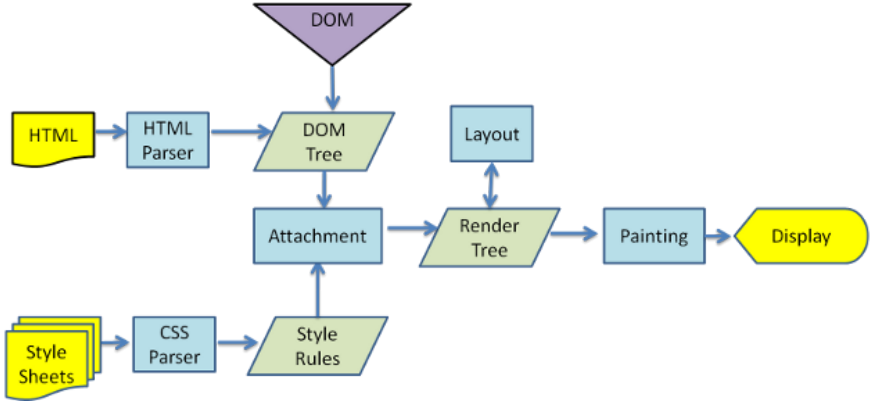
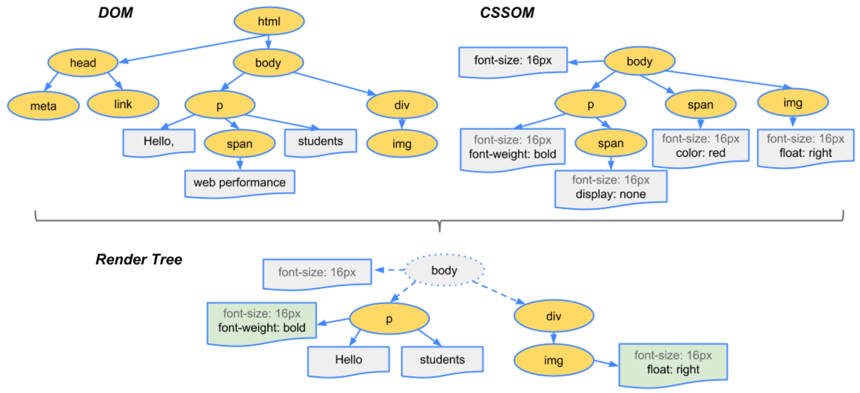
React.js

Outline

- Virtual DOM
- - CSR v.s. SSR
- Demo
- Getting Started
  - Create React App
  - Vite
  - CDN
- React Hooks
  - useState
  - useEffect
  - useMemo, useCallback
  - custom hooks
- state management
  - Context API
  - Mobx
  - Redux
- React Router
- UI Library
  - BootStrap
  - TailwindCSS, daisyUI
  - MUI

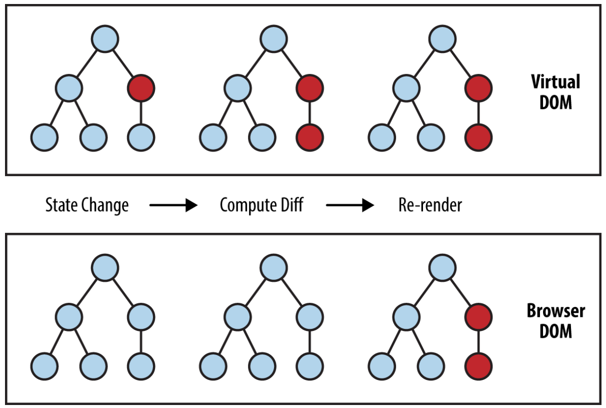
Document Object Model




DOM
html
head
body
meta
link
p
div
Hello,
span
students
img
web performance
CSSOM
font-size: 16px
body
p
span
img
font-size: 16px font-weight: bold
span
font-size: 16px color: red
font-size: 16px float: right
font-size: 16px display: none
Render Tree
font-size: 16px
body
font-size: 16px font-weight: bold
p
div
Hello
students
img
font-size: 16px float: right

Reference: DOM Virtual DOM Document Object Model

Virtual DOM



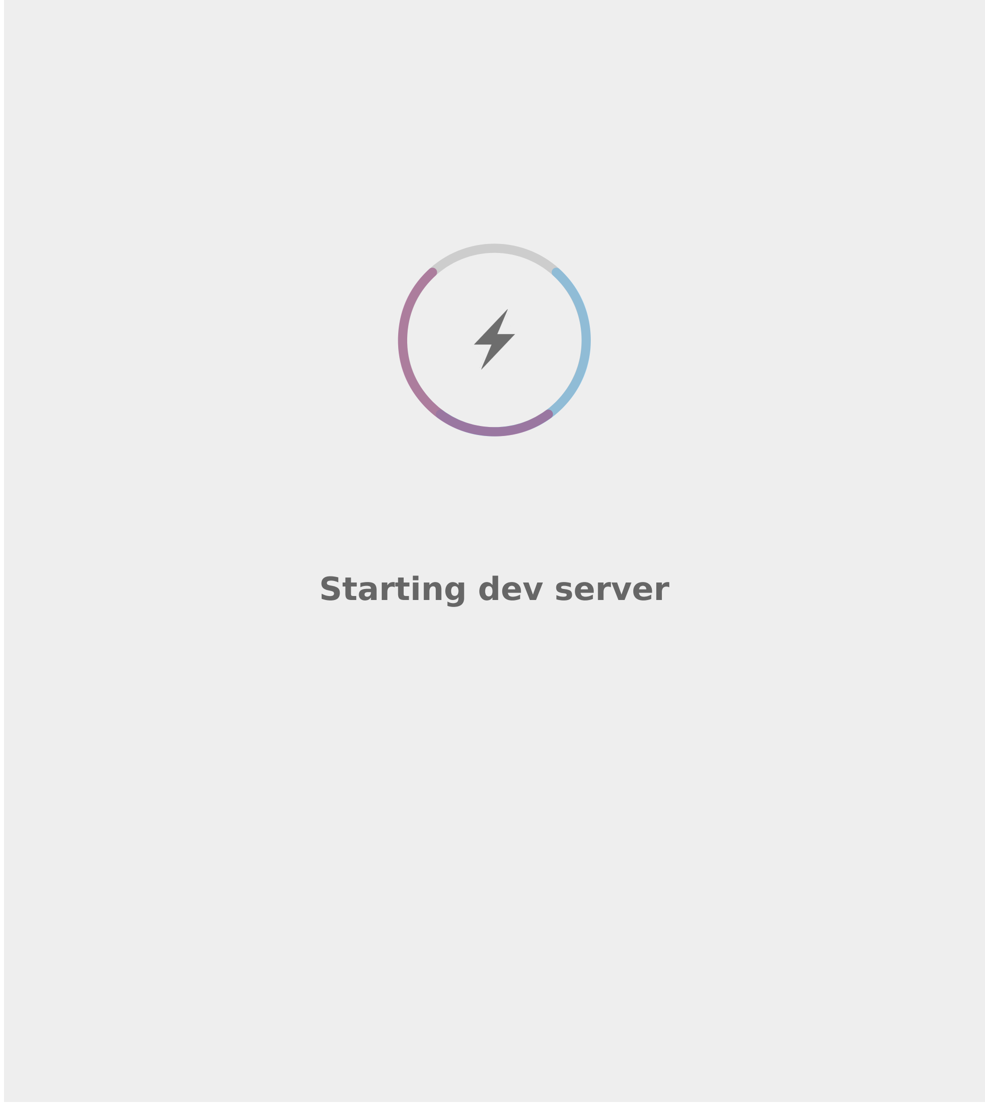
- Virtual DOM JavaScript memory
- virtual DOM virtual DOM
- reflow repaint
- Diffing
  - HTML element tree
  - "key"

- CSR(Client-Side Rendering)
  - Server Javascript HTML Server
- SSR(Server-Side Rendering)
  - Server CPU HTML

CSR	SSR
React.js	Next.js
Vue.js	Nuxt.js

> CSR SSR

Demo



Starting dev server

CRA, Create React App

- JavaScript

```
npx create-react-app [PROJECT_NAME]
```

- TypeScript

```
npx create-react-app [PROJECT_NAME] --template typescript
```

Vite

- JavaScript

```
npm create vite@latest [PROJECT_NAME] -- --template react
```

- TypeScript

```
npm create vite@latest [PROJECT_NAME] -- --template react-
```

> Vite 3.0 vs. Create React App: Comparison and migration guide

Create Project

```
npm create vite@latest react-todolist -- --template react-ts
cd react-todolist
npm install
npm run dev
```

- `npm install` to install dependencies
- file structure

  ```
  - react-todolist
    |- public/
    |- src/
    |- index.html
    |- vite.config.ts
  ```

- available scripts in `package.json`
  - `npm run dev` to run dev server
  - `npm run build` to build for productive usage
  - `npm run dev` to preview project in production environments

  ```
  {
    ...
    "scripts": {
      "dev": "vite",
      "build": "tsc && vite build",
      "preview": "vite preview"
    },
  }
  ```

Setup React Router

- install dependency

```
npm install react-router-dom
```

> if you are interested in react router, you could follow up official tutorial

- create `src/pages/index.tsx` as home page

```
const Home = () => { return <>Home</> }
export default Home
```

- create `src/pages/detail.tsx` as detail page

```
const Detail = () => { return <>Detail</> }
export default Detail
```

- setup router and routes in `src/App.tsx`, you'll see Home text on your browser

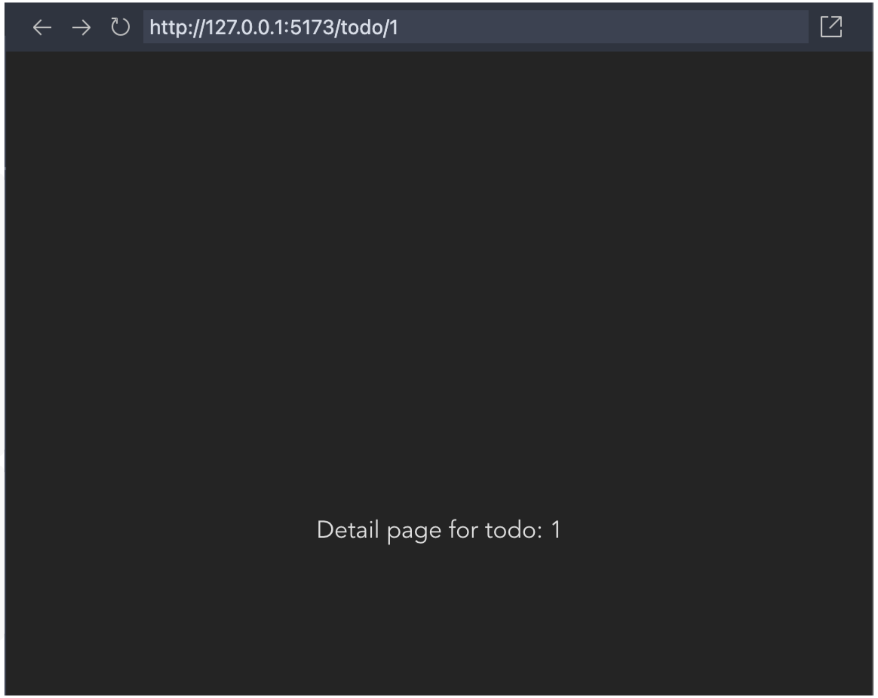
```
<BrowserRouter>
  <Routes>
    <Route index element={<Home />} />
  </Routes>
</BrowserRouter>
```

Setup Route for Detail Page

- retrieve params defined with route path `:id` via useParams hook provided by react router

```
// src/App.tsx
<BrowserRouter>
  <Routes>
    <Route index element={<Home />} />
    <Route path="todo/:id" element={<Detail />} />
  </Routes>
</BrowserRouter>
```

```
const Detail = () => {
  const { id } = useParams()
  return <>Detail page for todo: {id}</>
}
```



Detail page for todo: 1

Setup home page

```
const Home = () => {
  const [todos, setTodos] = useState<ITodo[]>([]);
  return <div>
      <h1>Todo List</h1>
      <div className="todolist-input">
        <input
          type="text"
          placeholder="input text..."
          value={title}
          onChange={(e) => setTitle(e.target.value)}
        />
        <button onClick={addTodo}>+</button>
      </div>
      <textarea
        value={description}
        onChange={(e) => setDescription(e.target.value)}
      ></textarea>
      <ul>
      // put todo items here
      </ul>
    </div>
}
```

```
{todos?.map((todo) => (
  <li key={todo.title} className="todolist-item">
    <input
      type="checkbox"
      checked={todo.done}
      onChange={(e) => toggleDone(todo, e.target.checked)}
    />
    {todo.title}
    <button onClick={() => removeTodo(todo)}>x</button>
  </li>
))}
```

- map function todo items jsx elements
array key react

Create Component

`` `src/components/TodoItem.tsx` ``

```
const TodoItem = ({ checked, title, onChecked, onDelete }: IP
  return (
    <li className="todolist-item">
      <input
        type="checkbox"
        checked={checked}
        onChange={(e) => onChecked(e.target.checked)}
      />
      {title}
      <button onClick={onDelete}>x</button>
    </li>
  );
};

export default TodoItem;
```

checked, title, onChecked, onDelete component props component

```
// src/pages/index.tsx
<ul>
  {todos?.map((todo) => (
    <TodoItem
      key={todo.title}
      title={todo.title}
      checked={todo.done}
      onChecked={(checked) => toggleDone(todo, checked)}
      onDelete={() => removeTodo(todo)}
    />
  ))}
</ul>
```

Create todo item

```
const [title, setTitle] = useState('');
const [description, setDescription] = useState('');

const addTodo = useCallback(() => {
  const data = [...todos, { title, description, done: false }
  setTodos(data);
  save(data);
  setTitle('');
  setDescription('');
}, [todos, title, description, save]);
```

```
<div className="todolist-input">
  <input
    type="text"
    placeholder="input text..."
    value={title}
    onChange={(e) => setTitle(e.target.value)}
  />
  <button onClick={addTodo}>+</button>
</div>
<textarea
  value={description}
  onChange={(e) => setDescription(e.target.value)}
></textarea>
```

- react ---------
- html js variables elements value, onChange, onCLick
- value onChange callback value(setValue) virtual DOM, real DOM
- onClick, onKeyDown, ... elements

```
const removeTodo = useCallback(
  (item: ITodo) => {
    const data = [...todos.filter((todo) => todo.title !== it
    setTodos(data);
    save(data);
  },
  [todos, save]
);

const toggleDone = useCallback(
  (todo: ITodo, checked: boolean) => {
    const data = [
      ...todos.map((item) =>
        item.title === todo.title ? { ...item, done: checked
      ),
    ];
    setTodos(data);
    save(data);
  },
  [todos, save]
);
```

- removeTodo, toggleDone jsx elements

```
<li key={todo.title} className="todolist-item">
  <input
    type="checkbox"
    checked={todo.done}
    onChange={(e) => toggleDone(todo, e.target.checked)}
  />
  {todo.title}
  <button onClick={() => removeTodo(todo)}>x</button>
</li>
```

use local storage API to persist todo items

- retrieve todo items from local storage and set to todos state

```
useEffect(() => {
  const data = localStorage.getItem('todos');
  if (data) {
    setTodos(JSON.parse(data));
  }
}, []);
```

- save changes to local storage

```
const save = useCallback(
  (todos: ITodo[]) => {
    localStorage.setItem('todos', JSON.stringify(todos));
  },
  [todos]
);
```

via packages, like react-use, usehooks-ts

- remove save function and modify addTodo, removeTodo, toggleDone functions

```
const [todos, setTodos] = useLocalStorage<ITodo[]>('todos', [
const addTodo = useCallback(() => {
  const data = [...todos ?? [], { title, description, done: f
  setTodos(data);
  setTitle('');
  setDescription('');
}, [todos, title, description]);
const removeTodo = useCallback(
  (item: ITodo) => {
    const data = [...(todos ?? []).filter((todo) => todo.titl
    setTodos(data);
  },
  [todos]
);
const toggleDone = useCallback(
  (todo: ITodo, checked: boolean) => {
    const data = [
      ...(todos??[]).map((item) =>
        item.title === todo.title ? { ...item, done: checked
      ),
    ];
    setTodos(data);
```

State Management

useState state component state

-
  - state state components props
- props

```
function App() {
  const [todos, setTodos] = useLocalStorage<ITodo[]>('todos',
  return (
    <BrowserRouter>
      <Routes>
        <Route index element={<Home todos={todos} setTodos={s
        <Route path="todo/:id" element={<Detail todos={todos}
      </Routes>
    </BrowserRouter>
  )
}
```

props
state react context API

```
// store/index.tsx
export const RootContext = createContext<IRootContext>({
  todos: [],
});

export const RootProvider = ({ children }: PropsWithChildren)
  const [todos, setTodos] = useLocalStorage<ITodo[]>('todos',
  return (
    <RootContext.Provider value={{ todos: todos ?? [], setTod
      {children}
    </RootContext.Provider>
  );
};
```

```
// src/App.tsx
// provider todos state into pages
<RootProvider>...</RootProvider>
// src/pages/index.tsx
// retrieve todos state from RootContext
const { todos, setTodos } = useContext(RootContext);
```

React Hooks

- React Function Hook
- function
- useState
  - [state, setter]
  - state re-render

```
const [todos, setTodos] = useState<ITodo[]>([]);
```

- useEffect
  - effect function, dependencies
  - return of effect function is cleanup function
  - dependencies effect function
  - dependencies

    component(ComponentDidMount)

    component (ComponentWillUnmount)

```
useEffect(() => {
  const data = localStorage.getItem('todos');
  if (data) {
    setTodos(JSON.parse(data));
  }
}, []);
```

Memorized Hook

- useMemo
  - dependencies
- useCallback
  - dependencies function

```
const addTodo = useCallback(() => {
  const data = [...todos, { title, description, done: false }];
  setTodos(data);
  save(data);
  setTitle('');
  setDescription('');
}, [todos, title, description, save]);
```

Custom Hook

- A custom Hook is a JavaScript function whose name starts with ”use” and that may call other Hooks.

```
function useLocalStorege<T extends any>(key: string, initialValue: T) {
  const [state, setState] = useState<T>(() => {
    const data = localStorage.getItem(key);
    try {
      return JSON.parse(data);
    } catch (e) {
      return initialValue;
    }
  });

  useEffect(() => {
    state && localStorage.setItem(key, JSON.stringify(state));
  }, [state]);

  return [state, setState] as [T, Dispatch<SetStateAction<T>>];
}
```

Packages of Custom Hooks

- react-use
- usehooks-ts

Component

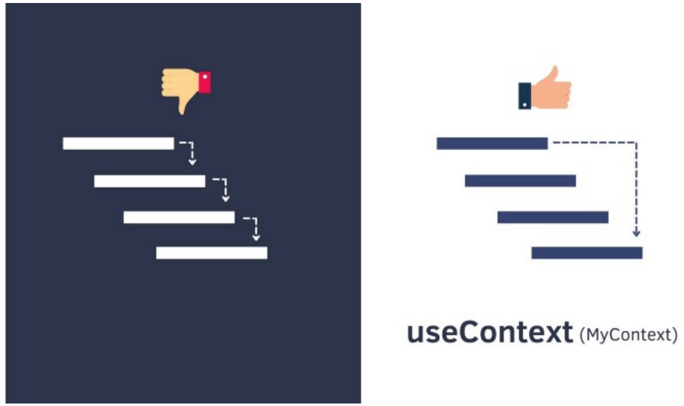
-
- PropsWithChildren

```
const TodoItem = ({ checked, title, onChecked, onDelete }: IProps) => {
  return (
    <li className="todolist-item">
      <input
        type="checkbox"
        checked={checked}
        onChange={(e) => onChecked(e.target.checked)}
      />
      {title}
      <button onClick={onDelete}>x</button>
    </li>
  );
};
```

```
<TodoItem
  key={todo.title}
  title={todo.title}
  checked={todo.done}
  onChecked={(checked) => toggleDone(todo, checked)}
  onDelete={() => removeTodo(todo)}
/>
```

State Management

- Context API



- Mobx
- Redux

```
export const rootContext = createContext<IRootContext>({
  todos: [],
});
```

UI Libries

- BootStrap
- TailwindCSS(, daisyUI)
- MUI

Some Usefull Packages

- Lodash
- React Hook Form