Reviewing Project1 Lessons

Key lessons:

- You cannot trust user data
- MVC is easiest solution to flow
- Thinking it through
- Software is hard to estimate!
- UI issues
- Generated Code
- Missed Requirements

You cannot trust user data

- cookies, url, params all user data
- It CAN be wrong
    - Mistakes
    - App changes
    - Deliberate

User Data comes in many forms

- Which url are they going to?
    - The order you intend can be bypassed
- Cookies & session id
    - May not be accurate/valid
    - Must check before changing data
    - Many students failed to handle/redirect
- Passed parameters
    - May not be the options you offered
    - Must allowlist

Easy to assume, but worth it not to

- We assume the intended/offered

But the issue isn't just "security"

- Defensive coding re: user data
    - Also better code in general
    - Easier to change
        - Fewer built in assumptions

Break out and name logic

- Usernames "dog" and "@!#$!@#" NOT the same!
    - One is not allowed, one is not valid
    - Many failed to distinguish
- Don't try to handle all in one check!
- Try `isValidUsername()` and `isPermittedUser()`
    - Or `isValidPassword` (No actual password)
        - Treating dog as bad password
- 400 - Client Error (generic)
- 401 - User Needs to Login
- 403 - User Is Logged in, not allowed
    - We used with "dog"

Web apps rapidly become complex

- Deal with most recent data
- Deal with data spanning multiple requests

How do we keep the code clean?

- Understandable
- Easy to change

Common issues

- All code in one pile (server.js)
    - Need to break up logic into "building blocks"
- Code broken up poorly
    - "Building blocks" not actual decoupled
    - Compare:
        - Textbook broken up by topic
        - Textbook broken up every 30 pages
- Error messages in controller
    - Set *state* in controller
    - Pass state to view
    - View shows error message for that state

Separate the state and the view

Remember the MVC Pattern?

- Good lessons on separating concerns
    - Even when not explicitly using MVC

Have a clear data model (state)

- Based on request data
    - Update state
- Render new view
    - Based on state

When writing a route handler

Ask yourself

- "What is the user asking to do?"
- "What state changes does this make?"

Write your code to answer those questions

THEN:

- Write returned view
    - View may have conditionals on state
    - View should never CHANGE state

Actual general programming advice

None of those points are specific to web

- General best practices
- Web just forces you to handle earlier
    - Client-Server
    - Stateless request-response
    - Multiple users asynchronously requesting
    - Poor separation of concerns hits you faster

Write and confirm in small chunks!

- MANY students not doing this!
    - Makes it harder to find bugs
    - Makes it harder to fix bugs
    - Have to rewrite a lot
        - Taking even MORE time

Write and confirm small bits

- Pieces you EXPECT not to have issues!
- Quick to do
- But most mistakes START small

Remove State Code Before Submission

- Remove debugging console.log
- Remove commented out code

Employer/Team won't want this!

Did Project1 take longer to write than expected?

Important Lesson

- Software always more time than you think
- Because it is all about abstraction
    - Describing a lot of work in a quick summary
- Never gets easier

Remember this lesson for future

- Bosses/Clients/Team want info ASAP
- You will be asked for timelines
    - ALWAYS allow extra time!
        - Not lying, you *know* your gut is wrong
    - Avoid guessing; Break problems down
    - More on this at end of semester
- This was even worse than you think
    - compare code was pre-written
    - Login code was pre-written

Project 2 will be more work!

Can you complete Project 2 without

- Extensions?
- Panic?
- Submitting work that isn't yours?

If not, how can you be sure about Final?

Final Project has no extensions

I have a grading deadline that can't be pushed back

- Final project is custom
    - No pre-written parts
- Final project has more parts
    - REST Services
    - React code
    - CSS

Don't treat project like a weekly assignment

- Start early, alongside assignments

Common HTML/CSS/UI Issues

- Logout button
- Labels not associated
- Styling element types
- Inline Styles
- Error Output

Logout button issues

- Logout button at end of form
    - Have you ever seen this?
    - Boss/Team will expect you to be "normal"
- FYI, you can style a button to look like a link!
    - **https://www.ratemyprofessors.com/**

Form Issues

- Labels were not associated
    - Label must have `for`
    - Field must have matching `id`
    - Easy to test: Click on label
        - Why didn't you test?

Label:

Inline Styles

- Don't use inline styles

```
<button style="background-color: lime;">
  DO NOT DO THIS
</button>
```

- Hard to edit
- Hard to maintain
- Impossible to reuse

"Works" doesn't make it "Good"

Don't style element types unless being generic

Bad:

```
form { /* Affects all forms, including any added in future */
    border: 1px solid black;
}

ul li a { /* All future lists, and also: what is this? */
    /*...*/
}
```

Fine:

```
p { /* Default for all paragraphs */
  line-height: 1.6;
}
```

Better:

```
.change-form { /*...*/ } /* Only this form! */
.nav-items a { /*...*/ } /* We know what and why!*/
```

Error Output

- Errors are views
- Should have full HTML
- Can pass in state to view
- View can decide exact language to display
    - Keep all "UI" in the View

Missed Requirements

- HTTP Status codes requirements were explicit
- Some messages were explicit
- Handling invalid sessions was explicit
- Associated Labels were explicit

Why were these missed?

README had MANY Requirements

- Absolutely *painful* to meet them all
- Work will have this many requirements
    - Often implicit (like "logout" button)
    - You will need to ask questions
- You will need to manage many requirements
    - "It's done! 2/3rds of what you requested!"
- Treat assignments as practice
    - And Projects as proof