TIA - React frontend

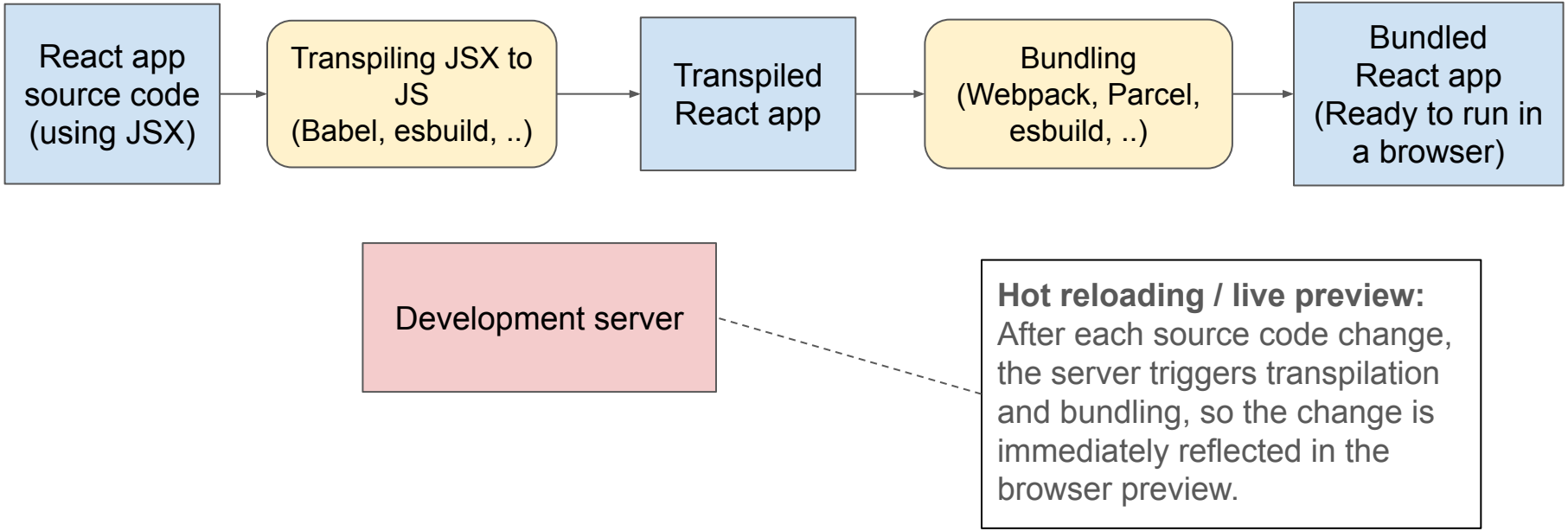
Jana Kostičová

Preliminary steps

- Create GitHub public repository
  - You should have it created already
- Install Git locally
  - https://git-scm.com/downloads - comes with built/in GUI tools git gui, gitk
  - (optional) Git 3rd party GUI client https://git-scm.com/downloads/guis
  - Git bash is sufficient
- Install Visual Studio Code or another IDE
  - You can use built-in IDE git support instead of git bash
- Clone your public git repository to …/my-app directory

```
git clone url
```

React dev environment



Dev environment setup 1 - Install Node.js locally

It is recommended to install latest LTS (long-term support) version

Basic installation

- https://nodejs.org/en/download/
  - Common issue in Win: “error 'ENOENT: no such file or directory' when creating a react project” https://stackoverflow.com/questions/76635661/npm-shows-an-error-enoent-no-such-file-or-directory-when-creating-a-react-pro

Advanced installation - via NVM / node version manager (enables flexible switching between node versions)

- Nvm (Linux, macOS): https://github.com/nvm-sh/nvm
- Nvm-windows:
  - https://learn.microsoft.com/en-us/windows/dev-environment/javascript/nodejs-on-windows#install-nvm-windows-nodejs-and-npm
  - https://github.com/coreybutler/nvm-windows/releases

Dev environment setup 2 - Bootstrap React App

Vite tool

- https://vite.dev/guide/

```
cd my-app
npm create vite@latest my-app-fe --template react
cd my-app-fe
npm install
npm run dev
```

- You can choose JavaScript or Typescript
- Commit changes to Git
  - We use the mono repo (check if it is suitable for your hosting)

React components

- React uses component-based design
- **Component** = “a piece” of user interface
- Functional components and class components
- Components are nested to form a tree structure

Example - component tree

```
function App() {
  return (
    <div>
      <Header />
      <MainContent />
      <Footer />
    </div>
  );
}
```

```
function Header() {
  return <h1>Welcome to my website!</h1>
}

function Sidebar() {
  return <div>Sidebar content</div>
}

function Article() {
  return <div>This is the main content of the
page.</div>
}

function MainContent() {
  return (
    <>
      <Sidebar />
      <Article />
    </>
  )
}

function Footer() {
  return <span>&#169; 2025 All rights reserved</span>
}
```

Passing data via props

```
function Parent() {
  const parentData = "Hello from Parent";
  return (
    <div>
      <h1>Parent Component</h1>
      <Child data={parentData} />
    </div>
  );
}

function Child(props) {
  return (
    <div>
      <h2>Child Component</h2>
      <p>{props.data}</p>
    </div>
  );
}
```

Child component gets data from the parent component.

React hooks

- Special functions that allow you to "hook into" component state and lifecycle
- Used only in functional components
- *useState, useEffect, …*
- Hooks must be placed at the top level of the component

Example - useState hook

```
function Counter() {
  // Declare a state variable 'count' with an initial
  // value of 0 and a setter function setCount
  const [count, setCount] = useState(0);

  return (
    <div>
      <h1>Counter: {count}</h1>
      <button
         onClick={() => setCount(count + 1)}>Increment
      </button>
    </div>
  );
}
```

Component Counter is **re-rendered** each time the value of ‘count’ changes

useState hook enables a component to re-render when the value of its state changes

eventHandlers

- Functions triggered by user actions / events
- Examples: *onClick, onChange, onSubmit, …*
- They are commonly used to update component's state based on user action
- Be careful to provide a function reference to an event handler, not a function call !

WRONG:

```
<button onClick={setCount(count + 1)}>Increment</button>
```

- function is executed when the component renders

CORRECT:

```
<button onClick={() => setCount(count + 1)}>Increment</button>
```

- function is executed when the user clicks the button

```
function Child(props) {
  return (
    <div>
      <h2>Child Component</h2>
      <p>{props.data}</p>
    </div>
  );
}

function Counter() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <h1>Counter: {count}</h1>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <Child data={count} />
    </div>
  );
}
```

Passing state via props

- “count” is passed from Counter to ChildComponent via props
- Both components re-renders if “count” changes

```
const Parent = () => {
  const [data, setData] = useState('Initial data');
  return (
    <div>
      <h1>Data: {data}</h1>
      <Child sendDataToParent={setData} />
    </div>
  );
};

const Child = ({ sendDataToParent }) => {
  return (
    <div>
      <button onClick={() => sendDataToParent('Data from child')}>
         Send Data to Parent
       </button>
    </div>
  );
}
```

Passing state from child to parent

- Via callback function sendDataToParent

Example - useEffect hook

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

  // useEffect hook to update the document title whenever 'count' changes
  useEffect(() => {
    document.title = `You clicked ${count} times`;
  }, [count]); // Dependency array, useEffect will run when 'count' changes

  return (
    <div>
      <h1>{count}</h1>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
```

useEffect hook

- Enables side effects after the component renders (fetching data, subscribing to a service, timer, ...)

Parameters

- Effect function (a function reference, not a function call !)
- Dependency array (optional)
  - Not provided - hook runs after each re-render
  - Empty [] - hook runs only ONCE after the first render (component mount)
  - Nonempty - hook runs after the first render and then if some of the dependencies changes

Returns

- Cleanup function (optional)

Example - useEffect hook, timer

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

  useEffect(() => {
    // create the interval to increment the count
    const intervalId = setInterval(() => {
      setCount(prevCount => prevCount + 1);
    }, 1000); // 1 second
    return () => clearInterval(intervalId); // clear the interval on unmount
  }, []);          // empty dependency array - this effect runs once on mount

  return (
    <div>
      <h1>Count: {count}</h1>
    </div>
  );
}
```

React component lifecycle

A. Mounting phase
   - **useState():** Initializes state variables. React creates and tracks these state variables across renders.
   - **useEffect():** Executes side effects after the component renders (each useEffect executes after component mounting, regardless its dependencies)

B. Updating phase
   - **useState():** Updates state with setState and triggers a re-render when the state changes
   - **useEffect():** The effect will re-run if
     i. Some of its dependencies changes
     ii. No dependencies are provided

C. Unmounting phase
   - **useEffect():** cleanup function is executed (if provided)

React router

- SPA internal routing
- Needed in case of multiple “pages”
- Install package: `npm install react-router-dom`
- (Restart development server)
- Import:

```
import { createBrowserRouter, RouterProvider } from 'react-router-dom';
```

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <App>
      <MessageListPage></MessageListPage>
    </App>
  },
  {
    path: "/compose",
    element: <App>
      <NewMessagePage></NewMessagePage>
    </App>
  }
])

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <RouterProvider router={router} />
  </StrictMode>,
)
```

React router example

- Modify main.jsx
- Navigation examples:

1. **Button navigating to /compose**

```
<Link className="btn btn-primary"
to="/compose">
    +
</Link>
```

2. **Programmatical navigation to /**

```
const navigate = useNavigate();
navigate("/");
```

Source structure (one of many options)

```
├── /src
│   ├── /assets          # Static assets (e.g., images, fonts)
│   ├── /components      # Reusable components
│   ├── /data            # Mock data or data-related logic
│   ├── /pages           # Page components
│   ├── App.jsx          # Main app component
│   ├── main.jsx         # React entry point
│   ├── style.css        # Global styles
```

Components

- Rule of thumb: One component per one JSX file
- Group of coupled components can optionally be placed into a single JSX file
  - For example when only one component is exported and the remaining components are its subcomponents that are not used anywhere else
- export / import

References

- https://react.dev/
- https://vite.dev/guide/