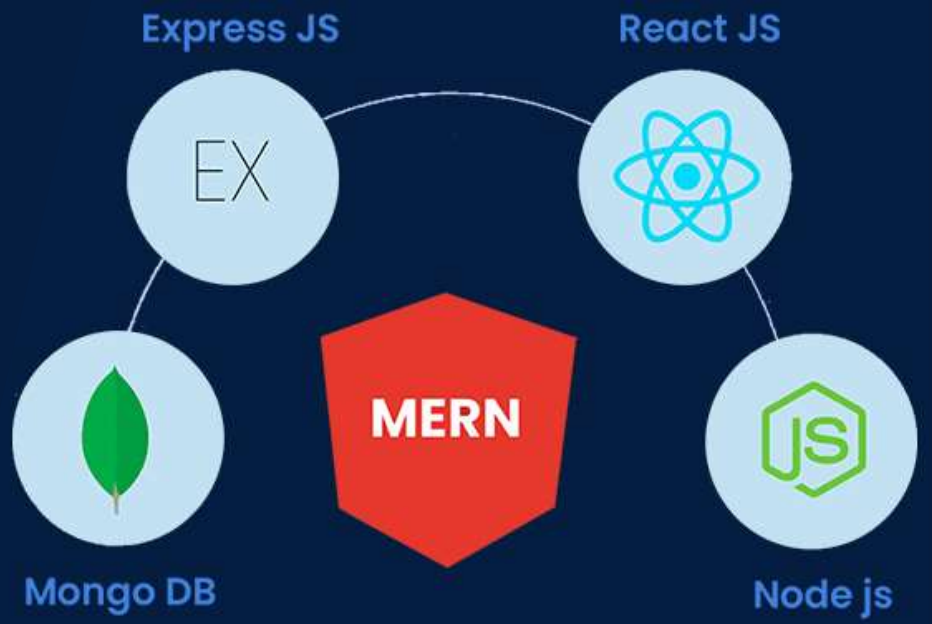
Class 01 & 02

MERN STACK DEVELOPMENT - MODULE B

Exploring the Pinnacle of Full Stack Development with MERN



INTRODUCTION TO MERN

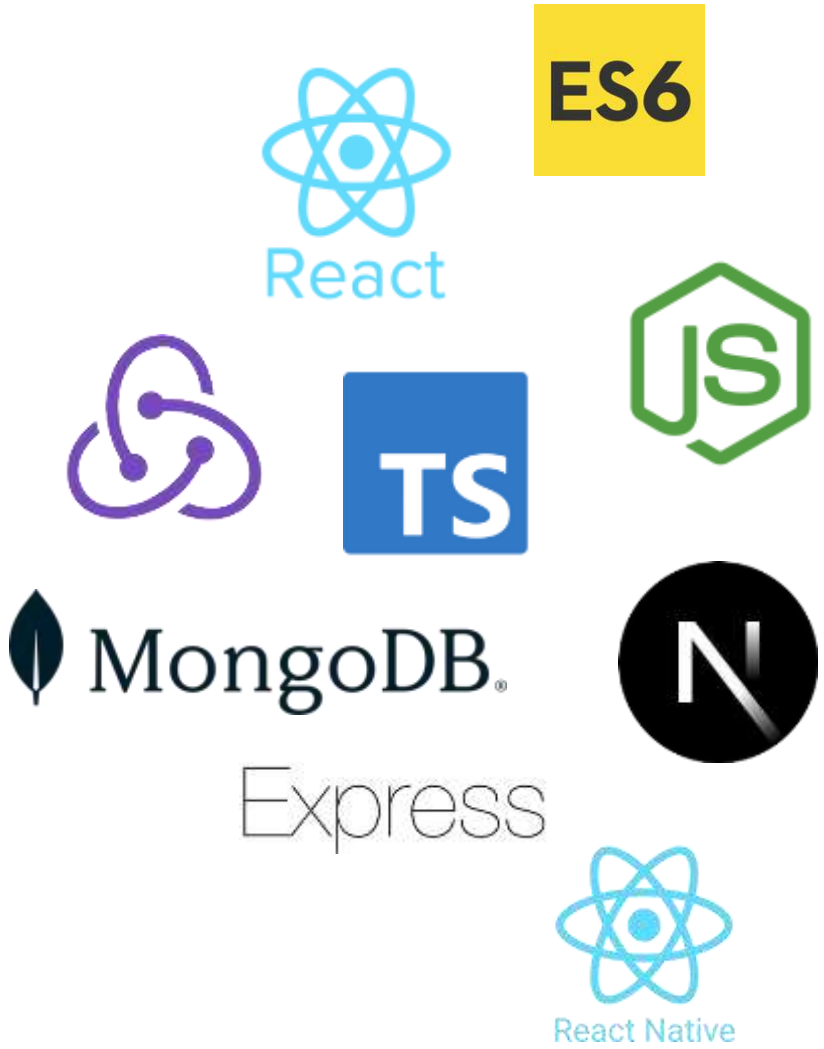
MERN is an acronym for **MongoDB, Express.js, React, and Node.js**, represents a powerful full-stack JavaScript framework for creating dynamic web applications.

MongoDB, a NoSQL database, offers flexibility and scalability, **Express JS** simplifies API development on the server-side, **React JS,** developed by Facebook, is a library for building user interfaces, & **Node JS** is a server-side JavaScript runtime, known for its asynchronous and event-driven nature.

Together, these components provide an end-to-end JavaScript solution, offering efficiency and versatility for web development projects.

COURSE CURRICULUM

- ECMAScript (ES6)
- React JS
- Redux Toolkit
- Node JS
- Express JS
- MongoDB
- TypeScript
- Next JS
- React Native



ECMASCRIPT 6 – (ES6)

ES6, or ECMAScript 2015, represents a pivotal moment in the evolution of JavaScript, introducing a host of new features and enhancements to the language.

ES6 Released in 2015, ES6 marked a significant step forward in making JavaScript more powerful, expressive, and developer-friendly. The term "ECMAScript" refers to the standardized scripting language upon which JavaScript is based, and ES6 is the sixth edition of this specification.

REACT JS

Developed by **Facebook** in 2013, React JS revolutionized frontend development by providing a solution to the complexities of user interface creation. Inventor, aimed to address the challenges of efficiently updating large and dynamic interfaces.

React simplifies UI development through a component-based structure, making it easier to manage and update individual parts of an application. It excels in crafting Single Page Applications (SPAs) where seamless, real-time updates are crucial. With its declarative syntax, virtual DOM, and efficient state management, React has become a go-to choice for building modern and responsive user interfaces.

REDUX / REDUX TOOLKIT

Redux is a Predictable State Container, introduced in 2015, Redux revolutionized state management in JavaScript applications, providing a centralized store and unidirectional data flow, particularly in React.

Why Redux / Redux Toolkit?

Crucial for developers dealing with complex data in apps, Redux ensures there's one clear place (a single source of truth) to keep track of everything. Meanwhile, Redux Toolkit simplifies the usual steps developers take, making it easier to manage this data. By using both, developers can build apps with organized, easy-to-understand code that's reliable and scalable.

NODE JS

In 2009, introduced Node.js, a runtime environment that enables JavaScript to run on the server side. This marked a pivotal moment, expanding JavaScript's domain beyond browsers.

Node.js addresses the challenge of handling concurrent operations efficiently by adopting a non-blocking and asynchronous design. It's a game-changer, allowing developers to use JavaScript for both front-end and back-end development, promoting code consistency and productivity across the entire application stack. With its exceptional scalability, performance, and versatility, Node.js has become a go-to choice for building server-side applications, APIs, real-time applications, and more..

EXPRESS JS

Developed in 2010, Express.js is a user-friendly framework for Node.js. It takes the complexity out of building web applications, offering a straightforward way to handle web development tasks.

Express.js is like a helpful toolkit that developers use to create web servers, APIs, and full-stack applications easily. It's popular because it's flexible, simple, and widely used in the coding community. If you're starting in web development, Express.js is your go-to tool for making things happen on the server side.

Express

MONGO DB

MongoDB is developed in 2007, MongoDB is a fresh take on databases. It's like a flexible magic box that solves the problems of traditional databases. Unlike rigid systems, MongoDB lets developers work with data more freely, making it perfect for modern applications. It's the go-to choice when you want your data to be easy to manage, grow effortlessly, and play well with web and mobile apps. In simple terms, MongoDB is the cool, modern way to handle data.

TYPESCRIPT

Developed by **Microsoft** in 2012, TypeScript is like JavaScript with an upgrade. It solves common development problems of unpredictable code by introducing "types" that catch errors before they happen.

While writing TypeScript, it's not directly used by browsers. Instead, it gets "compiled" into regular JavaScript that browsers understand. This compilation step is like translating code into a language everyone can comprehend, ensuring a smooth and error-free performance. The main benefit of TypeScript is that it can highlight unexpected behavior in your code, lowering the chance of bugs.

NEXT JS

Next JS introduced by **Vercel** in 2016, Next.js is a React framework for building full-stack web applications. React Components used to build user interfaces, and Next.js for additional features and optimizations.

Under the hood, Next.js also abstracts and automatically configures tooling needed for React, like bundling, compiling, and more. This allows to focus on building your application instead of spending time with configuration.

Whether you're an individual developer or part of a larger team, Next.js can help you build interactive, dynamic, and fast React applications.

REACT NATIVE

Developed by **Facebook** in 2015, React Native is a cutting-edge framework designed to streamline mobile app development. It addresses the challenge of maintaining separate codebases for iOS and Android by allowing developers to write code once and deploy it across multiple platforms.

Consider React Native as an advanced toolset, merging the efficiency of React with native mobile capabilities. It is the preferred choice for developers seeking a unified approach to building high-quality mobile applications while optimizing time and resources. React Native has become the cornerstone of cross-platform mobile development, offering a powerful and efficient solution for creating compelling user experiences on both iOS and Android.

Lets start with

ECMA SCRIPT 6

ECMASCRIPT

Introduction

ECMAScript (ES), standardized by Ecma International, originated in 1997 with the goal of establishing a consistent scripting language specification for web browsers. The term "ECMA" stands for the **European Computer Manufacturers Association**, the organization responsible for overseeing the development and maintenance of the ECMAScript standard.

Versions

Regular updates characterize ECMAScript, introducing new features and syntax enhancements. Notably, ES6, rolled out in 2015, brought essential elements such as classes, arrow functions, and template literals.

Latest Versions

In June 2023, the ECMAScript standard welcomed its 14th edition, ECMAScript 2023. This latest release brings forth a suite of new features, enhancing the capabilities and functionality of ECMAScript for developers.

SHORT HISTORY OF ECMA SCRIPT

1995
JAVASCRIPT ORIGINS
In 1995, JavaScript, initially Mocha and LiveScript, was created by Brendan Eich at Netscape. Initially named Mocha, then LiveScript, before settling on its now-familiar title, JavaScript.

1997
ES1 - First Edition
In 1997, ECMAScript (ES1) was formalized to standardize JavaScript, laying the groundwork for its evolution. This inaugural edition established fundamental features, shaping JavaScript's trajectory.

2009
ES5 – Fifth Edition
In 2009, ES5 (Fifth Edition) was a pivotal update to ECMAScript, introducing significant improvements and laying the groundwork for modern JavaScript development.

2015
ES6 - Sixth Edition
ES6, also known as ECMAScript 2015, released in 2015, revolutionized JavaScript. Its substantial updates transformed the language, making it more expressive and powerful for developers.

Topics

- Variable declaration keywords (let & const)
- Hoisting
- Pass by Value & Pass by Reference
- null & undefined
- Destructuring
- Template Literals

VARIABLE DECLARATION KEYWORDS (LET & CONST)

In JavaScript, when we want to store information for later use, we use variables. Two essential keywords for declaring variables are let and const.

let: Think of let as a tool that allows you to name a storage box and change its contents whenever needed. It's like having a flexible container that can hold different things at different times. We use let when we expect the value of our storage box to change during the course of our program.

const: On the other hand, const is like having a special box that can hold something very important but can never be replaced. Once you put an item in this box, you can't swap it for something else. We use const when we want to make sure our stored information stays the same throughout our program.

KEY DIFFERENCES

var	vs	let	vs	const
Redeclaration				
Can be redeclared within the same scope		Cannot be redeclared within the same scope.		Cannot be redeclared or reassigned after the initial assignment.
Reassignment				
Can be reassigned throughout the program.		Can be reassigned within its block scope.		Cannot be reassigned after the initial assignment.
Scope				
Function-scoped. Visible throughout the entire function, regardless of where it is declared.		Block-scoped. Visible only within the block (enclosed by curly braces) where they are declared.		Block-scoped. Visible only within the block (enclosed by curly braces) where they are declared.

HOISTING

Hoisting in JavaScript is a behavior where variable and function **declarations** are moved to the top of their containing scope during the code execution. This means you can use variables and functions in your code even before they are declared. However, it's important to note that only the declarations are moved, not the initializations (assigning values). This can sometimes lead to unexpected behavior if not understood correctly.

Variable Hoisting

example

```
console.log(x); // Output: undefined
var x = "Hello, World!"
console.log(x); // Output: 5
```

Function Hoisting

example

```
greet(); // Output: Hello, World!
function greet() {
console.log("Hello!");
}
```

PASS BY VALUE & PASS BY REFERENCE

Pass by value

when you pass a primitive data type (such as numbers, strings, or booleans) to a function, a copy of the actual value is passed to the function. Changes made to the parameter inside the function do not affect the original value outside the function. This behavior is characteristic of primitive types, and each variable holds its own distinct value.

Pass by reference

While JavaScript is "pass by value" for primitives, it exhibits a "pass by reference" (or more precisely, "pass by sharing") behavior when it comes to objects (including arrays and functions). When an object is passed to a function, what is actually passed is a reference to the object in memory. Consequently, changes made to the object's properties or elements inside the function are reflected in the original object outside the function.

EXAMPLES OF **PASS BY VALUE**

Pass by Value in Variable Assignments:

example

```
let a = 5;
let b = a; // b gets a copy of the value of a
b = 10;
console.log(a); // Output: 5 (unchanged)
console.log(b); // Output: 10
```

Pass by Value in Function Calls:

example

```
function mutiply_number(num) {
        num = num * 2;
        return num;
}

let x = 7;
let result = mutiply_number(x);

console.log(x); // Output: 7 (unchanged)

console.log(result); // Output: 14
```

EXAMPLES OF **PASS BY REFERENCE**

Pass by Reference in Variable Assignments:

example

```
let obj1 = { value: 10 };
let obj2 = obj1;
// obj2 now holds a reference to the same object as obj1

obj2.value = 20;

console.log(obj1.value); // Output: 20 (modified)
console.log(obj2.value); // Output: 20
```

Pass by Reference in Function Calls

example

```
function addToArray(arr, value) {
        arr.push(value);
}

let myArray = [1, 2, 3];
addToArray(myArray, 4);

console.log(myArray); // Output: [1, 2, 3, 4] (modified)
```

NULL & UNDEFINED

undefined

undefined is a **primitive** value in JavaScript, representing the default state of a variable that has been declared but not assigned a value. It is also associated with function parameters that are not provided and serves as the default return value for functions that do not explicitly return anything.

null

null is an object value in JavaScript, intentionally used to signify the absence of an object value. It is explicitly assigned to variables or object properties when you want to convey the deliberate absence of a value or when an object property has no applicable value.

DESTRUCTURING

Destructuring is a powerful feature in JavaScript that allows you to unpack values from arrays or properties from objects into distinct variables. It provides a concise and expressive way to work with complex data structures.

Array Destructuring

syntax

```
const [first, second, third] = [1, 2, 3];

console.log(first, second, third);
// 1, 2, 3
```

Object Destructuring

syntax

```
const { name, age } = { name: ‘Jack', age: 30 };

console.log(name, age);
// jack, 30
```

TEMPLATE LITERALS

In ES6, Template Literals redefine string manipulation in JavaScript, offering a concise and expressive alternative to traditional concatenation. Marked by the backtick (`) symbol, they enhance code readability with a natural syntax, allowing seamless variable injection using ${}. seamlessly handle variable interpolation and expressions, and simplify the creation of multiline strings.

Traditional String Concatenation

syntax

```
const name = “user name";

const greetingConcat = "Hello, " + name + "!";
```

Ternary Operator

syntax

```
const name = "user name";

const greetingTemplate = `Hello, ${name}!`;
```

EXERCISE:

```
const student_scores = [
        { name: ‘Harry', score: 85 },
        { name: 'David', score: 92 },
        { name: ‘Jack', score: 78 },
];
```

Grade Assignment:

Apply your understanding of loop (for loop), destructuring, & the ternary operator to assign grades to students based on their scores.

- Make use of for-loops for iteration.
- Utilize destructuring to simplify code when extracting properties from objects.
- Apply the ternary operator to concisely assign grades based on score criteria.

That's it for today!

Keep coding, stay curious, and enjoy your JavaScript journey!