Deeplinking

- **Deeplinking**: URLs for specific SPA content
    - 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

A simple app to demonstrate

App.jsx

```jsx
function App() {
  const [ page, setPage ] = useState('/');

  return (
    <>
      <Header setPage={setPage}/>
      { page === '/' && <Home/> }
      { page === '/about' && <About/> }
      <Footer/>
    </>
  );
}
```

Header.jsx

```
function Header({ setPage }) {
  function changePage(e) {
    e.preventDefault();
    setPage(e.target.pathname);
  }

  return (
    <header className="header">
      <a href="/" onClick={ changePage } >Home</a>
      <a href="/about" onClick={ changePage } >About</a>
    </header>
  );
}
```

- Example is missing `className`s, `<nav>`, etc

Remember: Concepts vs Libraries

We are learning and demonstrating **concepts**

- Understanding the "magic"
- Demystifying what is happening

Outside of class:

- You would use a **routing library**
- Not do it manually

Server Configuration and Paths

http://localhost:5173/dogs-drool

- Gives your SPA!
- Any url will!
- This is a **server configuration**
- Vite Dev server DOES do this
- `npx serve` does NOT do this i
    - Gives 404 instead unless `/`
    - `npx serve` does have an option
        - Outside our interest
- Other servers may or may not do this

Without Routing/Deeplinking

- 😻 App can change views ("pages")
- 😿 URL does NOT change on view change
- 😿 JS State resets on reload

Back/Forward

- 😿 Leaves the app unexpectedly
- 😿 Causes a page load (resetting state)
- 😿 Does NOT show last page state

Path-based Routing

- The urls for your app all use different paths
    - Like actual files
    - Might be without file extensions
    - Ex: `/`, `/about`, `/privacy`
- Server must give same page to browser!
    - Requires Server configuration
    - on load JS will create state matching url path
- Once loaded:
    - in-app navigation will update URL
    - NOT actual page loads
- Back/Forward in browser made to work

Path-based navigation

Links/Forms with paths

- Must `.preventDefault()` to stop navigation
- Must tell browser to update url

Other state changes

- Such as app-driven controls to change page
- Must update url to change state in url
- Must tell browser to update url

Usually only "page" state in url!

Telling browser to change URL

- "Navigation" inside app sets URL
    - Using JS to set without a page load
- Done with the **History API** (see MDN)

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`
    - So we update state to match url path
- Can be used for hash-based urls too!

window.history.pushState

- API is a little unusual (see MDN)
- `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
    - Server gets requested URLs on page load
- Better for Search Engines
    - Search Engines think SPA is different pages
- Requires Server/Framework configuration
    - Load `index.html` instead of path in URL

Modifying our App

- Tell Browser to change URL
  - When "page" changes
- Set "page" in state
  - On Page Load
- Confirm it all works

Changing the URL using History API

```
function Header({ setPage }) {
  function changePage(e) {
    e.preventDefault();
    window.history.pushState(null, '', e.target.pathname);
    setPage(e.target.pathname);
  }

  return (
    <header className="header">
      <a href="/" onClick={ changePage } >Home</a>
      <a href="/about" onClick={ changePage } >About</a>
    </header>
  );
}
```

Behavior after setting History on page change

- 😻 App can change views ("pages")
- 😻 URL DOES change on view change
- 😿 JS State resets on reload
    - Page State may not match URL

Back/Forward over pushed history entries

- 😻 Does NOT leave the app unexpectedly
- 😻 Does NOT cause a page load
- 😿 Doesn't yet change our page state

Setting page state on page load

- Read url and set page state!
- But when?
    - Easy option: First time App() renders
        - useEffect!
        - App WILL render "wrong" once
            - Consider if/how that is a problem

Modifying App.jsx

App.jsx

```jsx
function App() {  // Note: Don't use this!
  const [ page, setPage ] = useState(''); // set later

  useEffect( () => {
    setPage(document.location.pathname);
  }, []); // Important to have empty dependency array!

  return (
    <>
      <Header setPage={setPage}/>
      { page === '/' && <Home/> }
      { page === '/about' && <About/> }
      <Footer/>
    </>
  );
}
```

Common mistake!

`useEffect` should only be used for side effects

- Working with an **external** (to React) system
- Setting our own state isn't external!

App.jsx

```
function App() {
  const path = document.location.pathname;
  const [page, setPage] = useState(path);

  return (
    <>
      <Header setPage={setPage}/>
      { page === '/' && <Home/> }
      { page === '/about' && <About/> }
      <Footer/>
    </>
  );
}
```

After changes on page load

- 😻 App can change views ("pages")
- 😻 URL DOES change on view change
- 😻 JS State matches URL on load/reload

Back/Forward over pushed history entries

- 😻 Does NOT leave the app unexpectedly
- 😻 Does NOT cause a page load
- 😿 Doesn't yet change our page state

`popstate` event when Back/Forward

Back/Forward over pushed history entries

- Does NOT yet change state
- Will fire a `popstate` event
  - on `window`
  - `window` is not controlled by React

We need to add an eventListener to window

- `window` _is_ outside of React
- When? On Page Load
  - `useEffect()`

Adding `popstate` listener

```
function App() {
  const path = document.location.pathname;
  const [page, setPage] = useState(path);

  useEffect( () => {
    console.log('adding listener');
    window.addEventListener('popstate', () => {
      console.log('changing state');
      setPage(document.location.pathname);
    });
  }, []);

  return (
    <>
      <Header setPage={setPage}/>
      { page === '/' && <Home/> }
      { page === '/about' && <About/> }
      <Footer/>
    </>
  );
}
```

Listener is added twice

- Double the effect
- It "works"
    - But good practice to notice and fix
    - With **cleanup function**
- Removing event listeners is a bit weird
    - `.removeEventListener()`
    - With same value
    - Named handler callback

Cleanup popstate event listener

```
useEffect( () => {
  function handlePageLoad() {
    setPage(document.location.pathname);
  }

  console.log('adding listener');

  window.addEventListener('popstate', handlePageLoad);

  return () => {
    console.log('cleanup'); // Don't have in submitted code
    window.removeEventListener('popstate', handlePageLoad);
  }
}, []);
```

After adding popstate listener

- 😻 App can change views ("pages")
- 😻 URL DOES change on view change
- 😻 JS State matches URL on load/reload

Back/Forward over pushed history entries

- 😻 Does NOT leave the app unexpectedly
- 😻 Does NOT cause a page load
- 😻 DOES change our page state

DOES require server config to always load `index.html`

Hash-based Routing

- The urls for your app all use `#`
    - Often with a path-like string after it
    - Ex: `#/`, `#/about`, `#/privacy`
- Works like Path-based Routing
    - Does NOT require server configuration

Hash-based navigation

- COULD have normal links that use `#`
  - Those do not cause page loads
  - Automatically add to browser history
- But that causes problems later in process
  - How to update state on change
  - How to update state on Back/Forward
- Best to follow same process as Path-based routing
  - `preventDefault()` on navigation
  - Tell browser to add to history
  - Update state on `popstate`

How does using Hash-based URL work?

On Page load/`popstate`

- Use `document.location.hash`
  - Not `document.location.pathname`

On navigation

- Still `.preventDefault()`
- For links use `e.target.hash`
  - Not `e.target.pathname`

Notes about Hash-based Routing

- Easier to write for front-end 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

Generally Path-based Routing is "better"

- Sometimes, **like this course**, you use hash-based
- Because you don't control the server config

App.jsx for Hash-based Routing

```
function App() {
  const hash = document.location.hash;
  const [ page, setPage ] = useState(hash);

  useEffect( () => {
    function handlePageLoad() {
      setPage(document.location.hash || '#/'); //if no hash
    }

    window.addEventListener('popstate', handlePageLoad);
      setPage(document.location.hash);
    });

    return () => {
      window.removeEventListener('popstate', handlePageLoad);
    }
  }, []); // Important to have empty dependency array!

  return ( <> { /* same as path-based */ </> );
}
```

`Header.jsx` for Hash-based Routing Example

```
function Header({ setPage }) {
  function changePage(e) {
    e.preventDefault();
    window.history.pushState(null, '', e.target.hash);
    setPage(e.target.hash);
  }

  return (
    <header className="header">
      <a href="#/" onClick={ changePage } >Home</a>
      <a href="#/about" onClick={ changePage } >About</a>
    </header>
  );
}
```

These are just examples of the concepts!

You should be able to:

- Handle different links and pages
- Have other links and make them
    - Add to history stack and change browser URL
    - Update "page" state
- Have different navigation menu styles and HTML
- Have a form submit change the shown "page"
    - And add new "page" to history stack

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?