Week 09 — React - components, local state and list processing

Lukáš Grolig et al.

Outline

- Motivation
- SPA
- React conceptually
- React basics
- Some basic example
- React native
- Next.js and SSR
- Other frameworks

Motivation

- Most of the time, there is a need to update the content of a website dynamically
- Static HTML pages are unusable here, as you would need to manually update the pages
- To create a dynamic website, you could use a library like jQuery
- Nowadays, jQuery is still used on legacy systems but has mostly been replaced

What is SPA?

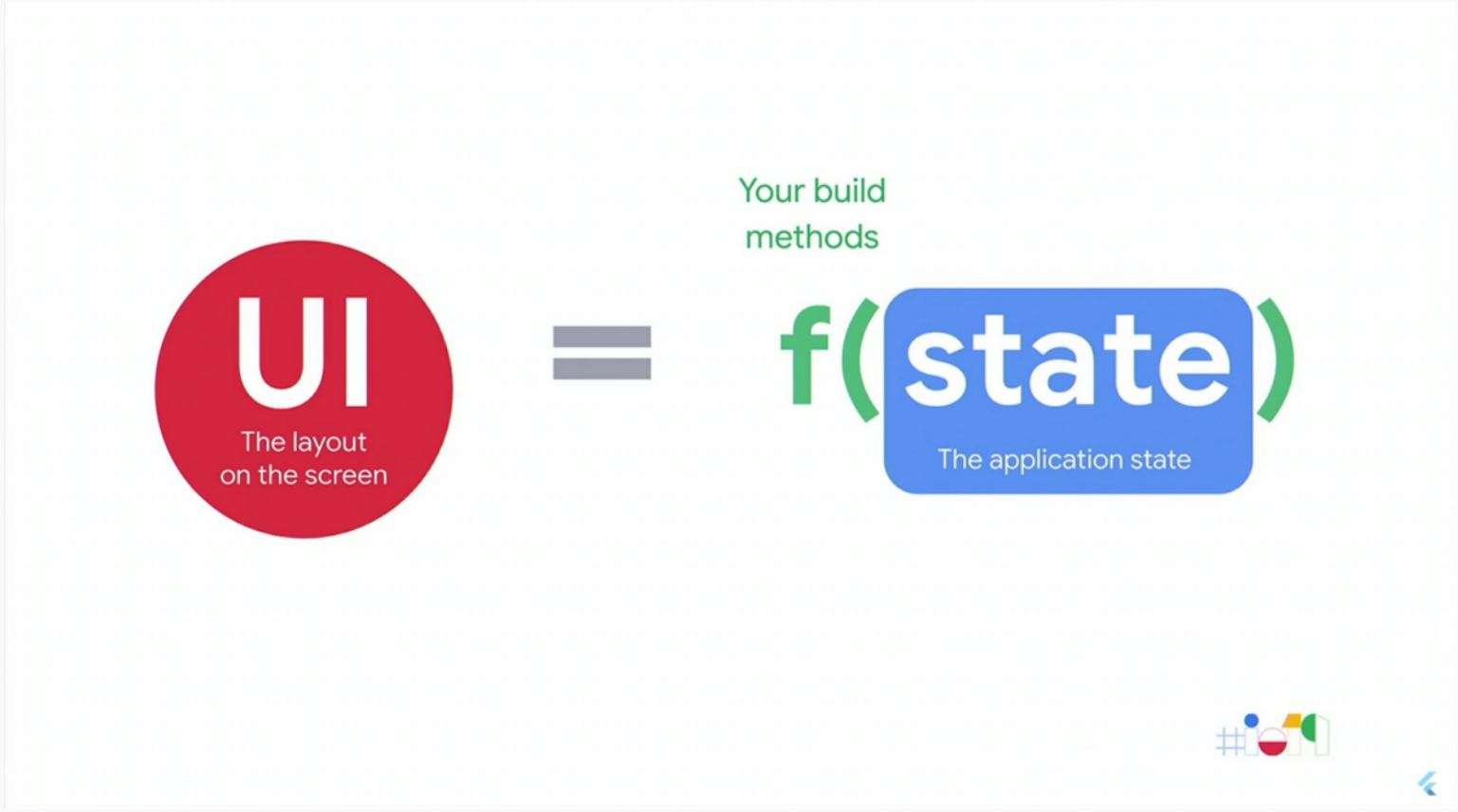
- SPA = Single Page Application
- The app is running on the client.
- Applications are reactive and responsive to changes on the server.
- Applications are communicating with the server to show the latest information or perform some state change.
- The same app can be used anywhere.
- Takes more time to develop.

Single Page Application	Server Side Application
can be written in JS or language targeting JS or WASM	can be written in any lang
low consumption of server resources	client receives only HTML
more performance on client required	server resources heavy
PWA is possible = apps can work without connection	not many requirements on client
no requirement for refresh upon change	application is fast to develop
	no offline application (PWA)
	every action needs reload

React.js

- JavaScript library for building user interfaces.
- Created to handle large-scale applications like Facebook, Messenger, Instagram, Airbnb, Netflix, and others. Even Microsoft is using it.
- They do a really good job delivering backward compatibility.
- React is only concerned with state management and rendering that state to the DOM, most of the time additional libraries must be used
- Uses Client Side Rendering - rendering pages directly in the browser using JavaScript
- All logic, data fetching, templating and routing are handled on the client rather than the server.

UI = f(state)


UI
The layout
on the screen
=
Your build
methods
f(state)
The application state

Javascript syntax extension (JSX) / (TSX)

```
const element = <p>You clicked 0 times</p>
```

- JSX is also called Javascript XML and is used to create React elements
- Any JS expression can be put into JSX and is an expression itself
- Using `class` is not permitted inside JSX, instead `className` is must be used

```
const element = <p className="clicksNumber">You clicked {count} times</p>
```

- JSX is nothing more than syntactic sugar for the React.createElement() function.

```
<MyButton color="blue" shadowSize={2}>
  Click Me
</MyButton>
```

```
React.createElement(
  MyButton,
  {color: 'blue', shadowSize: 2},
  'Click Me'
)
```

Components

- Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.
- Components are not limited to React, pure JS, and every other framework that uses them.
- For example, implementation of Components in pure JS can be done using Web Components
- Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return elements describing what should appear on the screen.

```
export const Counter = (props) ⇒ {
  <div className="counter">
      <p>You clicked {props.count} times</p>
  </div>
}

// Rendering our component
<Counter count=5 />
```

- This is the functional way of creating components in React, you could also use an ES6 class to define a component

Components in React

- The first letter of React component must be capitalized, otherwise React will recognize it as an HTML tag
- When React sees an element representing a user-defined component, it passes JSX attributes and children to this component as a single object. We call this object “props”.
- Props are Read-Only, therefore a component must never modify its props.
- While using TypeScript it is best practice to define an interface containing props

```
interface CounterProps {
    count: number;
}

export function Counter(props: CounterProps) {}
```

Components in React

- Methods can also be created inside a React component

```
export const Counter = () => {
  const incrementCount = () => {
      // method which increases count by 1
  }

  return (
    div className="counter">
      <p>You clicked {count} times</p>
      <button onClick={incrementCount}>
        Increment count
      </button>
  </div>
  );
}
```

Component Lifecycle

- Every component has a life cycle, which consists of 3 phases
  - First is mounting, which is when the component is rendered for the time
  - Second is updating, which is every subsequent render
  - Third and last is unmounting, which is when we remove the component from the DOM
- The Effect hook is called in all 3 phases, but we will talk about it later in the lecture

Conditional render

```
export const Counter = ({count}) => {
  if (count > 0) {
    return <p>You clicked {count} times<p/>;
  }
  return <p>You didn't click yet.<p/>;
}
```

How to prevent component render

```
export const WarningBanner = (props) => {
  if (!props.warn) {
    return null;
  }

  return (
    <div className="warning">
      Warning!
    </div>
  );
}
```

React hooks

- Hooks let you use more of React’s features without classes.
- It allows you to use state and other React features without writing a class. Hooks are the functions that "hook into" React state and lifecycle features from function components.
- Calling Hooks inside loops, conditions, or nested functions is not permitted.
- Instead, call hooks only from React function components or from custom hooks

For example:

- **State Hook**
- **Effect Hook**
- Context Hook
- Reducer Hook
- Layout effect hook

State hook

```
const [state, setState] = useState(initialState);
```

- Initially, the value of the state is set to initialState
- Value of the state is changed with setState
- setState not only changes the value but tells each component that uses the given state to rerender

- For example: Creates a button and a counter which increments with each click
- If normal variables were used, the value would change, but the component wouldn't rerender

```
function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Effect hook

```
useEffect(func);
```

- Adds an ability to perform side effects from a function component
- Calls passed function after every render
- Multiple effect hooks can be placed inside one component

`useEffect(func, [state]);`

- Alternatively, the Effect hook can have a second parameter, which is a list of states
- In this case, the function is called every time a component using given states change
- For example: The given function will be called only after the state 'count' changes

```
useEffect(() => {
  document.title = `You clicked ${count} times`;
}, [count]);
```

Lists

- In React transforming arrays into lists of elements is nearly identical.
- You can build collections of elements and include them in JSX using curly braces {}.

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) ⇒
  <li>{number}</li>
);

// later in the code
<ul>{listItems}</ul>
```

- This example is bad because it doesn't use `key`, more on the next slide
- You can also put this code into a component and pass the numbers as props

Keys

- In React, list keys help to identify which items have changed, are added, or are removed
- The best way to pick a key is to use a string that uniquely identifies a list item among its siblings. For example ID from data.
- Item index can be also used as a key, but only as a last resort
- Use index as a key only if these conditions are met, otherwise your application may break your application and display the wrong data
    - the list and items are static
    - the items in the list have no ids
    - the list is never reordered or filtered

Virtual DOM (VDOM)

- It is based on assumption that virtual is faster than regular DOM.
- The virtual DOM is a programming concept where an ideal, or “virtual”, representation of a UI is kept in memory and synced with the “real” DOM by a library such as ReactDOM. This process is called reconciliation.
- When using React, at a single point in time you can think of the `render()` function as creating a tree of React elements.
- On the next state or props update, that `render()` function will return a different tree of React elements.
- To efficiently update the UI to match the most recent tree, React implements a heuristic O(n) algorithm based on two assumptions:
  - Two elements of different types will produce different trees.
  - The developer can hint at which child elements may be stable across different renders with a key prop.

Hooks and objects

- Sometimes we need to store an object as a state instead of primitive type
- Changing objects properties does not trigger rerender
- Use the spread operator `...` to "recreate" the object

Shipping the react app

- Compile, minify typescript
- Process, bundle css
- Copy assets
- Now your app can be served by any conventional file server (nginx, caddy, apache)

Css in React

- There are many ways to use CSS in React
- Inline css
  `<p style={{ color: 'red'}}>You clicked {count} times</p>`
- Style object

```
const countColor = {
  color: 'red'
};

<p style={countColor}>You clicked {count} times</p>
```

- Importing a CSS stylesheet
  `import './Styles.css';`
- Css modules - create a special css file with `.module.css` extension

```
import * as styles from "./counter.module.css"

<p className={styles.count}>You clicked {count} times</p>
```

- All modules are locally scoped, therefore you don't have to worry about name collision

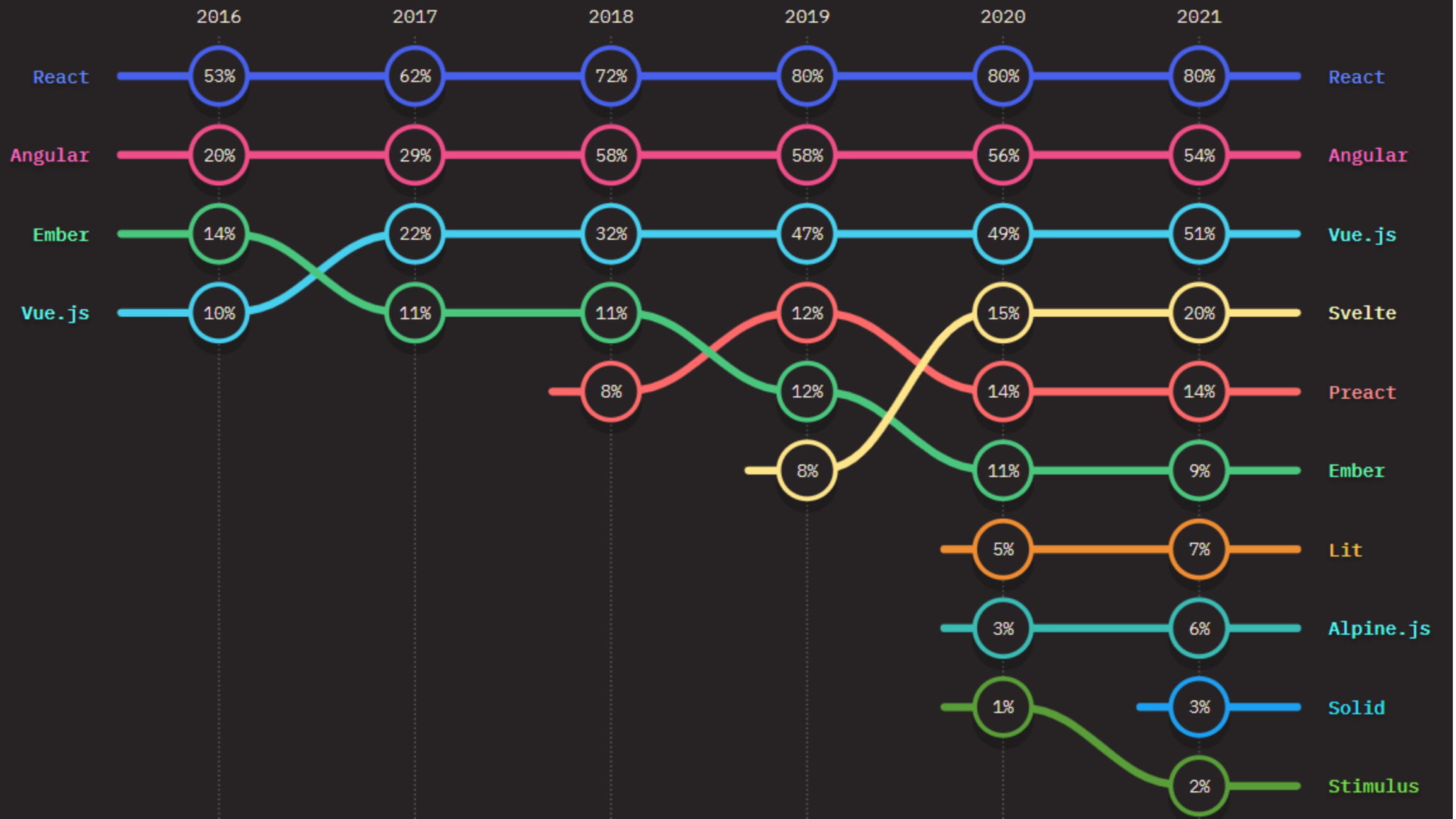
React native

- Cross-platform mobile framework that uses Reactjs for building apps and websites that run on different platforms
- Apps developed with React render HTML in UI while React Native uses JSX for rendering UI, which is nothing but javascript.
- Hot Reloading - Making a few changes in the code of your app will be immediately visible during development.
- Takes more time to initialize

Next.js and Remix

- Framework built on top of React, which enables it to use server-side rendering and to generate static websites
- Data Fetching allows you to render your content in different ways
- `ReactDomServer.renderToString(<div>p</div>);`
- Remix is another frontend based od React
- It provides APIs and conventions for server rendering, data loading, routing and more.
- Bundling is the process of following imported files and merging them into a single file (bundle)

Satisfaction | Interest | Usage | Awareness



Resources

- https://slides.com/lukasgrolig/pb138-introduction-to-react-72d059
- https://reactjs.org/docs/faq-internals.html
- https://reactjs.org/docs/reconciliation.html
- https://reactjs.org/docs/introducing-jsx.html
- https://reactjs.org/docs/jsx-in-depth.html
- https://reactjs.org/docs/components-and-props.html
- https://twitter.com/dan_abramov/status/981712092611989509

Resources cont.

- https://reactjs.org/docs/hooks-overview.html
- https://reactjs.org/docs/hooks-effect.html
- https://reactjs.org/docs/lists-and-keys.html
- https://robinpokorny.medium.com/index-as-a-key-is-an-anti-pattern-e0349aece318
- https://www.javatpoint.com/reactjs-vs-reactnative
- https://www.gatsbyjs.com/docs/how-to/styling/css-modules/
- https://developers.google.com/web/updates/2019/02/rendering-on-the-web#csr