CS144: Web Applications

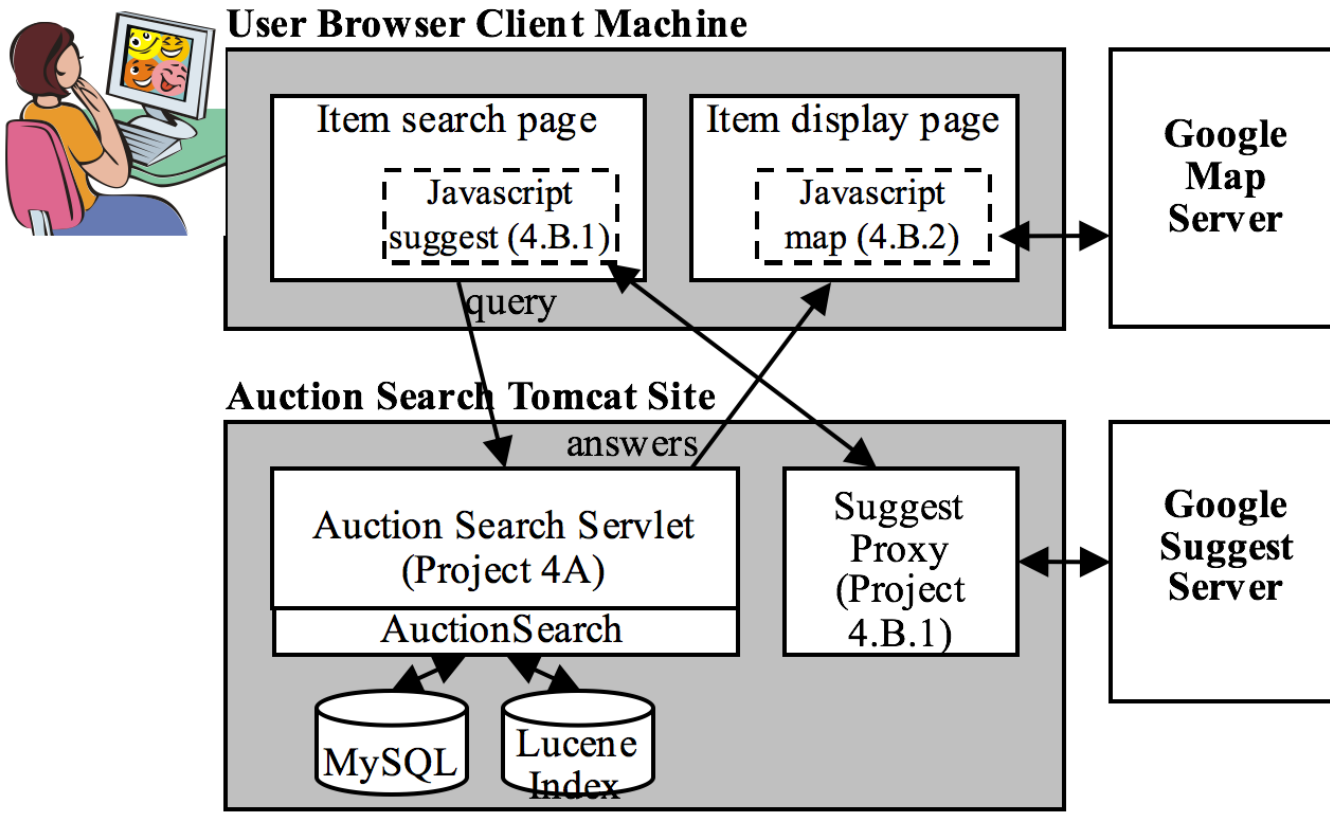
Project Part 4: *Building a Website*
Due on Friday, March 3, 2017, by 11:00PM

- **Submission deadline:** Programming work is submitted electronically and must be submitted by 11:00 PM. However, we recognize that there might be last minute difficulty during submission process, so as long as you started your sumission process before 11:00PM, you have until 11:55PM to completely upload your submission. After 11:55PM, you will have to use grace period as follows.
- **Late Policy:** Programming work submitted after the deadline but less than 24 hours late (i.e., by next day 11:00 PM) will be accepted but penalized 25%, and programming work submitted more than 24 hours but less than 48 hours late (i.e., by two days after, 11:00 PM) will be penalized 50%. No programming work will be accepted more than 48 hours late. Since emergencies do arise, each student is allowed a *total* of four unpenalized late days (four periods up to 24 hours each) for programming work, but no single assignment may be more than two days late.
- **Honor Code reminder:** For more detailed discussion of the Honor Code as it pertains to CS144, please see the Assigned Work page under *Honor Code*. In summary: You must indicate on all of your submitted work *any assistance* (human or otherwise) that you received. Any assistance received that is not given proper citation will be considered a violation of the Honor Code. In any event, you are responsible for understanding and being able to explain on your own all material that you submit.
- **Reminder:** Projects must be completed individually or by a team of two.

Change History

Overview

In Project 3, you developed the AuctionSearch Java class that searches and retrieves auction items within your database. In this part of the project, you will finally build a Web site that makes all auction items easily accessible by end users through a Web browser. You will implement the Web site as an *Apache Tomcat servlet* and it will integrate two Google services: the Google Maps (to display the location of an item) and Google suggest (to help users formulate keyword queries). Roughly, the overall architecture for this project is shown in the diagram below:



Note that the AuctionSearch class, MySQL database and Lucence index are what you implemented already as part of Projects 2 and 3. So your reponsibility for Project 4 is to implement the rest of the above architecture.

By completing this project you will be able to learn:

1. How to build a Web site using Java servlet and JSP
2. How to build dynamic user interfaces in Javascript and AJAX

We will be providing relevant tutorials and references to help you learn the above topics.

Part A: Implementing Your Basic Web Site

A.1. Learning Application Development on Tomcat

In this part of the project, you will use the Tomcat server for serving static and dynamic Web pages for Web browsers. As you may know, most web applications/sites consist of a fair number of files (like .html, .xml, .jsp, .jpg, .class, ...). In order to simplify the deployment of a Web application, Tomcat provides a mechanism to put together all files needed for an application in a single file, called *Web Application Archive*.

First, learn how you can develop and create your own Web application archive file for your application by reading the following tutorial:

- *Developing a Web Application on Tomcat*.

Before you proceed to the next step, make sure you understand how to develop and deploy a Web site using Java servlet, JSP, and Web application archive from the above tutorial.

A.2. Study the Provided Skeleton Code

Now that you understand the structure of a Web application archive file and how Java servlet and JSP pages can be used to serve dynamic contents, you need to implement the Java servlet that provides the search functionality for the eBay data.

The *project4.zip* file contains the basic skeleton code for this project. It contains three servlet classes, `SearchServlet`, `ItemServlet`, and `ProxyServlet`, that you will have to implement as part of Project 4. Go over the included `build.xml` file to understand what it does. Pay particular attention to the target "build", which essentially does the WAR file packaging for your eBay application. Note that all the files in `WebContents` directory are automatically added to the WAR file, including `WebContents/WEB-INF/web.xml`. Go over the `web.xml` file to see how request URLs are mapped to the provided servlet classes.

Note that *project4.zip* includes a `AuctionSearch.java` file (that you implemented as part of Project 3). To break the dependence of Project 4 on your earlier code (so that you won't be penalized for any bug in your code age), our provided `AuctionSearch.java` uses an important trick: Instead of obtaining the results from your locally installed local MySQL and Lucene, it obtains the results remotely from oak.cs.ucla.edu server. That is, the provided code does not depend on your local MySQL database or Lucene index! In addition, since it has the same interface as the `AuctionSearch` class in Project 3, so you can use the provided class exactly as you did in Project 3.

A.3. Implementing the Search Interface

Now it is time to create a search interface "page" in your web application. This "page" should be accessible at *http://localhost:1448/eBay/search* and contain at least one HTML input box where the user can enter and submit a keyword query. Once the query is submitted, your web application should display the list of top-k (k can be any reasonable number, say 20) matching items to the query. Your app should also provide ways for the user to "request" the next-k items, in case the user is not satisfied with the first-k items. Clicking on any item in the list should display the detailed information of the item, which will be implemented in A.4.

Note that the provided `web.xml` file maps the request to *http://localhost:1448/eBay/search* to the `SearchServlet` class, so you need to implement this search "page" by implementing the `SearchServlet` class (and the JSP page forwarded from this servlet class in case you follow the MVC approach).

Notes on CLASSPATH: (1) When you implement a Java servlet, your code will depend on the Tomcat servlet library, `$CATALINA_HOME/lib/servlet-api.jar`. To avoid the "class not found" error during compilation, you need to pass the jar file location as the classpath to your Java compiler in your Ant script. However, you should not include `servlet-api.jar` or any of the other standard Tomcat jars in your .war file's `WEB-INF/lib` folder. These libraries are already available to all servlets running within Tomcat (all jars in `$CATALINA_HOME/lib` are), and may actually cause classloader problems if included in your war file. (2) The provided `AuctionSearch` class in depends on the Axis2 library files. To avoid any "class not found" error during compilation, the Axis2 jar libraries at `$AXIS2_HOME/lib` should be passed to your Java compiler as a classpath. The library files are already available within Tomcat, so you do not need to include them in your war file. These two classpath issues are automatically taken care of if you use our provided `build.xml` and use the target "build" to compile and package your application.

The provided `build.xml` has another target "deploy" that copies the .war file built from the target "build" to `$CATALINA_BASE/webapps`. You can use this target to copy your packaged WAR file to Tomcat for testing. Take a look at the *Ant tutorial* if you do not understand the provided Ant script well.

A.4. Implementing the Item Interface

Similarly to the search interface, implement `ItemServlet` class that will return all of the details of an item (ItemId, name, start and ending time, bid history, etc.). The servlet is configured to be available at *http://localhost:1448/eBay/item* in the provided `web.xml` file. The item information should be deplayed "reasonably" and "intuitively" in your app. Do not simply show an XML dump to the user.

Part B: Adding Maps and Query Suggestion

Now that you have implemented the basic functionality of your Web site, it is time to add more help to the users by adding more dynamic UIs. Before you proceed, read about *JavaScript* as that is the language used throughout Part B. You may also find a Javascript debugger (for example, *Firebug* for FireFox) helpful in debugging your javascript code for this project.

B.1. Adding Google Maps

On your item interface, add Google Map to visually display the location of the item. In principle, you need to signup for a

Google Maps API key to use Google Map on your site, but for the basic Google Map API, you may be able to use it without your own API key. For instance, a sample HTML page like the following will work without an API key:

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
<style type="text/css">
  html { height: 100% }
  body { height: 100%; margin: 0px; padding: 0px }
  #map_canvas { height: 100% }
</style>
<script type="text/javascript"
    src="http://maps.google.com/maps/api/js?sensor=false">
</script>
<script type="text/javascript">
  function initialize() {
    var latlng = new google.maps.LatLng(34.063509,-118.44541);
    var myOptions = {
      zoom: 14, // default is 8
      center: latlng,
      mapTypeId: google.maps.MapTypeId.ROADMAP
    };
    var map = new google.maps.Map(document.getElementById("map_canvas"),
        myOptions);
  }

</script>
</head>
<body onload="initialize()">
  <div id="map_canvas" style="width:100%; height:100%"></div>
</body>
</html>
```

To see what will happen in the above example: *click here*. You should copy the relevant parts of this sample into the appropriate place in your app, so that your "item page" includes a small map that shows the location of your item and its immediate proximity. In case an item's latitude and longitude are not available, you may choose to simply not display a map, display a generic map such as of the entire world, or something else. Whatever you choose, it should be user-friendly, so avoid showing cryptic error messages or alert windows.

B.2. Adding Query Suggestion

Your auction site users may not know exactly what they want to search for, so in this last part you will be helping them formulate queries using *Google Suggest*. Try out their query interface to get a feel for what it does. When you are finished with Part B.2, the keyword-query box on your "search page" should behave similarly to the one on Google suggest.

The Google suggest interface uses AJAX to "silently" submit requests to a *REST-style Web Service* in response to user input (keypresses). The replies are parsed, and the returned suggestions are used to populate the drop-down list of suggestions. As we learned in the class, this interaction takes place using the XMLHttpRequest JavaScript object. Before you go any further, be sure you've read the required reading about *AJAX*, *JavaScript*, *DOM*, and *XMLHttpRequest*.

B.2.1. Developing a Google-Suggest Proxy

Due to the asynchronous nature of AJAX, security is naturally a big concern when using JavaScript. To reduce some security risks, *JavaScript security* enforces a "same-origin policy", which means that the scripts on a page may only access information within the same domain. For example, a script on a page like "http://localhost:1448/MySite/index.html"

cannot make a request for information from, say, google.com.

As a result, you will need to write a proxy (which runs on your Tomcat server) to pass requests between your query interface page and Google's suggest service. This proxy can be implemented by the `ProxyServlet` class that we provided which is configured to accept HTTP GET requests at *http://localhost:1448/eBay/suggest*. Your implementation should extract the passed-in query string, issues a request to the Google suggest service for that query (at `http://google.com/complete/search?output=toolbar&q=<your query>`), and returns the results back to the original caller. You may find the Java class *java.net.HttpURLConnection* helpful in building your proxy. Note that your servlet should return the *exact* XML data received from Google. You should thoroughly test your proxy (by issuing REST-style requests to it and checking the results) before moving on to the next step. Also note that, as your servlet should be returning XML, your browser may not render it properly (you may see just a blank page). Check the 'view source' option in your browser to see the data. Finally, if you intend to use responseXML to access Google suggest responses as an XML DOM, make sure that your proxy sets the "Content-Type:" HTTP header field to "text/xml" in its response. Otherwise, your browser wouldn't be able to recognize the response as an XML data, and won't be able to set responseXML correctly. You can set the content type of a Java servlet reponse by calling the `setContentType()` function of the response object.

B.2.2. Developing a Google-Suggest Client

After you have a functioning proxy, you'll need to write the "client" code in your query interface page which issues requests to your proxy as the user types and shows a list of suggestions. To do this, you will need to write a set of *event driven* (asynchronous) functions in JavaScript which submit requests and process results by populating a drop-down text box with a list of suggestions. By asynchronous, we mean that these functions are invoked as the result of "events", such as user key presses and data arrival, rather than by some control flow as is typical in procedural programming.

For the creation of a dynamic drop-down box in javascript, the following pages provide an excellent tutorial:

- *Creating an Autosuggest Textbox with JavaScript, Part 1* (*Local mirror*)
- *Creating an Autosuggest Textbox with JavaScript, Part 2* (*Local mirror*)

Go over the above tutorial to learn about the HTML elements and their associated events to create a drop-down-box like interface for query suggestion. The tutorial provides a number of code snippets, which you are welcome to use for your implementation. Your implementation of Google Suggest should behave similarly to the one on Google Suggest. **In particular, as the minimum requirement, you must ensure that as the user types input, a list of suggestions are populated based on the partial query, which the user can navigate with the mouse pointer or up/down arrows and choose with the enter key or mouse click button. Moreover, under no circumstance do your code crash when a user is navigating the list.**

Remember that, when you issue a request to your servlet, you'll likely be encoding the user's (partial) query in the URL. You need to "escape" special characters in URL. For example, a space in a URL should be converted to "%20". You do not need to manually do this replacement - most "web" languages, like javascript, contain functions to do the proper URL encoding for a string, which you simply need to call.

Additional Notes for Project 4

Please test your Web site using the latest version of Google Chrome Browser and make sure that it works correctly, so that we can avoid any problem due to cross-browser compatibility

For all of Project 4, the mentioned functionality is the minimum requirement. You are free, and of course encouraged, to add additional features, make the page more user friendly, etc. Also, you are not required to implement things exactly as indicated above. You may choose to, for example, display item details in a popup bubble using some AJAX UI library instead of on a separate page.

You must, however, abide by the requirement that your displayed result pages must be for "human consumption" - not just XML data. Also, your site should handle errors "gracefully". Your site should not show a cryptic Java error message to the user.

Showing your Web App

By now you should already have a working website, and in this section you are going to prove it. In order to do this, you

must "record" a **three-minute video**, demonstraing the basic functionality and interaction of your site. At the minimum, your video must demonstrate that

1. You can access the "search page" at http://localhost:1448/eBay/search
2. You can type query like "camera lens" and appropriate suggestions are made and can be selected
3. The top-k results from the query is displayed on the screen and the next-k results are easily accessible if needed
4. Clicking on a matching result will display the detailed information of the item together with a map of the item's location

There exist a number of excellent free/paid software for recording your computer screen while you demonstrate the functionality of your Web site. For example, *OBS Studio* is a free open-source screen software that is widely using for video recording and live streaming. Whatever software you use, make sure that your video is playable with the latest version of *VLC Media Player* (without installing any proprietary codec) to avoid any codec/format incompatibility issue. Recording your video using the H.264/MPEG4/WebM codec in the MP4/MOV/MKV container format will be a safe choice.

In order to help you get familiar with OBS Studio, ***a short tutorial*** is provided.

What to Submit

In this project you are creating a web site, consisting of both static content (HTML pages, images, etc.) and dynamic content (JSP pages, Java Servlets). This content is wrapped up as a Web application archive (.war) file which can be deployed on Tomcat.

Your submission should consist of (1) a **project4.zip** file that contains all of your Web-site related files and a (maximum) **three-minute video** file that demonstrate the functionality of your Web site. The **project4.zip** file should be structured as follows:

```
project4.zip
 |
 + team.txt
 |
 + README.txt
 |
 + build.xml
 |
 + WebContents
 |   + Web resource files (*.html, *.jsp, *.css, *.js, image files)
 |   + WEB-INF
 |       + web.xml
 + src
 |   + java source codes for the servlet (with your own naming/structure)
 |
 + lib
    + external java libraries used (not available in our VM)
```

The `team.txt` is a plain-text file (no word or PDF, please) that contains the UID(s) of every member of your team. If you work alone, just write your UID (e.g. 904200000).
If you work with a partner, write both UIDs separated by a comma (e.g. 904200000, 904200001). **DO NOT put any other content, like your names, in this file!**

The `build.xml` in the zip file should have the target "build" that builds your web site into a single .war file, and the target "deploy" that deploys that .war file on the Tomcat server pointed to by the environment variable `$CATALINA_HOME`. That is, **we should be able to simply unzip your submission and run "ant build" and "ant deploy" to deploy your web site on our machine.**

Note that your submitted **project4.zip** file should include *all* files needed for the proper operation of your Web site. Also note that your three-minute video is **not** included in the **project4.zip** file and need to be submitted separately. Add any additional notes or comments that you think will be helpful to the README.txt file and once your two submission files

(project4.zip and your video) are ready, submit via our submission page at ***CCLE*** .

As always, remember to allow sufficient time to prepare your submission once your work is complete.

Testing of Your Submission

The "grading script" for your submission is the file *p4_test*, which can be executed like:

```
cs144@cs144:~$ ./p4_test project4.zip
```

Add the path to the zip file if necessary after downloading the script and set its permission appropriately.

You **MUST** test your submission using the script before your final submission to minimize the chance of an unexpected error during grading. Please **make sure that your submission uses oak service**, not your own local Web service. When evertything runs properly, you will see an output similar to the following from the grading script:

```
Stopping tomcat server if it is running...
Running 'ant build' to build your war file...

   ... output from ant ...

Removing existing eBay application files on Tomcat...
Deploying your eBay application...

   ... output from ant ...

Now your Tomcat server is running with your application.
Please access your application through your browser.
Make sure that all application functionalities are working fine.
Don't forget to stop Tomcat server once you are done.
```

After you run the script, make sure to check the functionality of your site using the Firefox browser in the VM and stop the Tomcat server when you are done.

Grading Criteria

Overall grading breakdown is as below and each feature will be tested with simple cases and tricky cases.

- Servlet/JSP (40%)
- Google Map (20%)
- Google Suggestion / Search Interface (40%)