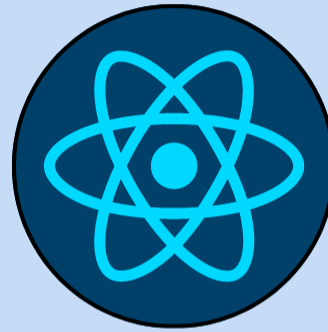


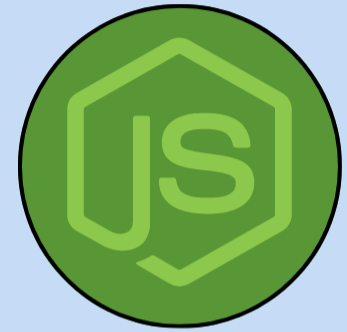
M



E



R



N

STACK

Module 1 : MongoDB

Module 2 : Express.js

Module 3 : Angular

Module 4 : Node.js

MERN Stack: A JavaScript Full Stack Solution

MERN stack is a popular choice for building dynamic web applications. It's a collection of JavaScript-based technologies that work together to handle different layers of the application.

Components of MERN Stack

- **MongoDB:** A NoSQL database that uses flexible JSON-like documents to store data.
- **Express.js:** A Node.js framework for building web applications and APIs.
- **React:** A JavaScript library for building user interfaces.
- **Node.js:** A JavaScript runtime environment that allows developers to execute JavaScript code on the server-side.

How MERN Stack Works

1.Client-Side (React): The user interacts with the application's user interface built using React.

2.Data Interaction (React and Node.js): React communicates with the server-side using HTTP requests to fetch or send data.

3.Server-Side (Node.js and Express.js): Node.js handles the server-side logic, and Express.js provides the framework for routing and handling requests.

4.Database (MongoDB): MongoDB stores and retrieves data as JSON-like documents.

Benefits of Using MERN Stack

- JavaScript Unification:** Consistent use of JavaScript across the entire stack simplifies development.
- Rapid Development:** The MERN stack offers a streamlined development process due to its cohesive nature.
- Scalability:** Both MongoDB and Node.js are designed for handling high traffic and large datasets.
- Open Source:** All components of the MERN stack are open-source, providing a large community and extensive support.
- JSON-Based:** Data consistency between the client and server due to the use of JSON.

When to Use MERN Stack

MERN stack is well-suited for building:

- Real-time applications (e.g., chat apps, online gaming)
- Single-page applications (SPAs)
- High-traffic web applications
- Cloud-based applications
- Mobile app backends

Module 1: MongoDB

Introduction to NoSQL Databases

- **Definition:** NoSQL databases provide a way to store and retrieve data that is modeled in a way other than the tabular relations used in relational databases (RDBMS).
- **Types of NoSQL databases:** Document, Key-Value, Column-Family, and Graph databases.
- **Advantages:** Scalability, flexibility, and handling unstructured data.

MongoDB Basics

- **Installation and Setup:** Install MongoDB Community Server on your local machine.
- Start the MongoDB server using the `mongod` command.
- Access the MongoDB shell using `mongo`.

CRUD Operations:

- **Create:** `db.collection.insertOne({name: "Alice", age: 25})`
- **Read:** `db.collection.find({name: "Alice"})`
- **Update:** `db.collection.updateOne({name: "Alice"}, {$set: {age: 26}})`
- **Delete:** `db.collection.deleteOne({name: "Alice"})`

Collections and Documents:

- **Collection:** Analogous to a table in RDBMS.
- **Document:** A record in a collection, stored in BSON (Binary JSON) format.

Example:

```
db.students.insertOne({  
  name: "John Doe",  
  age: 22,  
  courses: ["Math", "Physics"]  
});
```

Advanced MongoDB Schema Design:

- Schema-less design allows for flexible data models.
- Use embedded documents for one-to-many relationships.

Indexing:

- Improves query performance.
- Example: `db.students.createIndex({name: 1})`

Aggregation:

- Perform complex data manipulations and transformations.

Example

```
db.students.aggregate([
  { $match: { age: { $gt: 20 } } },
  { $group: { _id: "$courses", count: { $sum: 1 } } }
]);
```

Module 2: Express.js

Introduction to Express.js

Role in MEAN Stack:

- Express.js is a minimal and flexible Node.js web application framework that provides robust features for web and mobile applications.
- Facilitates routing, middleware support, and integration with databases.

Setting Up Express:

- Initialize a new Node.js project: `npm init`
- Install Express: `npm install express`
- Basic server setup:

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
  res.send('Hello World!');
});
app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}/`);
});
```

Routing in Express.js

•Routing Basics:

- Define routes to handle client requests.
- Example

```
app.get('/students', (req, res) => {  
  res.send('List of students');  
});
```

```
app.post('/students', (req, res) => {  
  res.send('Create a new student');  
});
```

Middleware:

- Functions that execute during the lifecycle of a request to the server.

Example

```
app.use(express.json()); // Middleware to parse JSON bodies
```

Error Handling:

Centralized error handling using middleware.

Example

```
app.use((err, req, res, next) => {  
  res.status(500).send({ error: err.message });  
});
```

Template Engines

Using Pug:

Install Pug: `npm install pug`

Set up Pug as the view engine:

```
app.set('view engine', 'pug');  
app.get('/profile', (req, res) => {  
  res.render('profile', { name: 'John Doe' });  
});
```

Building Dynamic Web Pages:

Use Pug templates to generate HTML with dynamic data.

Example

```
//- profile.pug
h1 Profile Page
p Name: #{name}
```

API Development

•RESTful API Design:

- Design principles: Statelessness, Uniform Interface, Resource-based URLs.
- Example: Implementing CRUD API for students

```
app.get('/students/:id', (req, res) => {
  // Fetch student by ID from database
});
app.post('/students', (req, res) => {
  // Create a new student record
});
app.put('/students/:id', (req, res) => {
  // Update student record
});
app.delete('/students/:id', (req, res) => {
  // Delete student record
});
```

Module 3 : React

Components

•Definition:

- Components are the building blocks of a React application. They can be thought of as reusable UI elements that are independent and self-contained.
- React components can be defined as either **functional components** (stateless) or **class components** (stateful).

```
import React from 'react';  
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}  
export default Welcome;
```

Explanation: The Welcome component is a simple functional component that receives props and returns a JSX element displaying the name passed as a prop.

Example: Class Component

```
import React, { Component } from 'react';

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

export default Welcome;
```

Explanation: The Welcome component is defined as a class, which extends React.Component. It has a render() method that returns JSX.

2. JSX (JavaScript XML)

•Definition:

- JSX is a syntax extension of JavaScript that looks similar to HTML. It is used in React to describe the UI.
- JSX is not required to use React, but it makes the code more readable and easier to write.

•Example:

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

Explanation: This JSX code represents an HTML h1 element that will render "Hello, world!" on the page.

3. Props

•Definition:

- Props (short for "properties") are read-only inputs passed to components. They allow data to be passed from one component to another, often from a parent to a child component.

•Example: Passing Props

```
function App() {  
  return <Welcome name="Alice" />;  
}
```

4. StateDefinition:

State is a built-in object used to contain data or information about the component. Unlike props, state is local to the component and can be changed using the `setState()` method. Example: Using State in a Class Component

```
import React, { Component } from 'react';
class Clock extends Component {
  constructor(props) {
    super(props);
    this.state = { date: new Date() };
  }
  componentDidMount() {
    this.timerID = setInterval(() => this.tick(), 1000);
  }
  componentWillUnmount() {
    clearInterval(this.timerID);
  }
  tick() {
    this.setState({
      date: new Date(),
    });
  }
  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}</h2>
      </div>
    );
  }
} export default Clock;
```

5. Handling Events

•Definition:

- Handling events in React is very similar to handling events in DOM elements, but with some syntax differences. React events are named using camelCase, and you pass a function as the event handler.

```
function Toggle() {  
  const [isOn, setIsOn] = React.useState(true);  
  
  function handleClick() {  
    setIsOn(!isOn);  
  }  
  
  return (  
    <button onClick={handleClick}>  
      {isOn ? 'ON' : 'OFF'}  
    </button>  
  );  
}  
  
export default Toggle;
```

6. Conditional Rendering

•Definition:

- In React, you can conditionally render elements based on the state or props of a component.

```
function Greeting(props) {  
  const isLoggedIn = props.isLoggedIn;  
  if (isLoggedIn) {  
    return <h1>Welcome back!</h1>;  
  }  
  return <h1>Please sign up.</h1>;  
}  
  
export default Greeting;
```

Explanation: The Greeting component renders different messages based on the value of isLoggedIn.

7. Lists and Keys

•Definition:

- Lists are used to display a series of similar items. Each item in a list needs a unique "key" prop to help React identify which items have changed, been added, or removed.

•Example: Rendering a List

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>{number}</li>
  );
  return <ul>{listItems}</ul>;
}

export default NumberList;
```

8. Forms

•Definition:

- Forms in React are similar to HTML forms but with additional handling of user input using state.


```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = { value: '' };

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({ value: event.target.value });
  }

  handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
  }
}
```

```
render() {
  return (
    <form onSubmit={this.handleSubmit}>
      <label>
        Name:
        <input type="text" value={this.state.value}
onChange={this.handleChange} />
      </label>
      <input type="submit
" value="Submit" />
    </form>
  );
}
}

export default NameForm;
```

Module 4 : Node.js

Introduction to Node.js

Role in MEAN Stack:

- Node.js is a JavaScript runtime built on Chrome's V8 engine, allowing JavaScript to be used for server-side scripting.
- It's non-blocking, event-driven architecture makes it suitable for I/O-heavy applications.

Setting Up Node.js:

- Install Node.js from nodejs.org.
- Verify installation: `node -v` and `npm -v`.

Core Modules and Features

File System (fs):

- Read and write files asynchronously.

```
const fs = require('fs');
fs.readFile('example.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log(data);
});
```

Networking with HTTP:

Create a basic HTTP server.

```
const http = require('http');
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello, World!\n');
});

server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

Working with Databases:

Use mongoose to interact with MongoDB from Node.js.

```
const mongoose = require('mongoose');  
mongoose.connect('mongodb://localhost/mydatabase', {  
  useNewUrlParser: true });
```

... THANK YOU ...