Javascript: Storage and State

Use Cases for Storage in a Browser

1. You want a mobile device to work in disconnected mode
2. You want to maintain conversational state
3. You want to remember where a user is in the app

- The 1st is relevant to this discussion, but also often involves the File API and/or the AppCache (things we will not cover in this class)
- The 2nd is a long held problem (since the first days of the dynamic web) for both server-side and client-side application development which we discussed already
- The 3rd is a step toward creating apps that more and more resemble what we are used to doing in a desktop or mobile application

Your design choices for storage
(persistent vs. non-persistent, client vs. server, etc.)
need to match the use case / type of stateful semantics
your application requires!

If the design does not meet the requirements & only those requirements,
you may have functional, performance, scalability, and security defects!

Javascript and Cookies

- We have already discussed cookies on the server
  - The server requests they be set
  - The client has to accept the cookie and send back on each request
- The document object has a cookie property
  - So, you read, write, and update it like any other property
  - It is multi-valued – several key-value pairs stored in a single cookie string delimited by semicolons
- So, just like the server-side, in a JS syntax of course
  - You can write/read values in a self-contained app
  - Main limitations are
    - Storage size
    - Multi-values as delimited strings
    - Marshaling state in a string-based format
    - Potential privacy concerns
    - They just feel like a hack
    - Make sure you expire them!

See http://www.w3schools.com/js/js_cookies.asp

New HTML5 Features : Deep-dive into Storage

Storage API

```
session[local]Storage.setItem(key, value)
session[local]Storage.getItem(key)
session[local]Storage.clear()
```

That's it, Any questions?

Actually…

1. What is the difference between them? *How long data lasts*
2. Does session here mean what it meant on the server? *NO*
3. Can I access like an associative array? *Yes, [] and '.' supported*
4. Is it secure/private? *Data is managed per subdomain, but still the data is on the device with no guaranteed encryption, etc.*
5. What is the capacity? *5MB in the spec but your mileage may vary*
6. What happens if the OS swaps out my app? *Sucks being you*

Web app developers on the server and on the client have historically been very poor at managing state

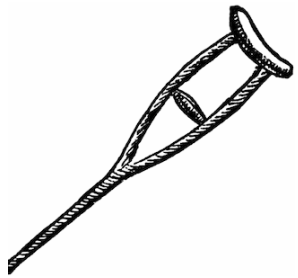
What about SQLite, WebSQL and IndexedDB?

Databases provide certain expected features:

- Understanding – the ability to manage lots of data in an organized way (according to an understood data model)
- Scale – Object/record management, indexing
- Querying – use of declarative or navigational queries

I struggle with the idea of a real need for this on clients

- Seems like a bit of a crutch
- But, as mobile devices get even more powerful…



A historically fragmented and convoluted space

- Most vendors started by supporting SQLlite
  - Kind of an “embedded database” approach
- Early standards efforts came up with WebSQL
  - So tied to SQLite it was considered useless as a standard
  - Firefox and IE decided not to support it
- IndexedDB is the latest standards effort (candidate), more open
  - But isn’t localStorage enough for most app’s needs?

Summary: Storage and State

Rich user interactions are *stateful* - The thing the user does is in fact dependent on things s/he has done before.

- If “before” == “a previous execution of the app” then use a permanent storage mechanism
  - Think of how your desktop OS tends to remember what you did the last time you used it
  - FileStorage, a web db, local storage
- If “before” == “a bounded interaction to accomplish some task in the immediate term” then use a session mechanism
  - This is conceptually referred to as *conversational state*
    - Programmatically we use the term “session”
  - *Bounded* means you should take precautions to ensure that once the end user goal is achieved the conversational state goes away
- If “before” means only those things the user has done “recently” (in this instance of the browser), use *session storage*.
  - “session” != conversational state as in Express middleware
  - “session” == instance of the browser (the runtime process lifecycle)
    - Like the way we talked about “session cookies”

Checkpoint: Where are we now?

We've talked about web app evolution from different angles:

- Web app architecture styles (6 of them)
- "thin client" (server-driven) vs. "fat client" (browser-based)
- "Page turning" (action-oriented) vs. "desktop model" (component-driven, with event handlers, like a GUI)

- We've discussed the DOM and events we can listen for an take action against on the DOM
  - We can write rich GUI-style apps on a single page load
  - "Single page apps" (SPAs) as they are known are like desktop apps; you load state and listen to events to change UI and model state
- As in desktop apps, we can leverage storage mechanisms to provide state (across process boundaries) to our apps
  - This talk on storage
- So do we need a server anymore? What happened to MVC?
  - Let's move on and talk about storage, AJAX, and then MVC/P