VELAMMAL COLLEGE OF ENGINEERING AND TECHNOLOGY

VIRAGANOOR, MADURAI – 625 009

AUTONOMOUS

Department of Information Technology

VCET Regulation 2021

21AD305 – Full Stack Development Laboratory

Lab Manual

Approved by
HoD/ IT
Dr. R. Kavitha, Prof

Prepared by
Ms. P. Loganayagi, AP-III/IT

Scanned Syllabus

Table of contents

Ex. No	Name of the Exercise	Page Number
1	Design Webpages for any given application	5
2	Write Server- side programming with Node.js	9
3	Perform Email applications using Node mailer Module	12
4	Write custom applications with Node.js and Mongo DB	16
5	Use React components, JSX, Class, Prop Events	23
6	Write custom applications Forms with React	26
7	Use Type Script for enhancing web application	29
8	Apply use Callback, use State, use Effect, use Ref Hook of React to applications	32
9	Use Web Pack for Application	35
10	Bind server and client side and deploy as a deliverable application	39
11	Deploy applications to Docker Hub	43

Ex No 1 **DESIGN WEBPAGES FOR AN APPLICATIONS**

Date:

Aim

To **design and develop responsive, interactive, and user-friendly web pages** for a given application using **HTML, CSS, and JavaScript**, ensuring proper layout, navigation, and usability that align with the application's functional and aesthetic requirements.

Algorithm

```
Step 1 : Start
Step 2 : Analyze the Requirements
            a. Identify the purpose of the application
            b. Determine the target audience
            c. List out the required features and pages (e.g., Home, About, Contact, Login)
Step 3 : Design the Layout
        a. Sketch basic structure/layout of each web page
        b. Decide the layout style (grid, flexbox, etc.)
        c. Plan navigation and links between pages
Step 4 : Create HTML Structure
        a. Use appropriate semantic HTML tags (<header>, <nav>, <main>, <footer>, etc.)
        b. Add content like headings, paragraphs, images, forms, and tables
        c. Link necessary internal/external resources (CSS, JS)
Step 5 : Apply Styling using CSS
            a. Design an external CSS file or use internal styles
            b. Style layout, typography, color schemes, spacing, and responsiveness
            c. Use media queries for responsive design
Step 6 : Add Interactivity with JavaScript
            a. Implement validation for forms (e.g., login, signup)
            b. Add dynamic effects like sliders, modals, or dropdowns
            c. Connect UI events (click, submit, hover) to functions
Step 7 : Test the Web Pages
            a. Open the web pages in multiple browsers
            b. Check for responsiveness on different devices (desktop, mobile, tablet)
            c. Fix layout or scripting issues
```

Step 8 : Deploy the Application (Optional)

a. Upload the files to a local or cloud server (e.g., GitHub Pages, Netlify)

b. Test live deployment

Step 9 : Stop

Concept Involved

HTML 5

HTML5 is the **fifth version** of **Hypertext Markup Language (HTML)**, a standard language used to **structure webpages.** It defines how content on a **webpage** should be **structured** and **displayed.** Here are some key points of HTML5

- **Multimedia Support:** Embeds audio and video without plugins.
- **New Form Controls:** Includes input types like date and email.
- **Web Storage:** Stores data offline for better performance.
- **Semantic Elements:** Uses tags like <header> and <footer> for better structure.
- **Improved Performance:** Faster and more efficient, especially on mobile.

Example:

```
<html>
<head>
  <meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Simple HTML Page</title>

</head>
<body>

 <header>
         <h1>Hello, World!</h1>
 </header>

<main>
  <p>This is a simple HTML5 page.</p>
</main>

 <footer>
           <p>Sanchhaya Education Private Limited, All rights reserved</p>
  </footer>
</body>
</html>
```

CSS - Cascading Style Sheet

CSS stands for **Cascading Style Sheets.** It is a stylesheet language used to style and enhance website presentation. CSS is one of the three main components of a webpage, along with **HTML** and **JavaScript**.

- HTML adds Structure to a web page.
- JavaScript adds logic to it and CSS makes it visually appealing or stylish. It controls the layout of a web page i.e. how HTML elements will be displayed on a webpage.

CSS was released (in 1996), 3 years after HTML (in 1993). The main idea behind its use is that it allows the separation of content (HTML) from presentation (CSS). This makes websites easier to maintain and more flexible.

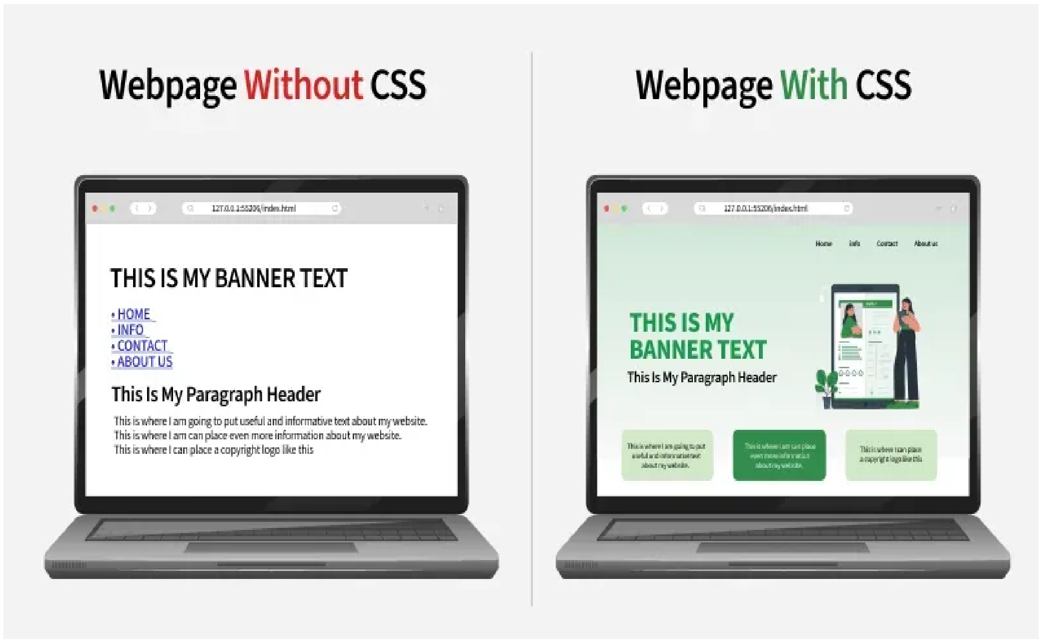


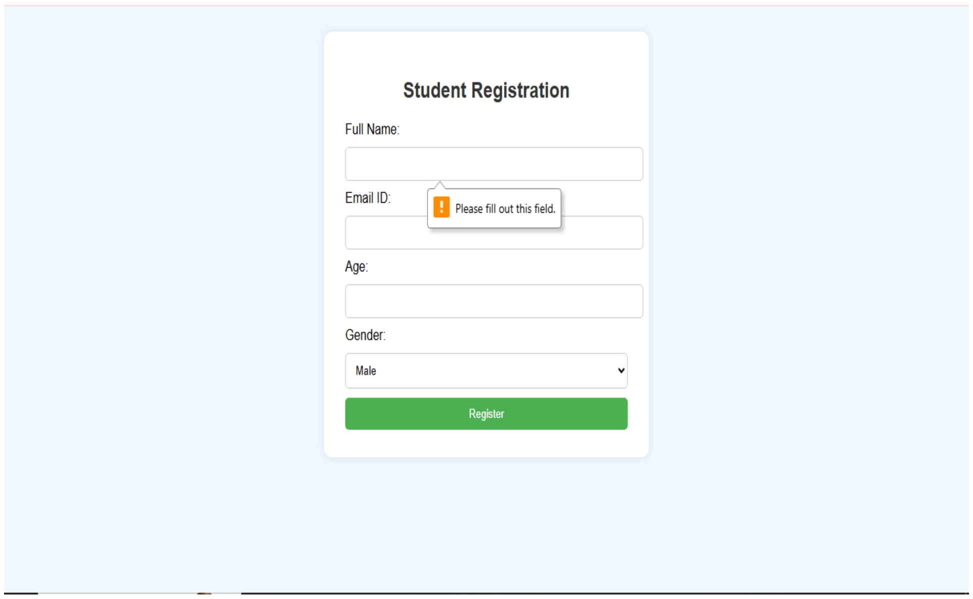
Fig 1.1

JavaScript

JavaScript is a programming language used to create dynamic content for websites. It is a lightweight, cross-platform, and single-threaded programming language. It's an interpreted language that executes code line by line, providing more flexibility**.**

- **JavaScript on Client Side:** On the client side, JavaScript works along with HTML and CSS. HTML adds structure to a web page, CSS styles it, and JavaScript brings it to life by allowing users to interact with elements on the page, such as actions on clicking buttons, filling out forms, and showing animations. JavaScript on the client side is directly executed in the user's browser.
- **JavaScript on Server Side**: On the Server side (on Web Servers), JavaScript is used to access databases, file handling, and security features to send responses**,** to browsers.

Output:



Result

The webpage was successfully designed and displayed correctly in the browser using HTML, CSS and JavaScript.

Ex No 2 **WRITE SERVER SIDE PROGRAMMING WITH NODE.JS**

Date:

Aim

To develop a basic server-side application using Node.js that handles client requests and sends appropriate responses.

Algorithm

Step 1: Import the http module.
Step 2: Create a server using http.createServer().
Step 3: Handle client requests based on the request URL.
Step 4: Send appropriate responses using res.write() and res.end().
Step 5: Start the server using server.listen() on a specified port.
Step 6: Run the server and test it in a browser or Postman.

Concepts Involved

Node.js Environment

The Node.js environment refers to the runtime in which Node.js applications execute. It is a free, open-source, and cross-platform JavaScript runtime environment that allows developers to build server-side applications, web applications, command-line tools, and scripts using JavaScript outside of a web browser.

V8 JavaScript Engine:

Node.js utilizes the V8 JavaScript engine, the same engine that powers Google Chrome. This allows Node.js to execute JavaScript code efficiently and provides high performance.

Event-Driven, Non-Blocking I/O:

Node.js operates on an event-driven, non-blocking I/O model, which makes it highly efficient and scalable, especially for applications handling many concurrent connections.

NPM (Node Package Manager):

NPM is the default package manager for Node.js, providing a vast ecosystem of open-source libraries and tools that developers can easily integrate into their projects.

Environment Variables:

Node.js applications often use environment variables (accessible via process.env) to manage configuration settings and sensitive data, such as API keys or database credentials, separately from the codebase. This promotes security and allows for easy adaptation to different deployment environments (development, staging, production).

Core Modules:

Node.js provides a set of built-in core modules for various functionalities, including file system operations, networking, and more.

Express.js

Express.js, commonly referred to as Express, is a minimal and flexible Node.js web application framework designed for building web applications and APIs. It is often considered the de facto standard server framework for Node.js due to its popularity and widespread use.

Example:

```
// Import Express
const express = require('express');
const app = express();

// Define a route
app.get('/', (req, res) => {
    res.send('Welcome to the Express.js Tutorial');
});

// Start the server
app.listen(3000, () => {
    console.log('Server is running on http://localhost:3000');
});
```

Output:

```
Welcome to the Express.js Tutorial
```

HTTP Methods

HTTP methods, also known as HTTP verbs, define the type of action a client wants to perform on a resource identified by a URL on a web server. These methods are a fundamental part of the Hypertext Transfer Protocol (HTTP) and are essential for interacting with web services and APIs.

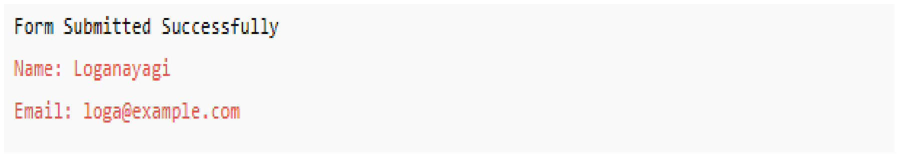
Method	Description
GET	The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
HEAD	Same as GET, but transfers the status line and header section only.

POST	A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
PUT	Replaces all current representations of the target resource with the uploaded content.
DELETE	Removes all current representations of the target resource given by a URI.
CONNECT	Establishes a tunnel to the server identified by a given URI.
OPTIONS	Describes the communication options for the target resource.
TRACE	Performs a message loop-back test along the path to the target resource.

Routing in Express

Routing in Express.js is the mechanism by which the application determines how to respond to a client's request for a specific URL (endpoint) and HTTP method (like GET, POST, etc.). Essentially, it's how your application decides what code to execute when a user accesses a particular page or submits data through a form.

Output:



```
Form Submitted Successfully
Name: Loganayagi
Email: loga@example.com
```

RESULT:

The server-side application using Node.js was successfully executed. The form data submitted by the user was received, processed, and a response was displayed in the browser, confirming successful data handling. This confirms that the server is functioning correctly and handling HTTP requests as expected.

Ex No 3 PERFORM EMAIL APPLICATIONS USING NODEMAILER MODULE

Date:

Aim

To create an email-sending application using Node.js and Nodemailer, which can send emails automatically through Gmail in a secure and easy way.

Algorithm

Step 1 : Start the program.
Step 2 : Import required modules: nodemailer and dotenv.
Step 3 : Load environment variables using dotenv to secure email credentials.
Step 4 : Create a transporter object using nodemailer.createTransport() with Gmail SMTP settings.
Step 5 : Define mail options like sender address, receiver address, subject, and message body.
Step 6 : Send the email using transporter.sendMail() with the defined mail options.
Step 7 : Check for errors or success in the callback function.
Step 8 : Display the status of the email sent operation on the console.
Step 10: Stop the program.

Concepts Involved

Node.js

Node.js uses an **event-driven**, **non-blocking** model.
It can handle many connections at once without waiting for one to finish before starting another.
This makes it great for real-time apps and high-traffic websites.
Here are some examples of what you can build with Node.js:

- Web servers and websites
- REST APIs
- Real-time apps (like chat)
- Command-line tools
- Working with files and databases
- IoT and hardware control

Node Mailer

- Nodemailer is a module for Node.js that simplifies sending emails.
- It allows you to send emails from your application using various email services and transport protocols like SMTP.
- To use Nodemailer, you need to install it via npm and configure a transporter with your email service credentials.
- Then, you define the email content and use the transporter's sendMail method to send the email.

Approach

To send email with Nodemailer using gmail

- Import the nodemailer module.
- Use **nodemailer.createTransport()** function to create a transporter who will send mail. It contains the service name and authentication details (user ans password).
- Declare a variable **mailDetails** that contains the **sender and receiver email id**, subject and content of the mail.
- Use **mailTransporter.sendMail()** function to send email from sender to receiver. If message sending failed or contains error then it will display error message otherwise message send successfully.

Procedure

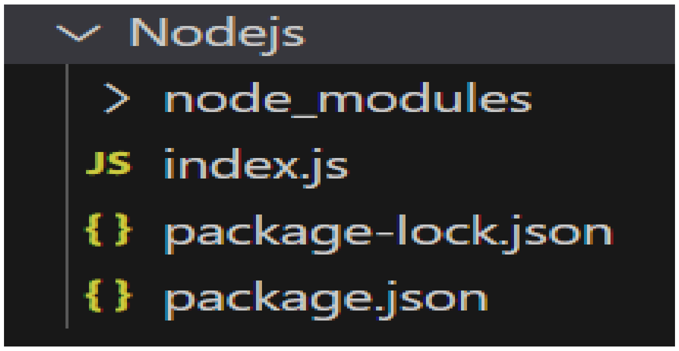
1. Install Node.js and npm.
2. Create a folder for the project and initialize it with `npm init -y`.
3. Install required modules:

```
npm install nodemailer dotenv
```

4. Create a `.env` file to store email and app password securely.
5. Write the code in `index.js` to send an email.
6. Run the script using:

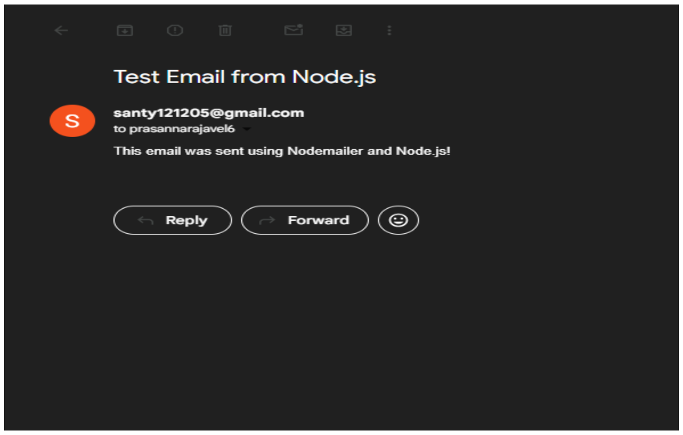
```
node index.js
```

Project Structure



```
Nodejs
  > node_modules
  index.js
  package-lock.json
  package.json
```

Output



Test Email from Node.js

santy121205@gmail.com
to prasannarajavel6
This email was sent using Nodemailer and Node.js!

Result

Thus, the email application using Node.js and Nodemailer was successfully created and tested

Ex No 4 WRITE CUSTOM APPLICATIONS WITH NODE.JS AND MONGO DB

Date:

Aim

To design and develop a custom CRUD application using Node.js as backend and MongoDB.

Algorithm

Step 1: Start
Step 2: Initialize Node.js Project
- Use the command: npm init -y
- Install required packages: npm install express mongoose dotenv

Step 3: Create Project Structure
- Create folders:
  - models/ → to define schema
  - routes/ → to define API routes
  - controllers/ *(optional)* → for logic separation

Step 4: Configure MongoDB Connection
- Create .env file to store connection string
- Connect MongoDB using mongoose.connect() in the main file

Step 5: Define Data Schema
- In models/, create a Mongoose schema with relevant fields
- Export the schema model

Step 6: Create RESTful API Endpoints
- Define CRUD routes:
  - **POST** for inserting data
  - **GET** for retrieving data
  - **PUT** for updating data
  - **DELETE** for deleting data

Step 7: Use Express Middleware
- Use express.json() to parse incoming JSON data

Step 8: Handle Database Operations
- Use Mongoose methods like save(), find(), findByIdAndUpdate(), findByIdAndDelete() inside route functions

Step 9: Start the Server
- Use app.listen(PORT) to run the server

Step 10: Test the Application
- Use Postman or any API testing tool to test the endpoints

Step 11: Stop

Concepts Involved

Node.js

Node.js is a powerful, open-source, and cross-platform JavaScript runtime environment built on Chrome's V8 engine. It allows you to run JavaScript code outside the browser, making it ideal for building scalable server-side and networking applications.

- ➢ JavaScript was mainly used for frontend development earlier. With Node.js (Introduced in 2009), JavaScript became a backend language as well.
- ➢ Non-blocking, event-driven architecture for high performance.
- ➢ Supports the creation of REST APIs, real-time applications, and microservices.
- ➢ Comes with a rich library of modules through npm (Node Package Manager).

What is Node.js?

- ➢ Node.js is not a programming language like Python, Java or C/C++. Node.js is a runtime, similar to Java virtual machine, that converts JavaScript code into machine code. It is , widely used by thousands of developers around the world to develop I/O intensive web applications like video streaming sites, single-page applications, and other web applications.
- ➢ With Node.js, it is possible to use JavaScript as a backend. With JavaScript already being a popular choice for frontend development, application development around MERN (MongoDB, Express, React and Node.js.) and MEAN (MongoDB, Express, Angular and Node.js) stacks is being increasingly employed by developers.

What Can You Build With Node.js?

- ➢ Node.js uses an **event-driven**, **non-blocking** model.
- ➢ It can handle many connections at once without waiting for one to finish before starting another. This makes it great for real-time apps and high-traffic websites. Here are some examples of what you can build with Node.js:
    - Web servers and websites
    - REST APIs
    - Real-time apps (like chat)
    - Command-line tools
    - Working with files and databases
    - IoT and hardware control

Applications of Node.js

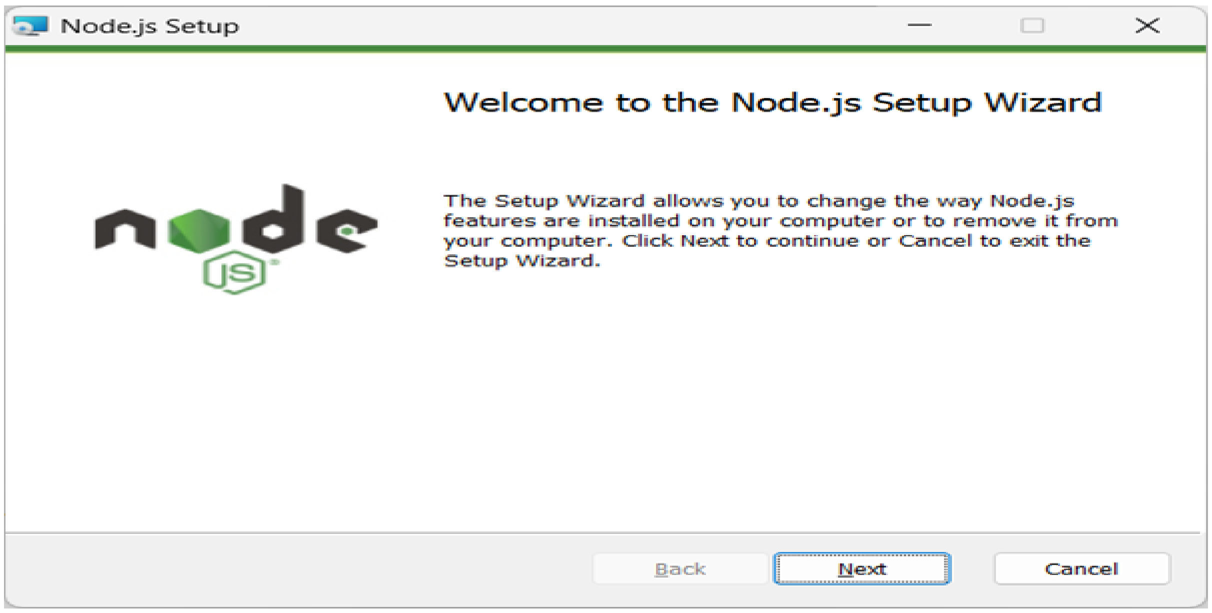
Node.js is used for building different type of applications. Some of the application types are listed below.

- **Streaming applications:** Node.js can easily handle real-time data streams, where it is required to download resources on-demand without overloading the server or the user's local machine. Node.js can also provide quick data synchronization between the server and the client, which improves user experience by minimizing delays using the Node.js event loop.
- **Single page apps:** Node.js is an excellent choice for SPAs because of its capability to efficiently handle asynchronous calls and heavy input/output(I/O) workloads. Data driven SPAs built with Express.js are fast, efficient and robust.
- **Realtime applications:** Node.js is ideal for building lightweight real-time applications, like messaging apps interfaces, chatbot etc. Node.js has an event- based architecture, as a result has an excellent WebSocket support. It facilitates real-time two-way communication between the server and the client.
- **APIs:** At the heart of Node.js is JavaScript. Hence, it becomes handling JSON data is easier. You can therefore build REST based APIs with Node.js.

These are some of the use cases of Node.js. However, its usage is not restricted to these types. Companies are increasingly employing Node.js for variety of applications.

Node.js Environment Setup

Assuming that you are working with Windows 10/Windows 11 powered computer, download the 64-bit installer for Windows: https://nodejs.org/dist/v20.9.0/node-v20.9.0-x64.msi, and start the installation b double-clicking the downloaded file.


Node.js Setup

Welcome to the Node.js Setup Wizard

The Setup Wizard allows you to change the way Node.js features are installed on your computer or to remove it from your computer. Click Next to continue or Cancel to exit the Setup Wizard.

The installation takes you through a few steps of the installation wizard. It also adds the installation directory of Node.js executable to the system path.
To verify if Node.js has been successfully installed, open the command prompt and type node -v. If Node.js is installed successfully then it will display the version of the Node.js installed on your machine.

Mongo DB

The **MongoDB** is an open-source document database and leading NoSQL database. MongoDB is written in C++. This tutorial will give you great understanding on MongoDB concepts needed to create and deploy a highly scalable and performance-oriented database. MongoDB features are flexible data models that allows the storage of unstructured data. This provides full support indexing, replication, capabilities and also user friendly APIs. The **MongoDB** is a multipurpose dataset that is used for modern application development and cloud environments. This scalable architecture enables us to handle system demands and also adding more nodes to distribute the load.

MongoDB Basic Commands

We have a list of standard MongoDb commands to interact with the database, These commands are CREATE, READ, INSERT, UPDATE, DELETE, DROP and AGGREGATE can be classified into following groups based on their nature −

Command	Description
CREATE	Creates a new table in the database and other objects in the database.
INSERT	Inserts collection name in existing database.
DROP	Deletes an entire table or specified objects in the database.
UPDATE	Updates the document into a collection.

Postman or Web Client

Postman and WebClient are both tools for interacting with APIs, but they serve different purposes and operate at different levels. Postman is a standalone API client, primarily used for testing and exploring APIs through a graphical user interface. WebClient, on the other hand, is a reactive HTTP client library built into Spring Boot, used within applications to make API calls programmatically.

Postman:

- Purpose: Primarily for API testing and exploration, offering a user-friendly interface to send requests, view responses, and manage collections of API calls.
- Level: User-level tool, separate from application code.
- Features: Graphical user interface, request building, response inspection, collections, test scripting, and collaboration features.
- Pros: Easy to use for testing, helps with debugging, good for exploring APIs.
- Cons: Can be resource-intensive with large collections, GUI-centric (limited CLI features).

Web Client:

- Purpose: To make HTTP requests from within an application, particularly useful for microservices communication and handling asynchronous operations.
- Level: Library integrated into application code.
- Features: Reactive, non-blocking operations, fluent API for request building, supports both synchronous and asynchronous programming models.
- Pros: Efficient for making API calls within applications, good for building scalable and high-performance applications.
- Cons: Requires coding to use, not designed for manual exploration like Postman.

Packages

Mongoose, Express.js, and dotenv are common packages used together in Node.js applications, particularly when building web APIs with a MongoDB database.

Express.js:

- This is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It simplifies the process of handling HTTP requests, routing, middleware, and creating RESTful APIs.

Mongoose:

- This is an Object Data Modeling (ODM) library for MongoDB and Node.js. Mongoose provides a straightforward, schema-based solution to model application data. It allows developers to define models, perform CRUD (Create, Read, Update, Delete) operations on MongoDB documents, and manage relationships between data.

dotenv:

- This package loads environment variables from a .env file into process.env. It is primarily used to manage sensitive information like database connection strings, API keys, or port numbers, keeping them separate from the codebase and allowing for easy configuration changes across different environments (development, production, etc.).

Installation: All three packages are typically installed using npm:

```
npm install express mongoose dotenv
```

dotenv setup: A .env file is created in the project root to store environment variables (e.g., MONGO_URI, PORT). In the main application file (e.g., app.js or server.js), dotenv is initialized at the very top to load these variables:

```
require('dotenv').config();
```

Mongoose connection: Mongoose uses the MongoDB URI from process.env to establish a connection to the database:

```
    const mongoose = require('mongoose');
mongoose.connect(process.env.MONGO_URI)
            .then(() => console.log('MongoDB connected'))
            .catch(err => console.error(err));
```

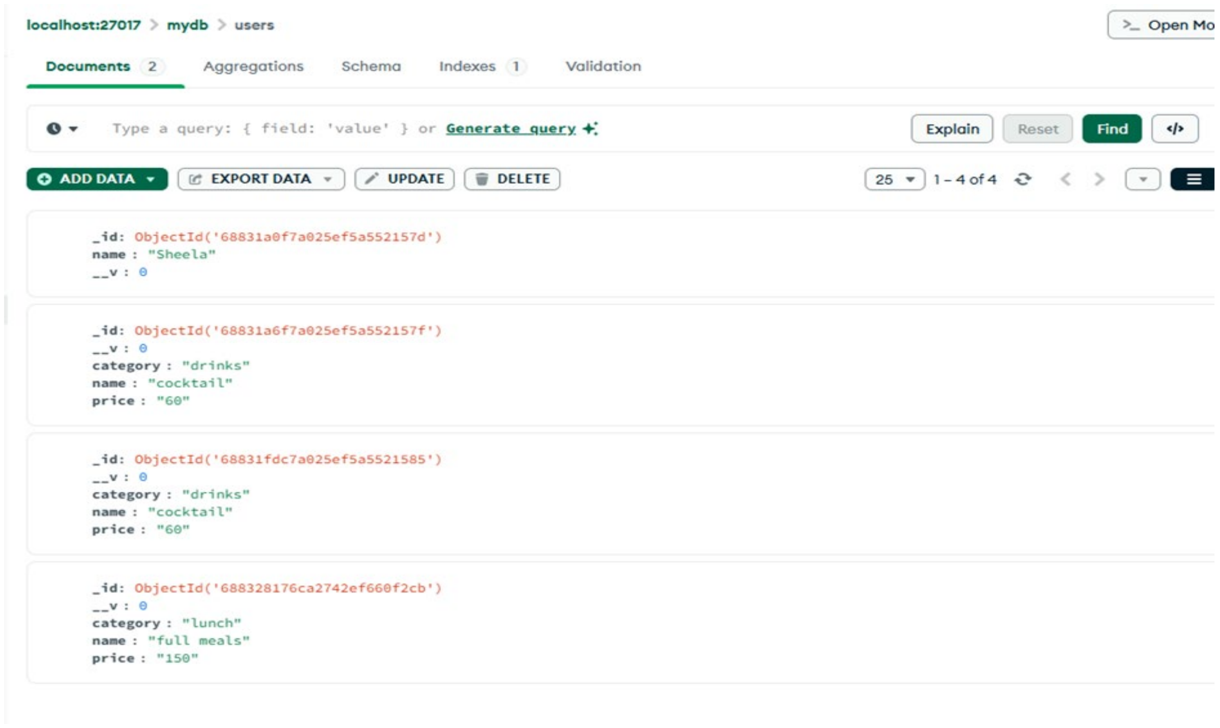
- Express server: Express is used to create and configure the web server, define routes, and handle requests. Mongoose models are then used within these routes to interact with the MongoDB database.

```
const express = require('express');
const app = express();
const port = process.env.PORT || 3000;

// ... define routes and use Mongoose models ...

app.listen(port, () => {
        console.log(`Server running on port ${port}`);
});
```

Output



```
_id: ObjectId('68831a0f7a025ef5a552157d')
name : "Sheela"
__v : 0

_id: ObjectId('68831a6f7a025ef5a552157f')
__v : 0
category : "drinks"
name : "cocktail"
price : "60"

_id: ObjectId('68831fdc7a025ef5a5521585')
__v : 0
category : "drinks"
name : "cocktail"
price : "60"

_id: ObjectId('688328176ca2742ef660f2cb')
__v : 0
category : "lunch"
name : "full meals"
price : "150"
```

RESULT:

Thus, A custom application was successfully created using Node.js and MongoDB. The project performed basic CRUD operations through RESTful APIs and was tested using Postman. The same development process can be applied across various domains by changing the data schema.

Ex No 5 **USE REACT COMPONENTS, JSX, CLASS, PROP, EVENTS TO BUILD AN APPLICATION**
Date:

Aim

To develop a form-based web application using React that demonstrates the use of Class Components, JSX, Props, and Event Handling.

Algorithm

Step 1: Set up a React environment using create-react-app.
Step 2: Create a class-based form component.
Step 3: Initialize state for each form input.
Step 4: Use JSX to create the form structure.
Step 5: Bind input fields using value and on Change.
Step 6: On form submission, prevent default behaviour and display submitted data.
Step 7: Pass the submitted data to a child component via props.
Step 8: Render the data conditionally.

Concepts Involved

JSX

JSX stands for JavaScript XML, and it is a special syntax used in React to simplify building user interfaces. JSX allows you to write HTML-like code directly inside JavaScript, enabling you to create UI components more efficiently. Although JSX looks like regular HTML, it’s actually a syntax extension for JavaScript.

Syntax:

```
const element = <h1>Hello, world!</h1>
```

How to Implement JSX in Action

JSX can be implemented in a React project to create dynamic and interactive UI components. Here are the steps to use JSX in a React application:

- **Create a React App**
- **Write JSX in the Component:** In the src/App.js file, write JSX to display a message:

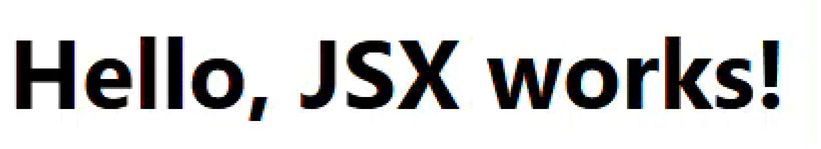
```
import React from "react";

 function App() {
 const message = "Hello, JSX works!";


 return <h1>{message}</h1>;
 }

 export default App;
```

Output


Hello, JSX works!

Class

In React, a "class" refers to a class component, which is a way to define components using ES6 JavaScript classes. Before the introduction of Hooks in React 16.8, class components were the primary method for creating stateful components and managing component lifecycle.

Example

```
import React from 'react';

class Greeting extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}!</h1>;
  }
}

export default Greeting;
```

Prop

In ReactJS, "props" (short for properties) are a mechanism for passing data from a parent component to a child component. They are a fundamental concept in React, enabling components to be dynamic and reusable.

Syntax

```
// Passing Props
 <DemoComponent sampleProp = "HelloProp" />
```

Syntax

```
// Accessing props in class components
this.props.propName;

// Accessing props in functional components
props.propName;
```

Events

In React.js, events represent user interactions or system occurrences within an application, such as clicks, key presses, form submissions, or mouse movements. React's event system is designed to handle these interactions and trigger corresponding actions within components.

Syntax

```
        <element onEvent={handlerFunction} />
```

RESULT:

Thus, successfully created a React application using Class Component, JSX, Props, and Events to handle and display form data.

Ex No 6 **WRITE CUSTOM APPLICATIONS FORMS USING REACT**

Date:

Aim

To develop custom form-based applications using React with proper component structure, JSX, Props, Events, and State Management.

Algorithm

Step 1: Set up React environment.
Step 2: Create a parent form component with fields: Name, Email, Feedback.
Step 3: Use state to manage input data.
Step 4: On submission, validate the form and pass data to a result component.
Step 5: Display a custom message using props and conditional rendering.

Concepts Involved

React Components

A component is a building block that encapsulates a piece of the user interface (UI). It defines the structure and behavior of the UI, either by managing its own state or by receiving data through props and rendering content accordingly. Components are reusable and can be composed together to build complex UIs in a modular way.

- Components can be reused across different parts of the application to maintain consistency and reduce code duplication.
- Components manage how their output is rendered in the DOM based on their state and props.
- React loads only the necessary components, ensuring optimized performance.
- Only the specific Component updates instead of the whole page.

Types of React Components

There are two primary types of React components:

1. Functional Components

Functional components are simpler and preferred for most use cases. They are JavaScript functions that return React elements. With the introduction of React Hooks, functional components can also manage state and lifecycle events.

- **Stateless or Stateful:** Can manage state using React Hooks.
- **Simpler Syntax:** Ideal for small and reusable components.
- **Performance:** Generally faster since they don't require a 'this' keyword.

```
function Greet(props) {
    return <h1>Hello, {props.name}!</h1>;
}
```

2. Class Components

Class components are ES6 classes that extend React.Component. They include additional features like state management and lifecycle methods.

- **State Management:** State is managed using the this.state property.
- **Lifecycle Methods:** Includes methods like componentDidMount, componentDidUpdate, etc.

```
class Greet extends React.Component {
    render() {
        return <h1>Hello, {this.props.name}!</h1>;
  }
}
```

Props in React Components

Props (short for properties) are read-only inputs passed from a parent component to a child component. They enable dynamic data flow and reusability.

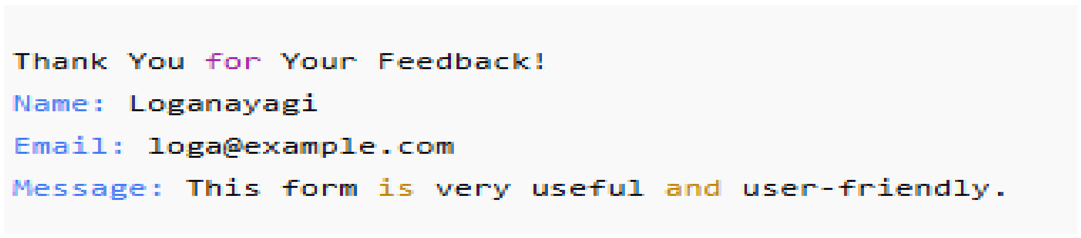
- ➢ Props are immutable.
- ➢ They enable communication between components.

State in React Components

The state is a JavaScript object managed within a component, allowing it to maintain and update its own data over time. Unlike props, state is mutable and controlled entirely by the component.

- ➢ State updates trigger re-renders.
- ➢ Functional components use the useState hook to manage state.

Output



```
Thank You for Your Feedback!
Name: Loganayagi
Email: loga@example.com
Message: This form is very useful and user-friendly.
```

<u>RESULT</u>

Thus, successfully developed a custom feedback form application using React that utilizes class components, props, events, and state.

Ex No 7 USE TYPE SCRIPT FOR ENHANCING WEB APPLICATION

Date:

Aim

To develop a strongly-typed, scalable web application using **React with TypeScript**, focusing on forms, props, state, and events.

Algorithm

Step 1: Setup React + TypeScript.
Step 2: Define a Student type.
Step 3: Create a form with typed inputs and events.
Step 4: Use props to pass submitted data to the display component.
Step 5: Validate and display the student details.

Concept Involved

TypeScript

TypeScript enhances JavaScript by adding strict type definitions, making your code more robust and maintainable. ReactJS, a popular library for building user interfaces, pairs excellently with TypeScript to create clean, efficient, and scalable codebases. Combining TypeScript with React offers a powerful approach to building robust and scalable web applications.

Steps to setup the Project Environment

Step 1: Create a React app

First, you need to set up a new React project. Use the following command to create a React app and navigate into the project directory.

npx create-react-app **project_name**

cd **project_name**

Step 2: Install TypeScript

Next, install TypeScript in your project. This allows you to use TypeScript's features in your React application.

```
npm install --save typescript
```

Step 3: Install React types definition

Install the latest type definitions for React and ReactDOM. These definitions allow TypeScript to understand the types used in React.

```
npm install @types/react @types/react-dom
```

Step 4: Install React dependencies

Once the TypeScript and type definitions are installed, install the rest of the React dependencies by running the following command.

```
npm install
```

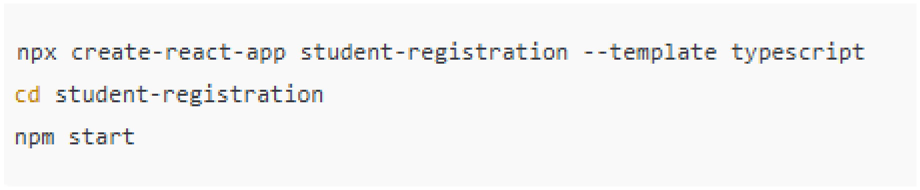
Step 5: Create files and Write code

Create different files with **.tsx** extension in your project directory and write the code using **TypeScript** and **React**.

Step 6: Run Project

Now, it's time to run your project and see the output in your web browser. Use the following command to start the development server:

```
npm run
```

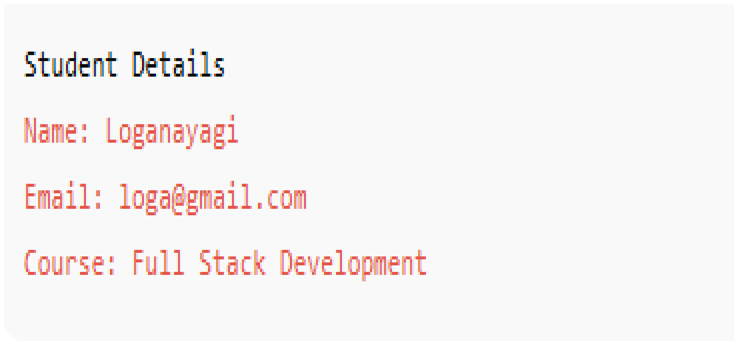


```
npx create-react-app student-registration --template typescript
cd student-registration
npm start
```

Props and State Typing

In React, props and state are fundamental concepts for managing data, but they serve different purposes. Props are used to pass data from parent to child components, while state is used to manage data within a component that can change over time. Props are read-only for the child component, meaning they cannot be modified by the component receiving them. State, on the other hand, can be updated by the component itself using the setState method.

Output:



```
Student Details

Name: Loganayagi

Email: loga@gmail.com

Course: Full Stack Development
```

RESULT

Thus, successfully created a typed React application using TypeScript.

Ex No 8 **APPLY USECALLBACK, USE STATE, USE EFFECT, USEREF HOOK OF REACT TO APPLICATIONS**

Date:

Aim

To develop a React application that demonstrates the usage of useState, useEffect, useCallback, and useRef hooks in real-world scenarios.

Algorithm

Step 1: Start

- Initialize the React application using TypeScript.

Step 2: Create Functional Component

- Create a functional component named StudentForm.

Step 3: Initialize State Variables

Use useState to declare:

- name → to store the student's name
- email → to store the student's email
- submitted → to track form submission status

Step 4: Set up useRef

Use useRef to reference the **Name input field** in order to set focus on load.

Step 5: Set Focus on Input (useEffect)

Use useEffect with an empty dependency array to run once:

- Focus on the name input using ref.current?.focus()

Step 6: Define Submit Handler (useCallback)

Use useCallback to define handleSubmit():

- Prevent default form submission
- Set submitted to true

Step 7: Reset Submission Message (useEffect)

Use useEffect with submitted as a dependency:

- If submitted is true, start a 3-second timer
- Reset submitted to false after timer ends
- Clear the timer using cleanup function

Step 8: Bind Input Fields
Bind the name and email input fields with value and onChange handlers using useState.

Step 9: Display Success Message
If submitted is true, display a success message like "Form submitted successfully".

Step 10: End

Concept Involved

React Hooks

ReactJS Hooks are one of the most powerful features of React, introduced in version 16.8. They allow developers to use state and other React features without writing a class component. Hooks simplify the code, make it more readable, and offer a more functional approach to React development. With hooks, React's state and lifecycle features can be used in functional components, making them much more powerful than they were before.

Types of React Hooks

React offers various hooks to handle state, side effects, and other functionalities in functional components. Below are some of the most commonly used types of React hooks:

1. State Hooks

State hooks, specifically useState and useReducer, allow functional components to manage state in a more efficient and modular way. They provide an easier and cleaner approach to managing component-level states in comparison to class components.

useState: The useState hook is used to declare state variables in functional components. It allows us to read and update the state within the component.

Syntax:

```
const [state, setState] = useState(initialState);
```

useReducer: The useReducer hook is a more advanced state management hook used for handling more complex state logic, often involving multiple sub-values or more intricate state transitions.

Syntax:

```
const [state, dispatch] = useReducer(reducer, initialState);
```

2. Context Hooks

The useContext hook in React is a powerful and convenient way to consume values from the React Context API in functional components. It allows functional components to access context values directly, without the need to manually pass props down through the component tree **const contextValue = useContext(MyContext);**

3. Effect Hooks

Effect hooks, specifically useEffect,useLayoutEffect, and useInsertionEffect, enable functional components to handle side effects in a more efficient and modular way.

useEffect: The useEffect hook in React is used to handle side effects in functional components. It allows you to perform actions such as data fetching, DOM manipulation, and setting up subscriptions, which are typically handled in lifecycle methods like componentDidMount or componentDidUpdate in class components.

Syntax:

```
useEffect(() => {
  // Side effect logic here
}, [dependencies]);
```

useLayoutEffect: The useLayoutEffect is used when we need to measure or manipulate the lawet before the browser paints, ensuring smooth transitions and no flickering.

Syntax:

```
useLayoutEffect(() => {
 // Logic to manipulate layout or measure DOM elements
}, [dependencies]);
```

useInsertionEffect: The useInsertionEffect is designed for injecting styles early, especially useful for server-side rendering (SSR) or styling libraries, ensuring styles are in place before the component is rendered visually.

Syntax:

```
useInsertionEffect(() => {
   // Logic to inject styles or manipulate stylesheets
}, [dependencies]);
```

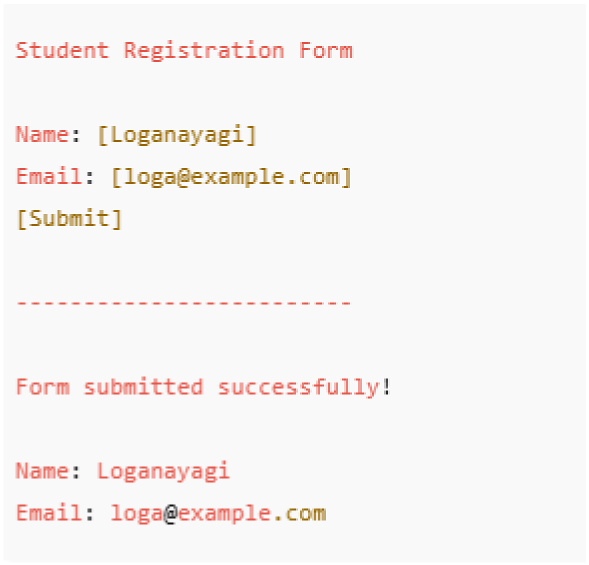
```
npx create-react-app student-hooks-app --template typescript
cd student-hooks-app
npm start
```

4. Performance Hook

Performance Hooks in React, like useMemo and useCallback, are used to optimize performance by avoiding unnecessary re-renders or recalculations.

useMemo: useMemo is a React hook that memoizes the result of an expensive calculation, preventing it from being recalculated on every render unless its dependencies change. This is particularly useful when we have a computation that is expensive in terms of performance, and we want to avoid recalculating it on every render cycle.

Output



```
Student Registration Form

Name: [Loganayagi]
Email: [loga@example.com]
[Submit]

-------------------------

Form submitted successfully!

Name: Loganayagi
Email: loga@example.com
```

RESULT

Thus, the application successfully demonstrates the practical usage of all 4 major React hooks effectively in a student form.

Ex No 9 **BUILD APPLICATION USING WEB PACK**

Date:

Aim

To configure and use **Webpack** for bundling a React application using **TypeScript**, and to demonstrate the use of React hooks (useState, useEffect, useCallback, useRef) in a application.

Algorithm

Step 1: Initialize a new npm project.
Step 2: Install required dependencies and devDependencies.
Step 3: Configure Webpack and Babel.
Step 4: Create React + TypeScript files.
Step 5: Run Webpack to build the bundle.
Step 6: Serve with a dev server or open index.html.

Concepts Involved

React JS

React is a powerful JavaScript library for building fast, scalable front-end applications. Created by Facebook, it's known for its component-based structure, single-page applications (SPAs), and virtual DOM,enabling efficient UI updates and a seamless user experience.

Module bundling with Webpack

Webpack is a static module bundler for modern JavaScript applications. It processes your application's modules and their dependencies, creating a dependency graph, and then bundles them into one or more optimized output files (bundles) for deployment.

Entry Points:
Webpack starts building the dependency graph from one or more entry points, which are the main files of your application (e.g., index.js).

Modules:
In Webpack, a module is any file that can be imported into another file. This includes not only JavaScript files but also CSS, images, fonts, and other assets, which are treated as modules through the use of loaders.

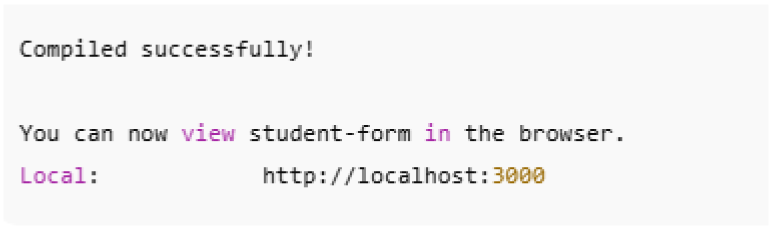
Loaders:
Webpack only understands JavaScript and JSON by default. Loaders are transformations that convert other file types (e.g., CSS, SCSS, images, TypeScript) into valid modules that Webpack can process and add to the dependency graph.

Plugins:
Plugins are more powerful tools that can perform a wider range of tasks, like optimizing bundles, managing assets, injecting environment variables, or generating HTML files.
Output:
Webpack bundles the processed modules and assets into one or more output files, typically placed in a dist directory. These bundles are optimized for production, often minified and compressed to reduce file size and improve load times.

<u>How Webpack Bundles Modules:</u>
Dependency Resolution:
Webpack analyzes the entry points and recursively identifies all dependencies (e.g., import statements in JavaScript, url() in CSS).
Dependency Graph Creation:
It builds a comprehensive dependency graph that maps out how all the modules in your project are interconnected.
Module Transformation (with Loaders):
As it traverses the graph, Webpack applies configured loaders to transform non-JavaScript modules into a format it can understand.
Bundle Generation:
Finally, Webpack combines the processed modules and their dependencies into the specified output bundles, optimizing them for production if configured to do so.

<u>TypeScript</u>

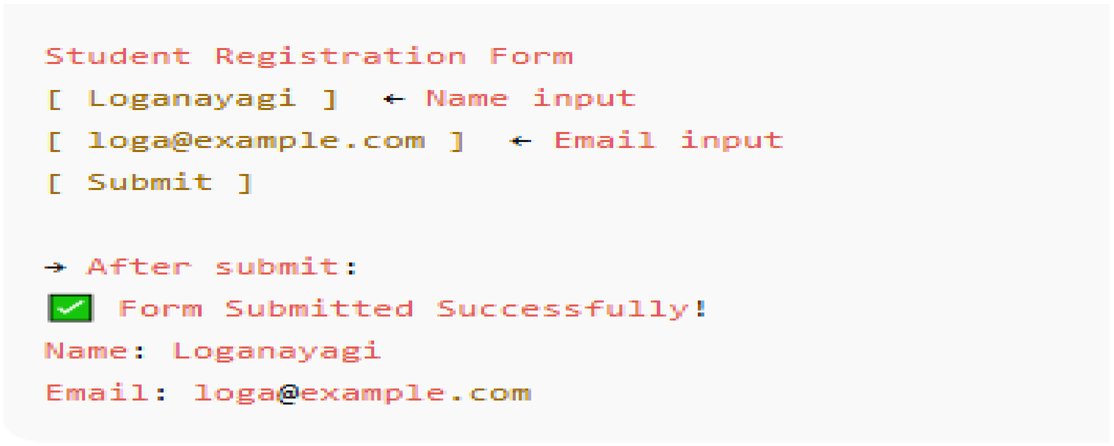
TypeScript is a strongly typed superset of JavaScript that compiles to plain JavaScript. It extends JavaScript by adding optional static typing, which allows developers to define the types of variables, function parameters, and return values. This type system helps in catching errors during development, before the code is executed, leading to more robust and maintainable applications.



```
Compiled successfully!

You can now view student-form in the browser.
Local:            http://localhost:3000
```

Output



```
Student Registration Form
[ Loganayagi ]  ← Name input
[ loga@example.com ]  ← Email input
[ Submit ]

→ After submit:
✅ Form Submitted Successfully!
Name: Loganayagi
Email: loga@example.com
```

RESULT

Thus, The form is successfully built using **React (Class Component + Hooks)** with **TypeScript**, bundled using **Webpack**.

Ex No 10 **DEPLOY AN APPLICATION BY BINDING SERVER-SIDE AND CLIENT SIDE**

Date:

Aim

To develop and deploy a full-stack application using React (frontend) and Node.js with MongoDB (backend) and bundle using Webpack.

Algorithm

Step 1: Frontend (React + TypeScript):

- Create a registration form with fields (Name, Email).
- Handle state using `useState`.
- Handle focus and validation using `useRef`, `useEffect`.
- Submit form data using `fetch` to the backend API.
- Display success message after submission.

Step 2: Backend (Node.js + Express + MongoDB):

- Create an Express server.
- Define a POST route `/register` to receive data.
- Store data in MongoDB using Mongoose.
- Enable CORS for frontend access.

Step 3: Bind Frontend and Backend:

- Set up proxy in React app to connect with backend.
- Or use full URLs like `http://localhost:5000/register`.

Step 4: Bundle Using Webpack:

- Use `webpack.config.js` to bundle the frontend.

Step 5: Deployment:

- Host the backend on **Render/Heroku**.
- Host frontend on **Vercel/Netlify**.
- Use `.env` for secure environment variable management.

Concepts Involved

Deployment:

Deployment platforms like Render, Vercel, and Heroku offer streamlined solutions for bringing web applications and services online. They abstract away much of the underlying infrastructure management, allowing developers to focus on code.

Render:

- A fully managed cloud platform designed for ease of use, often seen as a modern alternative to Heroku.
- Supports various application types, including web services, databases (Postgres, Redis), cron jobs, and static sites.
- Offers features like automatic deployments from Git, custom domains, SSL, and environment variables.

Vercel:

- Specializes in frontend deployments, particularly for static sites, serverless functions, and frameworks like Next.js.
- Known for its focus on performance, with features like automatic static optimization, edge deployments, and preview deployments for every Git commit.
- Integrates seamlessly with Git repositories for continuous deployment.

Heroku:

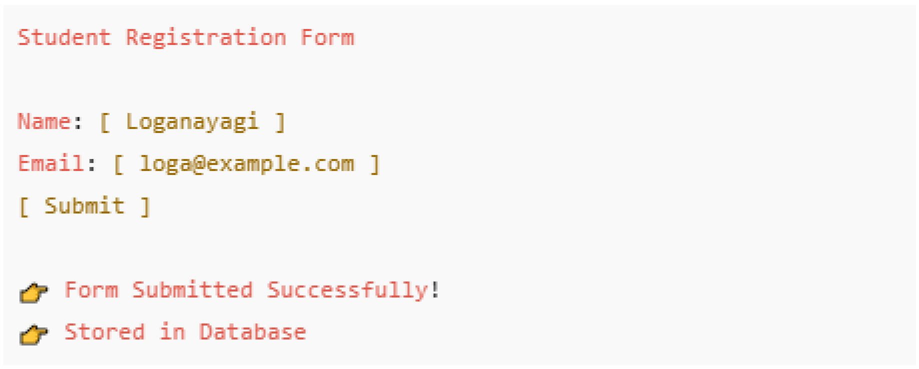
- A pioneering Platform-as-a-Service (PaaS) that simplifies application deployment and scaling.
- Supports a wide range of programming languages and frameworks through "buildpacks."
- Provides a comprehensive ecosystem of add-ons for databases, monitoring, and other services.
- Offers features like automatic scaling, custom domains, and a robust CLI for management.

General Deployment Process (common to these services):

- Connect to Git Repository: Link your project to a Git repository (e.g., GitHub, GitLab, Bitbucket).
- Configure Environment Variables: Set up necessary environment variables for your application (e.g., database connection strings, API keys).
- Define Build and Start Commands: Specify how your application should be built and how it should be started.
- Automatic Deployments: Configure automatic deployments on every push to a designated branch.
- Custom Domains & SSL: Add custom domains and enable SSL certificates for secure access.

- Monitoring & Logs: Utilize the platform's tools for monitoring application performance and accessing logs.

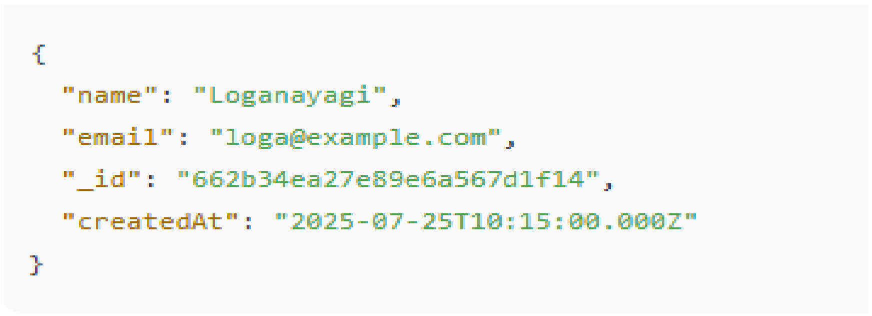
Output



```
Student Registration Form

Name: [ Loganayagi ]
Email: [ loga@example.com ]
[ Submit ]

👉 Form Submitted Successfully!
👉 Stored in Database
```



```
{
  "name": "Loganayagi",
  "email": "loga@example.com",
  "_id": "662b34ea27e89e6a567d1f14",
  "createdAt": "2025-07-25T10:15:00.000Z"
}
```

RESULT

Thus, the full stack web application successfully designed, developed, and implemented using **React (Frontend)**, **Node.js (Backend)**, and **MongoDB (Database)**. The integration between client and server sides was achieved seamlessly, and the application was tested with sample input to verify its functionality.

Ex No 11 **DEPLOY APPLICATIONS USING DOCKER HUB**

Date:

Aim

To **containerize a full stack web application** (built using **React**, **Node.js**, and **MongoDB**) and **deploy it to Docker Hub**, enabling easy sharing, scalability, and cross-platform execution.

Algorithm

Step 1: Start

Step 2: Create a **Dockerfile** in the backend (Node.js) project folder

- Define the base image (e.g., node:18)
- Set the working directory
- Copy project files
- Install dependencies
- Expose backend port (e.g., 5000)
- Define the startup command

Step 3: Create a **Dockerfile** in the frontend (React) project folder

- Use Node image
- Copy React app files
- Build the React project
- Use serve to run the static site
- Expose frontend port (e.g., 3000)

Step 4: Create a **docker-compose.yml** file

- Define services for frontend, backend, and MongoDB
- Set up network links between them
- Configure ports and environment variables

Step 5: Run the command docker-compose up --build to build and run all services

Step 6: Test the application locally in the browser (localhost:3000)

Step 7: Log in to **Docker Hub** using docker login

Step 8: Tag the built Docker images with your Docker Hub username

Step 9: Push the tagged images to Docker Hub using docker push

Step 10: Verify that the images appear in your Docker Hub repository

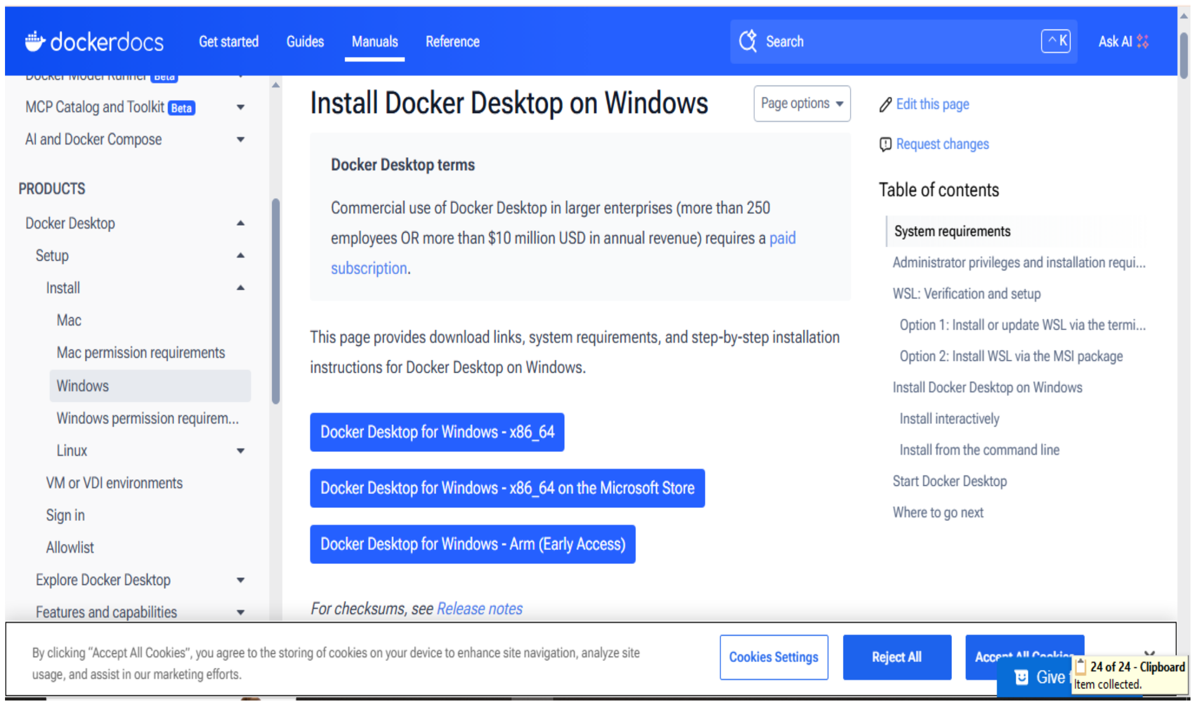
Step 11: End

Concepts Involved

Docker Installation

What is Docker?

Docker is an open-source container platform software tool, where you run your applications in the form of containers. Docker containers comes with light weighted softwares having all the dependencies and configurations so we can run them across different computing environments. It facilitates the developers to package their application with all its dependencies into a single entity in the form of images. These can be portable easily or sharable with other developers without worrying about the underlying OS.



Install Docker Desktop on Windows

Docker Desktop terms

Commercial use of Docker Desktop in larger enterprises (more than 250 employees OR more than $10 million USD in annual revenue) requires a paid subscription.

This page provides download links, system requirements, and step-by-step installation instructions for Docker Desktop on Windows.

Docker Desktop for Windows - x86_64

Docker Desktop for Windows - x86_64 on the Microsoft Store

Docker Desktop for Windows - Arm (Early Access)

For checksums, see Release notes

By clicking "Accept All Cookies", you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts.

Requirements of Windows For Downloading Docker

The following are the requirements of Windows on Docker:

1. **Windows 11 64-bit:** Home or Pro version 21H2 or higher, or Enterprise or Education version 21H2.
2. **Windows 10 64-bit:** Home or Pro version 21H2 or higher, or Enterprise or Education version 21H2.
3. 4GB RAM or Higher.
4. Hyper V feature must be enabled in Windows.

RESULT

Thus, the full stack student registration application was successfully developed and deployed using **React**, **Node.js**, **MongoDB**, and **Docker**, and published to **Docker Hub** for cross-platform access and testing.

References

1. https://www.w3schools.com/html/default.asp
2. https://www.w3schools.com/css/default.asp
3. https://www.w3schools.com/js/default.asp
4. https://react.dev/learn/typescript
5. https://www.geeksforgeeks.org/devops/docker-tutorial/
6. https://www.geeksforgeeks.org/reactjs/react/