Code.Hub

The first Hub for Developers

Ztoupis Konstantinos

Code Splitting

Code.Learn Program:
React

Single Page Application

- huge bundle size
- as the app grows, the bundle becoming too large and hence begins to impact page load times
- download a 1MB package of JavaScript to load only the first page

Bundling

- “bundled” using tools
- the process of following imported files and merging them into a single file: a “bundle”

Bundlers

- Webpack and Browserify provide support for code-splitting
- splitting the code into different bundles which can be loaded on demand (lazy-loaded)

The problem

- bundle includes large third-party libraries
- bundle contains code that might never run

The solution

Code splitting

the practice of only loading the JavaScript you need the moment when you need it

Code Splitting

Code Splitting improves

- the performance of your app
- the impact on memory, and so battery usage on mobile devices
- the downloaded KiloBytes (or MegaBytes) size

Code Splitting

Add code-splitting into an app through the dynamic syntax import()

- is not yet part of the JavaScript language standard
- import() for loading a module relies on JavaScript Promises
- a promise that is fulfilled with the loaded module or rejected if the module could not be loaded

Dynamic Imports

```
new Promise(resolve => resolve())
  .then(() => import('moment'))
  .then(moment => {this.setState({time: moment().format('LLLL')})})
  .catch(err => console.log('Failed to load moment library', err))
```

Code Splitting in React Part 1

- package for code-splitting React components called **react-loadable**
- provides a HOC for loading React components with promises, leveraging on the dynamic import() syntax

Code Splitting in React Part 1

```
import Loadable from 'react-loadable';

const Menu = Loadable({
   loader: () => new Promise(resolve => resolve()).then(() => import('./Menu')),
   loading: LoadingComponent
});
```

Code Splitting in React Part 2

- React.lazy
- React.Suspense

Lazy

- function lets you render a dynamic import as a regular component
- automatically load the bundle containing the component when this component gets rendered
- return a Promise which resolves to a module with a default export containing a React component

```
const Component = React.lazy(() => import('./Component'));
```

Suspense

- fallback: prop accepts any element while waiting for the component to load
- suspense: anywhere above the lazy component
- wrap multiple lazy components with a single suspense component

```
<Suspense fallback={<div>Please wait...</div>}>
  <Component />
</Suspense>
```

Route-based code splitting

- do not disrupt the user experience
- choose places that will split bundles evenly
- add code splitting to routes or different pages

Route-based code splitting

```
const Home = lazy(() => import('./routes/Home'));
const About = lazy(() => import('./routes/Info'));

const App = () => (
  <Router>
    <Suspense fallback={<div>Please wait...</div>}>
          <Route exact path="/home" component={Home}/>
        <Route path="/info" component={Info}/>
    </Suspense>
  </Router>
);
```

Error boundaries

- analogous to try{ }catch(e){ } statements
- components that catch JavaScript errors anywhere in their child component tree
- log errors
- display a fallback UI instead of the crashed component tree

Error boundaries

An error boundary can be placed anywhere above lazy components to show nice user experience if a lazy component fails to load

```
<ErrorBoundary>
  <Title>Welcome home</Title>
  <Suspense fallback={<Loader />}>
    <Home />
  </Suspense>
</ErrorBoundary>
```

Named Exports

- React.lazy supports default exports
- for importing named exports, create an intermediate module that reexports it as default

Named Exports

```
export const Home = () => {};      // Components.js
export const Info = () => {};



export { Home as default } from "./ Components.js";    // HomeComponent.js



import React, { lazy } from 'react';      // MyApp.js
const Home = lazy(() => import("./HomeComponent.js "));
```