Deeplinking

- **Deeplinking** is SPA urls for specific contents
  - Even though it is all the same html page
- Two options:
  - **hash-based urls**
  - **path-based urls**
- Both require:
  - Navigating SPA "pages" changes browser url
  - JS reads URL on page load and sets app state
  - Set app state on back/forward button

Why do we need Deeplinking?

A SPA means:

- One HTML page w/content based on JS state

Reloading a SPA means

- Current content lost

Loading happens when:

- Someone follows a link to SPA
- You hit Back/Forward
- You manually reload

We don't want these situations to reset state

Routing Libraries are normal solution

- Deeplinking has lots of subtleties
    - Libraries have solved those
        - Ex: `react-router`, `@tanstack/router`
        - But you CAN do it "the hard way"
    - You are not expected to do so for this course
- BUT
    - You must understand UX impacts of options
        - Impact is more UX than UI

Hash-based Routing

- The urls for your app all use `#`
    - Often with a path-like string after it
    - Ex: `#/`, `#/about`, `#/privacy`
- As you use App, url changes to reflect new state
    - Does not count as loading the page
- Copy/Save/Share/Reload link? Back/Forward?
    - It loads a matching state
    - IS a reload
        - But starts in different state
        - Some info may be lost

Hash-based navigation

Two options:

- Have normal links that use `#`
- Code changes to URL in browser

Links with `#`

- Automatically update URL w/o navigating
- Only happens when user clicks an actual link

Code Changes URL

- Can update URL at any time

How does loading Hash-based URL work?

- Page loads
- JS checks URL before/when `<App/>` renders
    - Reads `document.location.hash`
    - Sets initial app state
        - Dev has to decide what url matches what "state"
    - Conditionally Renders based on THAT state

Notes about Hash-based Routing

- Easier to write for developer
    - No special server configuration required
- Search Engines may not index pages of app
    - All URLs indicate same page!
- Server logs can't track which links are used
    - All URLs are same according to server

Setting the URL for hash manually

Two options:

- set `document.location.hash` (example: `#example`)
- use `history.pushState()` (more later)

Back/Forward

- Good: Changes url
- Good: Does not reload page
- Bad: Does not change state
    - Make sense, we are only changing state on load

We need to detect when the url hash changes

Detecting a hash change

- window emits a `hashchange` event
- But React can't add a listener to `window`
    - It's not an element from a component
- We will have to add a listener with plain JS
    - And do it via React
    - So it can change state
- We can use `useEffect` to do this!
- Remember to cleanup the listener!

Path-based Routing

- The urls for your app all use different paths
    - Like actual files
    - Might be without file extensions
    - Ex: `/`, `/about`, `/privacy`
- Server might give same page to browser!
    - Requires Server configuration
    - on load JS creates state matching url path
- Server might give different static generated pages
    - already starts with appropriate state
- After load, pages change as SPA either way

Path-based navigation

Links/Forms with paths

- Must `.preventDefault()` to stop navigation
- Must update url to change state in url

Other state changes

- Must update url to change state in url

How does loading Path-based URL work?

- Server might give same page to browser!
    - Requires Server configuration
    - CRA server DOES do this
    - `npx serve` does NOT do this
    - Server you deploy to does/does not?
    - on load JS creates state matching url path
    - Conditionally renders based on that state
- Server might give different static generated pages
    - Requires framework configured to do this
        - CRA does NOT do this
        - NextJS CAN do this
    - Page shows appropriate state

Changes after Loading

- "Navigation" inside app sets URL
    - Using JS to set without a page load
- Done with use the History API

History API

- We can add/replace/remove from history "stack"
    - The pages the browser uses in back/forward
- We can add entries
    - Change url without navigation when added
    - Change url w/o navigation if back/forward
- Emits a `popstate` event on `window` when changed
    - We can manually add listener with `useEffect`
        - Much like we did for `hashchange`
    - So we update state to match url path
- Can be used for hash-based urls too!

window.history.pushState

- API is a little unusual
- `window.history.pushState()` takes 3 arguments
    - First is an optional bit of data ("state")
        - Allows more state than contained in url
        - Doesn't help with deeplinking urls
        - We will simply use `null`
    - Second is a historical mistake
        - Doesn't do anything, but is required
        - We will use `''` (empty string)
    - Third is url string
        - absolute path or relative path

Notes about Path-based Routing

- Better for logging
- Better for Search Engines
- Requires Server/Framework configuration

State Changes

- URL can load different states
    - What state changes represent a URL change?
    - When you load a URL, what state to you set?
- Generally a "view" or "page"
    - What content is shown
    - Usually not other state
- Could be a particular state OF a page
    - Ex: Form details filled out?
    - Ex: "Character builds" editors

URL results can create UX differences

- What if diff user sees diff content for same URL?
- Expected or a surprise?

How to manually create deeplinking URLs?

- `window.location.hash`
    - Can be read/changed for hash-based URLs
    - Can listen to `hashchange` event on `window`
        - Not using React directly
- `window.history.pushState()`, `window.history.replaceState()`
    - Can be changed for hash- or path-based URLs
    - Can listen to `popstate` event on `window`
        - Not using React directly