Topics for Review

- Bundles and separation of concerns
- Promises and `.then()`, `.catch()`
- `fetch()`, `response.ok`, `response.json()`
- Client-side state
- state and render cycle
    - Not using `.style`
    - Not hiding/showing blocks
    - Changing classes on render
    - Error messages and immediate forms
- Differences between client state and server state
- Polling

Separation of Concerns

What files to create?

- Model
    - Might be multiple parts
- View
    - Might be multiple parts
- Controller
    - Attach listeners?
    - Polling?
    - initial load?
    - Service wrappers?
- Some of those not MVC
    - But called by MVC parts

Deciding on your files

- Watch for coupling
    - Using values outside function
        - Pass most values instead
    - Passing many values
- Is coupling necessary?
    - Be passed values?
    - Can you return a value?
    - Accept a callback/return a promise?

How to start

- Start with rendering an initial view
    - Determines needed state
- I recommend separating files early
    - Not required, but may be easier
        - Enforces reduced coupling
- Make minimal changes
    - Keep code runnable
    - Confirm it works
        - Makes it easier to chase bugs

Promises

- Promise will resolve or reject
    - pending until then
- Resolve calls .then() callback
    - passes resolve value
    - passes resolve value if promise
- Reject calls .catch() callback
    - passes reject value
    - passes reject value if promise
- Any thrown error OR "crash" is rejected

Service Wrappers

- Call fetch
- Return Promise
    - Resolve with parsed data
    - Reject with formatted error
- Does NOT modify state
    - Separation of Concerns
    - Allows for reuse in different situations
- My examples are ONE WAY to do it
    - Not everyone follows this pattern
    - But is a good pattern

In Case of Network Errors

`fetch()` only rejects on failure to reach service

- Not on error returned by service
- We `.catch()` BEFORE checking service response
    - Guarantees cause is network error
- We reject with formatted error object
    - Allows all errors to be handled by similar parsing
    - This is a choice, not a technical requirement
    - But is a good pattern
    - returning rejected promise
        - Will skip later `.then()`
        - Goes to later `.catch()`
        - Which can be in calling function

Checking service response

- a `response` object might be
    - A successful response
    - An error
- We check `response.ok` to see
    - Requires accurate status code!
        - REST does, other conventions may not
- On error status
    - We parse body data as JSON!
        - Requires service returns JSON body!
            - We do, not all services will
    - We then reject with that error
        - Requires consistent service error body
            - We are, not all services will be

On Service success

- We return `response.json()`
    - provides promise of parsed results
- Service wrapper doesn't use!
    - Just resolve of returned promise
    - Calling function can use results

Reporting Errors

Remember - `console.log()`/`console.warn()` are NOT error handling

Need to tell user what they need to do

Rarely wish to show direct error message service gave to user

- usually want to "translate" server error (or network error) to friendly message
- See todo sample for example
    - Just one way to do it
    - Important thing is result

Client Side State

- Why do we have this?

Why State

- Need state to make decisions about what to show
- There is an IMPLICIT State in the DOM
    - What we are showing IS the state!
- Changing output alters the DOM
    - add/remove classes
    - add/remove HTML
- As our application gets more complex
    - DOM becomes more complex
- Any HTML change can change reading the DOM
    - 1 output change can cause N other changes
- **Without client state, app cannot scale**
    - As it grows, harder to make changes

jQuery

This was a critical lesson learned from jQuery

- jQuery doesn't require you store state in DOM
- But most people did, it was easy
    - Had no explicit separate state variables
- jQuery applications were easy and common
    - But often got harder to change and buggy as they got large
- This wasn't jQuery's fault!
- But later frameworks/libraries had explicit state
    - Avoided this issue

An app without separate state will fail to scale

- It will "work"
    - Fast and easy to write
    - First few changes easy
- But complex UI means state reads/changes become hard
    - Easy to make mistakes about
- A list where items:
    - Might be filtered from view
    - Might be selected/not selected
    - Can be on different visible "pages" of results
- How do you say how many items there are?
    - How many times did that code have to change as above features added?

How is Client State different from Server State?

- Often similar
- Client deals with 1 user
    - Server deals with many
- Client cares about what it shows
    - Server cares about changing data

Example: Project 2

Client and Server state identical/near identical

- Client shows messages and who they are from
- Server keeps a list of messages and who from

Client and Server state different

- Client was a list of active users
    - No duplicates
    - Should never know sid of other users
- Server tracks who is logged in by sid
    - Can have duplicate usernames
    - Won't have duplicate sids

Who translates data to state?

- Service will send data to client
    - Based on data from server state
        - May not be identical to server state
            - Ex: user list
- Client gets data from service call
    - May not save directly to client state
        - Depends on how the client uses data
        - Ex: A newly posted message by this user
            - May add message to message list
                - Esp. if a post id was generated
        - Ex: a partial list of messages
            - Will merge into existing message list

Client Side Render

- Why rewrite HTML vs show/hide blocks?

Show/Hide doesn't scale either

- It works, it just doesn't scale
    - Though not as bad as no state
- Multiple sections, have to decide what to show hide in reaction to user action
    - Add new action, set every section
    - Add new section, alter every action result
    - Add 1 X, change N things
    - Linear pain
        - Algebraic growth, not exponential

Compare to the proscribed `render()`

Rendering tree only cares about relevant state

- Generally 1:1 pain
- 1 new section = 1 change to render
- 1 new action = 0 changes to render

Adding/Changing a section

- Don't need to know all other sections
- Just know the output of this section
    - Less mental work
    - Less "cognitive overhead"
    - Easier to make changes
    - Fewer mistakes/bugs

But isn't all that rewriting of HTML inefficient?

Not to mention causing pain with things like scrolling and in-progress typing

- Yes, but computer efficiency is only one part of programming
  - Faster, more accurate app updates/changes/fixes are important too
- Yes, but tools like React will fix that

Polling

- How to create
- When to start/stop
- Dealing with errors

Creating Simple Polling

- Know what data you want to poll
    - This may need a new service!
- Know when you want to poll it
    - Be careful with intervals/timeouts
        - Can get out-of-order results
            - If a single service call is slow
                - Internet traffic can be weird
    - Best to start next call after first call finished

Using results of Polling

- Write immediately?
  - Can interrupt user!
  - Perhaps show a message
    - Without full-screen HTML rewrite
  - Let user decide when to update UI?
  - Gets easier with smarter HTML replacement
    - Like React

Starting/Stopping polling

- If service polled requires login
    - Best to start/stop on login/logout
    - Can still have 1 bad attempt
        - See Internet traffic is weird
- Save `timeoutId`/`intervalId`
    - Probably in state

Errors in Polling

Handling depends on error

- Network error?
- auth error?
- Other error?

User didn't make request

- Response might need to change
- But if response indicate user needs to take action
    - Why wait?