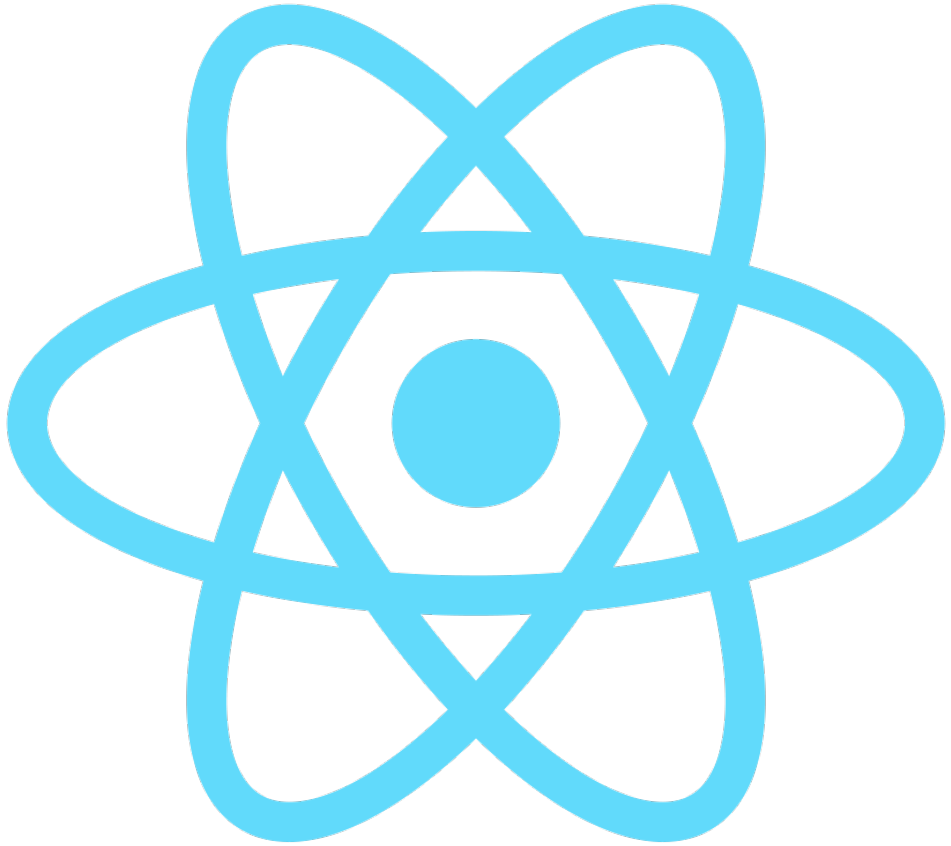
ReactJS
and Web Development



Richard Všianský, SAP
April 24th, 2023

Presenter: Richard Všianský

- Senior Frontend Developer at SAP Signavio
- 4+ years of experience with React
- 7+ years of experience with Web Development
- Master‘s Degree from Mendel University in Brno
- Maintainer of Data Driven Forms library

SAP Labs Czech Republic

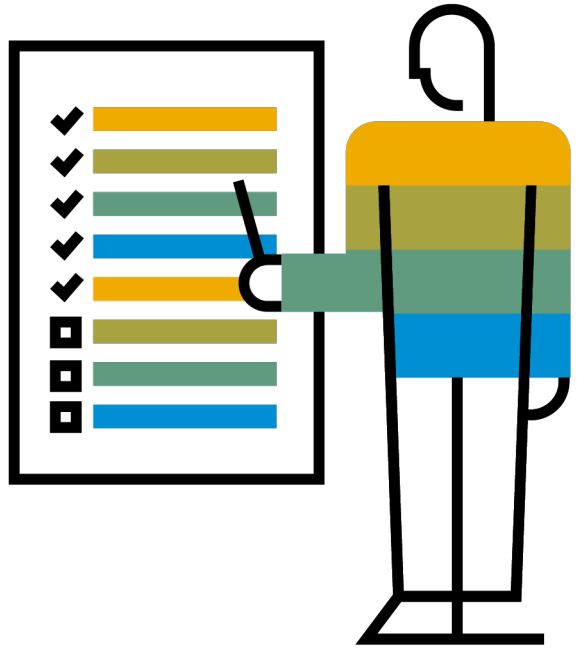
DEVELOPMENT

LOCALIZATION

SOFTWARE MAINTENANCE

Agenda

1) Introduction to React
2) Model View Controller Pattern and React
3) Advanced usage of React



React

- *An open-source JavaScript library for building user interfaces*
- **Declarative**
- **Component-based**

- First public release in 2013, released by Meta (Facebook)
- Most used front-end framework
- Web and mobile applications

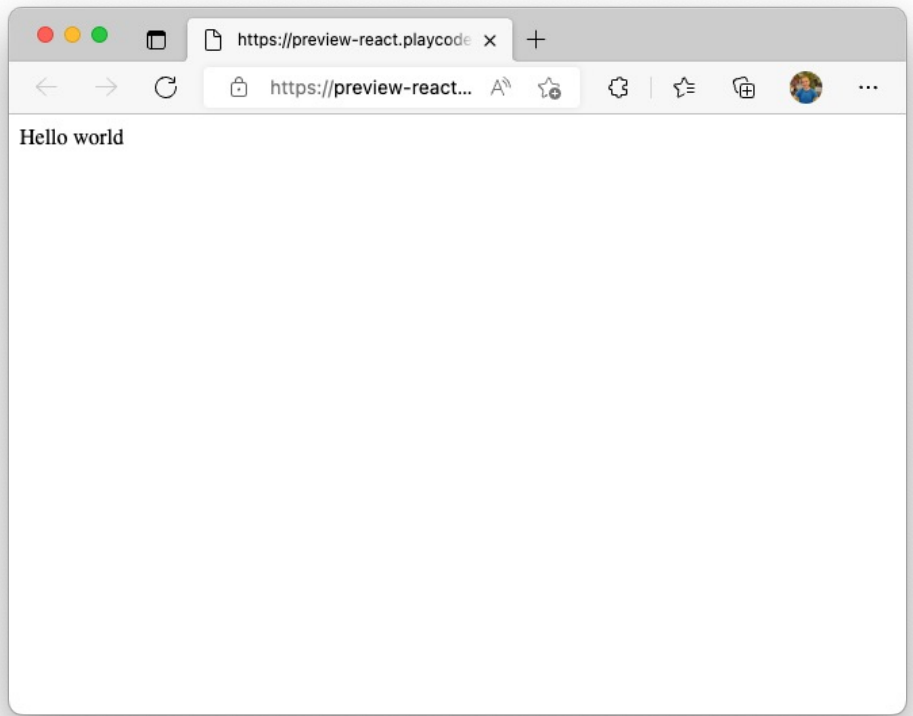
React – Component-based

- **Components = functions / class components**
- Components consists of elements (HTML, components)

- **State**
  - “internal“ attributes of component
- **Props**
  - “external“ attributes of component
  - parameters of function

Component is re-rendered only when state or props are changed / or when parent component is re-rendered

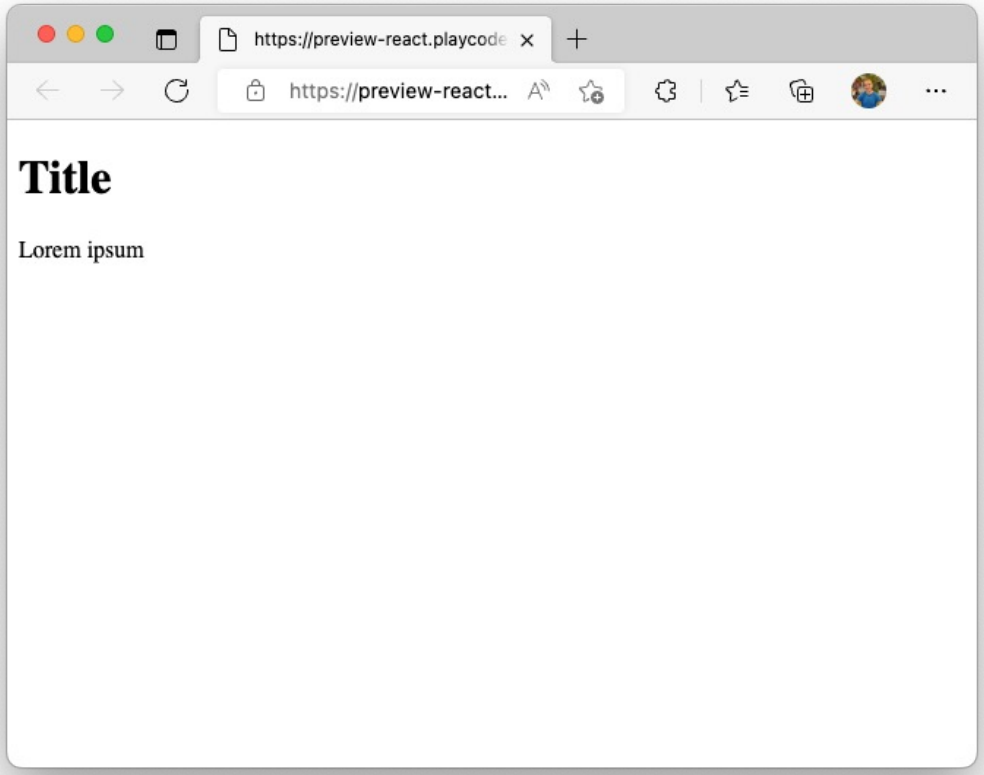
```jsx
// function component
const HelloWorld = () => 'Hello world'
// or
function HelloWorld() {
  return 'Hello world'
}

// class component
class MyComponent extends React.Component {
  render() {
    return 'Hello world'
  }
}
```



JSX

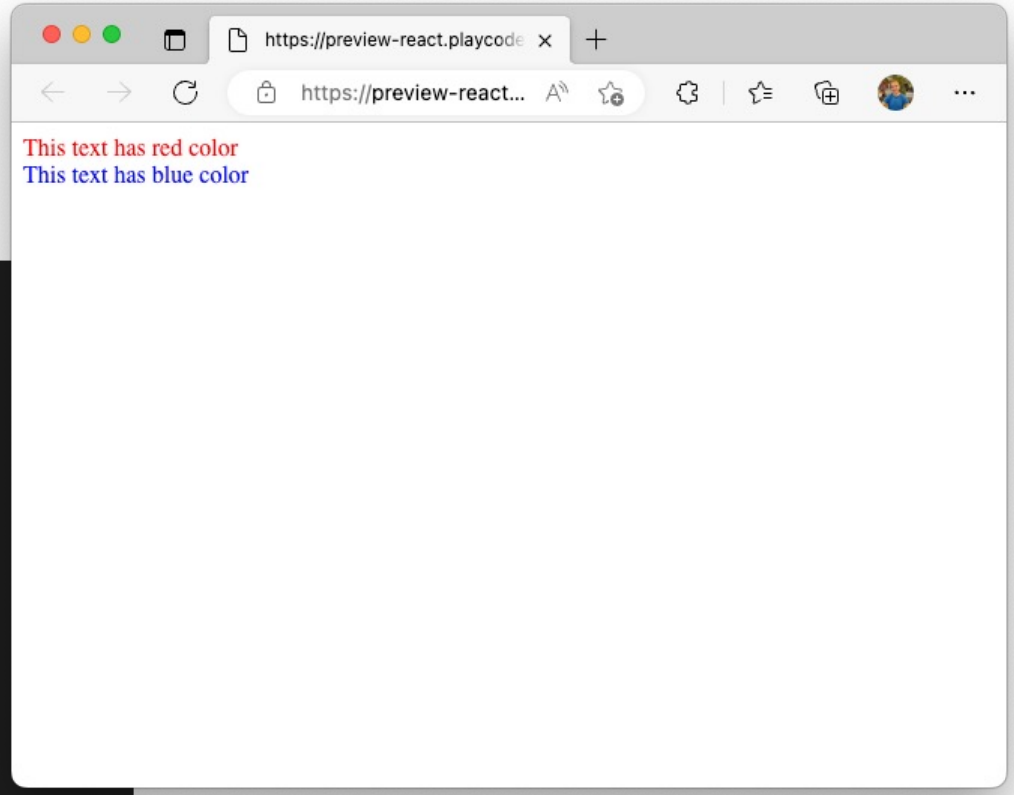
- Template (HTML) and logic (JavaScript) in one file
- Not required, React can be used without it (not recommended)

```jsx
const Component = () => <div>
    <h1>Title</h1>
    <p>Lorem ipsum</p>
</div>
```



Props

- Passing properties to components
- Can use *propTypes* for typing

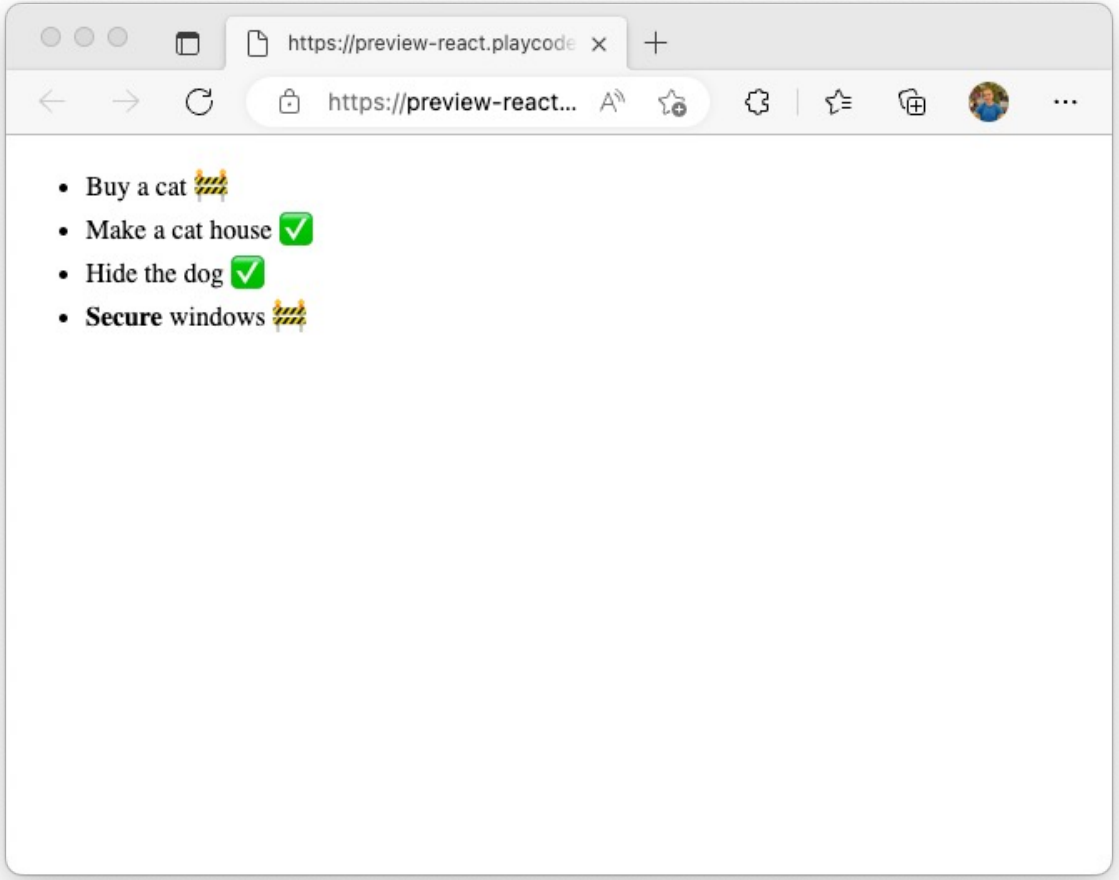



```
const Text = ({ color }) => (
    <div style={{ color }}>
      This text has {color} color
    </div>
)

const App = () => (
  <div>
    <Text color="red" />
    <Text color="blue" />
  </div>
)
```

```
import PropTypes from 'prop-types';

Text.proptypes = {
  color: PropTypes.string;
}
```

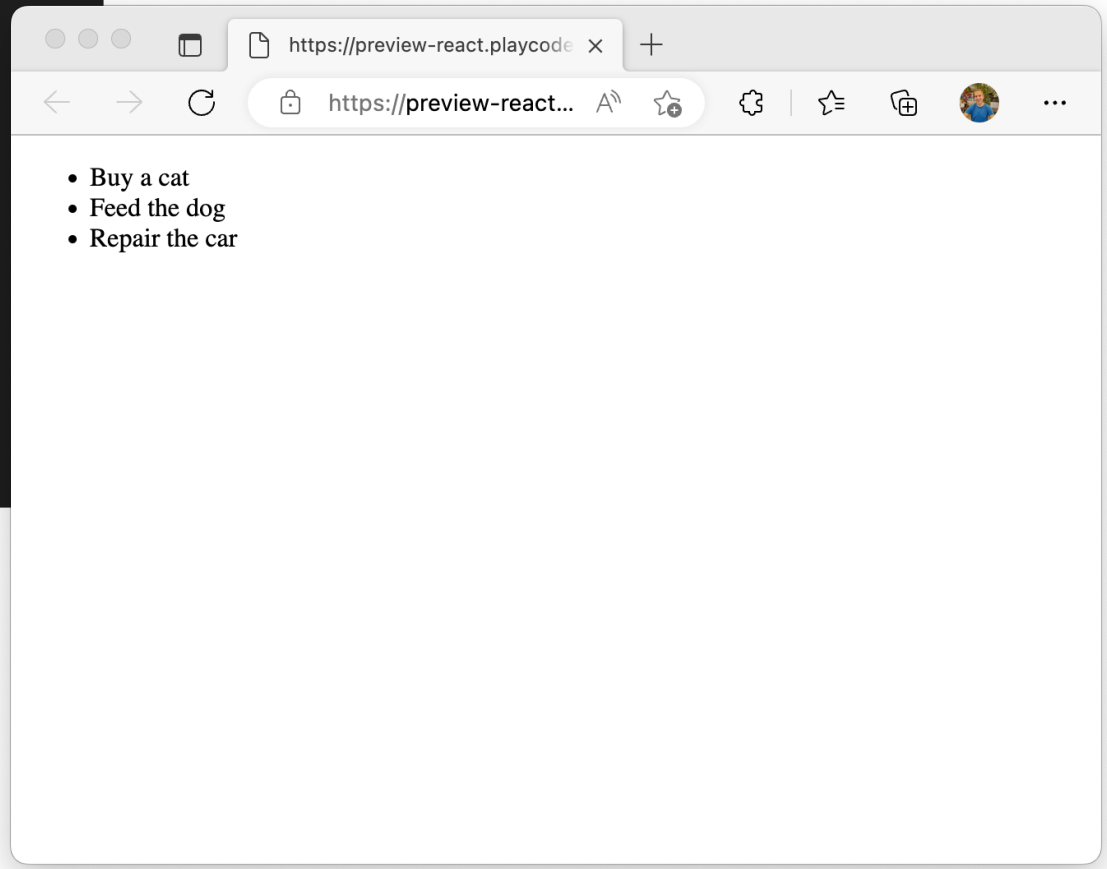
Conditions, Children and Default Props

```jsx
const Item = ({ isFinished = false, children }) => (
    <li>
      {children} {isFinished ? '✅' : '🚧'}
    </li>
)

const App = () => (
  <ul>
    <Item>Buy a cat</Item>
    <Item isFinished>Make a cat house</Item>
    <Item isFinished={true}>Hide the dog</Item>
    <Item><strong>Secure</strong> windows</Item>
  </ul>
)
```



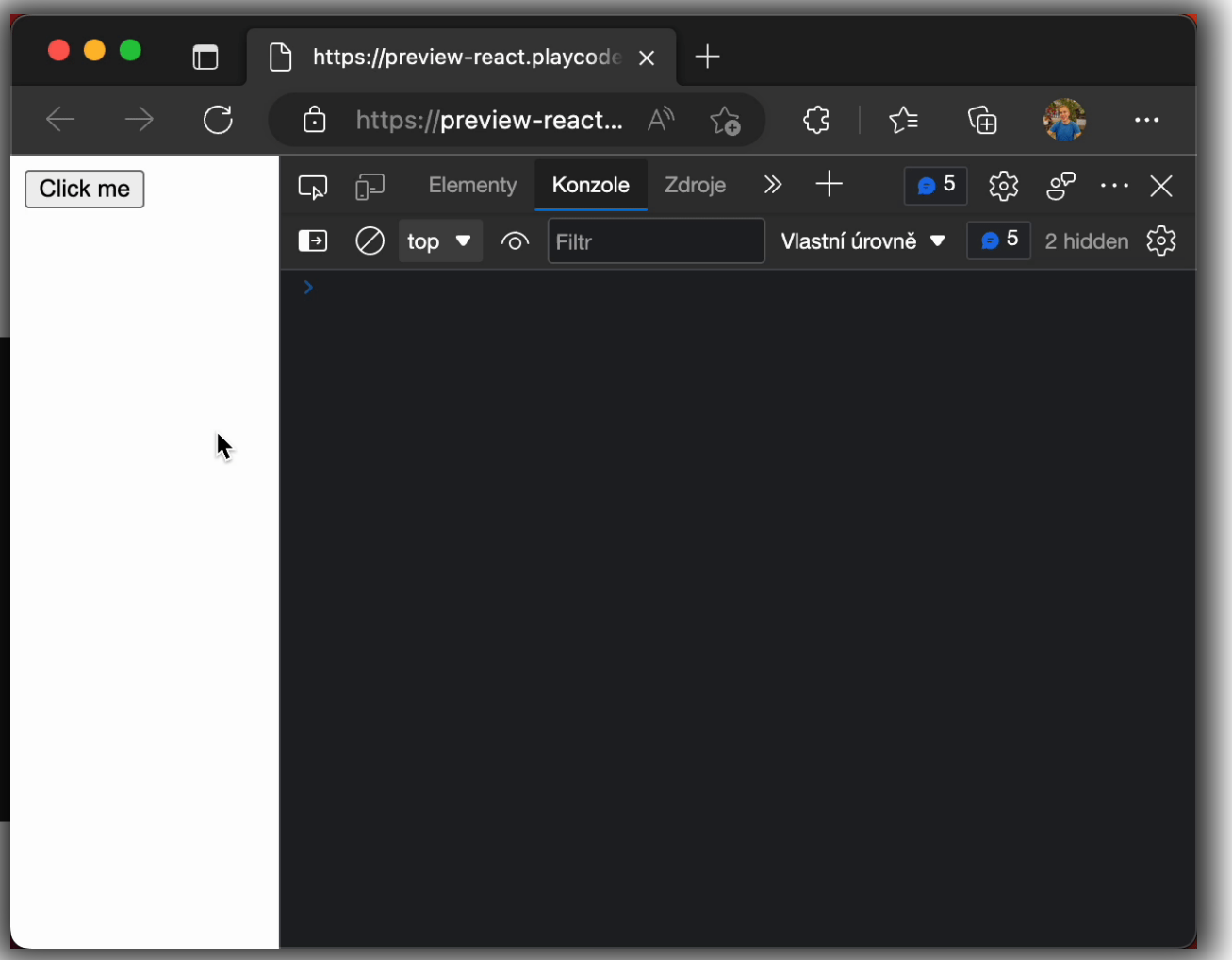
Loops in JSX

```jsx
const App = () => (
  <ul>
    {[
      "Buy a cat",
      "Feed the dog",
      "Repair the car"
    ].map(
      item => <li>
        {item}
      </li>
    )}
  </ul>
)
```



Event Handling

```jsx
const App = () => (
  <button
    type='button'
    onClick={() => console.log('Hello world!')}
  >
    Click me
  </button>
);
```



Hooks and Lifecycles Methods

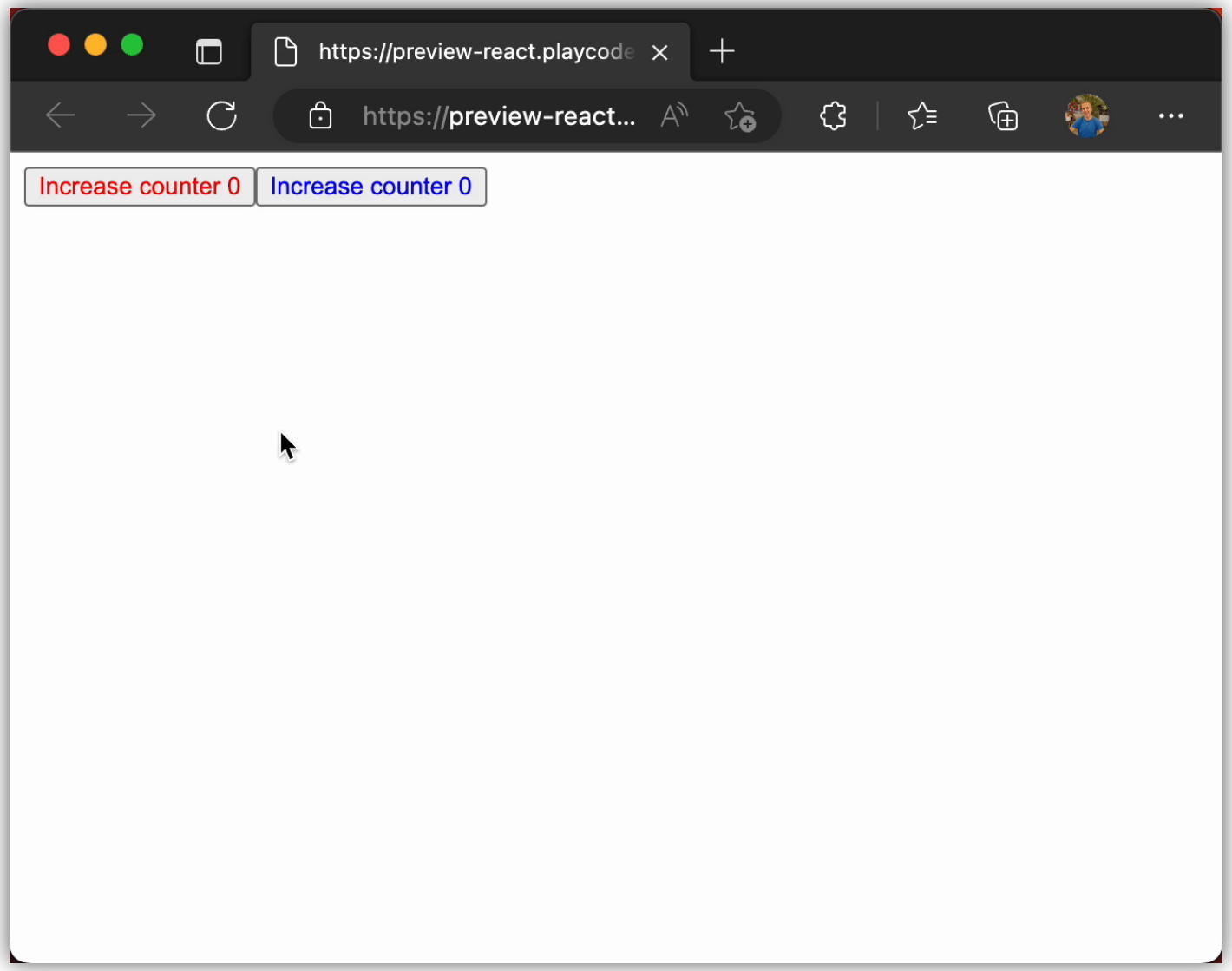
- Used to manage component states, trigger side-effects
- **Lifecycle** methods
  - Used in **Class** components only
  - Can be replaced by Hooks *(only ComponentDidCatch cannot be)*
  - High-order components
    - A component that returns component
    - To share functionality
- **Hooks** (recommended)
  - Used in **Function** components only
  - Introduced in React 16.8.0 (February 2019)
  - Shared functionality across components

Lifecycle Methods *(obsolete)*

- **constructor**()
  - Binds methods, sets props
- **render**()
- **componentDidMount**()
  - Called after component is inserted into the DOM tree
  - Usually used to load data from network
- **componentWillUnmnout**()
  - Called before component is destroyed
  - Usually used to clean subscriptions, cancel network requests
- Check other methods

Stateful Class Component Example

```
1 class Button extends React.Component {
2   constructor() {
3     super();
4     this.state = { counter: 0 };
5     this.increaseCounter =
6       this.increaseCounter.bind(this);
7   }
8
9   increaseCounter() {
10    this.setState(
11      prevState => ({ counter: prevState.counter + 1 })
12    )
13  }
14
15  render() {
16    const { counter } = this.state;
17    const { color } = this.props;
18
19    return <button
20      type='button'
21      style={{ color }}
22      onClick={this.increaseCounter}>
23      Increase counter {counter}
24    </button>;
25  }
26 }
27
28 const App = () => (
29   <React.Fragment>
30     <Button color="red" />
31     <Button color="blue" />
32   </React.Fragment>
33 );
```

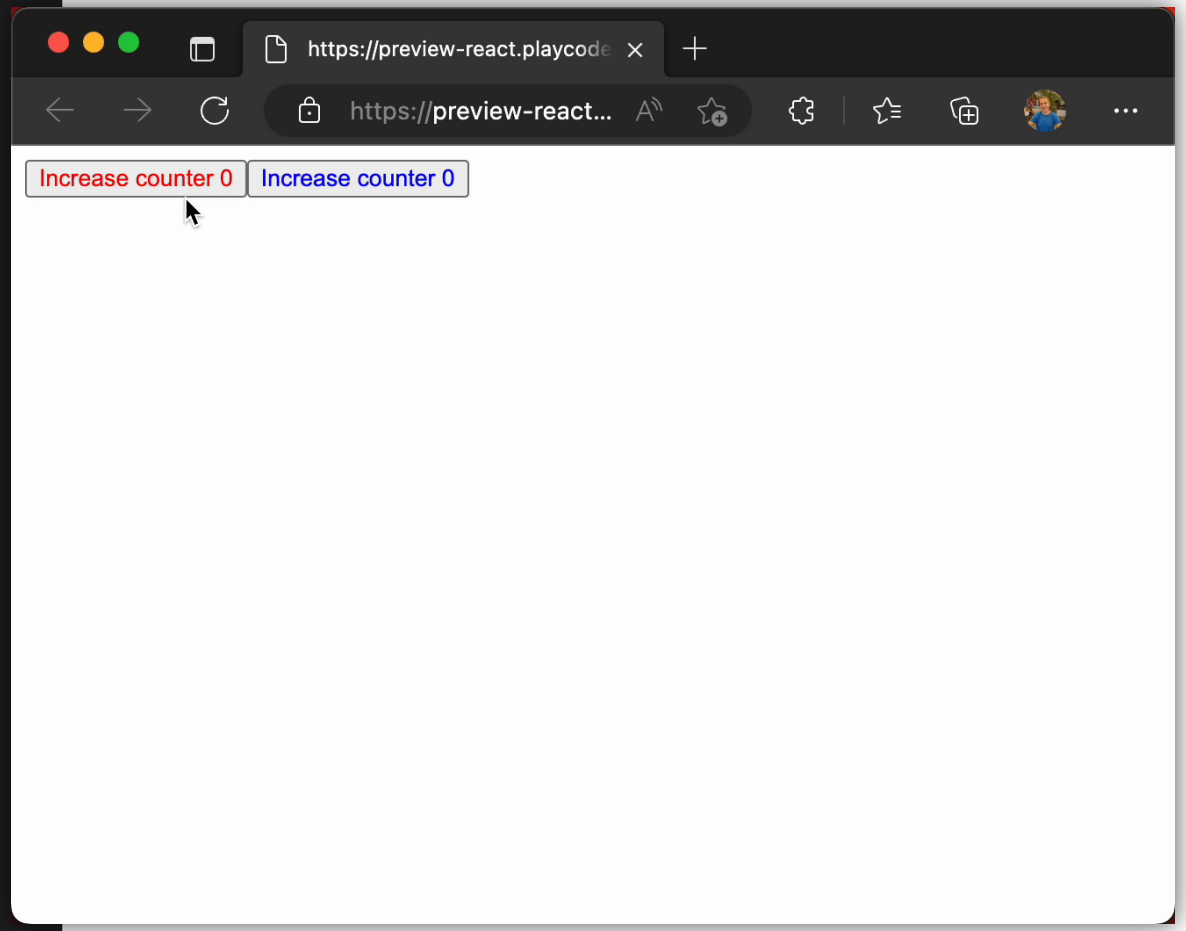


Hooks

- **useState, useReducer**
  - To initialize and store inner variables (state)
- **useRef**
  - To create an object that persists for the fulltime of the component
- **useEffect**
  - Triggers side-effects according to changes of dependencies
  - Replaces componentDidMount, componentDidUnmount, componentDidUpdate
- **useContext**
  - Accessing outside context
- Check other hooks, comparison with LifeCycle methods

Stateful Function Component Example

```jsx
const Button = ({ color }) => {
  const [counter, setCounter] = useState(0);

  const increaseCounter = () =>
        setCounter(prev => prev + 1);

  return (
    <button
      type='button'
      style={{ color }}
      onClick={increaseCounter}
    >
      Increase counter {counter}
    </button>
  );
};

const App = () => (
  <React.Fragment>
    <Button color='red' />
    <Button color='blue' />
  </React.Fragment>
);
```



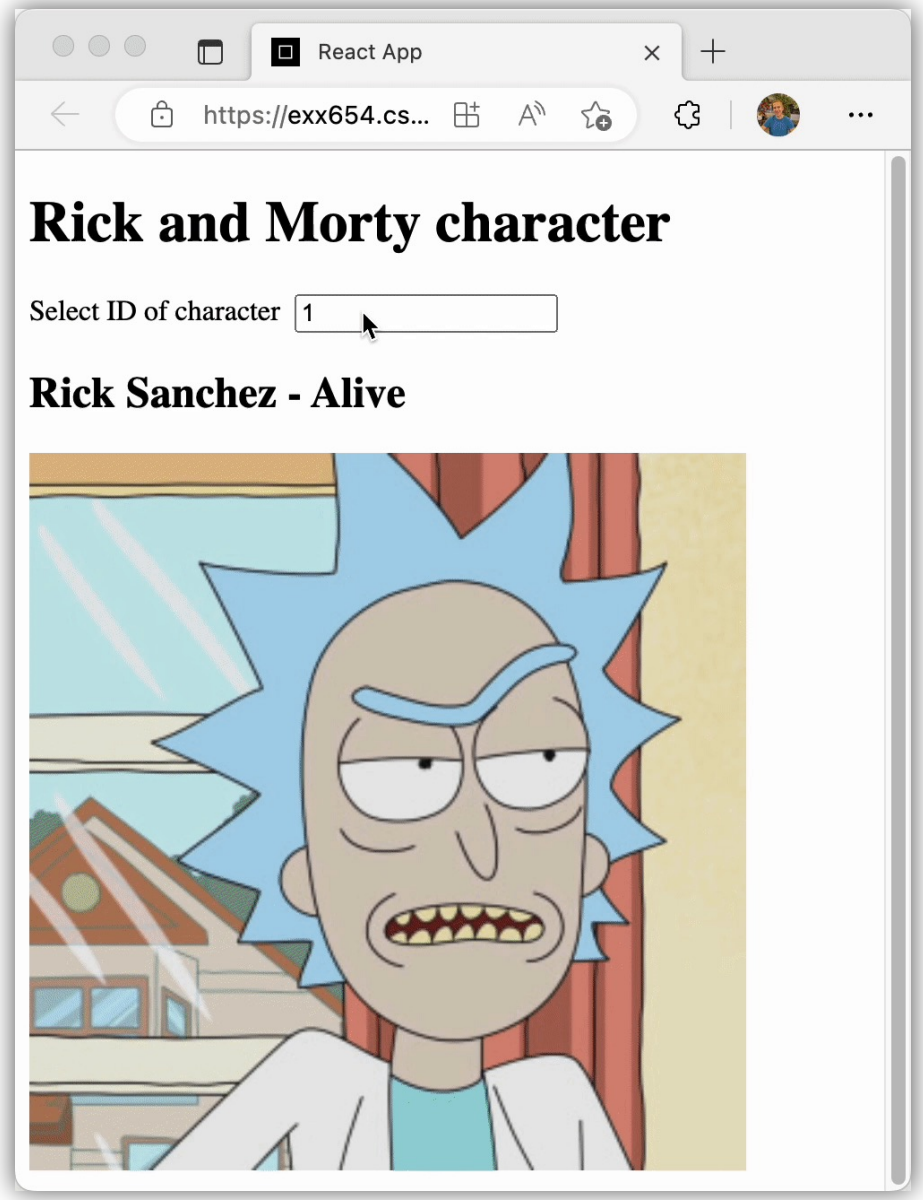
Custom Hooks

- A function calling other hooks
- To share functionality across components
- The name of a custom hook has to start with `*use*` (e.g. *useLoadData*)

```jsx
const useLoadData = (url) => {
  const [data, setData] = useState();
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState();

  useEffect(() => {
    setError(null);
    setData(null);
    setLoading(true);
    fetch(url)
      .then((response) => {
        if (response.ok) {
          return response.json();
        } else {
          return Promise.reject(response);
        }
      })
      .then((data) => {
        setData(data);
        setError(null);
        setLoading(false);
      })
      .catch(async (error) => {
        setError(await error.json());
        setLoading(false);
      });
  }, [url]);

  return { data, loading, error };
};
```

```jsx
const App = () => {
  const [number, setNumber] = useState(1);
  const { data, loading, error } = useLoadData(
    `https://rickandmortyapi.com/api/character/${number}`
  );

  return (
    <div>
      <h1>Rick and Morty character</h1>
      <div style={{ display: "flex", gap: 8 }}>
        <label>Select ID of character</label>
        <input value={number} onChange={(e) => setNumber(e.target.value)} />
      </div>
      {loading && <div>Loading ...</div>}
      {error && <div style={{ color: "red" }}>{error.error}</div>}
      {data && !error && (
        <div>
          <h2>
            {data.name} - {data.status}
          </h2>
          <img src={data.image} alt={data.name} width={400} height={400} />
        </div>
      )}
    </div>
  );
};
```

[Link to code]



ReactDOM

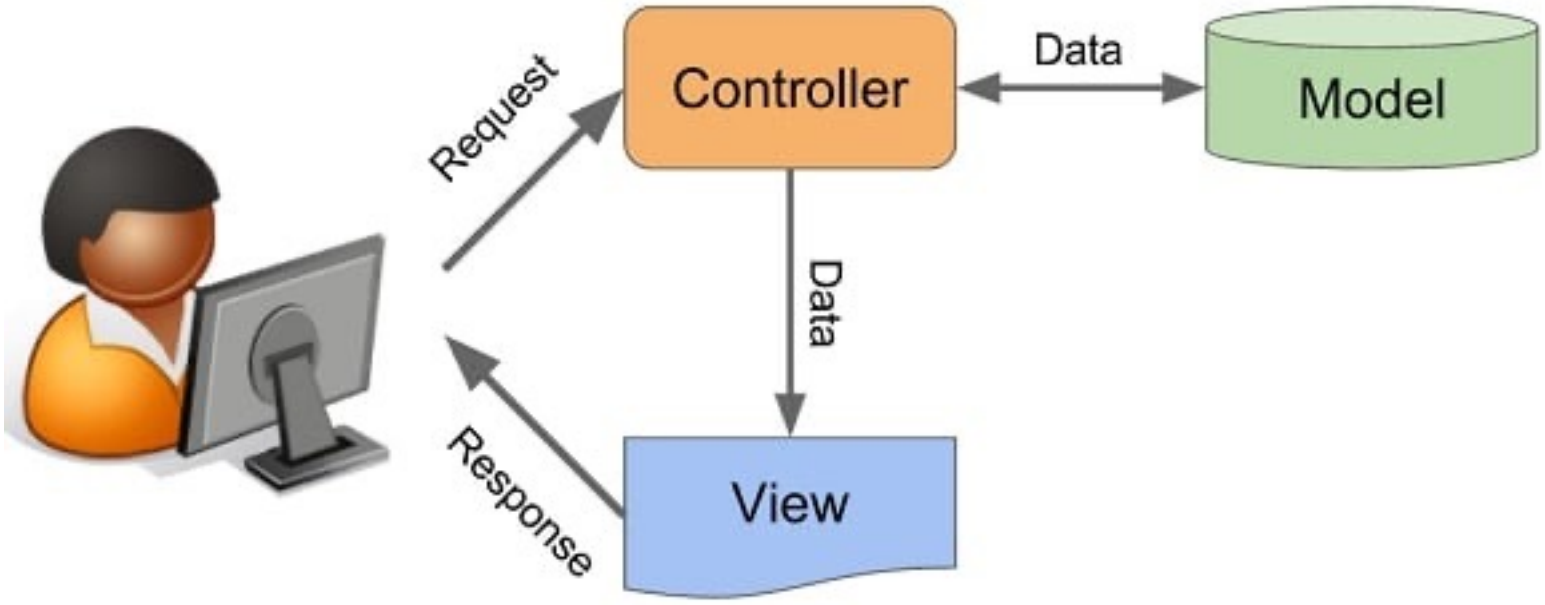
- A virtual DOM – a representation of React components tree
- Changes are first done in ReactDOM and after their comparison, they are rendered in DOM

```
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3
4 import App from './App.jsx'
5
6 ReactDOM.createRoot(
7   document.querySelector('#root')
8 ).render(<App />)
```

Model-View-Controller and React

Model-View-Controller

- A software architectural pattern used for developing GUI applications
- Separation of concerns



https://helloacm.com/model-view-controller-explained-in-c/

Model

- Manages data and logic – validation, saving, updating
- e.g. a table in database, ORM model, …
- Object-relational Mappers
  - A class in code represents table in the DB

Controller

- Receives input from View, updates Model and returns data to View
- Does not know how to handle data
- Sometimes called Presenter

View

- Displays data to users
- UI application (frontend)
- HTML, JavaScript, CSS => **REACT**

Model vs Controller

- Model/Controller is usually one application
- “Model” sometimes refers to a database
- e.g. In Rails Models and Controllers are different classes

MVC in Past

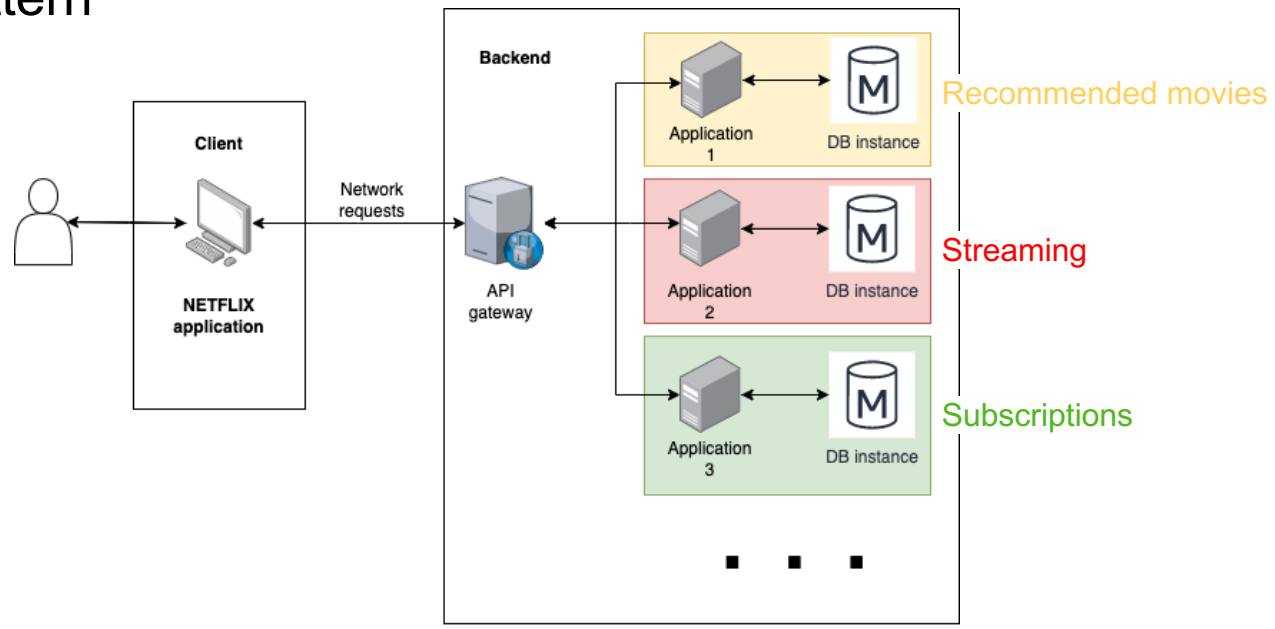
- MVC frameworks
- One application handled data, logic and presenting them to users (Monolith application)
- HTML templating (ERB, Haml)
- Ruby On Rails (2004) – Ruby
- Django – Python
- Laravel, Nette – PHP
- .NET – C#
- Spring - Java

MVC Today

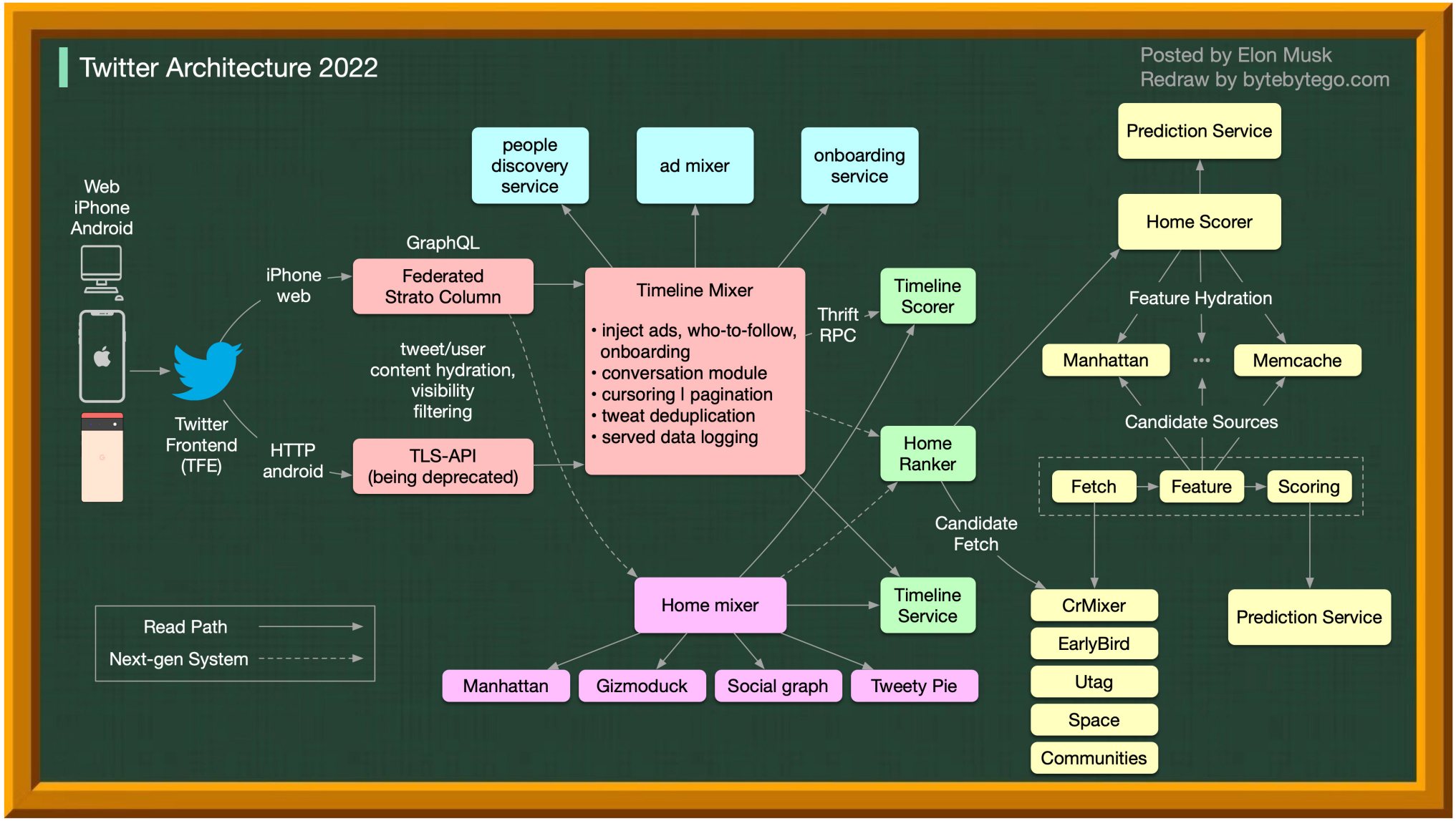
- Backend contains presenters + models
  - Handle users input and updates data
  - Usually one backend application (Java, PHP, Python, …)
- Frontend (UI) displays data to users
  - Usually one (React) application
- Frontend communicates with backend via API
  - Sends data
  - Receives data
  - No HTML templating
  - REST API / GraphQl / *RPC*

Microservices

- Usually one client, multiple backends (up to 100+)
- Each backend has own database, handles specific functionality
- Microservices interact with each other
- Each MS follows MVC pattern



Twitter

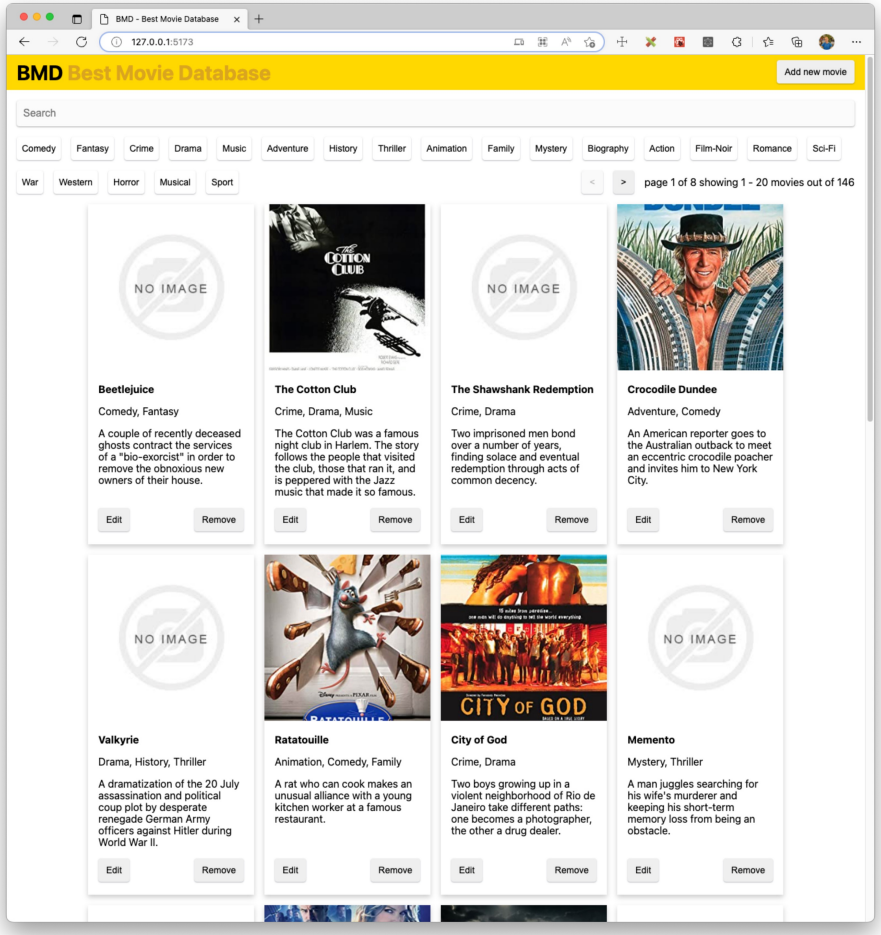

Twitter Architecture 2022
Posted by Elon Musk
Redraw by bytebytego.com
Web
iPhone
Android
Twitter
Frontend
(TFE)
iPhone
web
HTTP
android
GraphQL
Federated
Strato Column
TLS-API
(being deprecated)
tweet/user
content hydration,
visibility
filtering
people
discovery
service
ad mixer
onboarding
service
Timeline Mixer
• inject ads, who-to-follow, onboarding
• conversation module
• cursoring | pagination
• tweat deduplication
• served data logging
Thrift
RPC
Timeline
Scorer
Home
Ranker
Home mixer
Timeline
Service
Manhattan
Gizmoduck
Social graph
Tweety Pie
Prediction Service
Home Scorer
Feature Hydration
Manhattan
...
Memcache
Candidate Sources
Fetch
Feature
Scoring
Candidate
Fetch
CrMixer
EarlyBird
Utag
Space
Communities
Prediction Service
Read Path
Next-gen System

Microfrontends

- An approach to bring Microservices architecture to Frontend side
- One client consists of several UI applications
- Teams can use different frameworks, not blocked by others, code is separated
- Mostly used for code separations…
  - Using different frameworks is not scalable (sharing components, knowledge, programmers)
  - Frontend parts usually interact a lot with each other
  - Serving different applications can increase application size

MVC Example Using React and NodeJS

- Example https://github.com/rvsia/vut-2022-react/tree/main/02-mvc-integration
- Movie database web application
  - API REST connection
    - Custom local server with in-memory database
  - Create and update movies
  - Includes pagination, filtering



Common and Advanced Usage of React

React and CSS

- Inline styling

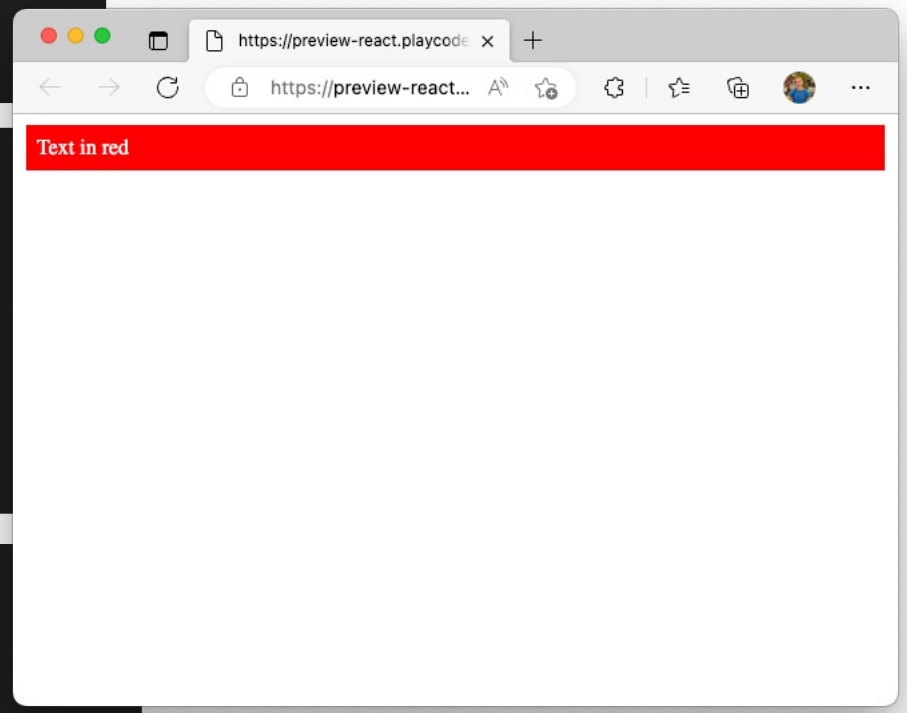
```
const App = () => <div
  style={{
    background: 'red',
    padding: 8,
    margin: 2,
    color: 'white'
  }}
>Text in red</div>
```

- CSS/PostCSS/SASS
  - Using *className* prop

```
// .text-in-red {
//  background: red;
//  padding: 8px;
//  margin: 2px;
//  color: white;
// }

const App = () => <div className="text-in-red">
  Text in red
</div>
```

- JSS (CSS-in-JavaScript)
  - [Multiple libraries]

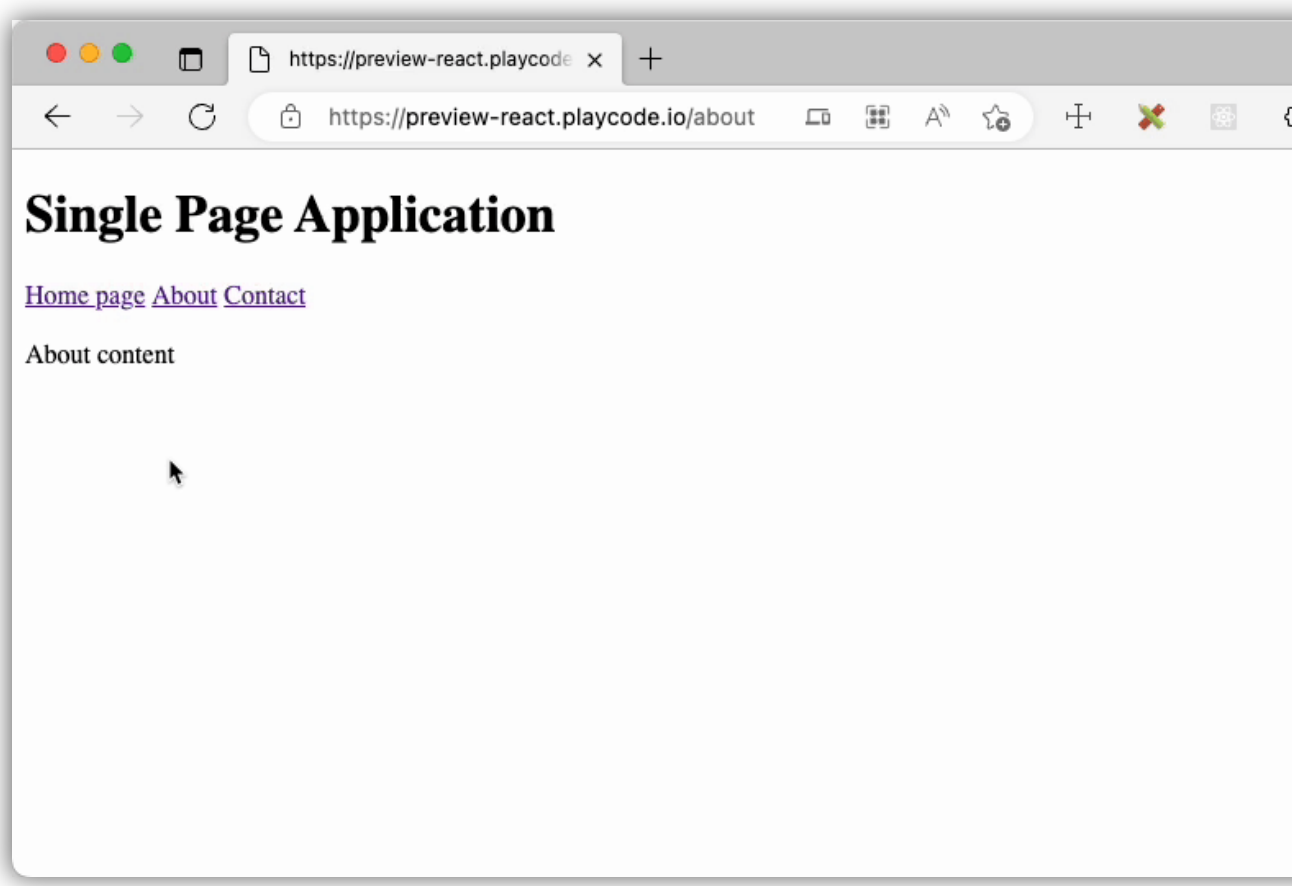
```
import styled from 'styled-jss'

const StyledDiv = styled('div')({
  background: 'red',
  padding: 8,
  margin: 2,
  color: 'white',
})

const App = () => <StyledDiv>Text in red</StyledDiv>
```



Routing – Single Page Application

- Only reloads data, page is not refreshed

```jsx
import { BrowserRouter, Link, Route, Routes } from "react-router-dom";

const Home = () => 'Home content';
const About = () => 'About content';
const Contact = () => 'Contact content';

const App = () => <BrowserRouter>
  <h1>Single Page Application</h1>
  <nav>
    <Link to="/">Home page</Link> 
    <Link to="/about">About</Link> 
    <Link to="/contact">Contact</Link> 
  </nav>
  <br />
  <Routes>
    <Route index element={<Home />} />
    <Route path="about" element={<About />} />
    <Route path="contact" element={<Contact />} />
    <Route path="*" element={<Home />} />
  </Routes>
</BrowserRouter>
```



Packages Management and Bundlers

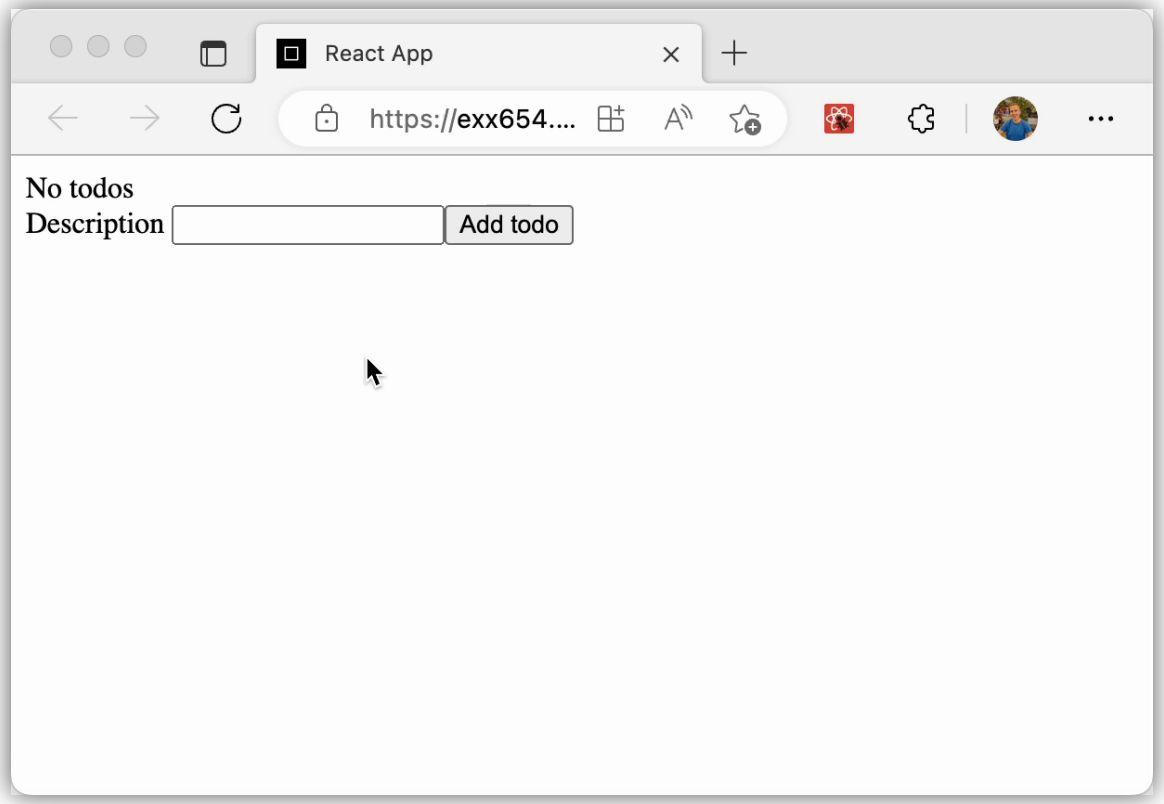
- Package management
  - Using open source / private packages
  - Contains scripts, configuration of the project
  - **NPM**, YARN, PNPM, *Bower* (obsolete)
  - Dependency tree
- Bundlers
  - Bundle JS, HTML, CSS and other resources altogether
  - **Webpack**, Snowpack, Vite, Rollup, Parcel
  - Uses Compilers to enhance JavaScript language (Babel)
    - Allows to use new features not supported by all browsers
  - Minifiers to make code/resources smallers
  - It is possible to use React without a bundler (*not recommend*)

Global State Management

- To share data across components
- Allows to pass data from bottom to top
- React Context
  - Simple, not optimized
- React Redux
  - The most used library
  - Optimized
  - Old, needs a lot of boilerplate
- Recoil
  - Developed by Meta, atomic approach
- Hook based libraries
  - Zustand

Zustand Global State Example

```jsx
import create from "zustand";

const useTodosStore = create((set, get) => ({
  todos: [],
  addTodo: (todo) =>
    set((state) => ({
      todos: [
        ...state.todos,
        // add ID to the todo
        { ...todo, id: state.todos[state.todos.length - 1]?.id + 1 || 0 }
      ]
    }))
}));

const Form = () => (
  <form
    onSubmit={(event) => {
      event.preventDefault();
      useTodosStore.getState().addTodo({
        description: event.target.description.value
      });
      event.target.description.value = "";
    }}
  >
    Description <input name="description" />
    <button type="submit">Add todo</button>
  </form>
);

const TodoList = () => {
  const todos = useTodosStore((state) => state.todos);

  if (!todos.length) return "No todos";

  return (
    <ul>
      {todos.map(({ description, id }) => (
        <li key={id}>{description}</li>
      ))}
    </ul>
  );
};

const App = () => (
  <div>
    <TodoList />
    <Form />
  </div>
)
```



Form State Management

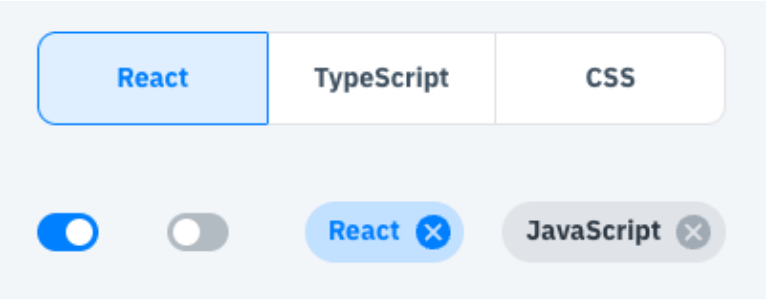
- Similar to global state, only on form level
- Performance optimization
- Form state: values, dirty, validation, conditions, …
- Handles event
- Based on field configuration libraries
  - Formik, react-hook-form, reactfinal-form, Unform, @shopify/reactform, felte, …
- Based on schema defined forms
  - Data-driven-forms, uniforms, React Json Schema Form, Winterfell

API Management

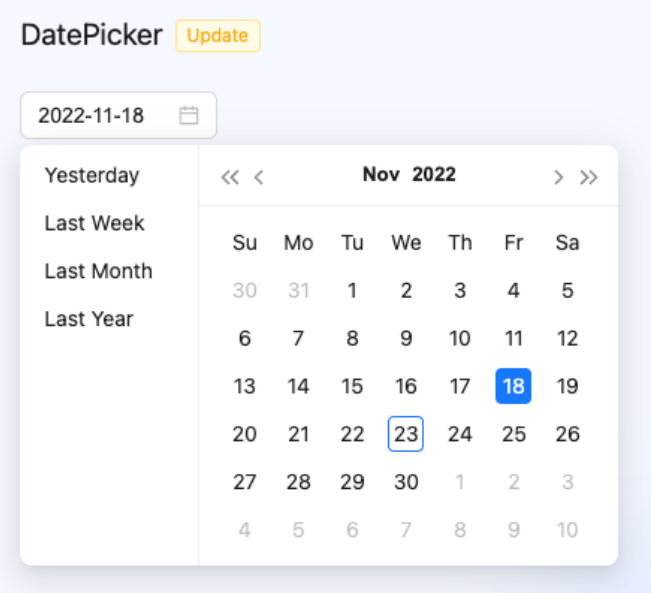
- Unify using of API across UI application
- Cache, validation, global store management
- Apollo GraphQl
  - For GraphQl requests
- Axios
  - Enhancement of the default *window.fetch* API
- SWR
  - Pagination, polling, cache, optimistic UI, …

Design System Libraries

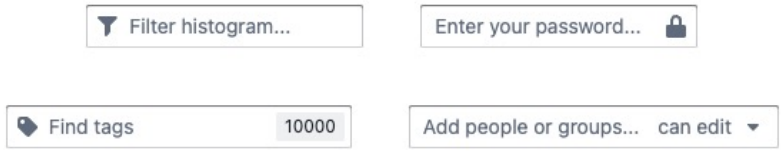
- Already implemented and designed components
- Rapid development, prototyping, production development
- MaterialUI, React Bootstrap, Ant Design, BlueprintJS, …



Material UI



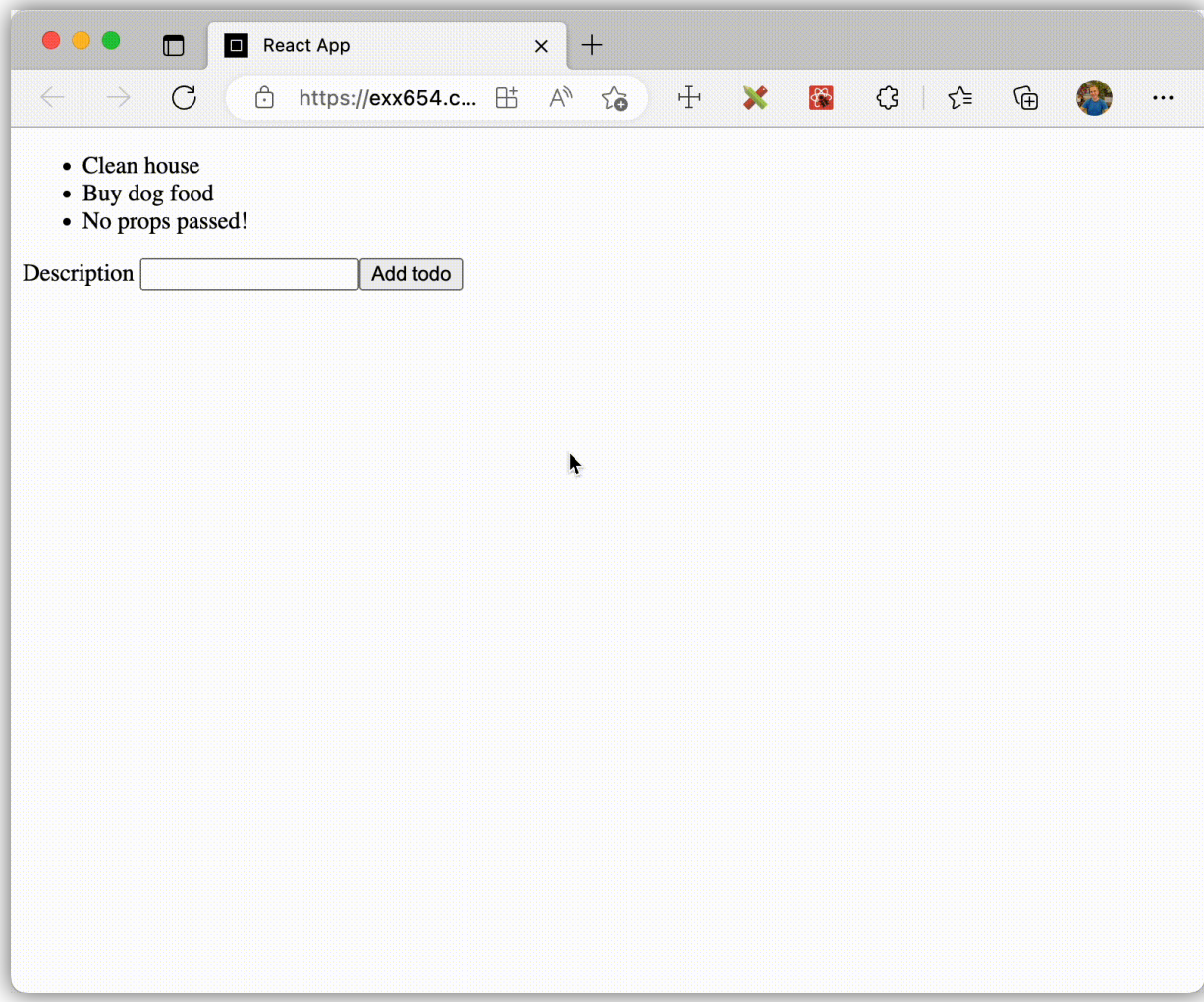
Ant Design



BlueprintJS

React DevTools

- Browser tools for debugging React application
- ReactDOM overview and Components inspector
  - Can check and change props, state
- Profiler
  - To track performance, rendering times



Server-side React Frameworks

- React can be also rendered on server
- Faster performance for users (only receive HTML), frontend directly accesses database and backend, better for SEO
- Components can still stay dynamic – in that case JavaScript is still sent
- NextJS, Remix
- React Server Components
  - Experimental
  - Streamed to browser, not just HTML, all the functionality is kept on server

Testing React

- Use React Testing Library for Unit testing
  - To test functionality, not styling
  - Runs in Node, not in a browser
  - Quick and cheap
  - Use when developing
  - Jest runs tests, RTL renders them
  - *Enzyme is no longer supported!*
- Cypress component testing
  - Runs in browser without backend
  - Slower
- E2E tests (Cypress, Playwright, …)
  - Runs in browsers with backend connected
  - The slowest approach
  - Use when approving code/changes


```
import "@testing-library/jest-dom/extend-expect";
import React from "react";
import { render, screen } from "@testing-library/react";

const Input = ({ name, value, label }) => (
  <div>
    <label htmlFor={name}>{label}</label>
    <input id={name} name={name} value={value} />
  </div>
);

it("React Testing Library works!", () => {
  render(<Input name="nickname" value="Johny" label="Provide nickname" />);

  expect(screen.getByLabelText('Provide nickname')).toHaveValue('Johny');
});
```

React Optimization

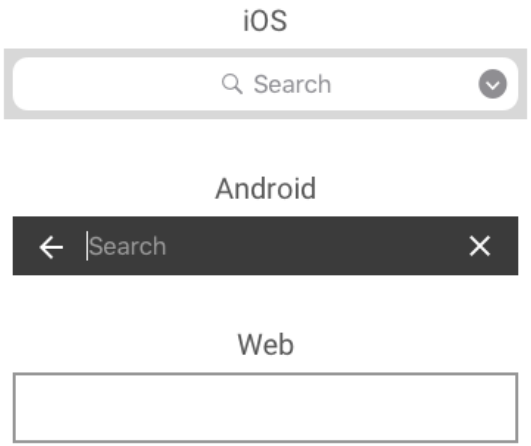
- React can be *slooooooow*
- Optimize:
- Rendering
  - Memoization (components are not rendered when props are not changed)
  - Subscription rendering (components are rendered only if their subscribed state is changed)
  - Debouncing (when typing, state/functionality is not triggered on each key stroke)
- Bundle size
  - Tree shaking (not used components are not bundled in the code)
  - Code splitting (components are loaded only when needed)
- Computation
  - Memo values (computed values are re-calculated only when a dependency is changed)
  - Cache values (do not recompute functions for the same values)

React Native – Mobile Applications

- React can be also used to develop native iOS and Android applications
- Instead of HTML elements it uses native elements of each system
- Slower than truly native applications, but codebase can be shared across platforms

```
1 import { Text, View } from 'react-native';
2
3 const HelloWorldApp = () => (
4     <View>
5       <Text>Hello, world!</Text>
6     </View>
7 )
```

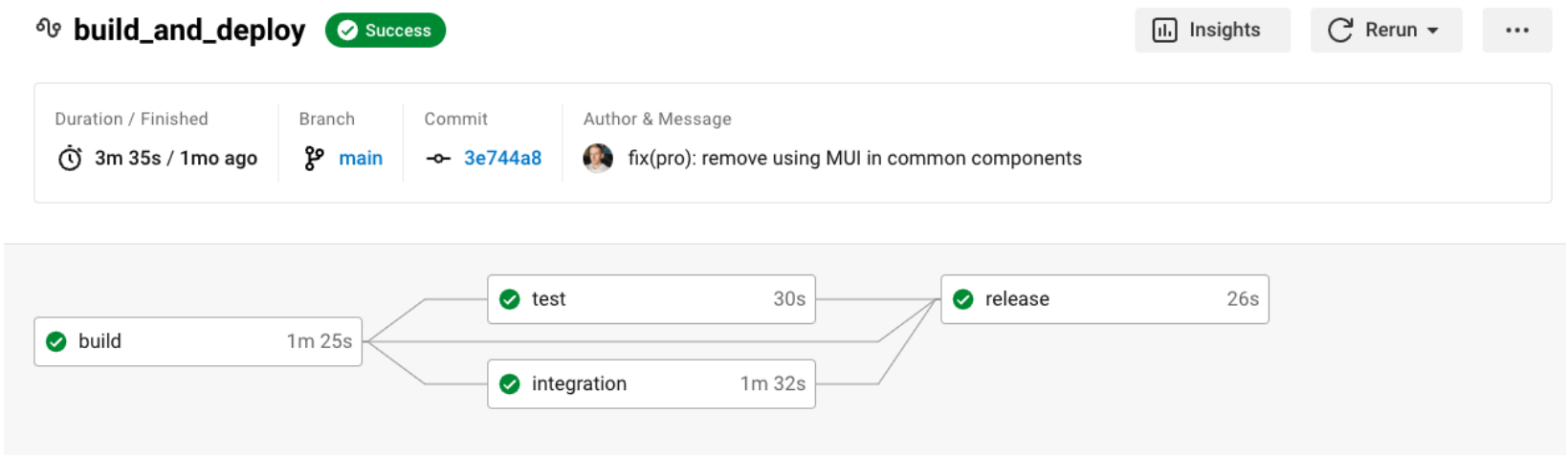
Not using HTML elements



Same component, different look

CI – Continuous Delivery

- Pipelines to automatically build, test and deploy applications
- CircleCI, Jenkins, Travis CI, GitHub Actions, …



React and TypeScript

- React and Typescript can be used together
- Providing typing to components

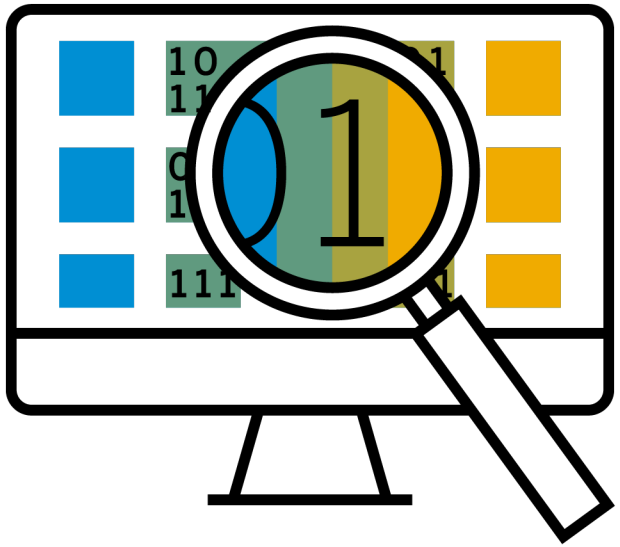
```tsx
interface ItemProps extends React.HTMLProps<HTMLDivElement> {
    title: string;
    description?: string;
}

const Item: React.FC<ItemProps> = ({ title, description, ...props }) => (
    <div {...props}>
        <h1>{title}</h1>
        {description && <small>{description}</small>}
    </div>
)
```

Alternatives

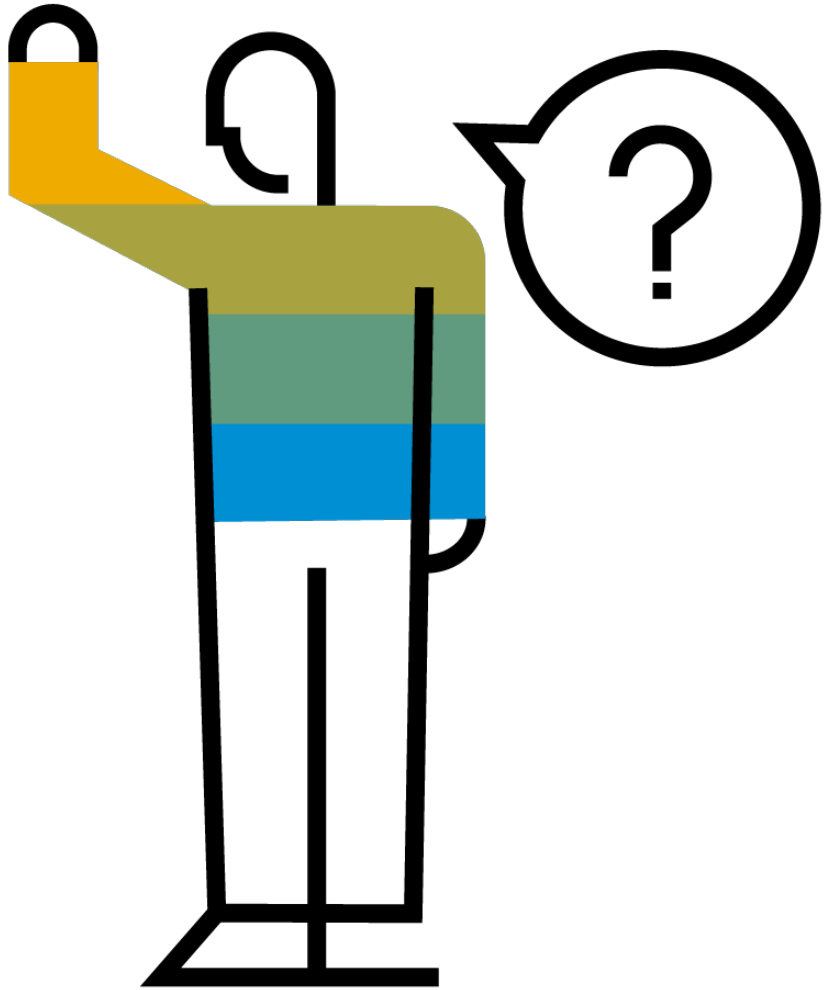
- MVC and templating frameworks
  - *Obsolete*, but still used
  - See MVC frameworks
- **Other JavaScript frameworks**
  - Vue – more developer friendly
  - Angular – includes more features, enterprise ready
  - Svelte – small community, similar to vanilla JS
  - SolidJS – similar to React, still early
- WebAssembly
  - Native code compiled to WebAssembly
  - Fast performance, still early to use
  - Blazor (.NET framework)

More Resources

- Official documentation: https://reactjs.org/, https://react.dev/
- YouTube
  - https://www.youtube.com/c/TheoBrowne1017
  - https://youtu.be/rzwaaWH0ksk
- Books
  - https://www.roadtoreact.com/
  - Eloquent JavaScript
- Open source projects



Thank you.



Contact information:

Richard Všianský
richard.vsiansky@sap.com

THE BEST RUN SAP