
JS
Factory Pattern
in JavaScript
like and share
codewithsloba.com
Swipe

Suppose you have to create a system that generates different types of UI components, such as buttons, inputs, and checkboxes.

You might directly instantiate these components throughout your codebase, leading to repetitive and error-prone code.

If you later need to modify or extend the types of components, it becomes challenging to manage and update every instance individually.

This is where you can use factory pattern

JS

Using the **Factory Pattern**, you **centralize** the creation of UI components through a factory function.

```
const createUIComponent = (type, props) => {
  }
};
```

This abstraction allows you to **dynamically create components** based on a **type**, making it easier to modify, extend, and maintain the code.

The factory function **encapsulates** the creation logic, promoting a more **modular and scalable** approach to building complex UIs.

Swipe for example

JS

Without Factory Pattern

```
const Button = ({ label }) => <button>{label}</button>;
const Input = ({ type, placeholder }) => <input type={type}
placeholder={placeholder} />;
const Checkbox = ({ label }) => (
  <label>
    <input type="checkbox" />
    {label}
  </label>
);

const AppWithoutFactory = () => {
  return (
    <div>
      <Button label="Click me" />
      <Input type="text" placeholder="Enter text" />
      <Checkbox label="Check me" />
    </div>
  );
};

export default AppWithoutFactory;
```

This approach may lead to repetitive code and maintenance challenges.

This code directly uses individual components, leading to potential code duplication.

JS

Using Factory Pattern

```
const Button = ({ label }) => <button>{label}</button>;
const Input = ({ type, placeholder }) => <input type={type}
placeholder={placeholder} />;
const Checkbox = ({ label }) => (
  <label>
    <input type="checkbox" />
    {label}
  </label>
);

const createUIComponent = (type, props) => {
  switch (type) {
    case 'button':
      return <Button {...props} />;
    case 'input':
      return <Input {...props} />;
    case 'checkbox':
      return <Checkbox {...props} />;
    default:
      throw new Error(`Unsupported component type: ${type}`);
  }
};
```

Factory function

centralizes the creation of UI components based on type

JS

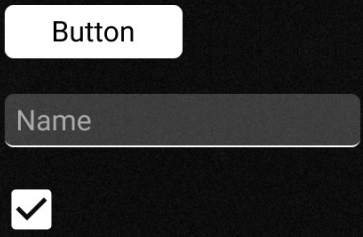
Example Usage

```
const AppWithFactory = () => {
  const button = createUIComponent('button', {
label: 'Click me' });
  const textInput = createUIComponent('input', {
type: 'text', placeholder: 'Enter text' });
  const checkbox = createUIComponent('checkbox', {
label: 'Check me' });

  return (
    <div>
      {button}
      {textInput}
      {checkbox}
    </div>
  );
};

export default AppWithFactory;
```

The client code now uses the factory function, promoting code reusability.



codewithsloba.com

Get a weekly digest of my tips and tutorials by subscribing now.