Malcolm Goldiner Components Doc

EasyRead Components Design

External API

MashApe - Webknox Words

MashApe is an API console and marketplace that allows you to use one API key to access all APIs using their platform. I chose to use the Webknox Words API to make a REST Request for the number of syllables in the input word. The data is returned in JSON and I parse that JSON to find the num array and then the syllables key.

Microsoft N-Grams

Microsoft N-Grams is a service that stores grams of words, up to 5 grams, that represent frequent search queries on Bing. They provide a RESTful API that returns JSON data to HTTP request directed at their endpoint. My project makes use of the bing body of searches from December 2013 and uses the conditional probability setting to query 3-grams in the system. This is used to evaluate a suggested word replacement.

External Libraries

WordNet (extjwnl) Extended Java WordNet Library
This project makes use of a WordNet Java Library to make API requests for a list of synonyms for each word when given an input of(Word, Part of Speech).

Stanford CoreNLP
Stanford CoreNLP provides a Java Library for parsing text and I use their sample code in its entirety to break down passages into their component sentences and words as well as get the part of speech for all of those words.

Components

Mashape Controller

This controller's function is to add syllable information, as well as keep track of the number of characters and total number of syllables in passages as a whole because they are necessary for the algorithms used to calculate Passage Grade Level.

Pseudocode

getNumSyllablesPerWord(Passage p, String word)

```
        p.numCharacters += word.length
        if word doesn't exist in db and is of length > 1
                getSyllableInformation
        else
                p.numSyllables += (fetch num syllables already in db)
end
```

// returns null if anything fails along the way, number of syllables won't be set and the system will try and fetch it again when it next encounters this word

getSyllableInformation(String word, Passage p)

```
        if word.length > 2
```

getSyllableInformation(String word, Passage p)

if word.length > 2
Asynchronous http request --> Mashape(word)
create a new word object
store num syllables from response in word
p.save
end

Interface PhraseValidator
To ensure future compatibility if we switch to Google N-Grams from Bing N-Grams this functionality is abstracted away behind an interface

required method
fetchFrequencies(String p)

MicrosoftNGramsValidator implements PhraseValidator

Pseudocode
fetchFrequencies(String p)
asynchronous http request
String strRes = response.toString()
//only response is the number corresponding to frequency in logarithmic form on success

if(strRes.length == 1)
return strRes
else
return 0
end

Possible Error: If this request fails the frequency will remain at it's default value and be ignored. The API always returns a number, no matter how infrequent the word is, so the only failure state is an invalid request. Invalid requests feature a full error message rather than the normal one number, thus the length check.
Parsing Controller
This controller encapsulates the recommended operations for parsing a string given on Stanford's Core-NLP website. The only additions are those needed to save the Part of speech, Sentences, and words to the database. The parsing of a new word also triggers a search for number of syllables. The saving of the passage with sentences now parsed triggers a determination of grade level. The call to parse is triggered by a UI element within the site and is surrounded in a try/catch. An error message is thrown if parsing fails, but it has no effect on site usability. In fact, users can continue to move around the site while parsing occurs in the background.

Pseudocode

parse(Passage p, String t)
performStanfordParseOperations

for every sentence parsed
passage.sentences.add(sentence)

for every word in that sentence
if length > 1 && first character is a letter
if we haven't queried already for this word already
MashapeController.getSyllableInformation()
passage.numWords++
if(passage.grade == 0)
PassageAnalysisController.generateSuggestions()

if(passage.grade == 0)
PassageAnalysisController.generateSuggestions()

p.save
p.determineGradeLevel
end

Passage Analysis Controller
This controller is responsible for running through our grade level determination flow, which uses the Flesch-Kincaid Readability Tests, Automated Readability Index, and Coleman-Liauu Index

Major Algorithms Used

Felsch-Kincaid Readability Index Grade Level from Wikipedia

$$0.39\left(\frac{\text{total words}}{\text{total sentences}}\right)+11.8\left(\frac{\text{total syllables}}{\text{total words}}\right)-15.59$$

Automated Readability Index from Wikipedia

$$4.71\left(\frac{\text{characters}}{\text{words}}\right)+0.5\left(\frac{\text{words}}{\text{sentences}}\right)-21.43$$

Coleman-Liau Index from Wikipedia

$$CLI = 0.0588L - 0.296S - 15.8$$

Pseudocode

double convertToGrade(double age)
if(age <= 6) return 0;
else if(age > 18) return 13;
return age - 6;
end

double gradeResolution(double grade)
if(grade > 12) return 13;
else return grade;
end

Implementation Note: After every change to grade level algorithms test against k5Learning.com Reading Comprehension sample sheets.
double averageAge(SimplePassage p)
double average = 0;
int count = 0;
for(Sentence s : p.sentences)
for(Word w : s.words)
if(this is not a stop word)
if(w.ageOfAcquisition greater than Kindergarten(6))
average += w.ageOfAcquisition;
count++
return average/count;
end

determineGradeLevel(SimplePassage p)

double ARI = calculateARI(p);

```
        double ARI = calculateARI(p);

        double FK = calculateFKScore(p);

        double CL = gradeResolution(calculateCLScore(p.text, p.numWords));

        if(p has more than 100 words)
                p.grade = (int) Math.round((ARI + FK + CL) / 3);
        else
                p.grade = (int) convertToGrade(averageAge(p));

        p.save();
end
```

SimplePassageController and ApplicationController

SimplePassage Controller performs all the rendering of the various pages with the site that aren't the index/header

Application Controller - Responsible for rendering the header and footer of every page as well as the index.

Pseudocode for these controller is omitted as they don't perform any algorithmically significant operations and are solely there to pass objects around and perform rendering + routing functions.

EasyRead will not export any public facing APIs.

Data Structure Choices

- For most of this project, a HashSet is my data table of choice. Anything that is persisted to the database or iterated through frequently is stored in a HashSet. Anything that must be accessed sequentially during the sequence of one method and then can be destroyed is in an ArrayList.
- The reasoning behind that is mostly that Play Framework uses Scala as a scripting language in the page templates and Scala is excellent at operations on Lists. Keeping my data in collections that lend themselves to both the static templates and normal Java Web Conventions has led to a Set and List heavy program.