Dynamic Web Sites/Apps

- Server-side is a bit…tedious
    - Entire page returned for every request
    - Even if only one tiny bit changes
    - Users don't like "click-thru" pages
- Example: If a form field has a typo
    - Don't send a page with just message and link
    - Send back a page with the form again
        - With added warning message

Client-side Javascript

- Server still has to handle errors
    - But can be "gruff" about it
- Client-side JS can
    - Provide "instant" error messages
    - Prevent "useless" navigation

Our Client-side JS so far is limited

- Still has to submit data to server for new data
  - Whether a search or a state change
- Submitting data to server
  - Always a navigation (so far)

What if we could talk to server without navigating?

AJAX / Web Service calls

Client-side JS can make a web request

- Gets back response
- Does NOT navigate the page
- JS code can read and use response data

Response could have HTML

- But usually better to just have "data"

Web Service is still a web request/reponse

Request

- a url (with or without query params)
- headers
- possibly a body

Response

- status
- headers
- possibly a body

The only difference is how it is used

Terminology

endpoint - the url for a web service

- Still a normal URL

payload - The body of request/response

- For all web requests/responses
- Not just services

Service concepts

There are different concepts a service might use

- REST
- GraphQL
- SOAP
- etc

Will discuss more when we write services

Service HTTP Methods

A few things ARE different about service calls

Such as the HTTP Methods

- HTML forms are just GET/POST
- HTTP also supports more methods
- Service calls don't use HTML Forms
    - So can use these other methods

Request body encoding

Encoding of the request body can also be different

- HTML forms are url-encoded by-default
  - Can also be "multipart/form-data"
- Lots of other encodings exist
  - such as JSON, XML, YAML, etc
- Service calls often use these other encodings

Handling Response

Lastly, a major difference with making a service call:

- HTML forms lead to navigation
    - Browser shows a page
- CSS, images, JS
    - Browser handles applying these files
- Other file types (PDF, documents)
    - Browser downloads, displays, whatever

Service calls are up to the JS you write to handle

- No automatic Browser handling

Parsing

Most common case

- Service returns formatted data
    - JSON in most cases
        - Even when service isn't written in JS
- Your JS gets the parsed JSON
    - Decides what to do with the data
    - Hint: Update state
        - Rewrite HTML based on state

Errors

No browser default action also means

- No default handling of errors
- Your JS must make decisions
    - Inform user when necessary

Errors come in different styles

- Couldn't reach service
- Service didn't like request
- Service itself has error

Waiting

Making a request takes time

- Browser doesn't show like page navigation
- Do you show a user the app is waiting?
    - If so, how?

Coding locally will be fast

- Real service calls will be slower
    - Sometimes many calls
    - Client device
        - Could be low bandwidth
        - Could be low CPU

Once you start using services

- Server-generated HTML feels more tedious
- Page in browser gets more state
    - Distinct from server-based state
    - Can replace small or large chunks of HTML
        - As needed
- Changing page in browser becomes painful
    - Existing client-side state is lost
    - More (re)loading data from server

How to avoid this?

Single Page Applications (SPA)

Initial Load of an HTML Page

- May have full HTML contents
    - Might not
- All other requests to change/read server state
    - Done via services
    - Any HTML changes done by client-side JS

The "app" is now a "single page"

- But the contents look/behave like multiple pages
    - "screens" / "views" ?
    - No real terminology to distinguish 😭

Web Services does NOT require SPA!

- Just a common result
- Incentives to maintain client-side state

Benefits to a SPA

- Client-side state fully maintained
    - Persists between *all* requests
        - More than server-side
- Services focus on their purpose only
    - Not Presentation
    - Services have benefits anyway
- Front-end devs may ignore server side language
    - And Service devs may ignore JS
- Less network overhead
    - No redundant web content

Costs of a SPA

- Requires client-side JS
- Extra effort to handle Back buttons
- Extra effort to appear on web as multiple pages
- One big app means one big complexity
    - (More/less complex than alternative ??)
- More CPU/Memory requirements on client
- Client-side all "visible"

How much work is done on server for a SPA?

- Most SPAs generate HTML on client
    - Then replace (parts of) HTML on page
    - Extreme version is an empty HTML page
        - All content generated in Client-side JS
- Increasing interest in changing/reducing this
- More on this once we get to React
    - First we SPA without React