# Flexbox; client/server architecture

UNC COMP 523

Wed Sep 16, 2020

Prof. Jeff Terrell

# Announcements

- music: Everlong, by The Foo Fighters, just 'cuz I like it (the song and the video) :-)
- grades are up-to-date as of yesterday afternoon

# Outline

- Flexbox, part 2
  - review
  - Flex sizing
- Grid layout (briefly)
- Responsive layout (briefly)
- Client/server architecture

# Flexbox review from last time

- Element with flex display property is *flex container*
- Direct children are *flex items*
- Enables flexible dimensions in one primary dimension, called the *main axis*
- Main axis controlled with `flex-direction` CSS property
- Main axis alignment controlled with `justify-content` CSS property
- Perpendicular axis is called the *cross axis*
- Cross axis alignment controlled with the `align-items` CSS property
- Demos for flex direction and flex alignment

# Flex basis

- *Flex basis* is how much *initial* space to give a flex item along the main axis
- The item's final size is also affected by growing and shrinking properties
- CSS property: `flex-basis`
- Valid values:
  - `auto` (default)
  - a size, like `100px` or `2rem`

# Flex grow

- Items may grow to fill the available space
- Disable this behavior with `flex-grow: 0` style applied to the item
- Enable it with a positive number for `flex-grow`
- What does the number mean?

# Available space

- First, determine initial size of all items from `flex-basis` values
- Then, there may be some space left over in the flex container
- Divide this up fractionally based on `flex-grow` values of items
- Example: values of 2, 1, 1 means that the first item gets twice the space of the other two

# Flex shrink

- Similarly, `flex-shrink` will shrink items if the main axis overflows
- Set `flex-shrink: 0;` to disable shrinking
- Shrinking takes into account the minimum size of items so may seem broken or inconsistent

# Sizing shorthand

- Combination of `flex-grow`, `flex-shrink` and `flex-basis` properties
- CSS: `flex: 1 1 auto`
- Special values:
  - `initial` is equivalent to `0 1 auto`
  - `auto` is equivalent to `1 1 auto`
  - `none` is equivalent to `0 0 auto`
  - a number, e.g. `1` is equivalent to `1 1 0`

# Demo

Demo 10: Flex Sizing

- set special/others to grow
- control relative ratios of growth by using different numbers
- note that final width of "One" and "Three" are different because of flex basis
- set p width to 200px and demonstrate flex shrink
- but can't shrink elements beyond their minimum width

# Flexbox Froggy

http://flexboxfroggy.com/

# Layout outline

- ~~Display types (e.g. block, inline, float)~~
- ~~Positioning elements (e.g. relative, absolute, fixed)~~
- ~~Flexbox layout~~
- **Grid layout**
- Responsive design

# Grid layout

- Like flexbox, but for flexible layout in *two* dimensions
- Sophisticated and flexible, but not difficult!
- Supported in all major browsers: https://caniuse.com/#feat=css-grid
- MDN overview
- Tutorial game: CSS Grid Garden

# Layout outline

- ~~Display types (e.g. block, inline, float)~~
- ~~Positioning elements (e.g. relative, absolute, fixed)~~
- ~~Flexbox layout~~
- ~~Grid layout~~
- **Responsive design**

# Responsive web design

- Goal: let your site look good on any sized screen
- An incantation:
  - `<meta name="viewport" content="width=device-width, initial-scale=1" />`
- Design for mobile first (because really basic phones don't support media queries)
- Then add media queries to use more horizontal space on bigger screens
- Turn on responsive design mode in your browser dev tools
- Example: The App Lab web site

# Poll on project sentiment

# Client/Server Architecture (CSA)

# Course context

- I intended to talk about infrastructure today, then realized this would be important background
- One student mentioned in feedback from last lecture that this would be helpful content
- This is useful for almost all projects for A8: walking skeleton

# Target audience

- Like with layout content, some will find this more useful than others
- If you haven't taken COMP 426 and aren't clear on what "frontend" and "backend" mean, this will be important for your project
- Otherwise, you may find the high-level overview clarifying and/or a helpful review

# CSA outline

1. Motivation: general app requirements and commonalities
2. The client/server architecture
3. The client/server interface, a.k.a. "the" API
4. HTTP for APIs
5. Implementation and debugging tips

# App commonalities

- All apps must do HCI: human-computer interface
  - output: show screens and information to users
  - input: handle events initiated by users
- This HCI requires code running on user devices
- Note: most apps are intended to run on multiple devices

# App commonalities: data needs

- Many apps want to provide a similar experience for the same user regardless of device
- And/or want to provide continuity in case of a lost or broken device
- And/or need to access data generated by other users
- How must this work? With data stored off-device, e.g. "in the cloud"
- So code on devices must talk to some other computer and fetch relevant data
- This is the basis of the client/server architecture

# CSA outline

1. ~~Motivation: general app requirements and commonalities~~
2. **The client/server architecture**
3. The client/server interface, a.k.a. "the" API
4. HTTP for APIs
5. Implementation and debugging tips

# Client/server architecture

- The client, or **frontend**, is the code running on a user's device
- The server, or **backend**, is the code running on a server somewhere
- Frontend code runs only when being used; backend code runs all the time (a "service")
- Frontend code runs on many devices; backend code runs on one (or a small number)
- Frontend code runs on untrusted devices; backend code runs on trusted devices

# The backend

The responsibilities of the backend include:

- Be available (at a known location) 24/7
- Identify the endpoint being requested
- Check authentication of the request: is a known user sending this request?
- Check authorization of the request: does the user have the needed access?
- Check the validity of the content type and payload, if any
- Apply other policies defined for this app
- Access the database to service approved requests

So **the backend is the gatekeeper for the central database**.

**Do all apps need a frontend?**

**What about a backend?**

# Necessity of frontend and backend

- Assuming "apps" necessarily interact with users, all apps have a frontend (though maybe a simple one)
- But not all apps need a backend
- Remember criteria necessitating a backend:
  - need to provide a similar experience for the same user regardless of device,
  - need to provide continuity in case of a lost or broken device, or
  - need to access data generated by other users
- Example: a calculator app doesn't need a backend
- Example: a photos app doesn't *need* a backend, but might have one anyway to back up photos

# CSA outline

1. ~~Motivation: general app requirements and commonalities~~
2. ~~The client/server architecture~~
3. **The client/server interface, a.k.a. "the" API**
4. HTTP for APIs
5. Implementation and debugging tips

# Client/server interface, a.k.a. "the" API

- Any interface consumed by a program is an API, or application programming interface
- But the interface between the frontend is often referred to as "the" API in C/S apps
- How does this work? The API has multiple **endpoints**. Each endpoint does a particular thing.
- Examples: authenticate a user, fetch user's feed, follow another user, create a new post, etc.
- Typically, most requests must be authenticated and carry a session identifier in a cookie, which the browser or frontend attaches to every request

# CSA outline

1. ~~Motivation: general app requirements and commonalities~~
2. ~~The client/server architecture~~
3. ~~The client/server interface, a.k.a. "the" API~~
4. **HTTP for APIs**
5. Implementation and debugging tips

# HTTP for APIs

- Most APIs use HTTP, the hyper-text transfer protocol, originally created to carry HTML data
- Each API endpoint defined by a unique combination of `(method, path)`
- HTTP method can be `{GET, POST, PUT, DELETE, ...}`
- HTTP path examples: `/login.html`, `/posts/`, or `/post/123`
- HTTP responses include a status code and optionally some data (interpreted by a content type)
- HTTP status levels: 200s imply success; 300s imply redirection, 400s imply client errors, 500s imply server errors
- HTTP status code examples: 200 (OK), 201 (Created), 204 (No Content), 403 (Forbidden), 404 (Not Found), 418 (I'm a teapot), and 500 (Internal Server Error)

# HTTP demo

- JSONPlaceholder
- Open browser's dev tools, Network tab, and click "Try it"
- See HTTP request and response details for the latest request

# CSA outline

1. ~~Motivation: general app requirements and commonalities~~
2. ~~The client/server architecture~~
3. ~~The client/server interface, a.k.a. "the" API~~
4. ~~HTTP for APIs~~
5. **Implementation and debugging tips**

# Defining an API

- If you define an API first, then you can implement the frontend and backend in parallel
- To define an API, specify for each endpoint:
  - method
  - path (might be a string or a pattern, e.g. `/post/:id`)
  - whether authentication is required
  - request data and content type (if any)
  - response conditions (e.g. not found vs. found, authorized vs. not)
  - for each condition:
    - status code
    - response data and content type (if any)
- Also specify how authentication works

# Implementation and debugging tips

- Let your clickable prototype drive your API; your API should support only the known use cases, not theoretical ones
- Say something API related isn't working in your app. Debugging suggestions:
  - Modify client code to fetch something from/post something to a known working API, e.g. JSONPlaceholder.com
  - Test backend code by using a known working client to send requests, e.g. a browser, Postman (desktop app), or a CLI tool like curl or httpie
  - (Then pay attention to response details)
  - Ensure that you can see full exception information on backend, either in HTTP response or from server process output
  - If a particular request triggers a backend exception, try using browser's network devtools to copy the request as a `curl` command for easy replay

# Poll: git comfort