CS1320
Creating Modern Web and Mobile Applications
Lecture 15
Web Application Architectures II
CDQuery User Library

• Suppose CDQuery were modified to know the user’s current collection
  o Understand what CDs they owned
  o Use this information in querying and display

• Then the application would need to know who the user was
  o Why is this problematic?
Web Applications and HTTP

• The web application assumes it knows the user
  o One request follows another
  o Common shopping cart for the user
  o Look up information based on the user
  o Server needs to know who the user is
    ▪ Even if they haven’t logged in

• HTTP is stateless
  o Each request is independent of previous requests
  o Requests come on different sockets at different times

• This disparity is addressed using sessions
What is a Session

• A mechanism for maintaining state
  ○ For the particular user and the particular web app
  ○ Within the server
  ○ Somewhat independent of the browser

• The session contains information about the current state
  ○ Information about the particular user
  ○ Information for the particular application
  ○ Information for this particular use of the application
Sessions

- **Represent a connected series of user actions**
  - Example: log-in, select-items to purchase, check-out, log-out
  - Example: select source/destination cities, dates; request schedules; select flights; login; purchase tickets

- **Needs to have a fixed start**
  - Might have a fixed end (log-out request)
  - More likely, time-out if unused; exit when browser closes
Session Properties

- **What information needs to be kept with the session**
  - Depends on the application

- **Sample information**
  - User id if one exists
  - Host, last-used time
  - Shopping cart
    - Associated with user id?
    - How to handle log in afterwards
  - Input values for forms (to be (re)filled automatically)
  - Previous searches or history
  - Site customization values
Tracking Sessions

• Should the CLIENT track the session
  o If you don’t browse off the page, these can be kept in html
    ▪ Hidden fields, JavaScript variables, separate DOM tree, etc.
  o But if you replace the page, they disappear
  o Also, if there are multiple pages up, what is used

• HTML 5 Local storage
  o Key-value pairs for the same domain
  o Settable and gettable from JavaScript
  o Works if the information is local & HTML5 is available
    ▪ And users always use the same browser and same machine (without incognito mode)
Tracking Sessions

• Should the SERVER track the session
  o Maintain as part of state for user
  o But need to send/get it from the browser
    ▪ Server needs to tell the browser the state for new pages
    ▪ Browser needs to tell the server the state for all requests
  o What happens if there are multiple pages displayed
  o What happens with back and forward buttons

• Client and Server both track the session
  o Typically using cookies
Tracking Sessions

HTTP Client

- Login POST
- username: david
- password: davida

- Set-Cookie: SESSIONID=66C530ACAF44D160588819EC80C737C

HTTP Server

- Login successful?
  1. create session id
  2. return session id in cookie
  3. store session id in database

- SESSION ID
  SessionId
  Username
  createDate
  expireDate
  lastAccessDate

- Database

- Lookup Session ID
  1. session match a username?
  2. session still valid?

HTTP is Stateless

Cookie: SESSIONID=66C530ACAF44D160588819EC80C737C

Content for 'david'
Cookies

• Cookies are a general mechanism
  ○ For conveying information between browser and server
  ○ Name-value pairs associated with a particular URL
    ▪ Can have multiple pairs
  ○ Sent automatically by the browser as part of the HTTP header
    ▪ With any request to that particular URL

• Can be set either by server or browser
  ○ Communications: header on a page can request a cookie set
  ○ Defining: JavaScript functions to define cookies
Cookie Properties

• **Name and the value associated with that name**

• **Maximum age**
  - When the cookie should be ignored/removed by browser
  - 0 means when the browser closes

• **Domain/port and path**
  - When to include the cookie in a HTTP request
  - Domains can be as specific as desired
  - cs.brown.edu, taiga.cs.brown.edu, taiga.cs.brown.edu/myapp

• **If you need security, use HTTPS**
  - Cookies can be restricted to only work with HTTPS
Cookie Management

- Libraries in server to manage cookies
  - Call to add/set a cookie (language-dependent)
  - Call to read cookie value
  - Added to headers on output pages
  - Used to extract session ids

- Similar libraries exist in the client (not widely used)
Session Identifiers

• How much information needs to be conveyed to and from browser?
  ○ We’ve talked about lots of things, some can be large
  ○ Really only need one piece of data
    ▪ Use this as an index to a table (or database) on the server
    ▪ Table holds all the information related to the session
  ○ This is the session ID

• Tracking Session IDs is difficult
  ○ Ensure validity (difficult to spoof; only server-generated IDs)
  ○ Ensure it is coming from same machine
  ○ Setting and checking cookies correctly
  ○ Time out if not used for certain amount of time
  ○ Handling explicit end of session
Session Management

• Use built in session-support
  o For your server
    ▪ Call to begin/enter session
      ■ Automatically looks at cookies or url
      ■ Validates the session
      ■ Makes session data available
    o Call to terminate session

• Can store arbitrary information with session
  o Can be stored in memory (not ideal)
  o Can be stored in application database
  o More often stored separately (REDIS)
Cookies, Sessions and Express

```javascript
var session = require('express-session');
var cookieparser = require('cookie-parser');
...
app.use(cookieparser("KEY"));
app.use(session { secret : "KEY", store: new RedisStore(), ...});
app.use(sessionManager);
...
function sessionManager(req,res,next) {
    if (req.session.uuid == null) {
        req.session.uuid = <unique id>
        req.session.save();
    }
    next()
}
...
req.session.<field>
```
Model-View-Controller

Controller

View

Data Display

Data Management

Model

Data Store
Data Manipulation
**DRY Principle**

- **Don’t Repeat Yourself**
  - Every piece of knowledge must have a single unambiguous authoritative representation within a system

- **Why have 3 different representations of data?**
  - More code to maintain
  - More code to change when data changes
  - More chance for bugs
Django and Ruby on Rails (and Flask)

- Widely used
  - Django: instagram, pinterest, …
  - Ruby/rails: github, basecamp, …
- Similar frameworks exist (e.g. Flask)
- Mostly a back end technology
  - Can be paired with a templating engine
  - Can be paired with front end templating as well
- Require knowing Python/Ruby
  - In addition to JavaScript, HTML, …
Django/Ruby Frameworks

• Express-like dispatch
  ○ Based on static tables, not executed code
  ○ With functions to handle the results

• Logic to control deployment, server setup, etc.

• Libraries to handle common web app

• Simple connection to database

• Simplified Data Management
DJANGO and Ruby/Rails

- **Map from internal objects to SQL automatically**
  - Changes in the object -> SQL updates
  - Objects created automatically from SQL database
  - SQL Tables created automatically from object definition
  - Changes to object definition change the database

- **Map from internal objects to HTML automatically**
  - Using templates

- **Map from internal objects to JSON automatically**
  - Changes in the object -> go to web site if needed

**OBJECT-RELATIONAL MODELING**
Object-Relational Modeling

- Not limited to Django-Ruby
  - There are libraries to provide some of this functionality
  - Even for Node.JS

- Not limited to SQL back ends
  - NoSQL databases can be used as
    - Direct mapping to object from json
  - Cache the current state in memory as objects
    - This allows fast query at times
  - Update updates memory and the database

- What are the problems with ORM?
RESTful Web Applications

• **Client-Server model**
  - Client handles presentation, server handles storage
  - MVC: client = view, server = model; controller can be either, generally client

• **Stateless**
  - All data needed for request is passed

• **Client maintains data**
  - Sends updates, requests to server
  - Using commands encoded in URL
RESTful API HTTP Methods

- **Collection API .../collection**
  - GET: return list of elements in the collection
  - PUT: Replace the entire collection
  - POST: Add an entry to the collection
  - DELETE: Delete the entire collection

- **Element API: .../collection/:item**
  - GET: Retrieve the given item
  - PUT: Replace or create the given item
  - DELETE: delete the given item

- **Action API: .../collection/:item/verb**
DJANGO/RUBY with REST

• **URL identifies the object in the server**
  - What field to access or change
  - New value of the field (using PUT)

• **Front end makes changes to long term**
  - By sending POST requests

• **Front end gets current state of objects**
  - By sending GET requests
Content Management Systems

• Creation and Modification of digital content
  o The contents of the web site

• Easy to create good-looking sites
  o With modern bells and whistles (e.g. slide shows)

• Easy to update the contents
  o For a non-programmer

• Standard interaction mechanisms often included
  o User accounts, …
  o Blogs, Wikis, …
Content Management Systems

- **WordPress**
  - The standard
  - PHP based
  - Extensible with modules or your own php code

- **Drupal**
  - Relatively common, more flexible
  - Fewer modules and features
  - PHP Based

- **Django-CMS**
  - Used for Brown CS web site
  - Python (Django) based

- **Lots of others available**
CMS Features

• **Templating engine**
  - MVC model - separate presentation from application logic
  - Reusable pieces

• **Roles and permissions**
  - Authentication
  - Roles: admin, author, editor, user, …
  - Hide complexity
CMS Features

• In-Browser Editing
  o Either separate editor on on-page editing
  o Layout and style
  o Images and media
  o Plugins such as Google maps
  o EXAMPLE: Brown CS web pages

• Publishing workflow
  o Create -> Edit -> Approve -> Publish -> Update -> Approve …

• Versioning
  o Revert, record of who did what
CMS Features

• Multilingual
  ○ Support for different languages

• Accessibility support

• Multi-site
  ○ Multiple sites running on one server

• Tree-like page structure
  ○ With appropriate permissions

• RESTful URLs

• Analytics
CMS Integration

- Can use CMS as a part of the web site
  - For the appropriate pages
- Code the other pages separately
  - Node.JS or other front end
- Integration in various ways
  - Django with Django-CMS
  - Reverse Proxy
    - Front end server redirects to appropriate back end
Next Time

• Node.JS lab
Next Time

• Node.JS lab
• Homework: Prelab for Node.JS
CDQuery (Again)

Find Your CDs

CD Search:

Find Your CDs

<table>
<thead>
<tr>
<th>Title</th>
<th>CD#1 title and major artist</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>* Track 1</td>
</tr>
<tr>
<td></td>
<td>* Track 2</td>
</tr>
<tr>
<td>Artist</td>
<td>CD#2 title and major artist</td>
</tr>
<tr>
<td></td>
<td>* Track 1</td>
</tr>
<tr>
<td></td>
<td>* Track 2</td>
</tr>
<tr>
<td>Track</td>
<td>CD#8 title and major artist</td>
</tr>
<tr>
<td></td>
<td>* Track 1</td>
</tr>
<tr>
<td></td>
<td>CD#4 title and major artist</td>
</tr>
</tbody>
</table>

Find Your CDs

<table>
<thead>
<tr>
<th>CD TITLE</th>
<th>ARTIST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Description</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TRACK Title</th>
<th>Artist</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Length</td>
</tr>
<tr>
<td></td>
<td>Description</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TRACK Title</th>
<th>Artist</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Length</td>
</tr>
<tr>
<td></td>
<td>Description</td>
</tr>
</tbody>
</table>
XMLHttpRequest

```javascript
var req = new XMLHttpRequest();
req.onreadystatechange = function () {
    if (req.readyState == req.DONE) {
        if (req.status == 200) << Handle returned data req.responseText>>
        else << Handle error >>
    });
req.open("POST", "/url/…");
rq.setRequestHeader("Content-type", "application/json");
rq.send(<data to send>);
```
Session Management

1. **Session setup**
   - Set up a new session or the target server
   - Extract the session ID from this session

2. **Session maintenance**
   - Keep the session alive by sending periodic requests

3. **Session fixation**
   - Wait for the user to login
   - Enter the session

4. **Session entrance**
   - Wait for the user to login

---

**Diagram:**

- **Authenticate**
  - Pass authentication info to the server
  - Generate unique SessionId on successful authentication
  - Store session Id at server
  - Mode = "SQLServer"
  - Mode = "ASP.NET State Service"
  - Same Worker Process (application.exe)
  - Out-of-process SQL Server Database table

- **Persist information**
  - Store sessionId
  - Store session

- **Request another page**
  - SessionId passed as cookie or appended to URL to server
  - Verify the sessionId is valid or not from server-side stored session

- **Logout**
  - Pass authentication info to the server
  - Generate unique SessionId on successful authentication
  - Store session Id at server
  - Mode = "SQLServer"
  - Mode = "ASP.NET State Service"
  - Same Worker Process (application.exe)
  - Out-of-process SQL Server Database table

- **Persist information**
  - Store sessionId
  - Store session
Sessions in URLs

• Putting sessions IDs in URLs is not a good idea
  ○ Especially if the URL is public (GET rather than POST)

• Problems
  ○ GET requests may be logged; server logs now contain private information
  ○ Copy and paste of URLs can confuse the server
  ○ Server might use the passed in session id, allowing attacker to steal information

• Solution: use cookies
  ○ But what if cookies aren’t enabled?
Session Tracking Mechanisms

• **Encode the session id in the URL**
  - All requests from the browser are URLs
  - The ID can be part of each request
    - `http://....?SID=xxxxxxxxxxxxxxxxx&...`

• **How to get this into the URLs on the page**
  - If requests come from forms, add a hidden field
  - Requests for new pages, replace the URL on generation
  - How to get all URLs on the page
  - Problems?
Question

• Which is not true about sessions in a web application?

A. Sessions represent a connected series of user actions
B. Sessions must have a fixed start
C. Sessions must have a fixed end
D. Sessions can include a variety of different types of information
E. Sessions can be supported by cookies or URL query or post data
XMLHttpRequest (using jQuery)

- **Syntax**
  ```javascript
  var req = $.ajax({
    method: "POST",
    url: "/url/…",
    data: { data to send },
    success: function(data,sts) { … },
    error: function(msg,sts,err) { … }
  });
  ```

  - Request gets sent when JavaScript returns
  - Other parameters and events are available
CQ Query Tasks

• Primary Tasks
  o Initial Search For CDs
  o Look at the details of a specific CD
  o Refine initial search by title, artist, track, genre; sort results

• Should these be done client-side or server-side?
  o All server side
  o Initial search server side, rest client side
  o Refinement & detail client side, rest server side
  o Detail page client side, rest server side
  o All client side
Node.JS Event Example

• Request comes in
  o JavaScript code creates database query based on parameters
  o Starts query and registers continuation

• When query completes (done asynchronously)
  o Template parameters computed from database result.
  o Template file is opened and a new continuation is provided

• When file is ready to read (done asynchronously)
  o A stream from the file to the client is established
  o The file is templated and output is output asynchronously
Asynchronous Operations

• **Node.js libraries define asynchronous operations**
  - File open, read
  - Network read
  - Database queries and updates
  - Web sockets

• **Common combined operations also defined**
  - Streams: copy from one place to another
    - From the file system to the network
    - All handled in background
Web Sockets
Model-View-Controller

View
- Data Display
  - Web Browser

Controller
- Data Management
  - Web Server

Model
- Data Store
  - Database
Model-View-Controller

• Basic idea is to separate the display, the data, and the logic
  - Each can be change independent of the others
• Exactly how this is done varies from case to case
  - Some do it with a common data abstraction
  - Some do it with callbacks
  - All call themselves MVC
• Different people mean different things
React-JS

- Templates mixed with JavaScript code
  - Expressed as functions
  - With HTML
  - And embedded code
- Can be done either server side or client side
  - Use for templating in the server
AngularJS and VueJS

- Templates that are executed at run time
- Automatically update the page as values change
- MVC (Model-View-Controller)
  - Model = the data structures
  - View = the template
  - Control = commands that modify the data
- Combine this with Object-Relational Modeling
  - Make a simple, consistent web application
What Information is Preserved

• **Between pages**
  - Authentication information
  - Current state (shopping cart, nearest store, ...)
  - History (videos watched, ...)

• **Between runs (between browsers)**
  - User information
  - History
  - Is this session based?
Cookies, Sessions and Express

var session = require('express-session');
var cookieparser = require('cookie-parser');
...
app.use(cookieparser("KEY"));
app.use(session { secret : "KEY", store: new RedisStore(), ...});
app.use(sessionManager);
...
function sessionManager(req,res,next) {
    if (req.session.uuid == null) {
        req.session.uuid = <unique id>
        req.session.save();
    }
    next()
}
...
req.session.<field>