- what else can I fuck around w/ during this final half hour? Perhaps Project Management literature?

- Man, at this point I kinda feel like I'm just burning through time → not really making solid progress. Is there a way to get 20 minutes more in to make my time good? Or should I just throw in the towel?? →→ I think I'm calling it a day!...

2017 | 09 | 11 MONDAY

[index.js]

```
definitions and metadata
  var express = require('express')
  var app = express();
  var router = app;
  app.use(bodyParser.json());

router.get('/form', function(req,res) {
  :
  :
  :
});

router.post('/formpost', function(req,res) {
  :
  :
  :
});

app.listen(3000, function() {
    console.log('Example app listening on port 3000!'),
});
```

- fax office page looks @ white board registry, sees Janey's in charge of handling router.get('/'...) today, then hands her info from fax...

- My code:
  - "part of Express"
  - "separate layer on top of Express"
  - "software @ runs w/ Express"
  - → populated offices down the hall (Janey / get('/'...) ; Steve...) w/ people, practices, and protocols

- My software offices are down the hall from the offices @ sit on top of fax machine (Express)
  - → my people don't know as much about how to use fax machine, which is why we've moved into an office complex w/ people who do know how to use fax machine/ mail hub (Express)

my software:
- dispatch someone from each of my offices to the fax room to (staff have Express) write on the whiteboard (registry): "come and contact Janey (router.get('/'...)) in this event"; "come and contact Steve in this other event"

- client → user
- lawfirm → browser
- govt office → Server
- fax (Express...)
- Janey's office (get('/'...)) room 217
- Steve's office (another router.get() or router.post())
- protocol → Express (rip off fax, scan metadata from coversheet
- pages → part of Express
- registry → of who handles what sort of request inside govt office
  - → white board: lists who's working today and what kind of request they can respond to

- URL registry
- backbone / one level of index.js file
  - when I say → router.get('/'...) → registering a URL handler (@'s what .get() handler does)

- (Express is software running on Server → people, practices, & protocols @ work in govt office)

router.get('/', function(req, res) {
  ...
});

① ② ③

→ first argument: string ① describes URL
→ second arg: function ② knows how to communicate over a certain protocol: receives 'request' object and 'response' object
→ unboxed 'req' and preparing-to-be-boxed 'response' object
• ~~① received~~ from web browser, ② we opened up
• we've got to respond to ②.
• we've got a prototyped, almost finished 'response' object ② were almost finished putting things into & then tape up to send off.

• INTERFACE TO BLACK BOX
• PROTOCOL

• Express people don't know Janey or Steven, all they know is ② they need to deliver the fax w/ the described request, and then deliver back the finished box → That's all Express has to do.

• Janey knows how to bridge the gap btwn reading the fax and filling up the box to be sent back out. → my function ② I write, what I register against the '/' URL.

==

• Janey and Steve walk down the hall to Fax Office (Express)
• J & S tell ~~Ex~~ Fax Staffperson what to write on whiteboard ('if '/', contact Janey, etc)
• I run a handful of those requests (J&S go to Fax office to request update to whiteboard) every time I run index.js on Server (node ./routes/index.js)

↳ when I first run index.js
① unpack some 'metadata' boxes in the whole office area: make sure lights turn on, phone lines are set up, etc.
② Intern/admin from Janey & Steven's office walk down the hall to Fax office/Mail Room to tell their staff what to write on whiteboard.

→ Janey & Steven are one-off functions ② appear w/in ~~rote~~ router.get, router.post (Registry) calls → (the contents w/in ~~functi~~ function(req, res) { }); ← Janey or Steven

• Important fact: router.get or router.post associates function to URL (via registry)
  • ②'s all it does, just creates association.
  • The association is remembered in Express, and used in the future when Express gets a call for someone who can handle a particular URL: it can call the correct function b/c it's been told in the past & it remembers. (memory is registry)
• Router.get / Router.post happens one time (and one time only) after index.js is called and run on server.

↑ tangential to cookies, lay the groundwork ↑

- Client walks into lawer's office: I need to ~~talk to~~ info from govt.
- Lawyer ~~puts out~~ [prepares] standardized fax cover sheet, then faxes to govt office

- fax rec'd @ govt office by staff trained by Express; scan coversheet, notice it's a request for '/' URL, looks up @ Express-internally-managed registry: send this fax to Janey's office. Janey's office staff performes the tasks w/in function, finally, when all done, Janey's office ~~#~~ standard form for response & fills out details
  - ↳ Standard form == index.ejs : the web page looks the same every time except for contents w/in <% %> brackets, which ~~change~~ is filled in based on certain logic; ~~#~~ values of logic depend on results of function w/in router.get('/'), as well as logic formed by values on static form (eg, new table row on index.ejs file).

- ~~When~~ all, send box back to Express office. When done, some clerk in Express office does the job of ~~to res.render~~ ~~unpacking data and sending~~ copying data from res.render(box) and sending it to ~~#~~.ejs file. Then sticks it in fax machine and hits send
- Fax gets sent back to lawyer! Lawyer rips off fax and gives to client: it's a beautiful graph of time/temp entries!

COOKIES [Lawyer is web browser]

Lawyer/web browser saves loads of temporary data about ~~their~~ their interaction w/ particular govt office/web site.
- Cookie exists on both sides of fax transaction: on cover sheet of both sides → Cookie doesn't have meaning unless it's stored on both sides of transaction
  - → Lawer stores card (cookie key) w/in client file and use every time interact w/ govt office/server site.
  - → identifies a particular conversation

===

- Instead of making user login every time, server sends browser cookie key to save and use each time web browser communicates w/ web server → ~~#~~ static, saved key works every time browser communicates w/ server, and browser can then have access to other data saved on server side.
- Lawer/browser automatically saves cookie key in browser cache.

09:00

cookie key
- Secret ⊕ browser's storing wasn't generated by browser, it was ~~gener~~ generated and sent by server.

- server needs to have another registry somewhere ⊕ remembers cookie keys → doesn't live in Janey's office, not in Steven's office, not in Express fax/mail room.

(authentication protocol) Somewhere there's a permanent records office in server / gov't dept → and they'll keep records on behalf of everyone. Janey and Steven and everyone can communicate w/ permanent records directly.

- Janey's office needs to know ⊕ communication w/ Firefox & Sons Lawfirm (browser) is safe: makes an internal check to permanent records (w/in Server) to verify authentication & to gain access to browser-specific data and history.

- It's up to me to build system for permanent records ⊕ Janey and Steven can access. → DB Table.
  → ~~common to make record Storage as general as~~ possible

id	cookey_key	session_data
01	Key:Value	Key:Value, Key:Value, Key:Value

- db table of all known cookies:
  keys are cookie values.
  values are session data — 2-layer dictionary

id	cookie-key : session_data
~~key~~ ~~data~~ ~~data~~	
01	cookie_key: {{session_key: session_data}, {session_key: session_data}}

{user_name: chauncy, last_login: 2017-Aug-22 05:17}

high level (object)

15:30

things ⊕ my URL handlers would like to know

~~to~~
- clear browser cache → browser forgets cookie key
- ⊖ log out of website → Server forgets cookie key.
  - don't delete any records, just invalidate data for future use.

- my setup: each office in govt bldg to validate cookie before doing anything
  ↳ (except for log in page -- should ⊖ get to ⊖ point)
  ↳ extract out as separate function, then call function for each URL handler.

global war: no asynchronous
db dive: asynchronous callbacks.

4

29:23