香港中文大學
The Chinese University of Hong Kong

CSCI2720 - Building Web Applications

Lecture 12: SPA and Routing

Dr Colin Tsang

Outline

- Page-based navigation
- Single-page apps
- Routing in app
- React-router

From web pages to web applications

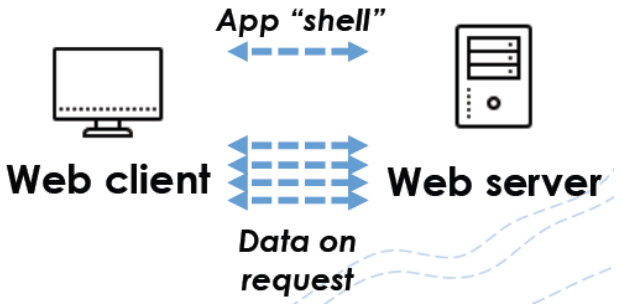
- The web originated as a system for managing *static content pages with hyperlinks*
  - Consider your experience with Wikipedia articles
- And then the web evolved with the use of CSS, JavaScript, and all kins of *dynamic contents*.
- Web applications
  - Presenting your contents/services in an environment like native applications, yet in a browser.

Page-based navigation

- Bandwidth waste by page structures and styling being *reloaded for every page*
  - Caching may help, i.e., reload from your local machine instead of server.
- The loading of a page (e.g., blanking out) gives a poor experience for the user
  - *User experience* (UX) became a term so ubiquitous when the web becomes part of everyone's life.

Single-page application (SPA)

- Ajax techniques became widely used around 2005
  - Asynchronous retrieval of data, without reloading
  - Now developers are gradually shifting to use the *Fetch API*
- The paradigm of SPA easily got popular
  - Initial load: the page framework (HTML), styles (CSS), and code (JS).
  - Subsequent loads: Only *fragments of data* required to be display.



Different views of SPA

- Pagination
  - Contents are displayed based on links
  - Good for locating items (on a certain page), with a sense of control
  - E.g., Google search
- Infinite scrolling
  - Contents are displayed without a finishing line
  - New contents are loaded slightly ahead
  - Good for content discovery, and optimized for mobile app experience
  - E.g., Instagram/Facebook post

- https://uxplanet.org/ux-infinite-scrolling-vs-pagination-1030d29376f1

Issues of SPA

better in mobile devices

- Page performance on various devices
- Search and location of items on page
- Scroll bar does not reflect amount of actual data
- Lacking a page footer

- Consideration of UX!

SPA: Pros and Cons

What's good?	*What's bad?*
Efficient use of bandwidth	Browser history is not trivial, not easy for bookmarking
Separation of view and data ➔ easier for maintenance	Search engine optimization: SEO contents not easily retrieved by robots (Not friendly to search engine) (PageRank Algorithm)
Imitating native applications and improved user experience without obvious reloading and waiting	JavaScript dependent ➔ thin server, thick client

Routing in the App

- Modern JS frameworks easily support SPA by allowing change of contents in *components*.
- Routing is a fundamental feature in frameworks
  - To display different contents basing on where the user is in the application

- http://rhymedcode.net/javascript-framework/angular-vs-react-vs-vue-routing/

File-system routing

- By default, a web server simply serves everything under a specified directory as **"/"** (root) of the URL
  - E.g., https://www.cse.cuhk.edu.hk/academics/ug-course-list/ could be pointing to **index.html** (default filename) in this directory list
  - All directories and files are served, basing on file permissions on the server
    - Admins often set permission *711* (others *no read*, *no write*, only *execute*) to the directory, so a list of files will not be shown if accessing the directory
- Even with the routing methods mentioned later, it is still possible to let React serve static content using the **/public** director
  - Usually for media or data files

```
/
+ about
+ academics
  + ug-course-list
    - index.html
    ...
+ research
+ people
...
```

Static and dynamic routing

- Static routing:
  - Routing before rendering takes place during initialization
  - Allowing inspection and matching of routes earlier
  - Used by Express, Angular, Vue, …
- Dynamic routing:
  - Routing takes place as the app is rendering
  - A component is rendered if a path is matched as a prop
  - Possible for responsive routes
  - Used by React-router
- See: https://www.techtarget.com/searchnetworking/answer/Static-and-dynamic-routing

History API

- In the browser, the URLs visited by the user can be found in the window.history object
  - **history.back()** loads the previous page
  - **history.forward()** loads the next page (if user went back before)
  - **history.go(num)** loads the specific item (*num = -1, -2,…*) in the history list
- It is possible to change the browser's current visited page with **window.location**
  - **location.assign()** loads a new URL
  - **location.replace()** replaces the current URL from the history with a new one
  - See: https://developer.mozilla.org/en-US/docs/Web/API/Window/location

History API

- New items can be pushed into the history stack with JS
  - **history.pushState(***data, title, url***)**
  - This allows SPA to enhance navigation since users can click on the familiar Back button
  - See: https://css-tricks.com/using-the-html5-history-api/
- The history stack also provides a **history.replaceState()** method, as well as a **onpopstate** event
  - **onpopstate** is triggered when the user clicks on Back or Forward buttons
  - See: https://developer.mozilla.org/en-US/docs/Web/API/Window/popstate_event
  -

React-router

- Installing react-router-dom with either
  - Linking from CDN (this is fading out from the current version v.6.4)
    - *https://unpkg.com/history@5/umd/history.development.js*
    - *https://unpkg.com/react-router@6/dist/umd/react-router.development.js*
    - *https://unpkg.com/react-router-dom@6/dist/umd/react-router-dom.development.js*
    - Set up the relevant components before using:

```
const {BrowserRouter, Routes, Route, Link} = ReactRouterDOM;
```

  - Using npm (e.g., with create-react-app):

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

    - Set up the relevant components before using:

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

React-router

- It has made a lot of changes to syntax from v4 to v5, to v5.1, to v6, and further to v6.4
- When referring to online tutorials, beware of version differences!
  - Sometimes “traditional” syntax could be supported as well, even if not found in modern tutorials…

React-router

- stackblitz.com/edit/react-router-vkexg8

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';

class App extends React.Component {
  render() {
    return (
      <BrowserRouter>
        <div>
          <ul>
            <li>
              <Link to="/">Home</Link>
            </li>
            <li>
              <Link to="/about">About</Link>
            </li>
          </ul>
          <hr />

          <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/about" element={<About />} />
          </Routes>
        </div>
      </BrowserRouter>
    );
  }
}

class Home extends React.Component {
  render() {
    return <h2>Home</h2>;
  }
}

class About extends React.Component {
  render() {
    return <h2>About</h2>;
  }
}

const root = ReactDOM.createRoot(document.querySelector('#app'));
root.render(<App />);

```

More to discover…

- There are more useful features of React-router:
  - URL/query parameters
  - Nesting
  - 404 error page
  - Sidebar
- Learn from examples:
  - Some may not correspond to the updated version though
  - https://github.com/remix-run/react-router/tree/dev/examples

Further readings

- React-router Docs
  - https://reactrouter.com/en/main

- Ultimate react-router guide (v6)
  - https://blog.webdevsimplified.com/2022-07/react-router/