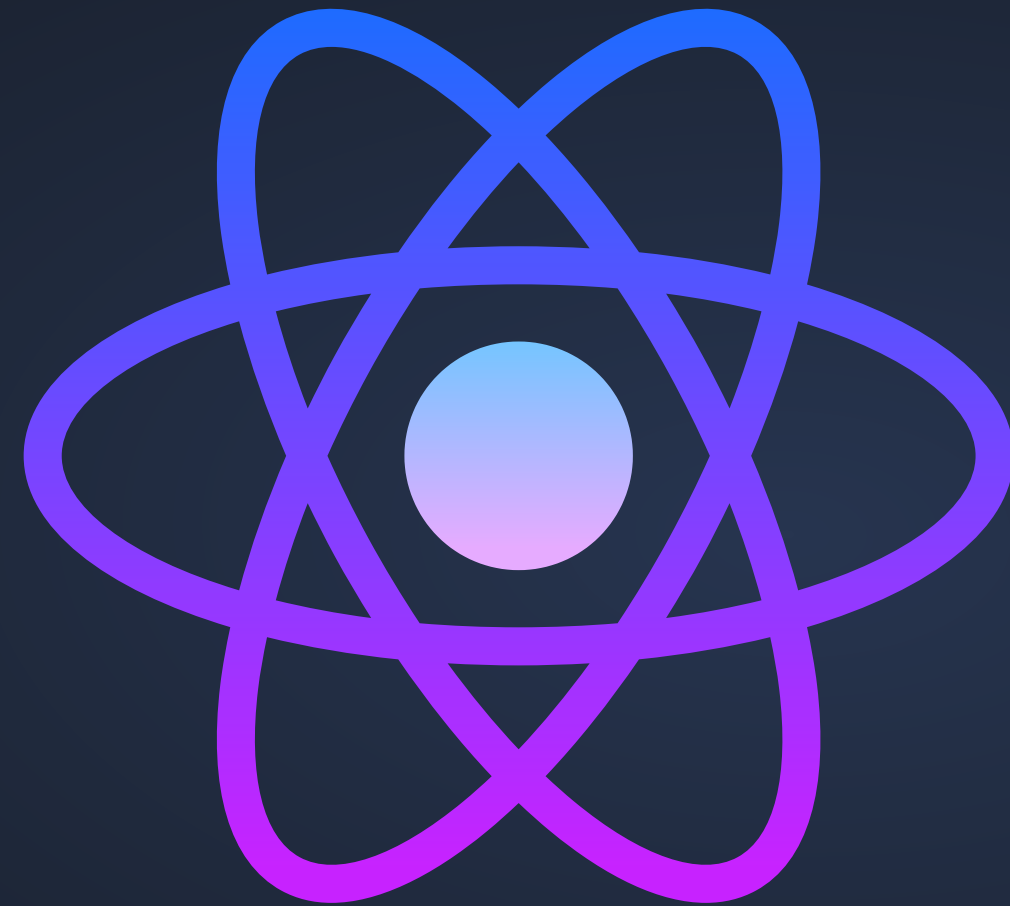


REACT JS

01. React is a popular **JavaScript library** used for **building user interfaces**.
02. It was developed by Facebook and is now maintained by a community of developers.
03. React uses a **component-based architecture**, where UI elements are broken down into reusable and modular pieces.



COMPONENT CONCEPT

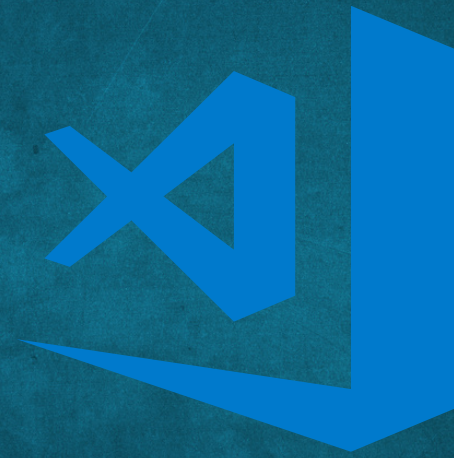
- Components are independent and reusable bits of code.
- They serve the same purpose as JavaScript functions, but work in isolation and return HTML
- Components come in two types, Class components and Function components

Content is no longer available

Tools you need to install



NODE JS



Visual Studio Code



WebStorm

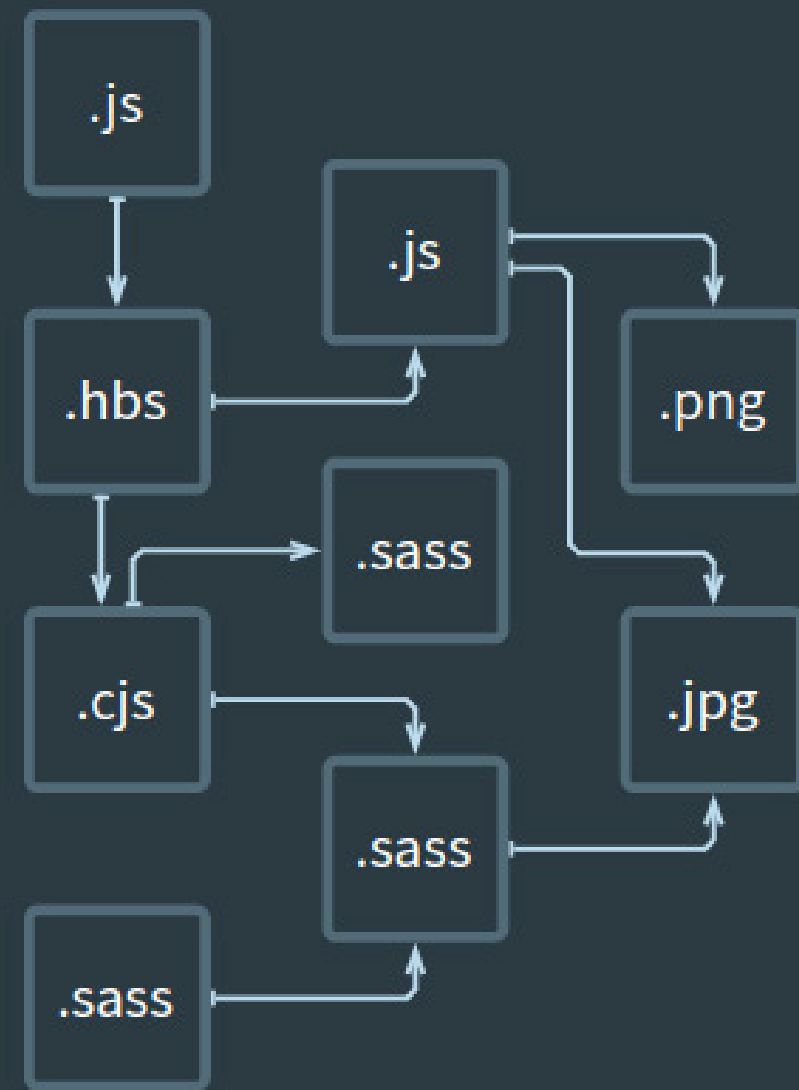
CREATE & RUN REACT APP USING VITE

ViteJS is a modern build tool and development server that aims to optimize the front-end development experience.

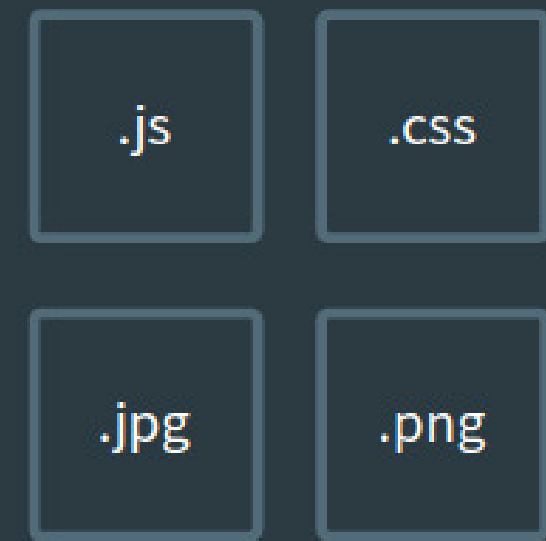
```
npm create vite@latest
```



ASSET BUNDLING



MODULES WITH DEPENDENCIES



STATIC ASSETS

WHY VITE

- 01.** **Faster build times:** Vitejs uses esbuild, which is a blazing fast compiler that can build projects in a fraction of the time that it takes other build tools.
- 02.** **Hot reload:** Vitejs has hot reload, which means that your code will be updated in real time as you make changes, so you can see the results of your changes immediately
- 03.** **Small bundle size:** Vitejs bundles are typically much smaller than bundles created by other build tools, which can improve performance and reduce bandwidth usage.



ESSENTIAL

Vite Commands



```
"dev": "vite",  
"build": "vite build",  
"preview": "vite preview"
```



REACT PROJECT

Structure

- | | |
|----------------|----------------|
| ▲ Distribution | ▲ Source |
| ▲ Node Modules | ▲ Package.json |
| ▲ Public | ▲ Vite Config |

```
> dist
> node_modules
> public
> src
🔒 .gitignore
<> index.html
{} package-lock.json
{} package.json
JS vite.config.js
```


MY FIRST FUNCTIONAL COMPONENT

In React, a functional component is a type of component that is defined using a JavaScript function.

- 01.** They are easier to read and write.
- 02.** They are simpler and lighter, making them faster to render.
- 03.** They do not require the use of the "this" keyword, making them less error-prone.
- 04.** They can take advantage of React's Hooks, which allow you to use state and other features without using class components.

```
App.jsx

1  import React from 'react';
2  const App = () => {
3    return (
4      <div>
5        <h1>
6          This is my
7          first component
8        </h1>
9      </div>
10 );
11 };
12 export default App;
```

JSX JAVASCRIPT XML

- JSX is a **syntax extension for JavaScript** that allows you to write HTML-like code in your JavaScript code
- It is commonly used in React applications to define the structure and content of UI components.
- JSX is not a separate language, but a preprocessor that converts the HTML-like code into plain JavaScript.
- It enables you to use **JavaScript expressions within your HTML-like code**, making it easier to dynamically generate content.
- JSX can improve code readability and maintainability by allowing developers to write declarative, intuitive code.

●●● App.jsx

```
4  <div>
5    <h1>Q: What time is it now</h1>
6    <h6>A: It is {new Date().getTime()} 0 Clock</h6>
7  </div>
```

JSX CONVENTIONS

- You need to return a **single parent element** in JSX
- You can implement **JS directly** in JSX
- All Tags **Self-close** in JSX
- **ClassName** and HTMLFor, not **class** and for in JSX
- Write all HTML **Attributes in camelCase** in JSX
- Write Inline **Styles as Objects** in JSX

JSX

Inline if else

```
App.jsx

1  const App = () => {
2    let marks=10
3    return (
4      <div>
5        {
6          marks>80?
7          <h1>Brilliant Result</h1>
8          :
9          <h1>Avarage Result</h1>
10       }
11     </div>
12   );
13 };
14 export default App;
```

IMMEDIATELY-INVOKED

Function expressions
inside your JSX

```
App.jsx

1  const App = () => {
2      let marks=10
3      return (
4          <div>
5              {(()=>{
6                  if(marks>80){
7                      return <h1>Brilliant Result</h1>
8                  }
9                  else{
10                     return <h1>Avarage Result</h1>
11                 }
12             })()}
13          </div>
14      );
15  };
16  export default App;
```

JSX

Loop Inside

App.jsx

```
1  const App = () => {
2    let item=['A','B','C','D'];
3    return (
4      <div>
5        <select>
6          {
7            item.map((item,i)=>{
8              return <option key={i.toString()}>{item}</option>
9            })
10         }
11       </select>
12     </div>
13   );
14 };
15 export default App;
```

JSX

Loop Inside Why
we use map

Method	Runs through each item	Executes given function	Returns the result	Number of elements in result (compared to original array)
.map	✓	✓	in array	=
.filter	✓	✓	if true, in array	= <
.forEach	✓	✓	no <i>return is undefined</i>	none
.reduce	✓	✓	in array or anything else	one (a single number or string) <i>Reduce transforms an array into something else</i>
for loop	✓	until condition is false <i>You know the number of iterations beforehand</i>	<i>They run code blocks. They aren't functions so don't need to return</i>	>, = or <
while loop	✓	while condition is true <i>You don't know the number of iterations beforehand</i>		>, = or <

CONDITIONAL RENDERING

Using an if...else Statement

```
●●● App.jsx

1  const LoginStatusBtn=(status)=>{
2      if(status){
3          return <button>Logout</button>
4      }
5      else{
6          return <button>Login</button>
7      }
8
9  }
10 const App = () => {
11     return (
12         <div>
13             <h1>Login Status</h1>
14             {LoginStatusBtn(false)}
15         </div>
16     );
17 };
18 export default App;
```

CONDITIONAL RENDERING

Using Switch Statement

```
●●● App.jsx

1  const App = () => {
2
3  const isLoggedIn=false
4
5  switch (isLoggedIn) {
6      case true:
7          return <button>Logout</button>;
8      case false:
9          return <button>Login</button>;
10     default:
11         return null;
12     }
13 };
14
15 export default App;
```

CONDITIONAL RENDERING

Using Ternary Operators

```
App.jsx

1  const App = () => {
2    let marks=10
3    return (
4      <div>
5        {
6          marks>80?
7          <h1>Brilliant Result</h1>
8          :
9          <h1>Avarage Result</h1>
10       }
11     </div>
12   );
13 };
14 export default App;
```

CONDITIONAL RENDERING

Using Logical &&

App.jsx

```
1  const App = () => {  
2  
3    let isLoggedIn = true  
4  
5    return (  
6      <div>  
7        <h1>Login Status</h1>  
8        {isLoggedIn && <button>Logout</button>}  
9      </div>  
10     );  
11  };  
12  
13  export default App;
```

CONDITIONAL RENDERING

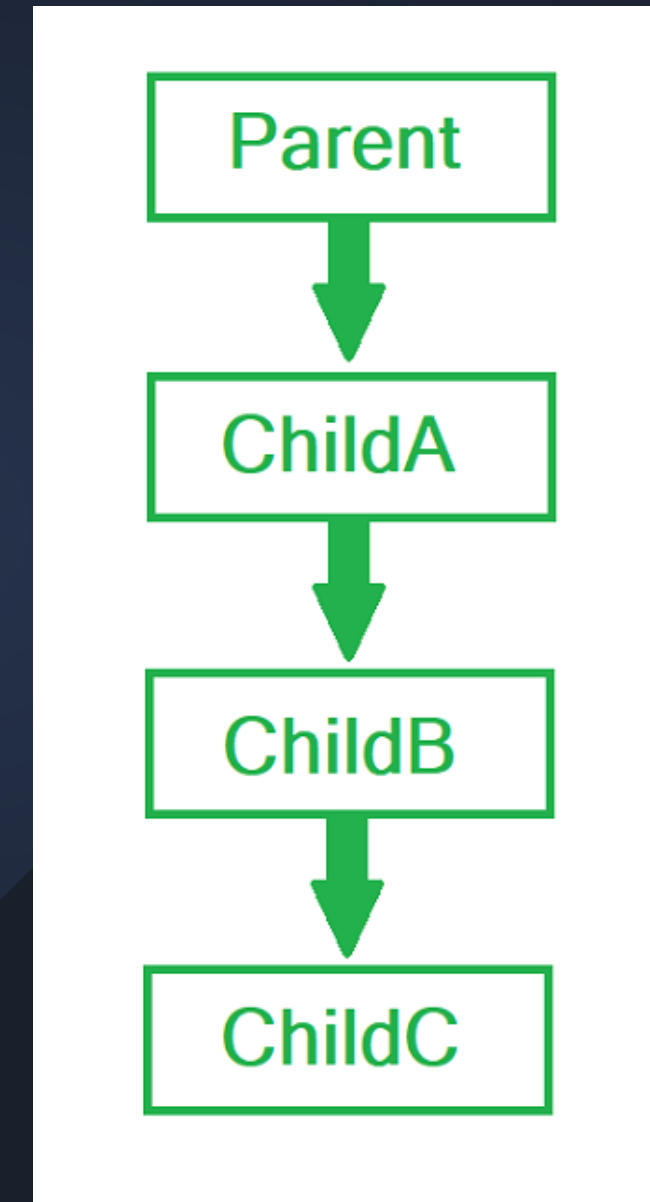
Using Immediately
Invoked Function

App.jsx

```
1  const App = () => {
2    let marks=10
3    return (
4      <div>
5        {(()=>{
6          if(marks>80){
7            return <h1>Brilliant Result</h1>
8          }
9          else{
10           return <h1>Avarage Result</h1>
11          }
12        })()}
13      </div>
14    );
15  };
16  export default App;
```

PASSING PROPS TO A COMPONENT

- The term 'props' is an abbreviation for 'properties'
- Used for passing data from one component to another.
- Props are being passed in a uni-directional flow means one way from parent to child
- Props data is read-only, which means that data coming from the parent should not be changed by child components



PASSING PROPS

Passing simple data

● ● ● App.jsx

```
5  <div>
6    <HeroSection time={new Date().getTime()} />
7  </div>
```

● ● ● HeroSection.jsx

```
3  const HeroSection = (props) => {
4    return (
5      <div>
6        <h1>Q: What time is it now</h1>
7        <h6>A: It is {props.time} O Clock</h6>
8      </div>
9    );
10  };
```


 App.jsx

```
4      const Item= {
5          "id": 1,
6          "name": "Product 1",
7          "description": "This is the description",
8          "price": 19.99,
9          "category": "Category 1"
10     }
11     return (
12         <div>
13             <ProductList Item={Item}/>
14         </div>
15     );
```

PASSING PROPS

Passing with object data

App.jsx

```
5     const handleClick = () => {
6         alert('Button clicked!');
7     };
8     return (
9         <div>
10             <MyButton handleClick={handleClick}/>
11         </div>
12     );
```

MyButton.jsx

```
3     const MyButton = (props) => {
4         return (
5             <div>
6                 <button onClick={props.handleClick}>
7                     Click
8                 </button>
9             </div>
10         );
11     };
```

PASSING PROPS

Passing function

RESPONDING TO EVENTS

Event handlers are your own functions that will be triggered in response to interactions like clicking, hovering, focusing form inputs, and so on

- Different ways to write an event handler
- How to pass event handling logic from a parent component
- How events propagate and how to stop them



RESPONDING TO EVENTS

Adding event handlers

```

●●● App.jsx

4  const App = () => {
5      function handleClick() {
6          alert('You clicked me!');
7      }
8      return (
9          <button onClick={handleClick}>
10             Click me
11          </button>
12      );
13  };

```

```

<button onClick={function handleClick() {
    alert('You clicked me!');
}}>Click me
</button>

```

```

<button onClick={() => {
    alert('You clicked me!');
}}>
Click me
</button>

```

```

<button onClick={alert('You clicked me!')}>
    Click me
</button>

```

RESPONDING TO EVENTS

Preventing default behavior

App.jsx

```
4  const App = () => {
5    function SubmitForm(e) {
6      e.preventDefault();
7      alert('You clicked me!');
8    }
9    return (
10     <form onSubmit={SubmitForm}>
11       <input />
12       <button>Send</button>
13     </form>
14   );
15 };
```