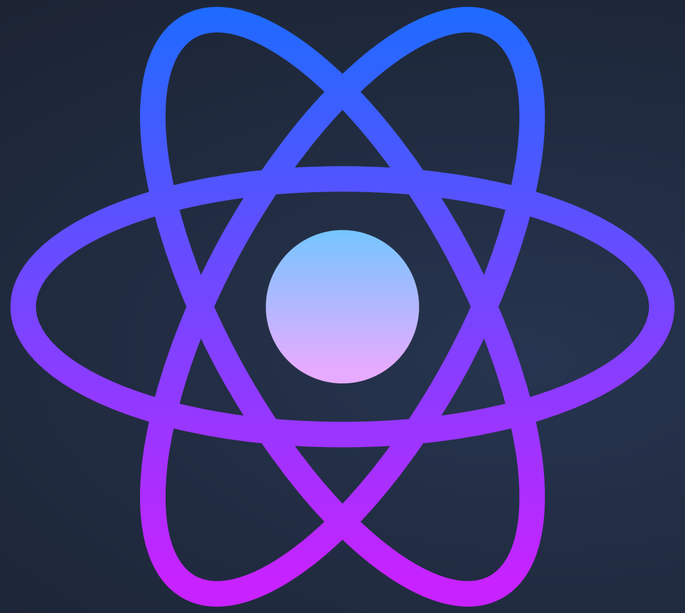
Learn with Rabbil

REACT JS

01. React is a popular **JavaScript library** used for **building user interfaces.**

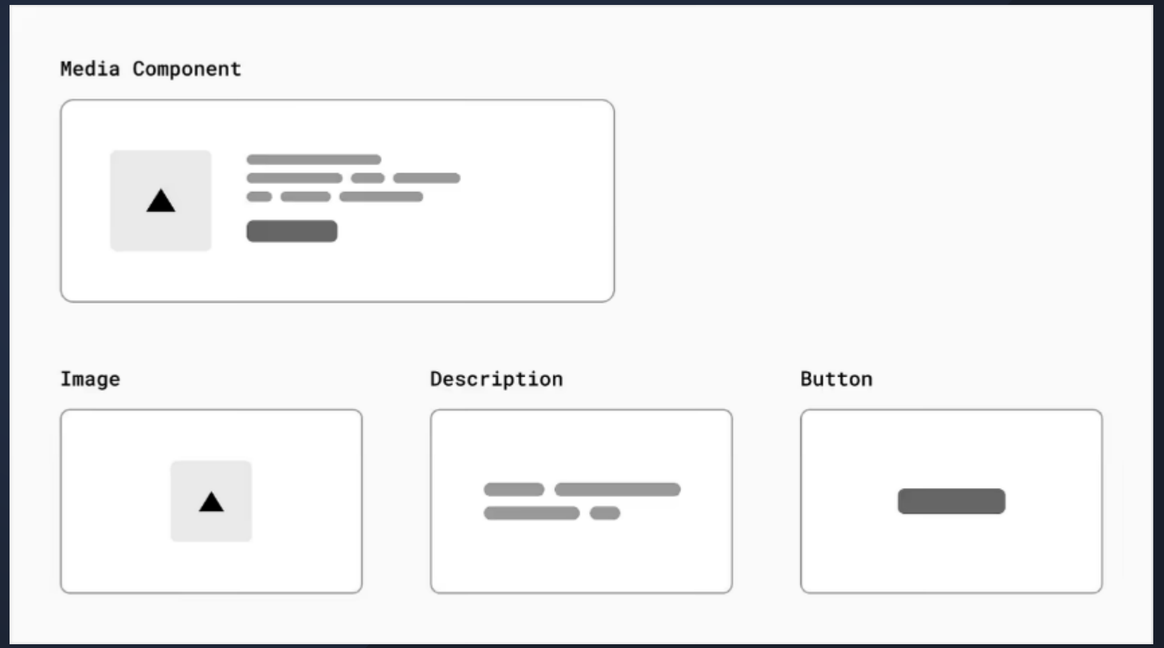
02. It was developed by Facebook and is now maintained by a community of developers.

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



Learn with Rabbil

Tools you need to install

NODE JS

Visual Studio Code

WebStorm

Learn with Rabbil

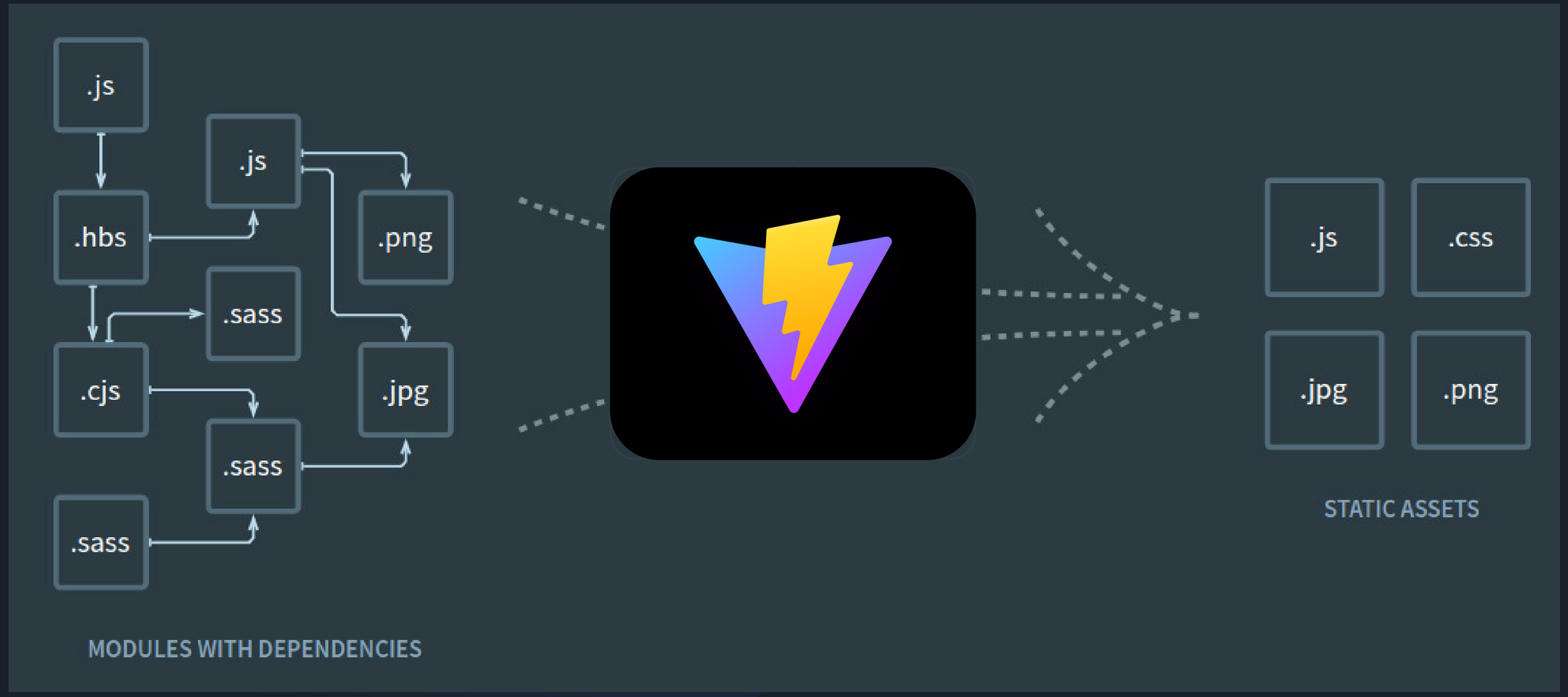
CREATE & RUN REACT APP USING VITE

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

```
npm create vite@latest
```

Learn with Rabbil

ASSET BUNDLING



Learn with Rabbil

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.

Learn with Rabbil

ESSENTIAL

Vite Commands

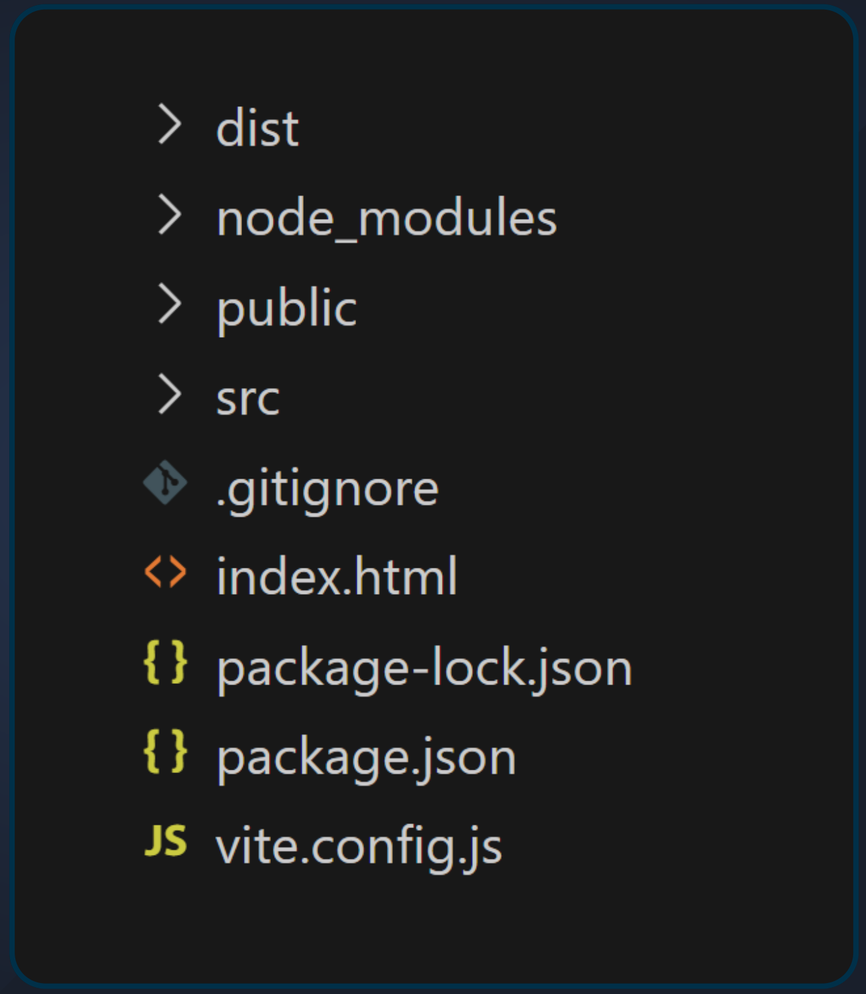
```
"dev": "vite",
"build": "vite build",
"preview": "vite preview"
```

Learn with Rabbil

REACT PROJECT

Structure

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



Learn with Rabbil

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

```jsx
import React from 'react';
const App = () => {
return (
     <div>
          <h1>
               This is my
               first component
          </h1>
     </div>
);
};
export default App;
```

Learn with Rabbil

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()} O Clock</h6>
7   </div>
```

Learn with Rabbil

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

Learn with Rabbil

JSX

Inline if else

App.jsx

```jsx
const App = () => {
    let marks=10
    return (
        <div>
            {
            marks>80?
            <h1>Brilliant Resutl</h1>
            :
            <h1>Avarage Result</h1>
            }
        </div>
    );
};
export default App;
```

Learn with Rabbil

IMMEDIATELY-INVOKED

Function expressions
inside your JSX

App.jsx

```jsx
const App = () => {
    let marks=10
    return (
        <div>
            {(()=>{
              if(marks>80){
                return <h1>Brilliant Result</h1>
              }
              else{
                return <h1>Avarage Result</h1>
              }
            })()}
        </div>
    );
};
export default App;
```

Learn with Rabbil

JSX

Loop Inside

App.jsx

```jsx
const App = () => {
let item=['A','B','C','D'];
return (
    <div>
        <select>
        {
            item.map((item,i)=>{
                return <option key={i.toString()}>{item}</option>
            })
        }
        </select>
    </div>
    );
};
export default App;
```

Learn with Rabbil

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 *return is undefined*	none
.reduce	✔	✔	in array or anything else	one (a single number or string) *Reduce transforms an array into something else*
for loop	✔	until condition is false *You know the number of iterations beforehand*	*They run code blocks. They aren't functions so don't need to return*	>, = or <
while loop	✔	while condition is true *You don't know the number of iterations beforehand*		>, = or <

Learn with Rabbil

CONDITIONAL RENDERING

Using an if...else Statement

App.jsx

```jsx
const LoginStatusBtn=(status)=>{
    if(status){
        return <button>Logout</button>
    }
    else{
        return <button>Login</button>
    }

}
const App = () => {
return (
    <div>
     <h1>Login Status</h1>
     {LoginStatusBtn(false)}
    </div>
    );
};
export default App;
```

Learn with Rabbil

CONDITIONAL RENDERING

Using Switch Statement

App.jsx

```jsx
const App = () => {

const isLoggedIn=false

switch (isLoggedIn) {
    case true:
      return <button>Logout</button>;
    case false:
      return <button>Login</button>;
    default:
      return null;
  }
};

export default App;
```

Learn with Rabbil

CONDITIONAL RENDERING

Using Ternary Operators

App.jsx

```jsx
const App = () => {
    let marks=10
    return (
        <div>
            {
            marks>80?
            <h1>Brilliant Resutl</h1>
            :
            <h1>Avarage Result</h1>
            }
        </div>
    );
};
export default App;
```

Learn with Rabbil

CONDITIONAL RENDERING

Using Logical &&

App.jsx

```jsx
const App = () => {

let isLoggedIn =true

return (
    <div>
        <h1>Login Status</h1>
        {isLoggedIn && <button>Logout</button>}
    </div>
    );
};

export default App;
```

Learn with Rabbil

CONDITIONAL RENDERING

Using Immediately Invoked Function

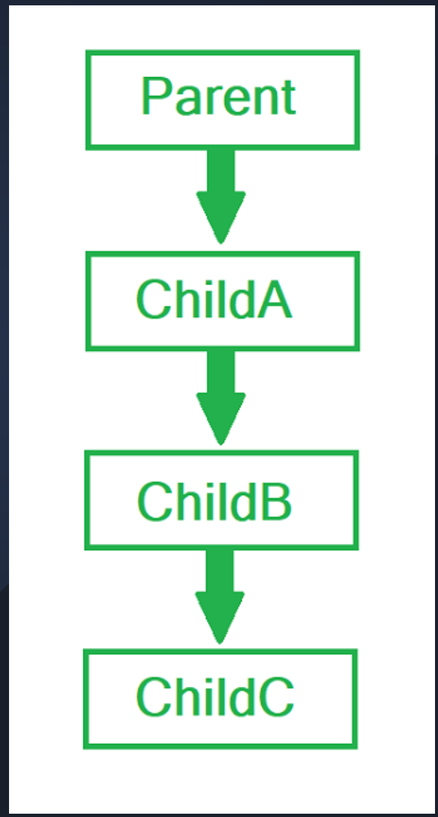
App.jsx

```jsx
const App = () => {
    let marks=10
    return (
        <div>
            {(()=>{
              if(marks>80){
                return <h1>Brilliant Result</h1>
              }
              else{
                return <h1>Avarage Result</h1>
              }
            })()}
        </div>
    );
};
export default App;
```

Learn with Rabbil

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



Learn with Rabbil

App.jsx

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

HeroSection.jsx

```jsx
const HeroSection = (props) => {
    return (
        <div>
            <h1>Q: What time is it now</h1>
            <h6>A: It is {props.time} O Clock</h6>
        </div>
    );
};
```

PASSING PROPS

Passing simple data

Learn with Rabbil

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

Learn with Rabbil

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

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

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

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

```jsx
<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  };
```