Announcements

- Maps grading: be timely, you’ll asked to reschedule if you’re late.
- Term Project: 4-way Checkpoint on Monday
- It’s all about your project now.
Web apps (thinking about multiple users)

John Jannotti

/course/cs0320/www/docs/lectures/

Apr 17, 2018
Maybe you don’t need this lecture

- Haven’t you have been making web apps all semester?
  - Your front ends are HTML, CSS, JavaScript
  - You could access them from any browser.
  - You’ve used forms to get input.
  - You’ve used AJAX to update them dynamically.
  - (Maybe) You’ve developed an intuition for “REST” APIs.

There’s a lot left (that might matter for projects)
- Your apps have no notion of a persistent user.
- They haven’t even needed a notion of session.
- You’ve been assuming every request came from the same user.
- Think about how you’d write a chat app, for example.
Maybe you don’t need this lecture

Haven’t you have been making web apps all semester?
  ▶ Your front ends are HTML, CSS, JavaScript
  ▶ You could access them from any browser.
  ▶ You’ve used forms to get input.
  ▶ You’ve used AJAX to update them dynamically.
  ▶ (Maybe) You’ve developed an intuition for “REST” APIs.

There’s a lot left (that might matter for projects)
  ▶ Your apps have no notion of a persistent user.
  ▶ They haven’t even needed a notion of session.
  ▶ You’ve been assuming every request came from the same user.
  ▶ Think about how you’d write a chat app, for example.
import java.io.*;
import java.net.*;

public class GetUrl {
    public static void main(String[] args) throws IOException {
        URL url = new URL(args[0]);
        try (InputStream is = url.openStream()) {
            // Read from 'is'
        }
    }
}
What’s going on?

- The URL class parses a string into a hostname and path
- The hostname is converted to an IP Address
- The OS is asked to create a socket
- The TCP socket is connected to the IP Address (and port)
- The HTTP protocol request the path.
- The openStream() method provides access to what’s returned.
What is an HTTP URL?

http: <host> [:port] / <path> [ ? <query> ] [ # <fragment> ]

- http is the scheme name, other urls exist.
  - https, ftp, telnet, mailto, etc.
  - These name *protocols*.
- host is a *domain name* or an *IP address*
  - www.amazon.com
  - 128.148.32.110
- port is a machine-local address, HTTP default is 80.
  - If an IP address is a street address, the port is a room number.
  - Allows multiple servers to run on one machine.
- The rest is opaque (for now).
TCP Sockets: two-way reliable channels

- TCP Sockets abstract over a lot of network machinery.
  - Packetization
  - Routing
  - Reliability (drops, reordering)
  - Flow control, congestion control
  - cs168 explains it all.

- Easiest to think of two kinds (particularly in Java)
  - Socket: Basic client socket, used to connect to servers.
  - ServerSocket: A special socket that waits for clients.
```java
try (Socket sock = new Socket(args[0], 80);
     PrintWriter out = new PrintWriter(sock.getOutputStream(), true);
     BufferedReader in = new BufferedReader((Socket sock.getInputStream())))
{
    out.println("GET " + args[1] + " HTTP/1.0");
    out.println("Host: " + args[0]);
    out.println(""");

    String line;
    while ((line = in.readLine()) != null) {
        System.out.println(line);
    }
}
```
HTTP Requests: Headers + Contents

- Header starts with HTTP line
  - GET <url> HTTP/1.1
  - POST <url> HTTP/1.1
  - Then has a set of name-value pairs
    - Content-type: text/html
    - Content-length: 20393
    - User-Agent: Mozilla/3.2
  - Then has a blank line
  - Then has any content

- Request specifies length
  - Or text itself can be length, data elements
HTTP Responses

- Headers starts with status line
  - 1xx: OK
  - 2xx: Success
  - 3xx: Redirection
  - 4xx: Client error
  - 5xx: Server error

- Name-value pairs (as before)

- Blank line

- Content
And what about servers...

How does Amazon’s web server connect with web browsers?

- `new Socket("www.amazon.com", 80)`?
- Servers have special needs.
  - They do not know who they will connect with.
  - They must accept multiple connections.
A (too simple) server

```java
ServerSocket serverSocket = new ServerSocket(5555);
while (true) {
    Socket clientSocket = serverSocket.accept();

    out = new PrintWriter(sock.getOutputStream(), true);
in = new BufferedReader(...(sock.getInputStream()));

    String line;
    while ((line = in.readLine()) != null)
        out.println(line);

    out.close(); in.close(); clientSocket.close();
}
```
A better server

```java
ServerSocket serverSocket = new ServerSocket(5555);  
while (true) {
    Socket clientSocket = serverSocket.accept();
    Thread t = 
        new Thread(new ConnectionHandler(clientSocket));
    t.start();
}
```
ConnectionHandler implements Runnable

```java
public class ConnectionHandler implements Runnable {
    private Socket s;

    public ConnectionHandler(Socket s) { this.s = s; }

    public void run() {
        out = new PrintWriter(s.getOutputStream(), true);
        in = new BufferedReader(...(s.getInputStream()));

        String line;
        while ((line = in.readLine()) != null)
            out.println(line);

        out.close(); in.close(); s.close();
    }
}
```
Better to use a threadpool

```java
Executor pool = new ThreadPoolExecutor(
    poolSize, maxPoolSize,
    keepAliveTime, TimeUnit.SECONDS,
    new LinkedBlockingQueue());
ServerSocket serverSocket = new ServerSocket(5555);

while (true) {
    Socket clientSocket = serverSocket.accept();
    pool.execute(new ConnectionHandler(clientSocket));
}
```
And whenever you use threads...

- Remember to think about thread safety.
- Servers are often quite parallelizable.
  - In the example, only $s$ was shared.
  - But the main thread immediately "dropped" it.
- But you will run into
  - Reuse of Singletons (globals)
  - Statistics, logging.
  - Metadata
  - Caching
The architecture of the web is “interesting”

- Basic web architecture is *stateless*
- No inherent concept of state associated with “users” or “sessions.”
- Each GET or POST is an independent event.
- Remember Boggle
  - `/play` generated a new board.
  - How did the `/results` POST handler know what to check?
- Can we do something more general and convenient?
Evolution of the web

- Early web was entirely static.
  - Web servers were basically file servers.
  - Server has “root” directory and URLs are defined implicitly from there.
  - Excitement came from HTML (nice display and hyperlinks)

- Next step - ISINDEX
  - Web pages could display a (single!) text entry field.
  - Mostly intended to search a local site.
  - Search term appended (after a ’?’) and same url rerequested.
  - But what code implements the search?
    - Webservers are generic “I/O machines.”
    - Keep the application specific search logic out of the web server!
Common Gateway Interface - CGI

- Webserver executes an external program for certain URLs.
  - The webserver is configured with certain patterns as CGI urls.
- Request info is passed as \textit{environment} variables.
  - \texttt{SCRIPT\_NAME}, \texttt{PATH\_INFO}, \texttt{QUERY\_STRING}, etc.
  - \texttt{HTTP\_*} for HTTP headers.
- and on stdin (POSTs)
- The program starts up, reads info, and generates HTML.
- Which is “piped” back to the server to send to client.
- A language-agnostic form of Spark.get or Spark.post functionality.
Like embedded languages, this kind of modularity can rescue you when you need to make changes at run-time on a long running server.

John Jannotti (cs32)  Web apps (thinking about multiple users)  Apr 17, 2018  20 / 33
**Dynamic content**

- ISINDEX was generalized to “Forms.”
  - Multiple fields.
  - Multiple types of fields (checkbox, radio, select, password)

- [http://bank.com/cgi-bin/calculate?loan=200000&interest=6.5&term=32](http://bank.com/cgi-bin/calculate?loan=200000&interest=6.5&term=32)

- Early uses depended only on a single request.
  - Search results
  - Mortgage calculator
  - No need to tie together multiple requests of the same user.

- More complicated sites need to keep track of users and their data.
  - Shopping cart
  - Personalization
Where can we stash state?

- URLs
  - http://www.bog.com/board?lines=tntb,gdar,elte,sish
  - http://www.bog.com/user?name=jj
  - Fiddly, (often) insecure, ends up with “ugly”, unsharable URLs.

- In form data
  - Put user specific data in “hidden” fields.
  - Each submission identifies itself to server.
  - All requests must be form posts.
  - Unbookmarkable, unshareable.

- Boggle only needs state for one submission, so it used the form “trick”.
The hidden `board` field tells the next Handler what to work on. This is tedious, not to mention insecure (allows cheating).
QueryParamsMap qm = req.queryMap();

Board board = new Board(qm.value("board"));
Set<String> legal = board.play();

Iterable<String> guesses = BREAKWORDS.split(qm.value("guesses"));

Compare legal to guesses, and make the HTML response.
What if I want to display a game count?

- /results Should say how many games you’ve played.
- How?
What if I want to display a game count?

- `/results` Should say how many games you’ve played.
- How?
- We *could* add `&plays=4` or hidden fields to *everything*...
What if I want to display a game count?

- */results* Should say how many games you’ve played.
- How?
- We *could* add &plays=4 or hidden fields to *everything* . . .
- We also *could* use a “global” count in the Java server.
- But that would be wrong for multiple users. (Would have been acceptable in your projects thus far.)
Cookies to the rescue

- Cookies are extra (name, value) pairs maintained by browsers.
- Their content is set by servers (usually).
- They are resubmitted by browsers in every future request.
- Eliminates constant form submission, or polluting URLs.
- Now we could increment a cookie value to track plays.
- Drawbacks?
Cookies to the rescue

- Cookies are extra (name, value) pairs maintained by browsers.
- Their content is set by servers (usually).
- They are resubmitted by browsers in every future request.
- Eliminates constant form submission, or polluting URLs.
- Now we could increment a cookie value to track plays.
- Drawbacks?
  - Cookies are limited in size, so can’t keep a lot there.
  - Each cookie also bulks up every request.
Leverage cookies into server “sessions”

- We can use just one cookie to index into a server-side Map containing as many values as we want.
  - Domain: doodle.com
  - Name: JSESSIONID
  - Content: 917FDD3FB3365A8E71EFC9A5F1F06285.worker2
  - Also: Path, Protocol, Expiration

- Web frameworks, including Spark, do this for you, making a “session” (Map<String,Object>) available with each request.

- If a client comes in without a cookie, a new empty session is created, and the cookie set.
Counting plays with Spark

```java
private static final String PLAYCOUNT = "PLAYCOUNT";
private static class PlayHandler implements TemplateViewRoute {
    ... 
    Session session = req.session();
    Integer plays = session.attribute(PLAYCOUNT);
    if (plays == null)
        plays = 0;
    plays ++;
    session.attribute("PLAYCOUNT", plays);
    ...
}
```
“Users” are more than sessions

- Sessions last for “a while”. (Depends on cookie expiration.)
- Sessions are not used across different devices.
  - But you’d like to maintain a shopping cart at home/work, computer/phone.
  - How can you keep a running total (or complex stats) of every boggle game I’ve ever played?

Another cookie that identifies the user.
Server side Handler checks a password, sets the extra cookie.
Server is responsible for persistence (usually a database).
USER=jj is not a good cookie. Why?

Do NOT trust clients. Encrypt cookie, or use long random cookies.
“Users” are more than sessions

- Sessions last for “a while”. (Depends on cookie expiration.)
- Sessions are not used across different devices.
  - But you’d like to maintain a shopping cart at home/work, computer/phone.
  - How can you keep a running total (or complex stats) of every boggle game I’ve ever played?
- Another cookie that identifies the user.
- Server side Handler checks a password, sets the extra cookie.
- Server is responsible for persistence (usually a database).
- USER=jj is not a good cookie. Why?

Do NOT trust clients. Encrypt cookie, or use long random cookies.

John Jannotti (cs32)  Web apps (thinking about multiple users)  Apr 17, 2018  29 / 33
“Users” are more than sessions

- Sessions last for “a while”. (Depends on cookie expiration.)
- Sessions are not used across different devices.
  - But you’d like to maintain a shopping cart at home/work, computer/phone.
  - How can you keep a running total (or complex stats) of every boggle game I’ve ever played?
- Another cookie that identifies the user.
- Server side Handler checks a password, sets the extra cookie.
- Server is responsible for persistence (usually a database).
- USER=jj is not a good cookie. Why?
- Do NOT trust clients. Encrypt cookie, or use long random cookies.
Web security is challenging

- The advice to not trust clients keeps coming up.
- Remember how easy it was to interact with web servers?
- Easy examples:
  - Don’t place price in web pages, and trust them on form submission.
  - Don’t build up SQL queries by concatenating user input.
  - Don’t “eval” JSON.
  - Quote user input properly when displaying it. (XSS)
- One more, Cross Site Request Forgery gets its own slide.
Cross Site Request Forgery

- Boy, cookies are convenient.
- The browser automatically identifies us, each time we make a request.
- What could go wrong?
Cross Site Request Forgery

- Boy, cookies are convenient.
- The browser automatically identifies us, each time we make a request.
- What could go wrong?
  - What if my page at www.jannotti.com makes an AJAX request to www.facebook.com?
- Fortunately, browsers don’t allow that, “Same Origin Policy”
- Are there other ways my page can make requests to Facebook?
  - IMG and SCRIPT tags, among others.
- Fortunately, I probably can’t get data that way.
- But I can still cause actions. Buy an item?
- Mitigation?
Cross Site Request Forgery

- Boy, cookies are convenient.
- The browser automatically identifies us, each time we make a request.
- What could go wrong?
  - What if my page at www.jannotti.com makes an AJAX request to www.facebook.com?
  - Fortunately, browsers don’t allow that, “Same Origin Policy”
  - Are there other ways my page can make requests to Facebook?
Cross Site Request Forgery

- Boy, cookies are convenient.
- The browser automatically identifies us, each time we make a request.
- What could go wrong?
  - What if my page at www.jannotti.com makes an AJAX request to www.facebook.com?
  - Fortunately, browsers don’t allow that, “Same Origin Policy”
  - Are there other ways my page can make requests to Facebook?
  - IMG and SCRIPT tags, among others.
  - Fortunately, I probably can’t get data that way.
  - But I can still cause actions. Buy an item?
Cross Site Request Forgery

- Boy, cookies are convenient.
- The browser automatically identifies us, each time we make a request.
- What could go wrong?
  - What if my page at www.jannotti.com makes an AJAX request to www.facebook.com?
  - Fortunately, browsers don’t allow that, “Same Origin Policy”
  - Are there other ways my page can make requests to Facebook?
  - IMG and SCRIPT tags, among others.
  - Fortunately, I probably can’t get data that way.
  - But I can still cause actions. Buy an item?
  - Mitigation?
Webapps: Now that we (mostly) can, should we?

- When is a web app a good idea?
- When is it a bad ideas?
- Even trickier: what about mobile?
  - Two compromises: Phonegap-like and Titanium/React Native-like.
Phonegap or Cordova

- A “native app” on iOS or Android uses the platform libraries.
- Usually written in Objective-C, Swift, Java or even C++.
- Access to “native widgets” and APIs.
- But hey, web browsers are also native apps!
- And a native app can embed an HTML “View”.
- So you can easily make a native app that displays “canned” web pages.
- With native APIs in the Javascript runtime, you can do whatever you want (geolocation, access contacts, etc) from Javascript.
- It’s very hard to get the “feel” just right.
- Great for rapid prototyping, reuse of web styles, or “uncomplicated” parts of an app.
The Phonegap approach really has two parts.
  ▶ Javascript access to native device-oriented APIs.
  ▶ WebViews used to display content.
Let’s push harder on the first part, and ignore the second.
Provide access to the UI APIs through Javascript.
“Hide” that behind nice libraries, so it seems like creating and styling HTML from Javascript.
End up with UIs that “feel” right (because they are truly native).
Retain the ease of web-like app development.
I’ll talk about this approach more next week.