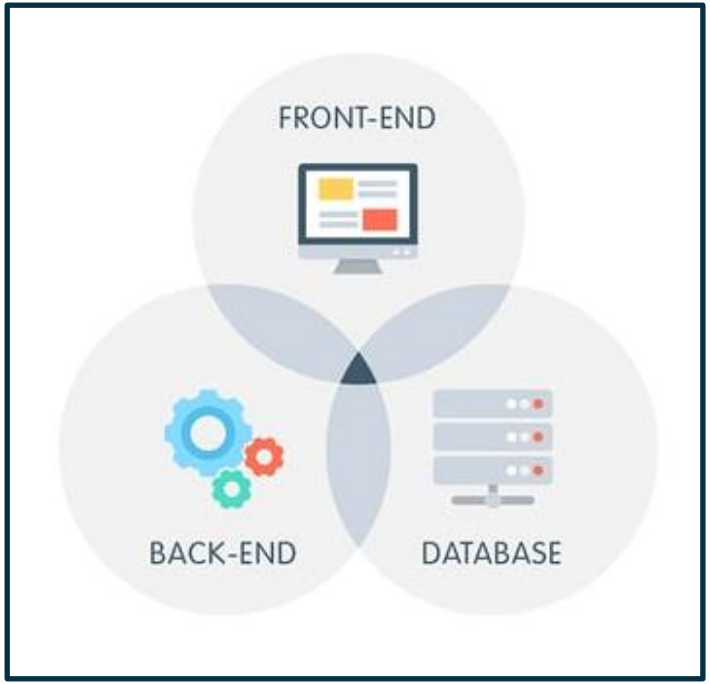
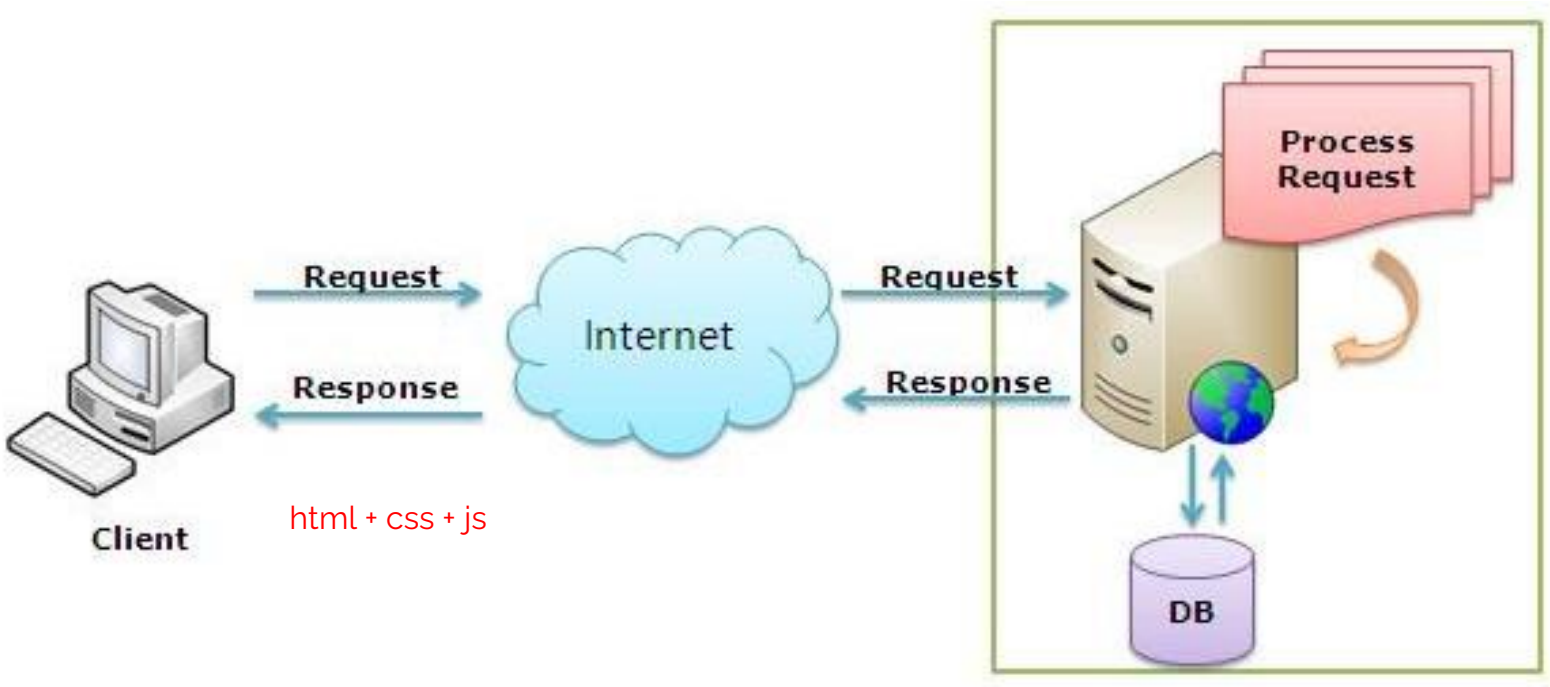
React

Roshan David Jathanna
roshan.jathanna@manipal.edu

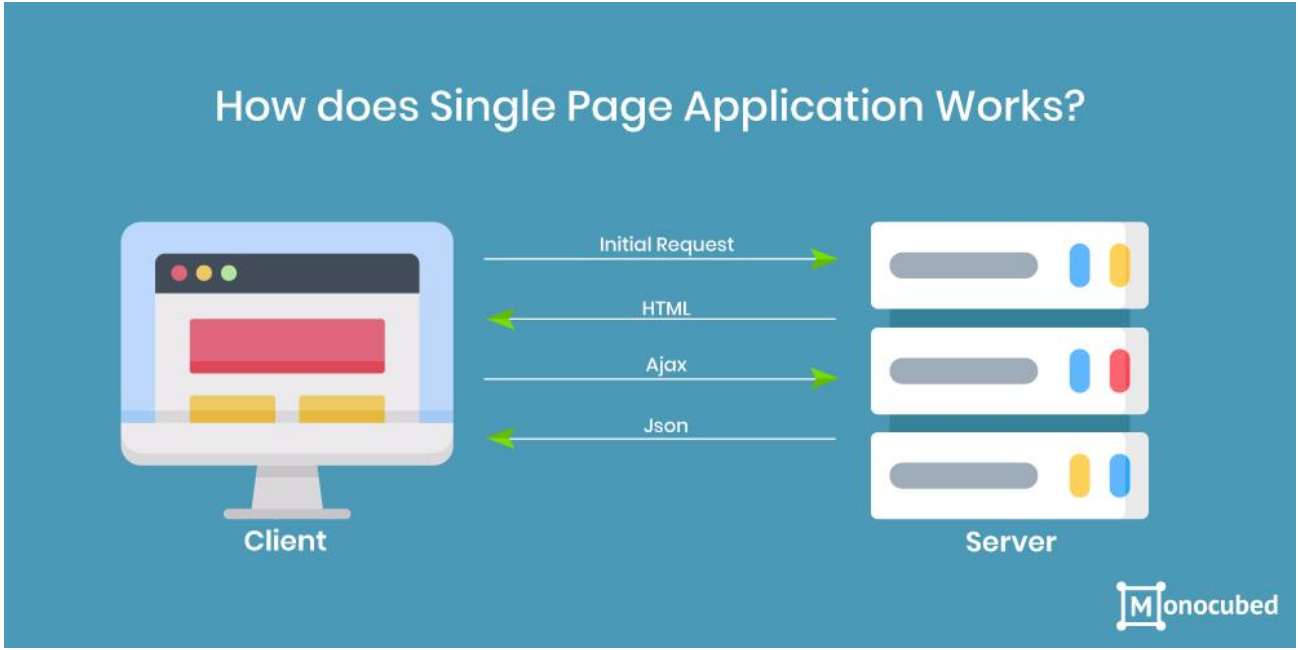


Traditional Web Application

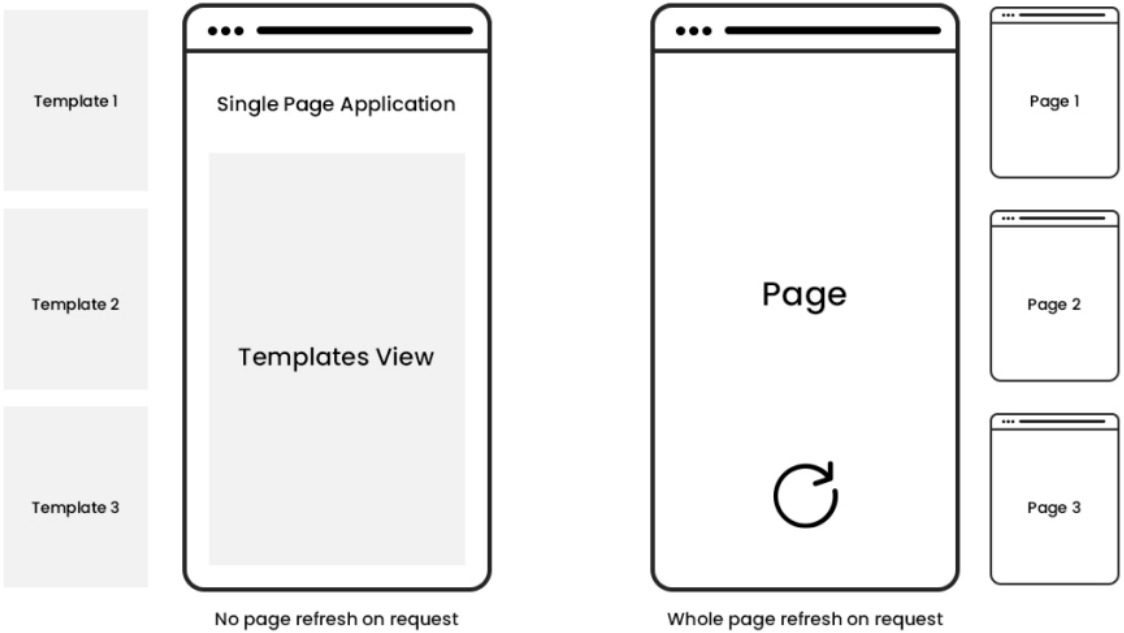


html + css + js

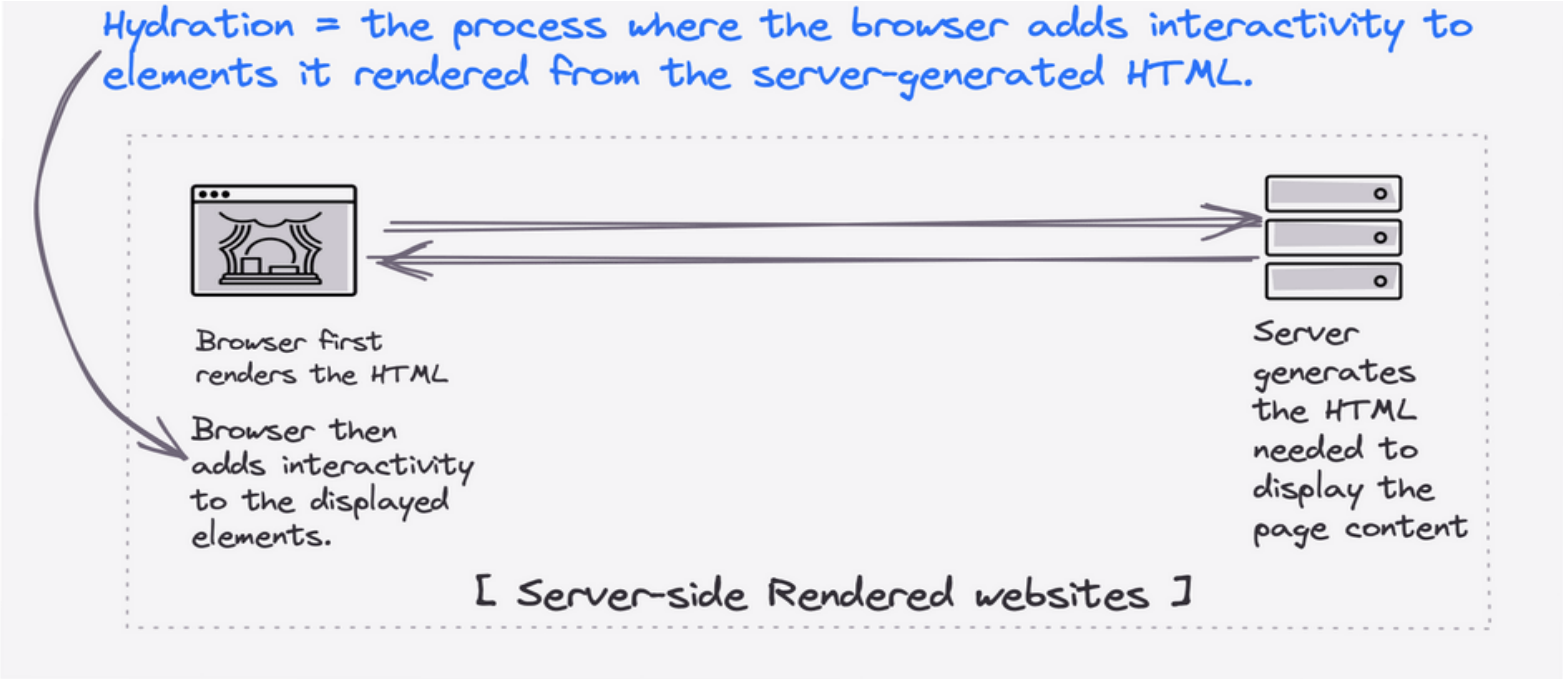
Single Page Application



Comparison

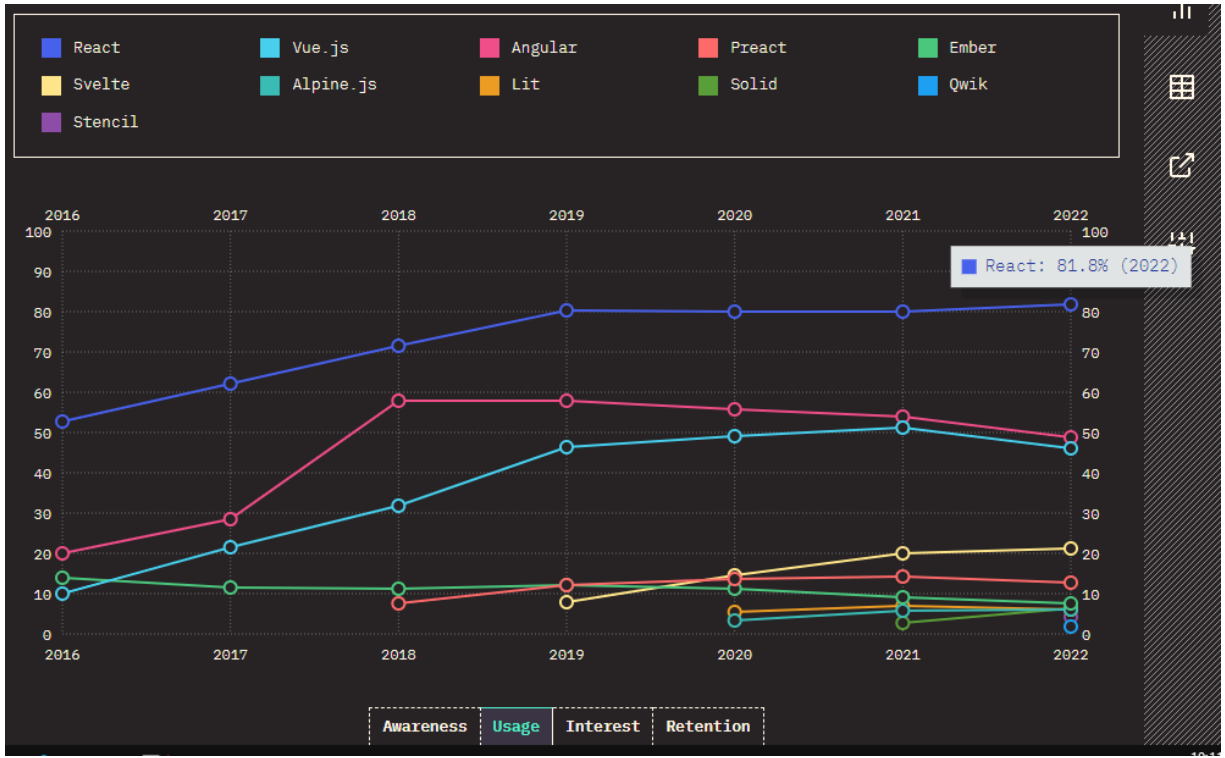

Template 1
Template 2
Template 3
Single Page Application
Templates View
No page refresh on request
Page
Whole page refresh on request
Page 1
Page 2
Page 3

Hydration



https://www.reddit.com/r/webdev/comments/xqd4i8/what_is_hydration/

Front-end frameworks

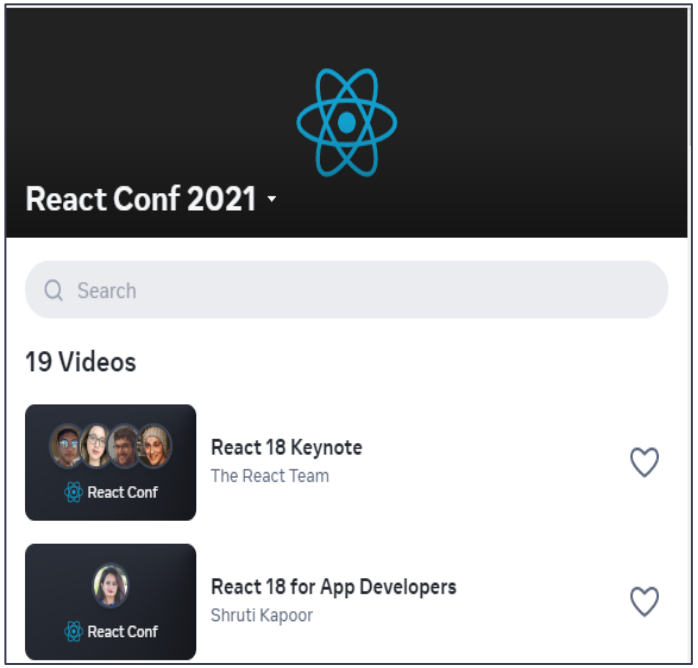


https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/

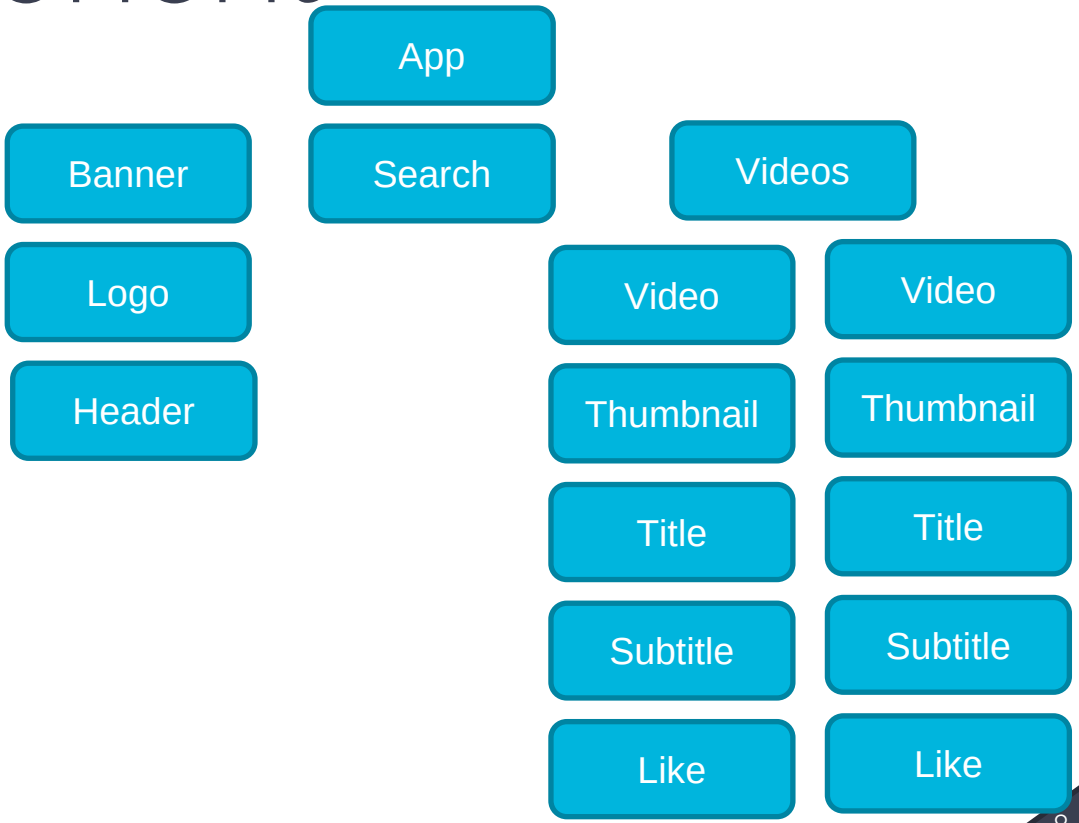
React

- JavaScript library for building fast and interactive user interfaces
- Developed at Facebook in 2011
- Most popular javascript library for building user interfaces

React Component



Website

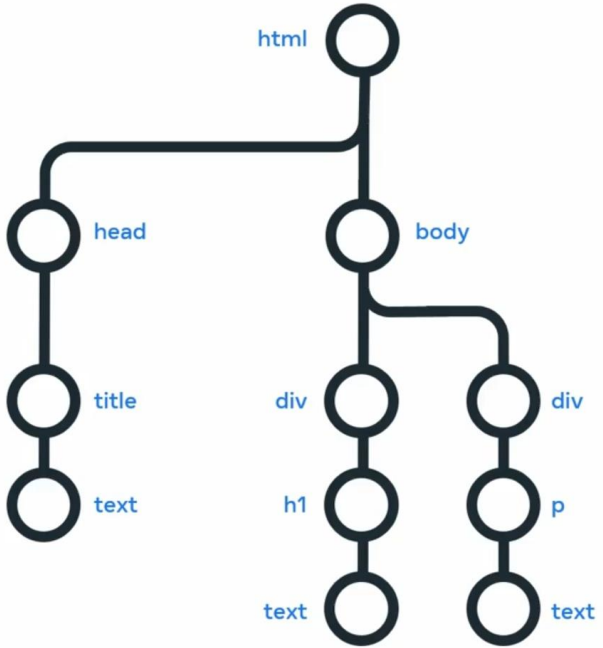


React Component

- React app is all about components
  - Piece of the UI (user interface) that has its own logic and appearance
  - Isolated development
  - Isolated testing
  - Reusable

Document Object Model (DOM)

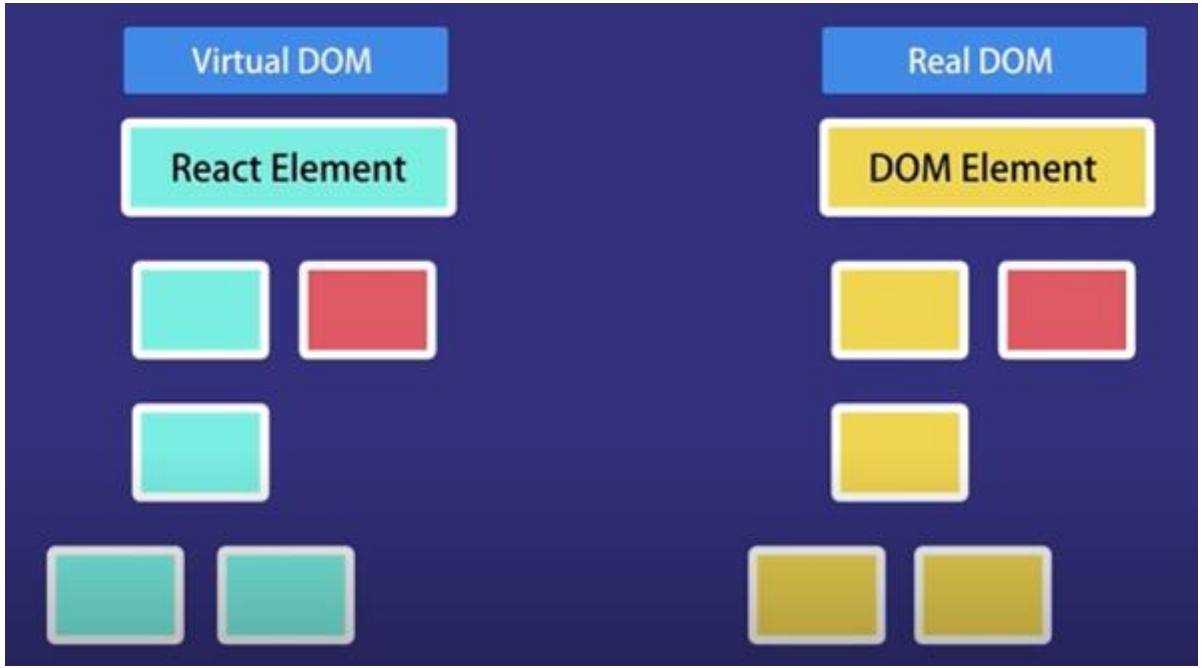
```
<!DOCTYPE html>
<html>
  <head>
     <title>My Website</title>
  </head>
  <body>
     <div>
        <h1>This is the heading</h1>
     </div>
     <div>
        <p>This is the paragraph</p>
     </div>
  </body>
</html>
```



Virtual DOM

- Actual DOM
  - Hard to keep track of changes
  - Slow to update
- React creates Virtual representation of DOM
- React will take care of changing actual DOM whenever virtual DOM is modified
- Virtual DOM uses
  - Efficient diffing algorithm
  - Update subtrees
  - Batch updates

Virtual DOM



React Setup

- Nodejs
- npm i –g create-react-app
- VS Code
  - Simple react snippets – Bruke
  - Prettier
  - Emmet
- npx create-react-app first
- npm start

Folder Structure

For the project to build, these files must exist with exact filenames:

- public/index.html is the page template
- src/index.js is the JavaScript entry point

Put any JS and CSS files inside src, otherwise webpack won't see them

Only files inside public can be used from public/index.html

React ES6 Modules

Default Import

```
const message = () => {
    return 'Hello World';
};
export default message;

import message from "./message.js";
```

Can be any name

Named Import

```
const message = () => {
    return 'Hello World';
};
export {message};

import {message} from "./message.js";
```

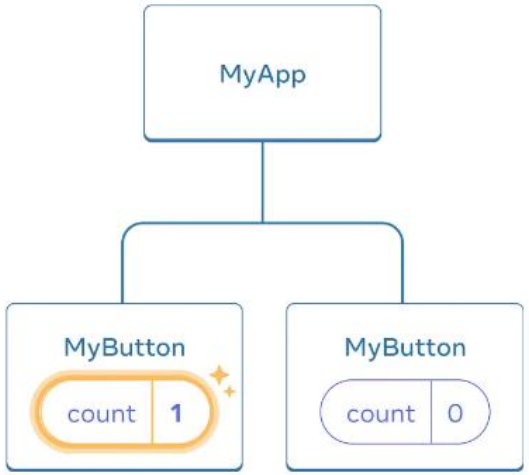
Has to be same name as export

React

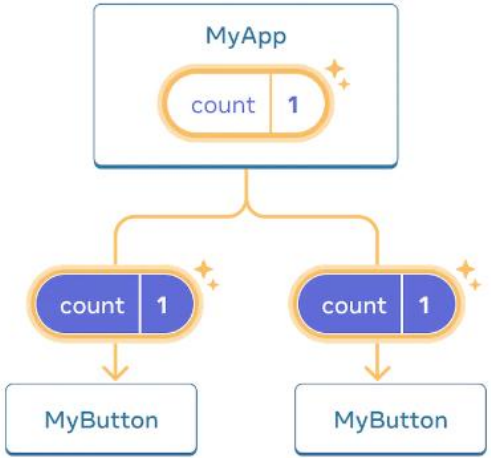
https://react.dev/learn

- Creating and nesting components
- Writing markup with JSX
- Adding styles
- Displaying data
- Conditional rendering
- Rendering lists
- Responding to events
- Updating the screen

Sharing data between components



The first `MyButton` updates its `count` to `1`



On click, `MyApp` updates its `count` state to `1` and passes it down to both children

State

- Does it remain unchanged over time? If so, it isn't state.
- Is it passed in from a parent via props? If so, it isn't state.
- Can you compute it based on existing state or props in your component? If so, it definitely isn't state!

Keep Components Pure

- **It minds its own business.** It does not change any objects or variables that existed before it was called.

```
let guest = 0;

function Cup() {
  // Bad: changing a preexisting variable!
  guest = guest + 1;
  return <h2>Tea cup for guest #{guest}</h2>;
}
```

- **Same inputs, same output.** Given the same inputs, a pure function should always return the same result.

Thinking in React

https://react.dev/learn/thinking-in-react

```
[
  { category: "Fruits", price: "$1", stocked: true, name: "Apple" },
  { category: "Fruits", price: "$1", stocked: true, name: "Dragonfruit" },
  { category: "Fruits", price: "$2", stocked: false, name: "Passionfruit" },
  { category: "Vegetables", price: "$2", stocked: true, name: "Spinach" },
```

Thinking in React

Start with the mockup

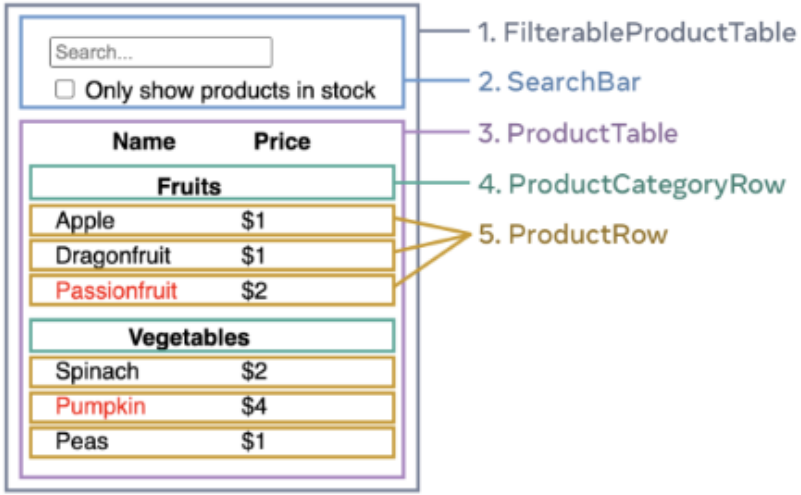
```
[
  { category: "Fruits", price: "$1", stocked: true, name: "Apple" },
  { category: "Fruits", price: "$1", stocked: true, name: "Dragonfruit" },
  { category: "Fruits", price: "$2", stocked: false, name: "Passionfruit" },
  { category: "Vegetables", price: "$2", stocked: true, name: "Spinach" },
  { category: "Vegetables", price: "$4", stocked: false, name: "Pumpkin" },
  { category: "Vegetables", price: "$1", stocked: true, name: "Peas" }
]
```

Thinking in React

Step 1: Break the UI into a component hierarchy

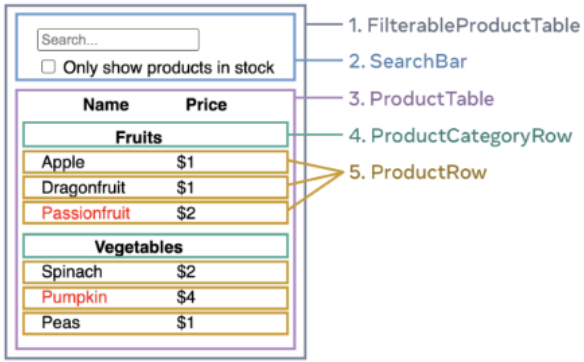
Same techniques for deciding if you should create a new function or object.
One such technique is the single responsibility principle, that is, a component should ideally only do one thing. If it ends up growing, it should be decomposed into smaller subcomponents.

1. FilterableProductTable (grey) contains the entire app.
2. SearchBar (blue) receives the user input.
3. ProductTable (lavender) displays and filters the list according to the user input.
4. ProductCategoryRow (green) displays a heading for each category.
5. ProductRow (yellow) displays a row for each product.

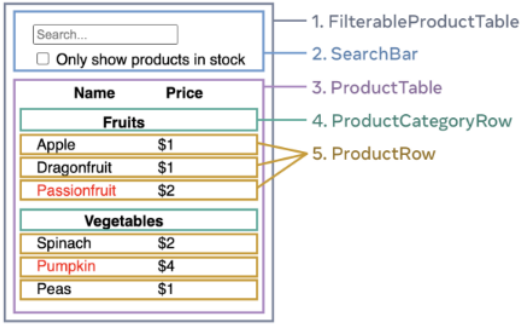


Thinking in React

Step 2: Build a static version in React




```jsx
function ProductCategoryRow({ category }) {
  return (
    <tr>
      <th colSpan="2">
        {category}
      </th>
    </tr>
  );
}

function ProductRow({ product }) {
  const name = product.stocked ? product.name :
    <span style={{ color: 'red' }}>
      {product.name}
    </span>;

  return (
    <tr>
      <td>{name}</td>
      <td>{product.price}</td>
    </tr>
  );
}
```




```jsx
function ProductCategoryRow({ category }) {
  return (
    <tr>
      <th colSpan="2">
        {category}
      </th>
    </tr>
  );
}

function ProductRow({ product }) {
  const name = product.stocked ? product.name :
    <span style={{ color: 'red' }}>
      {product.name}
    </span>;

  return (
    <tr>
      <td>{name}</td>
      <td>{product.price}</td>
    </tr>
  );
}
```

```jsx
function ProductTable({ products }) {
  const rows = [];
  let lastCategory = null;

  products.forEach((product) => {
    if (product.category !== lastCategory) {
      rows.push(<ProductCategoryRow
          category={product.category}
          key={product.category} />
      );
    }
    rows.push(<ProductRow
        product={product}
        key={product.name} />
    );
    lastCategory = product.category;
  });

  return (
    <table>
      <thead>
        <tr>
          <th>Name</th>
          <th>Price</th>
        </tr>
      </thead>
      <tbody>{rows}</tbody>
    </table>
  );
}
```

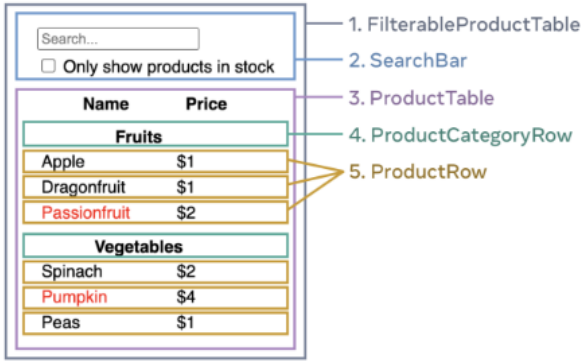
```jsx
function SearchBar() {
  return (
    <form>
      <input type="text" placeholder="Search..." />
      <label>
        <input type="checkbox" />
        {' '}
        Only show products in stock
      </label>
    </form>
  );
}

function FilterableProductTable({ products }) {
  return (
    <div>
      <SearchBar />
      <ProductTable products={products} />
    </div>
  );
}

const PRODUCTS = [
  {category: "Fruits", price: "$1", stocked: true, n
  {category: "Fruits", price: "$1", stocked: true, n
  {category: "Vegetables", price: "$2", stocked: tru
  {category: "Vegetables", price: "$1", stocked: tru
];

export default function App() {
  return <FilterableProductTable products={PRODUCTS}
}
```

Thinking in React

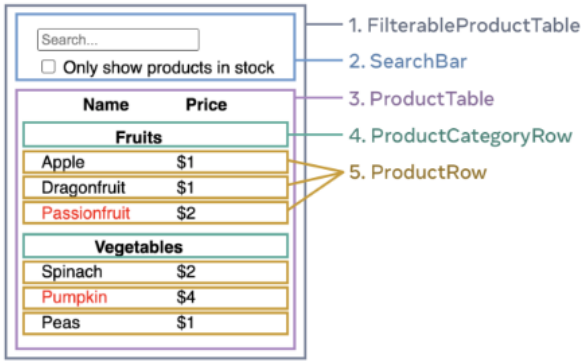
Step 3: Find the minimal but complete representation of UI state

All Data

1. The original list of products is **passed in as props, so it's not state.**
2. The search text seems to be state since it changes over time and can't be computed from anything.
3. The value of the checkbox seems to be state since it changes over time and can't be computed from anything.
4. The filtered list of products **isn't state because it can be computed** by taking the original list of products and filtering it according to the search text and value of the checkbox.



Thinking in React



Step 4: Identify where your state should live
React uses one-way data flow, passing data down the component hierarchy from parent to child component.

For each piece of state in your application:

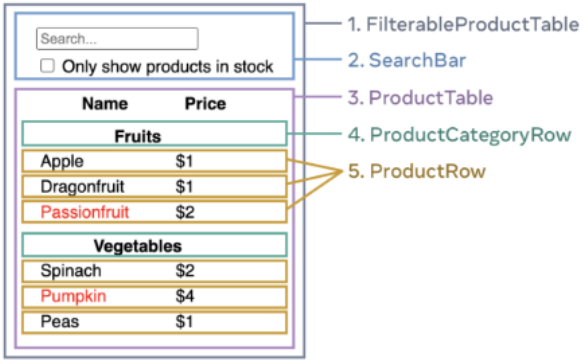
1.Identify every component that renders something based on that state.
ProductTable, SearchBar

2. Find their closest common parent component—a component above them all in the hierarchy.
FilterableProductTable

3. Decide where the state should live:

1. Often, you can put the state directly into their common parent.
2. You can also put the state into some component above their common parent.
3. If you can't find a component where it makes sense to own the state, create a new component solely for holding the state and add it somewhere in the hierarchy above the common parent component.

Thinking in React



Add state to the component with the useState() Hook.

```
function FilterableProductTable({ products }) {
  const [filterText, setFilterText] = useState('');
  const [inStockOnly, setInStockOnly] = useState(false);
```

pass filterText and inStockOnly to ProductTable and SearchBar as props:

```
 <div>
   <SearchBar
     filterText={filterText}
     inStockOnly={inStockOnly} />
   <ProductTable
     products={products}
     filterText={filterText}
     inStockOnly={inStockOnly} />
 </div>
```

Thinking in React

Step 5: Add inverse data flow

```
function FilterableProductTable({ products }) {
  const [filterText, setFilterText] = useState('');
  const [inStockOnly, setInStockOnly] = useState(false);

  return (
    <div>
      <SearchBar
        filterText={filterText}
        inStockOnly={inStockOnly}
        onFilterTextChange={setFilterText}
        onInStockOnlyChange={setInStockOnly} />
```

Inside the SearchBar, you will add the onChange event handlers and set the parent state from them

```
<input
  type="text"
  value={filterText}
  placeholder="Search..."
  onChange={(e) => onFilterTextChange(e.target.value)} />
```

Resources

- https://react.dev/learn