Elliot Liskin December 2015
Design Homework 2

On first load, the user is presented with a window that offers the option to choose a previously created Course from disk or to create a new Course. A Course, in Class Extractor, is a file that contains processed information of the user's input. So, for instance, a Course will contain the most important topics in a class as determined by the natural language processing (NLP) algorithms applied to a student's audio recordings of a professor. A Course is a collection of Lectures that has the ability to display an aggregate of all the topics discussed in the course as well as the topics discussed in any individual Lecture.

If the student chooses a previously created Course from disk, that Course file is loaded and its information is displayed. If the user chooses to create a new Course, a dropdown appears, offering the ability to select one or more raw audio recordings for processing. The user is presented with a determinate progress indicator for the time estimated until completion ("completion" is defined as all processing of the audio file(s) being finished).

After Class Extractor has successfully received the results from Watson, sent the data to and received the return array from Aylien, and calculated the importance of all topics, Class Extractor will close the Course File Selector window, opening a new window with Word Cloud and Timeline. Word Cloud is a series of ovals that each contain a topic name and hyperlinks to the audio recording and to the presentation where the topic can be found. Additionally, the sizes of Word Cloud's ovals are determined by the importance of the topic they represent. Word Cloud allows for direct comparison of importance between topics, as the size of each cloud is a function of the importance of its topic. Clicking on a topic's cloud causes the interface to zoom into that cloud and a new word cloud is generated with subtopics of the clicked cloud's topic. The hyperlinks in each cloud allow fast access to hearing about that topic, making it easy for a student who wants to review a topic to listen to it again. The second interface, Timeline, provides the student with a high-level temporal overview of when different concepts are discussed in a lecture, which concepts they overlap with, and what their subconcepts are. At the bottom of the interface is a timeline bar representing the lecture recording, and concepts are overlaid on top of it. Each topic overlay displays its name, so if a student wants to hear about mitosis in a biology class, he or she can simply click on the mitosis overlay, and the section of the lecture recording concerning mitosis will automatically start playing. If the student wants to review telophase, one of the stages of mitosis, the student can select the telophase overlay to listen to the segment about telophase. There will be a tab bar navigator at the bottom of the window, allowing the user to switch between these two interfaces.

Word Cloud will be backed by an array of topic objects, each of which has fields such as the name of the topic, where the topic can be found in the audio recording (represented by a start time and a duration), and the calculated importance of the topic. Since Class Extractor follows MVC (model-view-controller), Timeline can use the same backing as Word Cloud.

The Word Cloud interface controller will calculate the correct size of the cloud using the following algorithm: the cloud diameter is double the importance rating plus the multiplier times the reciprocal of the importance rating. This algorithm slightly favors less important topics, as if there is no weighting, there might be a clouds of size 200 and of size 10. Clouds of size 10 cannot display reasonably readable text or other information, so the algorithm makes them bigger. Note, however, that even though it does favor them a little bit, the lack of importance of these smaller topics will still be evident in the size of the cloud.

At the left side of the window in a side bar controller, there will be a hideable pane that allows the user to scroll through all of the Lectures in the Course. At the top of this pane is a list entry for Course Overview, which, when selected, displays a Word Cloud and Timeline for the

Course as a whole. This interface shows an aggregate of topics discussed so far. If, in a Course, the professor spent three lectures discussing a specific idea, then that topic is very important for the course holistically and will be displayed prominently; however, a topic being discussed in only one lecture does not make it unimportant, and it should still be studied and learned for any exams. As such, it will be accounted for accordingly. The Lectures in the side bar controller will be backed by an array of Lecture objects, which will each contain a file URL to the location of where the Lecture file can be found (which contains all of the information about that Lecture).

When the user saves the Course file and closes the application, Class Extractor will save all relevant information to disk in a Course file. When the user wants to view his or her Course, he or she can double click the file to open up the Course overview page. The user can then select a specific Lecture if desired.

In the File menu, the user can also add Lecture recordings to a Course, which will then initiate the analytical process as described above and incorporate that Lecture's information into the Course overview and create an entry in the Lecture side bar. There will also be an option to delete a Lecture if desired, which will remove that Lecture's entry from the side bar and delete any information from the Lecture in the Course overview.

The major algorithm used in Class Extractor is "frecency," which is a combination of frequency and recency. The amount of times a professor talked about a topic, the length of time he or she discussed it, as well as how recently the professor discussed that topic impacts how important that concept is for the Lecture, for the class as a whole, and how important it is in relation to other topics.

To calculate frecency, Class Extractor starts with calculating frequency. Frequency sums up the number of times the topic was mentioned in a class, and its pseudocode is below:

```
int curWordCount = 0;

for (int i to [words count])
{
     while (NSNotFound != compTextRange.location)
     {
          ++curWordCount;

          compText = [compText substringFromIndex:
               compTextRange.location + compTextRange.length];
          compTextRange = [compText rangeOfString: curWord];
     }
}

return curWordCount;
```

To calculate the total time that a professor talked about a topic, Class Extractor adds up the total duration of that topic, as follows:

```
int totalDuration = 0;

for (int i to [allCMTimes count])
{
     totalDuration += [allCMTimes objectAtIndex: i].duration
}

return totalDuration;
```

To calculate the recency of a topic, Class Extractor compares the time since the topic was last discussed to the current time:

```
int timeDiff = curTime.start - lastDiscussionTime.start;
```

And combines them all together using weightings to calculate the overall importance of this topic:

```
int total = curWordCount * 60 + totalDuration*50 - timeDiff*10
```

This end calculation is what is used to determine the sizes of the clouds in Word Cloud.

Class Extractor uses two sets of APIs: IBM's Watson speech-to-text transliteration and Aylien's Concepts. After the user selects an audio recording for processing, Class Extractor chops up the file into five minute segments and uploads the files to Watson for speech-to-text transliteration. Watson provides powerful speech-to-text transliteration in the cloud, and returns the result of the transliteration as a Javascript Object Notation (JSON) file. The one downside to Watson's API is that it is streaming, which means that the length of the audio file is the amount of time it takes to upload it to IBM's servers. The API leverages CURL, and looks as follows:

```
curl -u USERNAME:PASSWORD -X POST --limit-rate 40000 --header
"Content-Type: audio/wav" --header "Transfer-Encoding: chunked" --
data-binary @/Users/elliot/Desktop/test.wav "https://
stream.watsonplatform.net/speech-to-text/api/v1/recognize?
continuous=true"
```

However, to execute this command in an Objective-C application, Class Extractor must use NSTask, which is an Apple-provided API for launching Terminal commands from Objective-C code. A boundary case that must be considered is the user attempting to upload a file that is not an audio file; it is easy to check if a file has a correct extension, but the actual content of the file should be verified to prevent any security vulnerabilities.

After receiving the JSON file from Watson, Class Extractor will iterate through the JSON, parsing out the transcript and appending it to a string. This algorithm is represented with the following pseudocode:

```
    NSDictionary* resultDict = [audioData JSON];
    NSMutableString* resultString;
    for (i to [resultsArray count])
    {
        [resultString appendString: [resultDict objectForKey:
          @"transcript"]];
    }
```

resultString is passed into Aylien. The Aylien API is a powerful set of NLP libraries that can perform concept abstraction to determine what was discussed in a string of text. The Aylien API is also a cloud API, accessed through Mashape's API vendor service. All Class Extractor has to do is send the string, and Mashape returns an array of three to seven concepts. Accessing Aylien is represented by the pseudocode below:

```
NSURLSessionConfiguration* sessionConfig = [NSURLSessionConfiguration
defaultSessionConfiguration]
    [sessionConfig setHTTPAdditionalHeaders: @{@"X-Mashape-Key" :
credentials, @“Accept" : @"application/json"}]

    NSURLSession* urlSession = [NSURLSession sessionWithConfiguration:
sessionConfig]
    NSURLRequest* urlRequest = [NSURLRequest requestWithURL: fullURL]

    NSURLSessionDataTask* dataTask = [urlSession completionHandler:
^(NSData* data, NSError* error) {
        if (!error)
          [[data JSON] parse]
    }];
```

As mentioned, Class Extractor accesses the network under two circumstances: when connecting to Watson and when connecting to Aylien. Class Extractor connects to both of them using HTTPS.

Class Extractor accesses Watson using a CURL command. NSURLSession, an Apple-provided class for initiating network connections, is typically used when making network requests, but in this case it cannot be used because it does not support all of the arguments that this particular CURL command needs. To remedy this issue, Class Extractor uses NSTask to launch the same CURL executable that Terminal launches when executing a CURL command, and passes in the appropriate arguments as required. Even though the user gives Class Extractor his or her full lecture recording (which may be an hour or more in length), Class Extractor chops up the file and asynchronously sends five minute audio segments to Watson because Watson cannot accept audio files that are longer than five minutes. Watson returns a JSON file, and, for security reasons, Class Extractor will verify that it is a valid JSON file before attempting to parse it. For each value that Class Extractor parses from the JSON file, Class Extractor will also verify that it is of the expected data type before using it in any way. These checks will ensure that the data Class Extractor is receiving is safe and secure.

Class Extractor also connects to Aylien using CURL, except that for this command, NSURLSession supports all of the arguments that Aylien requires. As such, Class Extractor will use this library because it is much more efficient: no context switches are needed as they are for using NSTask. Class Extractor sends a string of text, and Aylien returns a JSON file with one to seven topics that are discussed in the string. Similarly to how Class Extractor handles the Watson return values, Class Extractor will verify the JSON and all parsed types.

In either network case, there is the potential of network failure, and in both cases, failure will be handled the same way: as gracefully as possible. If there is no network connection before the connection is initiated, the user will be shown a notification alerting them to this situation and asking them to connect to the internet, or if the computer is connected, to try again in a few minutes. If the network connection fails during a call, Watson will delete all data received from any ongoing calls (but not ones that have completed) and alert the user, asking them to verify their connection and try again.