# Sessions

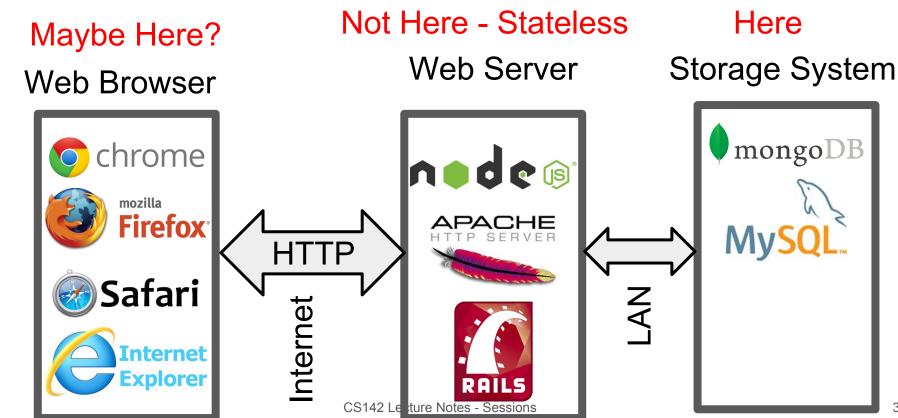
Mendel Rosenblum

# How do we know what user sent request?

- Would like to authenticate user and have that information available each time we process a request.
- More generally web apps would like to keep state per active browser
  - Called session state
- Concretely:

```
expressApp.get('/user/:user_id', function (httpRequest, httpResponse) ...
   // Need to make a decision to accept the request or reject it
   var sessionState = GetSessionState(httpRequest);
```

### Where could we get the session state from?



### Session state lookup problem

- HTTP request just come into a web server
  - Not a lot information to uniquely identify "session"
- Solution: Include something in the request to tells us the session
  - Care must taken to avoid forgeries
- Early HTTP solution: Cookies
  - State set by web server that browser attaches to every request
  - Useful but with a checkered history
- Modern browser support local storage API

### HTTP Cookies: Basic Idea

Web server adds Set-Cookie: to HTTP response header

```
Set-Cookie: cookie_name1=cookie_value1

Set-Cookie: cookie_name2=cookie_value2; expires=Sun, 16 Jul 2016 06:23:41 GMT

Each cookie is just a name-value pair.
```

• Future requests from browser to same server should include the Cookie: header

```
Cookie: cookie_name1=cookie_value1; cookie_name2=cookie_value2
```

### Cookie contents

- Cookie: name and data
  - Domain for this cookie: server, port (optional), URL prefix (optional)
  - The cookie is only included in requests matching its domain
  - Expiration date: browser can delete old cookies

#### Limits:

- Data size limited by browsers (typically < 4 KB)</li>
- Browsers limit the number of cookies per server (around 50)

# Cookies as web app storage

- User can:
  - View cookies
  - Modify/corrupt cookies
  - Delete cookies
  - Create cookies
  - Lose cookies to hackers
- Simply switching browsers looks like you deleted the app's cookies
  - Cookies have been used in bad ways (more later in class): Users are suspicious of them
- Pretty unreliable web app storage
  - Limited to hint, shortcut, etc. that can be recovered if missing
  - While actively communicating with web app: **Session cookies**

### Session state with cookies

- Early web frameworks (e.g. Rails) supported storing session state in cookies
  - Rails provided session, a JavaScript-like object, that you could store anything
    session[:user id] = "mendel"
- Rails packaged session into a cookie and added to HTTP response
  - o Data will be available in all future requests from the same browser
- Rails automatically checks for a session cookie at the start of each request:
  - Cookie exists? use it to find session data
  - No cookie? Create new session, new cookie
- End of each request: save session data where it can be found by future requests. (where?)

### Session state in cookies

- Early approach: Store session state in cookie
  - Since cookies can be viewed, changed, deleted, stolen, etc. care must be taken. Example:
    - session.user\_id = "mendel";
    - session.password = "foobar";
  - Using cryptography you can:
    - Hide content from viewers, hackers
    - Detect forgeries and changes
    - Can't do much about deletions
- An alternative is to put a pointer to the session state in the cookie:

```
Set-Cookie: session=0x4137fd6a; Expires=Wed, 09 Jun 2012 10:18:14 GMT Less transfer overhead but still need to protect with cryptography
```

# Options for storing session state

- Web server's memory
  - Fastest access
  - May be too large (many active users)
  - Makes load balancing across web servers hard

#### Storage system

- Easily shared across all the web servers
- May be overkill: Don't need the super reliability of storage system
- May be too much load for the storage system

#### Specialized storage system

- Support fast fetching of small, short-lived data
- Example: memcache, redis in memory key-value stores

### var session = require('express-session');

- ExpressJS has a middleware layer for dealing with the session state
  - Stores a sessionID safely in a cookie
  - Store session state in a session state store
  - Like Rails, handles creation and fetching of session state for your request handlers

#### Usage:

```
app.use(session({secret: 'badSecret'}));
    secret is used to cryptographically sign the sessionID cookie
app.get('/user/:user_id', function (httpRequest, httpResponse) ...
    httpRequest.session is an object you can read or write
```

### Express session usage example

- Login handler route can store into httpRequest.session.user\_id
- All other handlers read httpRequest.session.user\_id
  - If not set error or redirect to login page
  - Otherwise we know who is logged in
- Can put other per-session state in httpRequest.session
- On logged out you will want to destroy the session

```
httpRequest.session.destroy(function (err) { } );
```

# Express Session: Session Store

- Default session store is in the Node.js memory
  - OK for development but not production
- Has session store backends for many storage systems
- Hooking up to MongoDB via Mongoose

```
var MongoStore = require('connect-mongo')(express);
expressApp.use(session({
    store: new MongoStore({ mongooseConnection: mongoose.connection })
}));
```

# Cookie replacement: Web Storage API

- sessionStorage Per origin storage available when page is open
- localStorage Per origin storage with longer lifetime
- Standard key-value interface:

```
localStorage.appSetting = 'Anything';
localStorage.setItem('appSetting', 'Anything');
sessionStorage['app2Setting'] = 2;
```

Limited space (~10MB) and similar reliability issues to cookies