

Full Stack Development (23CSPE311)

Module-2:MERN and React Components

Introduction to MERN: MERN components, Serverless Hello world program. React Components: Issue Tracker, React Classes, Composing Components, Passing Data Using Properties, Passing Data Using Children, Dynamic Composition

Introduction to MERN: MERN components

What is Stack and SPA?

Any web application is built using multiple technologies. The combinations of these technologies is called a “stack,” popularized by the LAMP stack, which is an acronym for Linux, Apache, MySQL, PHP, which are all open-source software.

As web development matured and their interactivity came to the fore, Single Page Applications (SPAs) became more popular.

An SPA is a web application paradigm that avoids fetching the contents of an entire web page from the server to display new contents. It instead uses lightweight calls to the server to get some data or snippets and changes the web page.

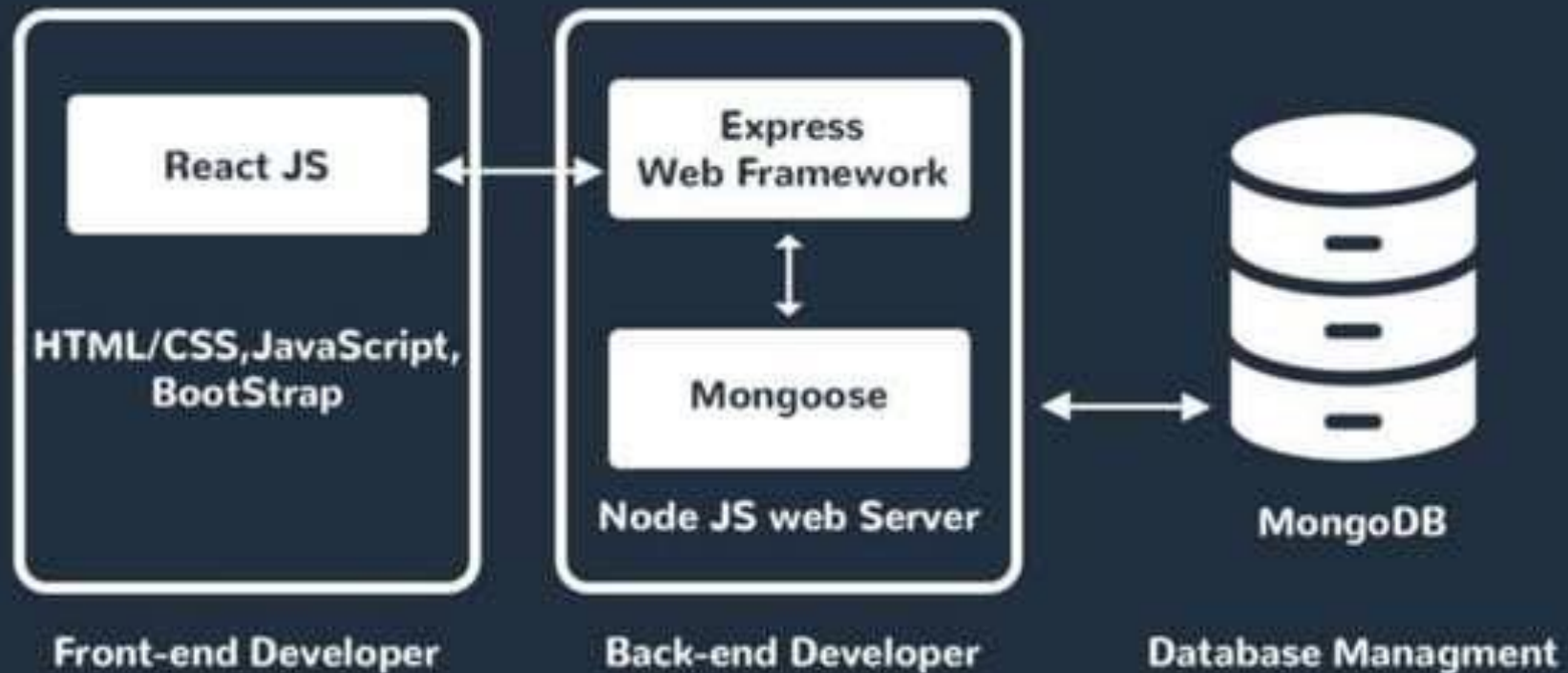
MERN Components

- 1. React
- 2. NODEJs
- 3. Express
- 4. MongoDB

Introduction

- MERN stands for MongoDB, Express, React, Node, after the four key technologies that make up the stack that is used for easier and faster deployment of full-stack web applications.
- MongoDB - document database
- Express(.js) - Node.js web framework
- React(.js) - a client-side JavaScript library
- Node(.js) - the premier JavaScript web server

MERN Stack Development



How does the MERN Stack work?

- The MERN architecture allows you to easily construct a 3-tier architecture (frontend, backend, database) entirely using JavaScript and JSON.



ReactJS

- A JavaScript library for building user interfaces for web and mobile applications.
- React is used to build single-page applications.
- React allows us to create reusable UI components.
- React-router to handle the front-end routing.
- React was created by Jordan Walke, a Software Engineer at Facebook.

Why to use React?

- JSX (JavaScript XML) makes it easier and simpler to write React components.
- ReactJS supports Components. These components also promote code reusability and make the overall web application easier to understand and debug.

How does React work?

- Instead of manipulating the browser's DOM directly, React creates a Virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM.
- React finds out what changes have been made, and changes **only** what needs to be changed.

MERN components: React

1. React, an alternate front-end technology created by Facebook, has been gaining popularity and offers an alternative to AngularJS. It thus replaces the "A" with an "R" in MEAN, to give us the MERN Stack.
2. React is an open-source JavaScript library maintained by Facebook that can be used for creating views rendered in HTML. Unlike AngularJS, React is not a framework. It is a library. Thus, it does not, by itself, dictate a framework pattern such as the MVC pattern
3. A React component declares how the view looks, given the data. When the data changes, if you are used to the jQuery way of doing things, you'd typically do some DOM manipulation.
4. If, for example, a new row has been inserted in a table, you'd create that DOM element and insert it using jQuery. But not in React. You just don't do

React: Component based

- The fundamental building block of React is **a component** that maintains its own state and renders itself.
- In React, all you do is build components. Then, you put components together to make another component that depicts a complete view or page.
- A component encapsulates the state of data and the view, or how it is rendered. This makes writing and reasoning about the entire application easier, by splitting it into components and focusing on one thing at a time.
- Components talk to each other by sharing state information in the form of read-only properties to their child components and by callbacks to their parent components.

React: No templates

- ❖ Many web application frameworks rely on templates to automate the task of creating repetitive HTML or DOM elements.
- ❖ The templating language in these frameworks is something that the developer will have to learn and practice. Not in React.
- ❖ React uses a full-featured programming language to construct repetitive or conditional DOM elements. That language is none other than JavaScript.
- ❖ For example, when you want to construct a table, you'd write a `for(...)` loop in JavaScript or use the `map()` function of `Array`. There is an intermediate language to represent a Virtual DOM, and that is JSX (short for JavaScript XML), which is very much like HTML.

React: Isomorphic

- React can be run on the server too. That's what isomorphic means: the same code can run on the server and the browser. This allows you to create pages on the server when required,



NodeJS

- JavaScript run-time environment built on Chrome's V8 JavaScript.
- Node.js allows you to run JavaScript on the server.
- It is free & open source, written in C++.
- Ryan Dahl developed Node.js in 2009. He embedded C++ code with Chrome's V8 Engine and gave the name as Node.js.
- Node.js runs single-threaded, non-blocking, asynchronous programming, which is very memory efficient.

Where to use NodeJS?

- Back-end services such as APIs.
- Highly scalable, data-intensive and real-time apps.
- I/O bounds applications.
- Single page applications.

Where not to use NodeJS?

Node.js is not used in CPU-intensive apps which requires calculations done by CPU.

NodeJs

The creators of Node.js just took Chrome's V8 JavaScript engine and made it run independently as a JavaScript runtime.

If you are familiar with the Java runtime that runs Java programs, you can easily relate to the JavaScript runtime: the Node.js runtime runs JavaScript programs.

Netflix, Uber, and LinkedIn are a few companies that use Node.js in production, and that should lend credibility as a robust and scalable environment to run the back-end of any application

NodeJs Modules

In a browser, you can load multiple JavaScript files, but you need an HTML page to do all that. You cannot refer another JavaScript file from one JavaScript file. But for Node.js, there is no HTML page that starts it all.

In the absence of the enclosing HTML page, Node.js uses its own module system based on CommonJS to put together multiple JavaScript files.

Modules are like libraries. You can include the functionality of another JavaScript file (provided it's written to follow a module's specifications) by using the `require` keyword (which you won't find in a browser's JavaScript).

You can therefore split your code into files or modules for the sake of better organization and load them using `require`. I'll talk about the exact syntax in a later chapter; at this point it's enough to note

NodeJs -npm

npm is the default package manager for Node.js. You can use npm to install third-party libraries (packages) and manage dependencies between them.

The npm registry (www.npmjs.com) is a public repository of all modules published by people for the purpose of sharing.

Although npm started off as a repository for Node.js modules, it quickly transformed into a package manager for delivering other JavaScript based modules, notably, those that can be used in the browser.

NodeJs is event driven

- Node.js has an asynchronous, event driven, non-blocking input/output (I/O) model, as opposed to using threads to achieve multi-tasking. Most other languages depend on threads to do things simultaneously.
- But in fact, there is no such thing as simultaneous when it comes to a single processor running your code. Threads give the feeling of simultaneousness by letting other pieces of code run while one piece waits (blocks) for some event to complete.
- Typically, these are I/O events such as reading from a file or communicating over the network. For example, on one line, you make a call to open a file, and on the next line, you have your file handle ready.
- What really happens is that your code is blocked (doing nothing) while the file is being opened. If you have another thread running, the operating system or the language will switch out this code and start running some other code during the blocked period.

NodeJs is event driven

- Node.js, on the other hand, has no threads. It relies on callbacks to let you know that a pending task is completed. So, if you write a line of code to open a file, you supply it with a callback function to receive the results—the file handle.
- On the next line, you continue to do other things that don't require the file handle. If you are used to asynchronous Ajax calls, you will immediately know what I mean.
- Event driven programming is natural to Node.js due to the underlying language constructs of JavaScript such as closures. Node.js achieves multi-tasking using an event loop. This is nothing but a queue of events that need to be processed, and callbacks to be run on those events.
- In the previous example, the file being ready to be read will be an event that will trigger the

Express



JS

ExpressJS

- Flexible Node.js framework that provides robust set of features for web for web and mobile application.
- It provides easy routing of requests based on HTTP methods and URLs.
- It allows to set up middlewares to respond to HTTP Requests.
- Allows to dynamically render HTML Pages based on passing arguments to templates.

Express

- Node.js is just a runtime environment that can run JavaScript. To write a full-fledged web server by hand on Node.js directly is not easy, neither is it necessary.
- Express is a framework that simplifies the task of writing the server code. The Express framework lets you define routes, specifications of what to do when an HTTP request matching a certain pattern arrives.
- The matching specification is regular expression (regex) based and is very flexible, like most other web application frameworks. The what-to-do part is just a function that is given the parsed HTTP request.
- Express parses the request URL, headers, and parameters for you. On the response side, it has, as expected, all functionality required by web

Express middleware

- Further, you can write Express middleware, custom pieces of code that can be inserted in any request/response processing path to achieve common functionality such as logging, authentication, etc. Express does not have a template engine built in, but it supports any template engine of your choice such as pug, mustache, etc.
- But, for an SPA, you will not need to use a server-side template engine. This is because all dynamic content generation is done on the client, and the web server only serves static files and data via API calls. In summary, Express is a web server framework meant for

Express middleware

- Further, you can write Express middleware, custom pieces of code that can be inserted in any request/response processing path to achieve common functionality such as logging, authentication, etc. Express does not have a template engine built in, but it supports any template engine of your choice such as pug, mustache, etc.
- But, for an SPA, you will not need to use a server-side template engine. This is because all dynamic content generation is done on the client, and the web server only serves static files and data via API calls. In summary, Express is a web server framework meant for



mongoDB®

MongoDB

- NoSQL database used for high-volume data storage.
- Open-source document-oriented database.
- MongoDB is written in C++.
- It stores data in JSON format.
- Can be easily used with Node.
- MongoDB uses BSON to query database.
- Documents containing key-value pairs are the basic units of data in MongoDB.



Database Concepts

Tabular (Relational)	MongoDB
Database	Database
Table	Collection
Row	Document
Index	Index
Join	\$lookup
Foreign Key	Reference

Why MongoDB?

- **Fast** – Being a document-oriented database, easy to index documents. Therefore a faster response.
- **Scalability** – Large data can be handled by dividing it into several machines.
- **Use of JavaScript** – MongoDB uses JavaScript which is the biggest advantage.
- **Schema Less** – Any type of data in a separate document.
- Data stored in the form of JSON.
- **Simple Environment Setup** – Its really simple to set up MongoDB.

MongoDB

- MongoDB is the database used in the MERN stack. It is a NoSQL document-oriented database, with a flexible schema and a JSON-based query language
- NoSQL stands for "non-relational," no matter what the acronym expands to. It's essentially not a conventional database where you have tables with columns and rows and strict relationships among them.
- There are two attributes of NoSQL databases that differentiate them from the conventional. The first is their ability to horizontally scale by distributing the load over multiple servers.
- They do this by sacrificing an important (for some) aspect of the traditional databases: strong consistency. That is, the data is not necessarily consistent for very brief amounts of time across replicas.

MongoDB : document-oriented

- Compared to relational databases where data is stored in the form of relations, or tables, MongoDB is a document-oriented database. The unit of storage (comparable to a row) is a document, or an object, and multiple documents are stored in collections (comparable to a table).
- Every document in a collection has a unique identifier, using which it can be accessed.
- The identifier is indexed automatically. Imagine the storage structure of an invoice, with the customer name, address, etc. and a list of items (lines) in the invoice.
- If you had to store this in a relational database, you would use two tables, say, invoice and invoice_lines, with the lines or items referring to the invoice via a foreign-key relation. Not so in MongoDB. You would

MongoDB : Schema-Less

- Schema-Less Storing an object in a MongoDB database does not have to follow a prescribed schema. All documents in a collection need not have the same set of fields.
- This means that, especially during early stages of development, you don't need to add/rename columns in the schema

MongoDB : JavaScript Based

- MongoDB's language is JavaScript. For relational databases, we had a query language called SQL. For MongoDB, the query language is based on JSON. You create, search for, make changes, and delete documents by specifying the operation in a JSON object.
- The query language is not English-like (you don't SELECT or say WHERE), and therefore much easier to construct programmatically. Data is also interchanged in JSON format.
- In fact, the data is natively stored in a variation of JSON called BSON (where B stands for Binary) in order to efficiently utilize space. When you retrieve a document from a collection, it is returned as a JSON object.
- MongoDB comes with a shell that's built on top of a JavaScript runtime like Node.js. This means that you

React Library

- React library is available as a JavaScript file that we can include in the HTML file using
- the `<script>` tag.
- It comes in two parts: the first is the React core module, the one that is responsible for
- dealing with React components, their state manipulation, etc.
- The second is the ReactDOM module, which
- deals with converting React components to a DOM that a browser can understand. These two libraries
- can be found in unpkg, a Content Delivery Network (CDN) that makes all open-source JavaScript libraries
- available online. Let's use the development (as opposed to production) version of the libraries from the
- following URLs:
 - React: `https://unpkg.com/react@16/umd/react.development.js`
 - ReactDOM: `https://unpkg.com/react-dom@16/umd/react-dom.development.js`
- These two scripts can be included in the `<head>` section using `<script>` tags like this:
- ...
- `<script src="https://unpkg.com/react@16/umd/react.development.js"></script>`
- `<script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>`

Serverless Hello world program

```
<!DOCTYPE HTML> //index.html
<html>
<head>
<meta charset="utf-8">
<title>Pro MERN Stack</title>
<script src="https://unpkg.com/react@16/umd/react.development.js"></script>
<script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
</head>
<body>
<div id="contents"></div>
<script>
const element = React.createElement('div', {title: 'Outer div'},
React.createElement('h1', null, 'Hello World!')
);
ReactDOM.render(element, document.getElementById('content'));
</script>
</body>
</html>
```

Hello world in React



Figure 2-1. A Hello World written in React

index.html: Changes for Using JSX

- `<!DOCTYPE HTML>`
- `<html>`
- `<head>`
- `<meta charset="utf-8">`
- `<title>Pro MERN Stack</title>`
- `<script src="https://unpkg.com/react@16/umd/react.development.js"></script>`
- `<script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>`
- `<script src="https://unpkg.com/@babel/standalone@7/babel.min.js"></script>`
- `</head>`
- `<body>`
- `<div id="contents"></div>`
- `<script type="text/babel">`
- `const element = React.createElement('div', {title: 'Outer div'},`
- `React.createElement('h1', null, 'Hello World!')`
- `);`
- `const element = (`
- `<div title="Outer div">`
- `<h1>Hello World!</h1>`
- `</div>`
- `);`
- `ReactDOM.render(element,`
- `document.getElementById('contents'));`
- `</script>`
- `</body>`
- `</html>`

Issue Tracker

- **Introduction**

- The Issue Tracker is a core example application used to demonstrate key React principles.
- It is a simple, full-stack application for managing a list of software issues.
- This example showcases how to build a complex UI by composing smaller, more manageable components.

Issue tracker

- In the case of the Issue Tracker, we'll only deal with a single object or record, because that's good
- enough to depict the pattern. Once you grasp the fundamentals of how to implement the CRUD pattern in
- MERN, you'll be able to replicate the pattern and create a real-life application.
- Here's the requirement list for the Issue Tracker application, a simplified or toned-down version of
- GitHub Issues or Jira:
 - • The user should be able to view a list of issues, with an ability to filter the list by
 - various parameters.
 - • The user should be able to add new issues, by supplying the initial values of the
 - issue's fields.
 - • The user should be able to edit and update an issue by changing its field values.
 - • The user should be able delete an issue

Issue tracker

- An issue should have following attributes: • A title that summarizes the issue (freeform long text)
- • An owner to whom the issue is assigned (freeform short text)

react Components

- • A status indicator (a list of possible status values)
- • Creation date (a date, automatically assigned)
- • Effort required to address the issue (number of hours)

Issue tracker

- An issue should have following attributes:
- A title that summarizes the issue (freeform long text)
- An owner to whom the issue is assigned (freeform short text)
- A status indicator (a list of possible status values)
- Creation date (a date, automatically assigned)
- Effort required to address the issue (number of days, a number)
- Estimated completion date or due date (a date,

Core Components

- **Core Components**

- The Issue Tracker app is composed of a few key components, organized in a hierarchy:
 - **IssueList**: The main container component that holds the other parts of the UI. It manages the list of issues.
 - **IssueFilter**: A component for filtering the issues based on their status or other criteria.
 - **IssueTable**: A component that displays the list of issues in a table format.
 - **IssueRow**: A component that represents a single row in the IssueTable. It is a child of IssueTable.
 - **IssueAdd**: A component for adding a new issue to the list.

Component Hierarchy and Data Flow

- **Component Hierarchy and Data Flow**
- The components are arranged in a tree-like hierarchy:
 - IssueList
 - IssueFilter
 - IssueTable
 - IssueRow (multiple instances)
 - IssueAdd
- Data flows in a unidirectional manner. The IssueList component holds the state (the array of issues) and passes it down to IssueTable as a prop.
- IssueTable then passes a single issue object as a prop to each IssueRow component

Building the IssueTable Component

- The IssueTable component is a functional or class component that receives an array of issues as a prop.
- Its job is to render a <table> and map over the issues prop to create a list of IssueRow components.

- **Example Code Snippet:**

```
function IssueTable(props) {
  const issueRows = props.issues.map(issue =>
    <IssueRow key={issue.id} issue={issue} />
  );
  return (
    <table>
      <thead>
        <tr>
          <th>ID</th>
          <th>Status</th>
          <th>Owner</th>
          <th>Title</th>
        </tr>
      </thead>
      <tbody>
        {issueRows}
      </tbody>
    </table>
  );
}
```

The IssueRow Component

- **The IssueRow Component**
- The IssueRow is a simple, stateless functional component.
- It receives a single issue object as a prop.
- Its sole purpose is to display the data for one issue in a table row (<tr>).

- **Example Code Snippet:**

```
function IssueRow(props) {
  const issue = props.issue;
  return (
    <tr>
      <td>{issue.id}</td>
      <td>{issue.status}</td>
      <td>{issue.owner}</td>
      <td>{issue.title}</td>
    </tr>
  );
}
```

State and Re-rendering

- **State and Re-rendering**

- The IssueList component uses state to manage the list of issues.
- When a new issue is added via the IssueAdd component, the IssueList component updates its state.
- This state change triggers a re-render of IssueList and all its children (IssueTable, IssueRows), automatically updating the UI to show the new issue.
- This demonstrates a key benefit of React: automatic UI updates based on state changes

- The Issue Tracker application is a practical example of the React principles of **components**, **unidirectional data flow**, and **state management**.
- By breaking down the problem into small, reusable components, the application becomes more organized, maintainable, and easier to scale.
- This approach is a cornerstone of "The React Way" and is fundamental to building any modern React application.

Introduction to React Components

- **Introduction to React Components**
- **What are they?** The fundamental building blocks of a React application.
- **A simple analogy:** Just as a car is made up of many individual parts (engine, wheels, chassis), a React app is made up of many components.
- Each component is a self-contained, reusable piece of code that defines a part of the user interface.
- Components allow for a modular and organized approach to UI development.

Props in React Components

- • Props (short for properties) are inputs to components.
- • They are passed from parent to child components.
- • Props are read-only.

State in React Components

- • State is managed within the component.
- • It allows components to create dynamic and interactive UIs.
- • State can be updated using `setState()` in class components or `useState()` in functional components.

Lifecycle Methods

- • Mounting: `componentDidMount()`
- • Updating: `componentDidUpdate()`
- • Unmounting: `componentWillUnmount()`
- • Used for managing side effects and component behavior.

Types of Components

- **Class Components:**

- Defined using ES6 classes.
- Extend `React.Component` and have a `render()` method that returns JSX.
- Can have their own state and lifecycle methods.
- Example:

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

- **Functional Components:**

- Defined as a JavaScript function that returns JSX.
- Simpler syntax and are generally preferred in modern React development, especially with the introduction of Hooks.
- Example:

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

Class Component Example

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

Props - The Component's Inputs

- **Props (Properties):** A way to pass data from a parent component to a child component.
- **Unidirectional Data Flow:** Props are read-only. A child component cannot modify the props it receives from its parent. This is a core principle of "The React Way" discussed in the book.
- **Example:**

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

// Child Component
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

State - The Component's Memory

- **State:** An object that holds data that can change over time.
- When the state of a component changes, React automatically re-renders the component to reflect the new data.
- **Class Components:** Use `this.state` and `this.setState()`.
- **Functional Components:** Use the `useState` Hook.
- **Key Concept:** State should be managed by the component that needs it, and passed down to child components as props.

Composing Components

- **Composition:** The practice of building complex components by combining simpler, more focused components.
- This leads to highly reusable, maintainable, and readable code.
- **Example from the book's context:** Building an "Issue Tracker" app by composing smaller components like IssueTable, IssueRow, and IssueFilter.

```
function IssueList() {  
  return (  
    <div>  
      <IssueFilter />  
      <IssueTable />  
    </div>  
  );  
}
```

Component Hierarchy and Communication

- **Hierarchy:** Components are organized in a tree-like structure. Data typically flows down the tree from parent to child via props.
- **Child-to-Parent Communication:** While props flow down, a child component can communicate back to its parent by calling a function passed down as a prop.
 - The parent defines a function.
 - The parent passes this function as a prop to the child.
 - The child calls the function to notify the parent of an event (e.g., a button click).

Stateless Components

- **Stateless Components:** Functional components that do not manage their own state. They receive data solely through props and are responsible only for rendering the UI.
- These components are also referred to as "pure components" or "presentational components" in some older literature.
- They are highly predictable and easy to test, making them a best practice in modern React development.

Summary-Components

- Components are the fundamental building blocks of React applications.
- They can be simple functions or classes with state and lifecycle methods.
- Data flows from parent to child using **props**, which are read-only.
- Internal, mutable data is managed by a component's **state**.
- **Composition** is key to building complex and reusable user interfaces.

React.JS – Styling the Components

Introduction

- Different ways to style React Components
 - **Inline CSS** - using style attribute
 - CSS in JS - Using Libraries JSS and Styled Components
 - CSS modules – css loader and Sass & SCSS
 - Stylable



React.JS – Styling the Components

Inline CSS

- To style an element with the inline **style attribute**, the value must be a JavaScript object
- In JSX, JavaScript expressions are written inside curly braces and JavaScript objects also use curly braces

- Example:

```
class MyHeader extends React.Component {  
  render() { return (  
    <h1 style={{color: "red"}}>Hello Style!</h1>  
    <p>Add a little style!</p>  ); }  
}
```

- camelCased Property Names

```
<h1 style={{backgroundColor: "lightblue"}}>Hello Style!</h1>
```



React.JS – Styling the Components

Inline CSS continued..

- Creation of javascript object which contains the css properties and values in the form of key value pair
- Passing this as a value to style attribute
- Example:

```
class Letter extends React.Component {
  render() {
    var letterStyle = {
      padding: 10,
      margin: 10,
      backgroundColor: "#ffde00",
      color: "#333",
      display: "inline-block",
      fontFamily: "monospace",
      fontSize: 32,
      textAlign: "center"
    };
    return (
      <div style={letterStyle}>
        {this.props.children}
      </div>
    );
  }
}
```



React.JS – Complex Components

- Introduction
- Approach followed
- Complex Component creation
- Sub component creation
- Properties from parent to sub component
- Modified parent and sub components
- Transferring properties

React.JS – Complex Components

Introduction

- Combining components to create the complex one
- Advantage is composability
- Approach
 - Identify the major visual elements
 - Breaking them into individual components



React.JS – Complex Components

Approach followed



tiger | Facts, Information, & Habitat ...
britannica.com



tiger | Facts, Information, & Habitat ...
britannica.com

- The complex component consists of **Image rendering**, **caption rendering** and **link rendering** on the web page



React.JS – Complex Components

Creation of complex component

```
class SrchResult extends React.Component {  
  render() {  
    return (  
      <div>  
        <ResImage/>  
        <ResCaption/>  
        <ResLink/>  
      </div>  
    );  
  }  
}
```



React.JS – Complex Components

Creation of sub components

```
class ResImage extends React.Component
{
  render() {
    return (
      <div> 
      </img> </div> );
  }
}
```

```
class ResLink extends React.Component {
  render() {
    return (
      <div> <a
      href="https://www.britannica.com/animal/tiger">
      britannica.com </a> </div> );
  }
}
```

```
class ResCaption extends React.Component {
  render() {
    return (
      <div> <p>tiger | Facts, Information and
      Habitat...</p> </div> );
  }
}
```



React.JS – Complex Components

Properties from parent to sub component

To avoid hard coding, properties have to be passed from parent Components to it's sub components

- `src`
- `href`
- `linktext`
- `caption`

```
<SrcResult      src="tiger.jpg"      href="https://www.britannica.com/animal/tiger"  
  linktext="britannica.com" caption="tiger | Facts, Information and Habitat..." />
```



React.JS – Complex Components

Modified Parent and sub components

```
class SrchResult extends React.Component {  
  render() {return ( <div>  
    <ResImage {...this.props}/> <ResCaption {...this.props}/> <ResLink {...this.props}/>  
    </div> ); }  
}
```

```
class ResImage extends React.Component {  
  render() {  
    return (  
      <div> <img src={this.props.src}>  
      </img> </div>  
    ); } }  
}
```

```
class ResLink extends React.Component {  
  render() {  
    return (  
      <div> <a href={this.props.href}> {this.props.linktext}  
      </a> </div> );  
    } }  
}
```

```
class ResCaption extends React.Component {  
  render() {  
    return (  
      <div> <p>{this.props.caption}</p> </div>  
    ); } }  
}
```

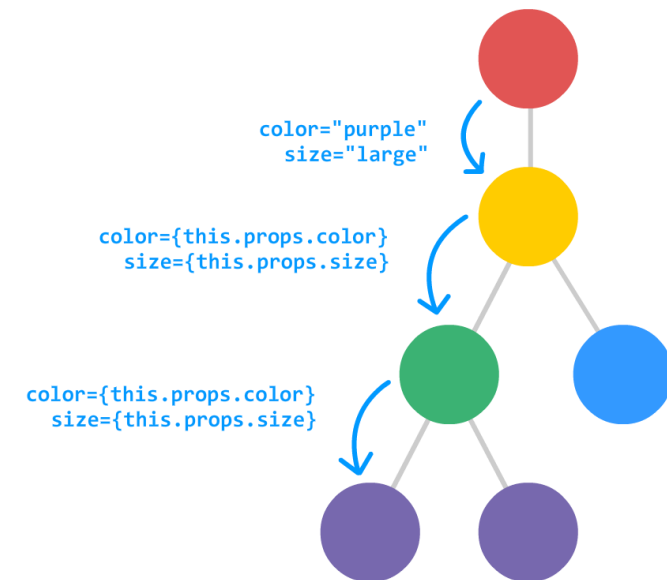


React.JS – Complex Components

Transferring Properties

- Transferring properties was not a tedious as there was only one level of hierarchy
- If there are multiple levels of components, transferring properties will be very cumbersome
- To overcome that, we use the **spread operator (...)**

`<Display {...this.props}/>`



Dynamic composition

- **Dynamic composition** in React is a powerful technique for creating flexible and reusable user interfaces. Instead of hard-coding the components you want to render inside another component, you can pass a component as a **prop** or use it as a **child** element. This allows you to choose which component to display at runtime based on certain conditions or data.
- Think of it like this:
- **Static Composition:** Building a car with a specific, built-in engine. You can't swap it out later. The engine component is fixed inside the car component.
- **Dynamic Composition:** Building a car with a special slot for an engine. You can then choose to put a V8, an electric motor, or a hybrid engine in that slot. The car component is generic and can be composed with different "engine" components.
- In React, this means a parent component can decide what to render inside it. It doesn't need to know the specific details of the components it's rendering; it only needs to know how to handle them as generic placeholders.

How Dynamic Composition is Implemented?

- How Dynamic Composition is Implemented in React
- Dynamic composition is typically implemented using one of two main patterns:
- Using `props.children`: This is the most common method. A component is designed to accept any other component or HTML element as a child. The parent component then renders the child using `props.children`. This is used extensively in React for creating layout components, such as a `<Card>` component that can contain any content you place inside it.

example

- `// Generic Card component`
- `function Card(props) {`
- `return (`
- `<div className="card">`
- `{props.children}`
- `</div>`
- `);`
- `}`

- `// Usage: The content inside <Card> is passed as `props.children``
- `<Card>`
- `<h2>Title</h2>`
- `<p>Some content here.</p>`
- `</Card>`

Passing a Component as a Prop:

- **Passing a Component as a Prop:** This method involves passing a component itself as a prop to another component. The receiving component can then render the passed component wherever it needs. This is useful when you need to change a specific part of a component's internal structure.
- ```
// Component that takes another component as a prop
```
- ```
function UserProfile({ ProfileComponent }) {
```
- ```
 return (
```
- ```
    <div>
```
- ```
 <h1>User
```
- ```
      Profile</h1>
```
- ```
 <ProfileComponent
```
- ```
    />
```
- ```
 </div>
```
- ```
);
```
- ```
}
```

# Passing a Component as a Prop:

- `// Different profile components`
- `function BasicProfile() {`
- `return <div>Basic user info...</div>;`
- `}`
  
- `function AdminProfile() {`
- `return <div>Admin-specific controls...</div>;`
- `}`
  
- `// Usage: Dynamically choose which component to render`
- `<UserProfile ProfileComponent={isUserAdmin ? AdminProfile : BasicProfile} />`