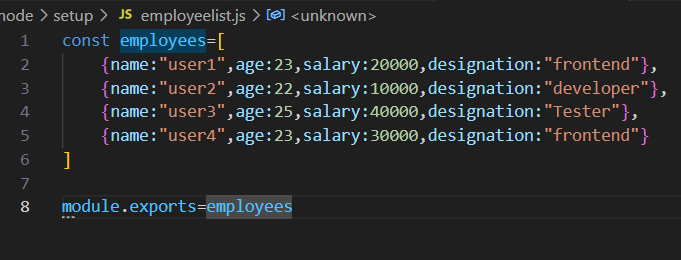
Here's an example using the axios library to make an HTTP GET request:

****

In this example, the axios.get() method is used to make an HTTP GET request to 'https://api.example.com/data'. The response data is accessed through response.data.

**Task:**

**Employeelist.js**

****

**Index.js:**

****

CORS stands for Cross-Origin Resource Sharing. It is a mechanism that allows resources (such as web fonts, JavaScript, or APIs) on a web page to be requested from another domain outside the domain from which the resource originated. CORS helps to enable cross-domain communication while maintaining the security of the web browser.

The body-parser middleware is added to parse incoming request bodies in JSON format.

The server defines three routes:

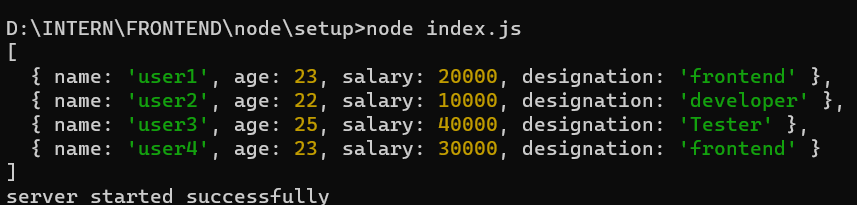
The /bring route responds with the message "hai good morning everyone" when accessed.

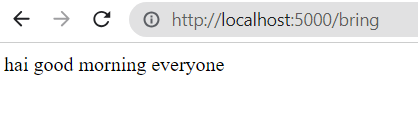
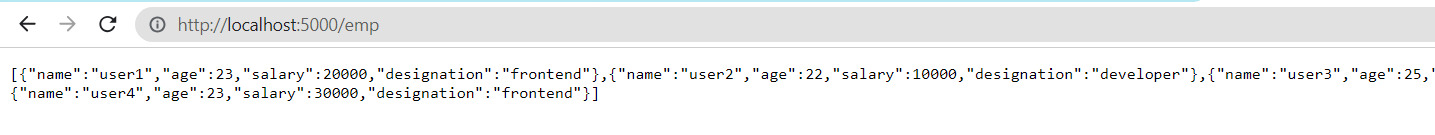
The /emp route returns the JSON data of the employees array imported from the employeelist.js file.

The /store-employee route handles a POST request and logs the username and password values from the request body.

The server listens on port 5000, and when it starts successfully, the callback function logs a message.

**cmd:**

****

**** ****

**WEEKLY REPORT**

WEEK-6 (From Dt 19-06-2023 to Dt 24-06-2023)

**TOPIC:** MongoDB

**ACTIVITY LOG FOR SIXTH WEEK**

| DAY & DATE | BRIEF DESCRIPTION OF DAILY ACTIVITY | LEARNING OUTCOME | PERSON INCHARGE SIGNATURE |
| --- | --- | --- | --- |
| DAY-1 | * Introduction to MongoDB |  |  |
| DAY-2 | * MongoDB With ReactJS |  |  |
| DAY-3 | * Database in MongoDB |  |  |
| DAY-4 | * ReactJS with Database |  |  |
| DAY-5 | * MySql * Firebase |  |  |
| DAY-6 | * MongoDB |  |  |

**OBJECTIVES:**

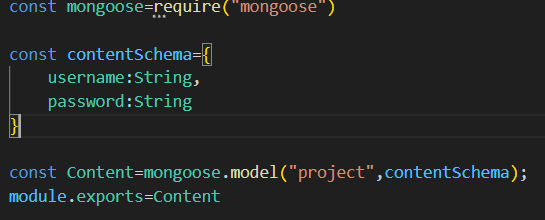
* Introduction to MongoDB
* Creating database with MongoDB
* Connecting MongoDB with ReactJS

**DETAILED REPORT:**

1.To create a database using MongoDB, create an account in MongoAtlas.

2.Create a new project and a new database. A new cluster will be created.

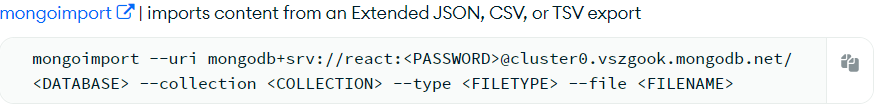
3.To insert the data in database, firstly a schema has to be defined. Open vs code and create a file in your project folder.

****

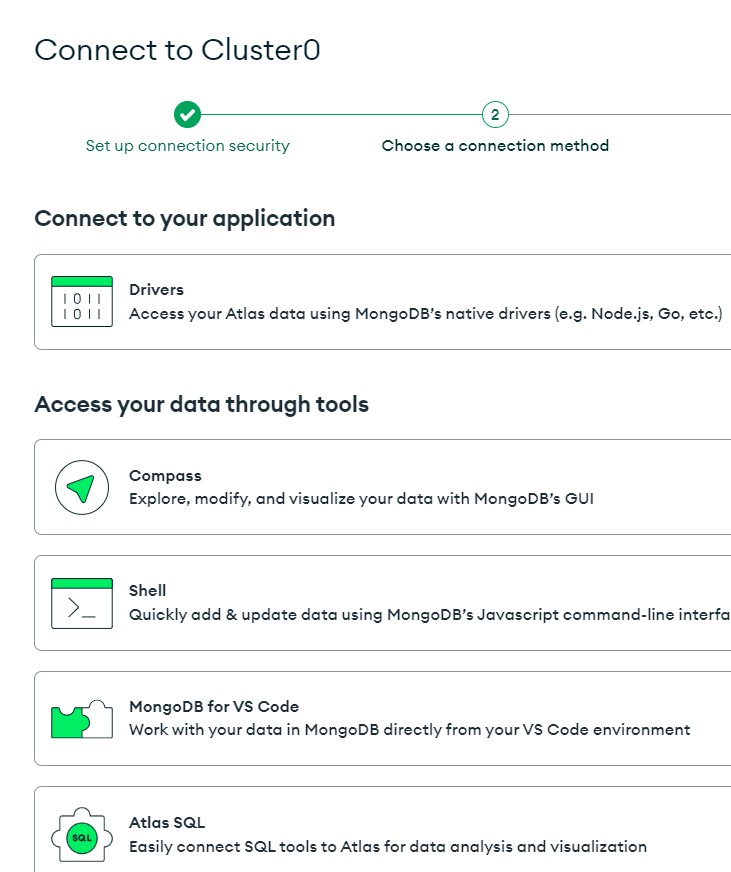
4.This is the schema in which the database stores the username and password. It creates a collection named “project”

5.Install mondodb by “npm install mongodb.”

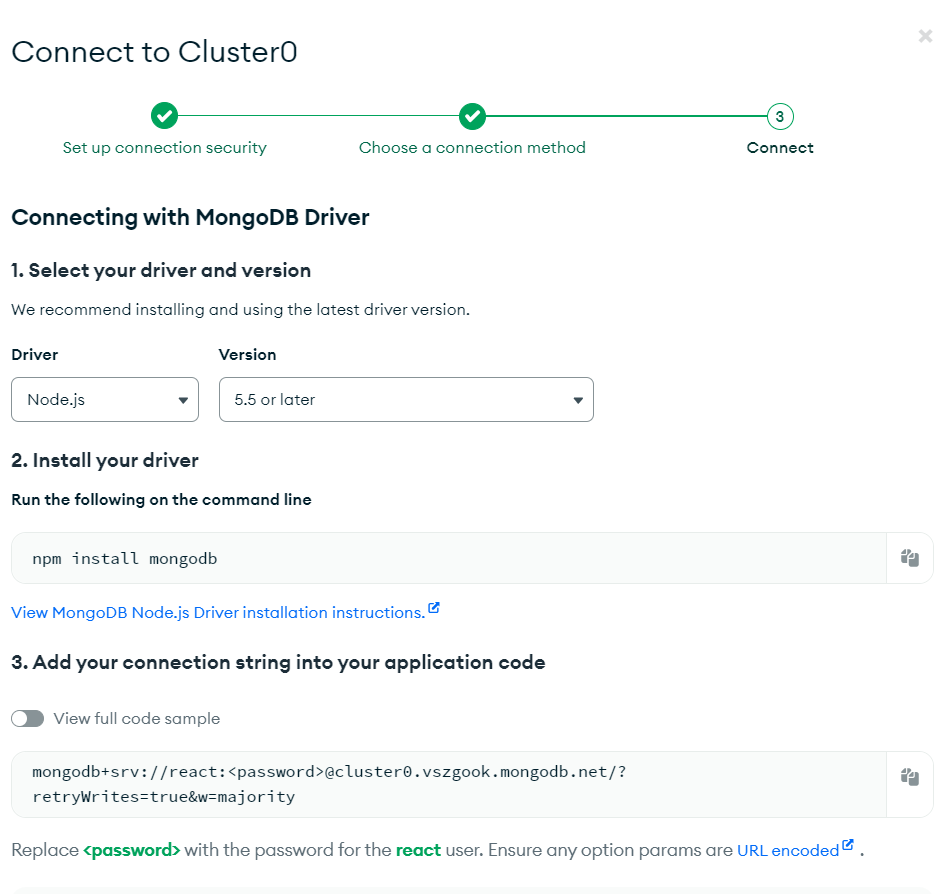
6.If we want to import the existing database collection to the mongodb, then copy the following command:

****

Now, to connect the cluster to our project. Click on the connect button and then a window appears.

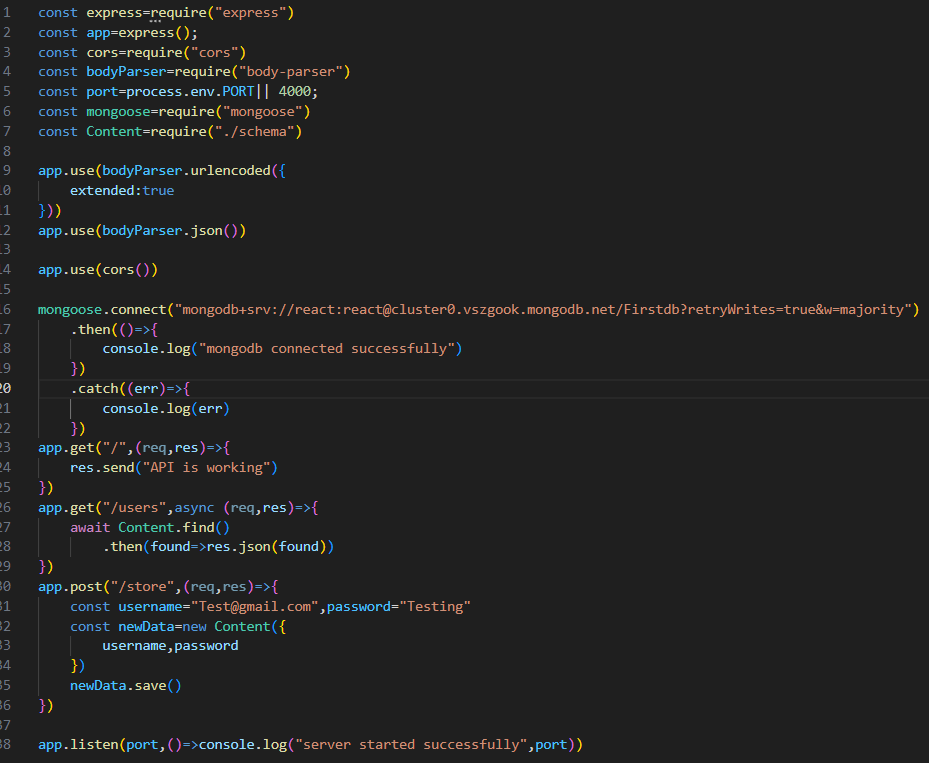


**7.Click on the drivers. Then select the driver which is usually node.js and also choose the version.**

****

8.At last, to connect this mongodb cluster to our application code, copy the url and replace our cluster password in the place of <password> and enter our database name before the ‘?’. After replacing these the URL becomes:

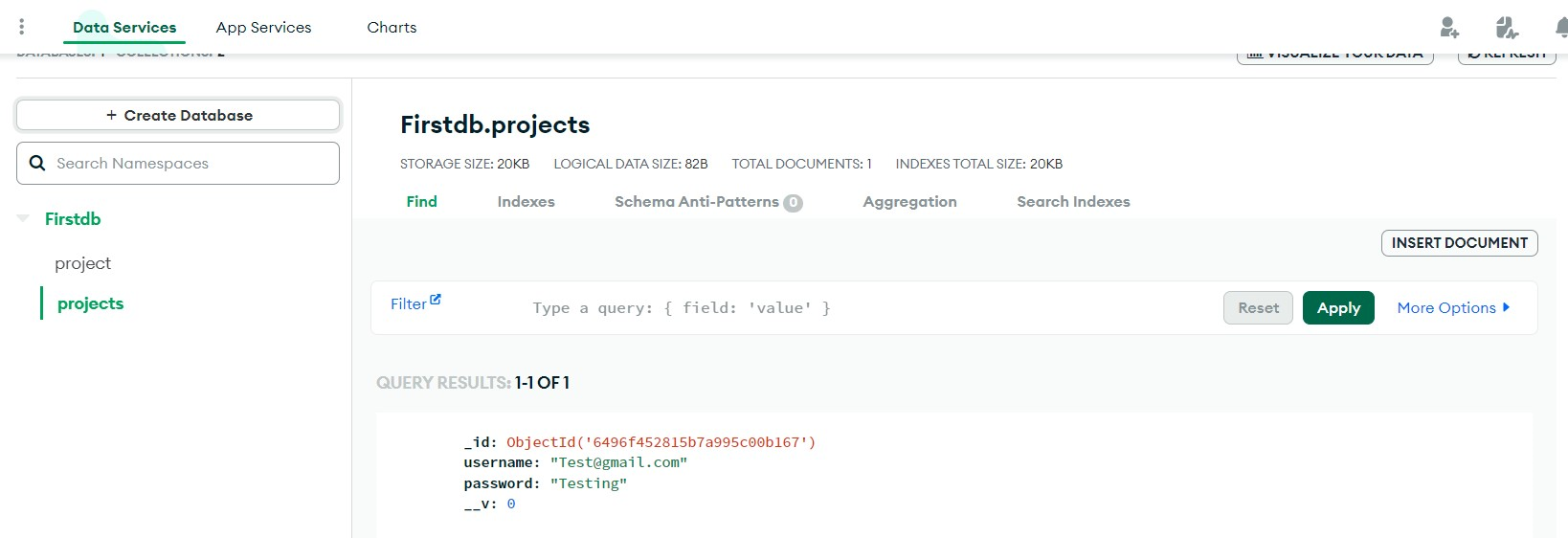
Mongodb+srv://react:react@cluster0.vszgook.mongodb.net/Firstdb?retryWrites=true&w=majority

****

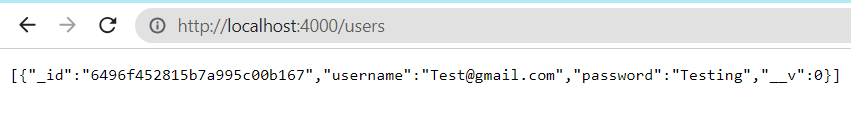
9. MongoDB is connected using mongoose.connect(). We provided the connection URL ("mongodb+srv://...") and any required credentials. If the connection is successful, a success message is logged; otherwise, an error message is logged.

10. We sent a post request to store the credentials. The "/store" route creates a new document in the Content collection with a predefined username and password. The document is saved to the database using newData.save().

11. Here we are giving the username “[test@gmail.com](mailto:test@gmail.com)” and password “Testing”. Now check the database. Open the MongoDB Atlas and go to your respective cluster and Click on the “Collection” to view the username and password we stored. Each query is assigned with some unique id.



12. In the above code, "/users" route performs an asynchronous operation to fetch all documents from the Content collection using content.find(). The retrieved documents are then sent as a JSON response.

****

This is the data stored in the database.

**WEEKLY REPORT**

WEEK-7&8 (From Dt 26-06-2023 to Dt 08-07-2023)

**PROJECT**

**AN APPLICATION TO ORDER FOOD FROM RESTAURANTS**



**ABSTRACT**

**AN APPLICATION TO ORDER FOOD FROM RESTAURANTS**

The project aims to develop a user-friendly web application that facilitates the convenient ordering of food from various restaurants. This application provides a platform where users can browse through a comprehensive list of participating restaurants, view their menus, and place food orders directly from their mobile devices. The development of the application involves a combination of front-end and back-end technologies. The front-end interface is designed to be intuitive and visually appealing, ensuring that users can navigate through the app effortlessly. The back-end infrastructure employs robust database management systems and secure API integrations to facilitate seamless communication between the application and the participating restaurants. The MERN stack is chosen for its advantages in building dynamic and efficient web applications. MongoDB serves as the database to store user and restaurant information, while Express.js handles the backend logic and API development. React.js, a powerful frontend library, provides a responsive and interactive user interface, and Node.js powers the server-side functionality.

The project follows an agile development approach, incorporating regular testing, user feedback, and iterative improvements. By leveraging the MERN stack's capabilities, this application aims to provide users with a seamless and intuitive platform to order food from restaurants, enhancing convenience, choice, and overall user satisfaction.

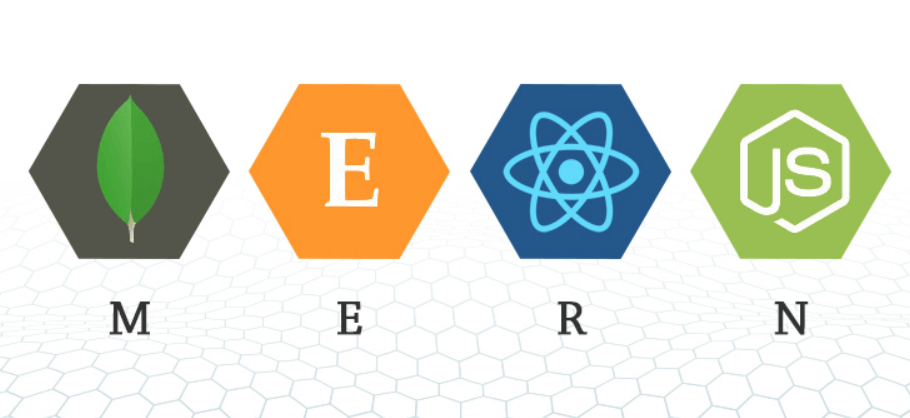
To enhance the user experience, the application will feature a user-friendly interface with intuitive navigation, visually appealing menus, and clear order tracking. Users will be able to know the items in his cart along with that he can see the list of items he has order over the time.

By developing this application, the project aims to revolutionize the way people order food, providing them with a convenient, personalized, and efficient solutionThe application will save users time and effort, allowing them to explore a wide range of culinary options, place orders effortlessly, and enjoy the convenience of doorstep delivery. By bridging the gap between restaurants and customers, this application aims to enhance the overall food ordering ecosystem and contribute to the growth of the online food delivery industry

# INTRODUCTION

**MERN STACK**

The MERN stack is a popular web development framework that combines four key technologies: MongoDB, Express.js, React.js, and Node.js. Each component of the MERN stack plays a specific role in building robust and scalable web applications.



**MongoDB** is a NoSQL database that stores data in a flexible, JSON-like format. It provides a scalable and efficient solution for managing large amounts of data and supports dynamic schemas, making it suitable for agile development processes.

**Express.js** is a minimalist web application framework for Node.js that simplifies the process of building server-side applications. It provides a set of robust features for handling routes, middleware, and HTTP requests, allowing developers to create APIs and handle server-side logic effectively.

**React.js** is a JavaScript library for building user interfaces. It enables the creation of interactive and reusable UI components, making it easier to build complex front-end applications. React.js follows a component-based approach, promoting code reusability, maintainability, and efficient rendering.

**Node.js** is a JavaScript runtime environment that allows developers to run JavaScript on the server side. It provides an event-driven, non-blocking I/O model, making it highly scalable and efficient for building server-side applications. Node.js enables developers to handle concurrent connections, perform file operations, and interact with databases.

The MERN stack offers several advantages for web development projects. It allows for a consistent and unified development experience, as all components use JavaScript, making it easier for developers to work across the entire application stack. The stack also provides a full-stack JavaScript solution, reducing the need for developers to switch between different languages or frameworks.

Furthermore, the MERN stack supports a reactive and component-based development approach, which improves code organization, reusability, and testability. The real-time capabilities of Node.js enable the creation of responsive and dynamic web applications.

**1.2 MOTIVATION**

In recent years, online food ordering and delivery have experienced a significant surge due to changes in consumer behaviour, convenience, and the availability of smartphones. This growing market demand presents a remarkable opportunity for businesses to capitalize on the trend by providing a seamless and user-friendly platform for customers to order food from their favourite restaurants. This project aimed to develop a user-friendly and efficient web application that allows customers to order food from various restaurants with ease.

**1.3 PROBLEM STATEMENT**

The aim of this project was to create a comprehensive and user-friendly web application that allows customers to conveniently order food from their favourite restaurants. By leveraging the power and flexibility of the MERN stack, the project aimed to provide seamless integration between the frontend and backend, efficient data management, and a responsive user interface. The insights gained from this project will serve as a valuable resource for future developers and stakeholders interested in leveraging the MERN stack for similar projects in the domain of food delivery and e-commerce as well.