

Code Reviews

Team: Pear
Jonathan Pan
Richard Liu
Michael Gonzalez

April 23, 2017

Project Span: January 2017 - April 2017

Reading Plan	
<ol style="list-style-type: none"> 1. Checklist 2. Use Cases 3. Description 4. Source Code <ol style="list-style-type: none"> a. OfficeHours.rb b. RateMyProfessor.rb c. SjsuCourses.rb 5. MVC Diagram 6. UML Class Diagrams 7. Data Models <ol style="list-style-type: none"> a. ER Diagram b. Relational Schema 	

CheckList	
<input type="checkbox"/> Documentation <input type="checkbox"/> Testing <input type="checkbox"/> Error Handling <input type="checkbox"/> Thread Safety <input type="checkbox"/> Performance	

Use Cases	
<ul style="list-style-type: none"> • Gather, organize, and present a list of current available SJSU CMPE courses to the user. • Gather, organize, and present professor office hours to the user. • Gather, organize, and present RateMyProfessors stats for each professor to the user. 	

Description	
OfficeHours	This script uses Google Sheet's API to fetch data from the Google spreadsheet provided by SJSU's CMPE department for professor office hours. For each professor/row in the

	spreadsheet a dictionary is created to store every category of information.
SjsuCourses	<p>This script uses the Nokogiri gem to scrape the SJSU course catalog for courses. Under the class SjsuCourses, one function scrape(department) scrapes its corresponding URL. The main URL contains links for each course. The number of links is usually around the 100 range. Since the number of HTTP requests is somewhat large, the run time of the function can be extremely slow. However, with the use of Ruby threads, one for each HTTP request, the run time is significantly sped up. Sometimes the requests to the SJSU website can time out so a queue of timed out requests is stored. Each link is parsed for course information and stored in a dictionary. At the end of the function the leftover unparsed links are looped in a queue until every link is parsed without exception.</p>
RateMyProfessor	<p>This script also uses the Nokogiri gem. It scrapes the RateMyProfessor website for professor information and reviews. It also uses threads to speed up the run time of the HTTP requests and stores information in dictionaries.</p>

Source Code

OfficeHours.rb(LOC:50)

```
5 class OfficeHours
6   def scrape
7     apikey = '?key=AIzaSyA6xICFHA0oPk5oJ_ae0DAeTBSkgYtNywM'
8     baseUrl = 'https://sheets.googleapis.com/v4/spreadsheets/1hFJbOwV10iOHMC_TnoD-TonyuePtvRK00kI1M5rjM70/values/'
9
10    o = OpenSSL::SSL::SSLContext.new
11    o.verify_mode = OpenSSL::SSL::VERIFY_NONE
12
13    # columns = JSON.parse(HTTP.get(baseUrl+'1%3A1'+ apikey, ssl_context: o))['values'][0]
14    names = JSON.parse(HTTP.get(baseUrl+'A2%3AA'+apikey, ssl_context: o))['values']
15    days = JSON.parse(HTTP.get(baseUrl+'B2%3AB'+apikey, ssl_context: o))['values']
16    times = JSON.parse(HTTP.get(baseUrl+'C2%3AC'+apikey, ssl_context: o))['values']
17    room = JSON.parse(HTTP.get(baseUrl+'E2%3AE'+apikey, ssl_context: o))['values']
18    phoneNumber = JSON.parse(HTTP.get(baseUrl+'F2%3AF'+apikey, ssl_context: o))['values']
19    email = JSON.parse(HTTP.get(baseUrl+'G2%3AG'+apikey, ssl_context: o))['values']
20
21    all_professors = []
22
23    size = names.size
24
25    size.times do |i|
26      professor = {}
27
28      unless names[i].first.nil?
29        professor['name'] = names[i].first.gsub("\n", ' ').squeeze(' ')
30      else
31        next
32      end
33
34      professor['days'] = days[i].first.nil? ? "N/A" : days[i].first
35      professor['times'] = times[i].first.nil? ? "N/A" : times[i].first
36      professor['room'] = room[i].first.nil? ? "N/A" : room[i].first
37      professor['number'] = phoneNumber[i].first
38      professor['email'] = email[i].first
39      all_professors.push(professor)
40    end
41
42    # all_professors.each do |p|
43    #   puts "Name: #{p['name']}, Days: #{p['days']}, Time: #{p['times']}, Room: #{p['room']}, Number: #{p['number']}"
44    # end
45
46    return JSON.pretty_generate(all_professors)
47  end
48 end
```

RateMyProfessor.rb(LOC:110)

```
1 require 'nokogiri'
2 require 'open-uri'
3 require 'openssl'
4 require 'json'
5
6
7 class RateMyProfessor
8   def scrape
9     main_url = "https://www.ratemyprofessors.com/search.jsp?queryBy=teacherName&dept=computer+engineering
10      &queryoption=HEADER&query=san+jose+state+university&facetSearch=true"
11     base_url = "https://www.ratemyprofessors.com"
12
13     d = Nokogiri::HTML(open(main_url,{ssl_verify_mode: OpenSSL::SSL::VERIFY_NONE}))
14
15
16     # Array of pages
17     pages = Array.new
18     pages.push(main_url)
19
20
21     # Next Pages
22     d.css("div#searchResultsBox > div.result-pager > div > a[href]").each do |page|
23       pages.push(base_url + page['href'].to_s)
24     end
25     pages.pop # Removes the last link because the last link is 'Next' which is already included
26     #puts pages
27
28
29     # Link to professors
30     links = []
31     pages.each do |page|
32       p = Nokogiri::HTML(open(page, {ssl_verify_mode: OpenSSL::SSL::VERIFY_NONE}))
33       p.xpath('//*[id="searchResultsBox"]/div/ul/li/a/@href').each do |link|
34         links.push(link)
35       end
36     end
37     #puts links
38
39
40     threads = []
41     all_professors = []
42
```

```

42
43
44 # Parse Each Page
45 links.each_with_index do |link, index|
46   pageurl = base_url + link
47   threads << Thread.new do
48     begin
49
50       # Open Page
51       page = Nokogiri::HTML(open(pageurl, {ssl_verify_mode: OpenSSL::SSL::VERIFY_NONE}))
52
53
54       # Get Name, Quality, Difficulty
55       fname = page.xpath("//div[@class='right-panel']/div[1]/div[@class='result-info']/
56         div[@class='result-name']/h1[@class='profname']/span[@class='pfname']/text()").first.to_s.strip!
57       lname = page.xpath("//div[@class='right-panel']/div[1]/div[@class='result-info']/
58         div[@class='result-name']/h1[@class='profname']/span[@class='pname']/text()").first.to_s.strip!
59       quality = page.xpath("//div[@class='rating-breakdown']/div[1]/div[1]/div[1]/div[1]/div/div/text()").to_s
60       difficulty = page.xpath("//div[@class='rating-breakdown']/div[1]/div[1]/div[2]/div[2]/div/text()").first.to_s.strip!
61
62
63       # Get Comments
64       comment_text = []
65       1.times do
66         page.css('td.comments > p.commentsParagraph').each do |t|
67           comment_text.push(t.text.strip.to_s)
68           break
69         end
70       end
71
72
73       # Get Comment Type
74       comment_type = []
75       page.css('span.rating-type').each do |t|
76         comment_type.push(t.text.strip.to_s)
77         break
78       end
79
80       professor = {'fname' => fname, 'lname' => lname,
81                   'difficulty' => difficulty, 'quality' => quality,
82                   'comment_text' => comment_text, 'comment_type' => comment_type,
83                   'url' => pageurl}
84
85       all_professors.push(professor)
86
87       rescue RuntimeError => e
88         puts "Missing Professor"
89       next
90     end
91   end
92 end
93 threads.map(&:join)
94
95
96 # Sort professors by first name
97 all_professors = all_professors.sort { |a,b| a['fname'] <=> b['fname']}
98
99
100 # # Data To Console
101 # puts JSON.pretty_generate(all_professors)
102
103 # # Data To File
104 # file = File.new('rmp_output.json', 'w')
105 # file.write(JSON.pretty_generate(all_professors))
106 # file.close
107
108 #render text:
109 return JSON.pretty_generate(all_professors)
110 end
111 end

```

SjsuCourses.rb(LOC:145)

```
1 require 'nokogiri'
2 require 'open-uri'
3 require 'json'
4 require 'http'
5 require 'pp'
6 require 'thread' # queue
7
8 class SjsuCourses
9
10   def scrape(department)
11
12     # Links : sjsu.edu
13     base_url = 'http://info.sjsu.edu'
14     case (department)
15     when "CMPE"
16       courses_url = 'http://info.sjsu.edu/web-dbgen/schedules-fall/d67626.html'
17     when "SE"
18       courses_url = 'http://info.sjsu.edu/web-dbgen/schedules-fall/d67627.html'
19     end
20
21     # Lists used in this method
22     courses = []
23     links = []
24     threads = []
25
26     # Exceptions queue for retrying
27     @exceptions = Queue.new
28
29     # Open main page
30     main_page = Nokogiri::HTML(open(courses_url))
31
32     # Scrape all links from main page
33     main_page.css("table > tr > td > a[href]").each do |page|
34       link = page['href'].to_s
35       if link[0..3] == "/web"
36         links.push(link)
37       end
38     end
39
40     # Open each link with a new thread
41     links.each_with_index do |link, index|
42
43       page_url = base_url + link
44
45       threads << Thread.new do
```

```

46
47     # Scrape page
48     course_info = scrapeCourse(department, page_url)
49
50     # Push dictionary to list of courses
51     if !course_info.nil?
52         courses.push(course_info)
53     end
54
55 end
56
57 end
58
59 # Synchronize and join threads
60 threads.map(&:join)
61
62 # Retrying all caught exceptions
63 while !@exceptions.empty?
64     page_url = @exceptions.pop
65     course_info = scrapeCourse(department, page_url)
66     if !course_info.nil?
67         courses.push(course_info)
68     end
69 end
70
71
72 # Sort courses by ID
73 courses = courses.sort { |a,b| a['ID'] <=> b['ID'] }
74
75
76 # pp courses
77 return courses
78 end
79
80 def scrapeCourse(department, page_url)
81
82     # Dictionary to store course information
83     course_info = {}
84
85     # Open link of current course
86     begin

```

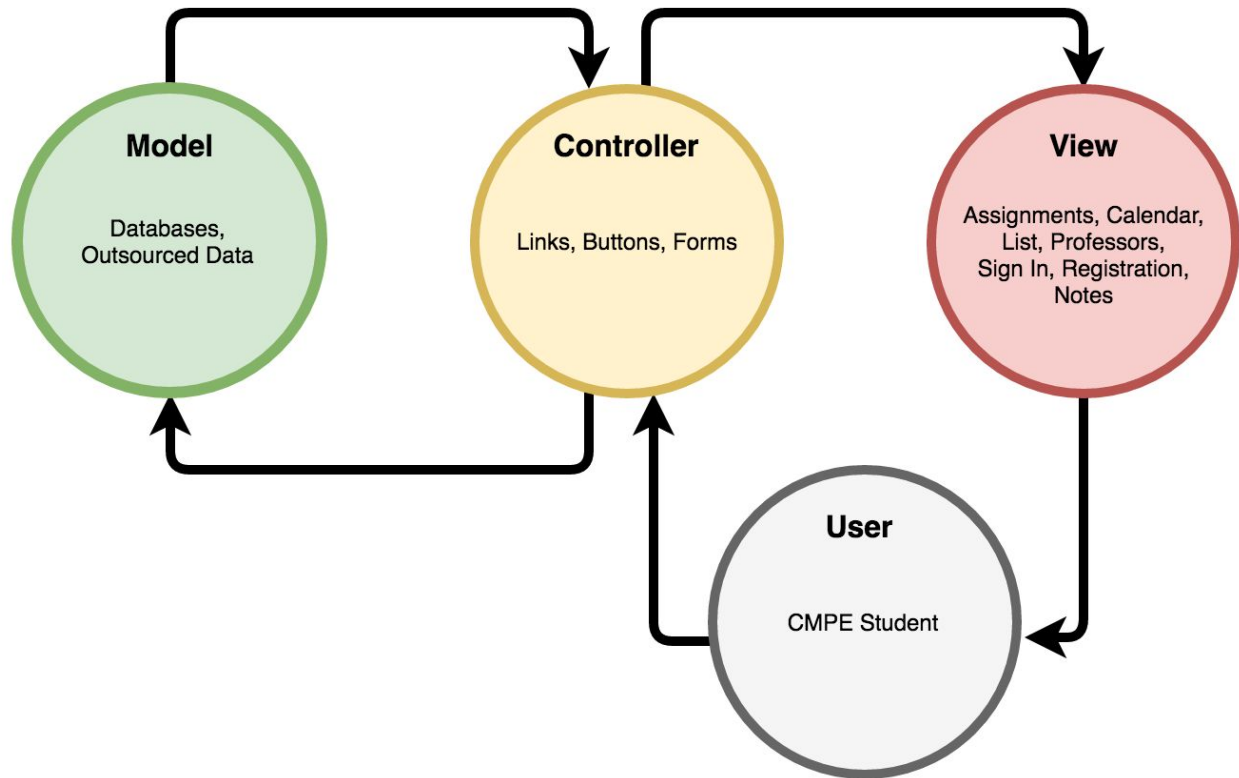


```

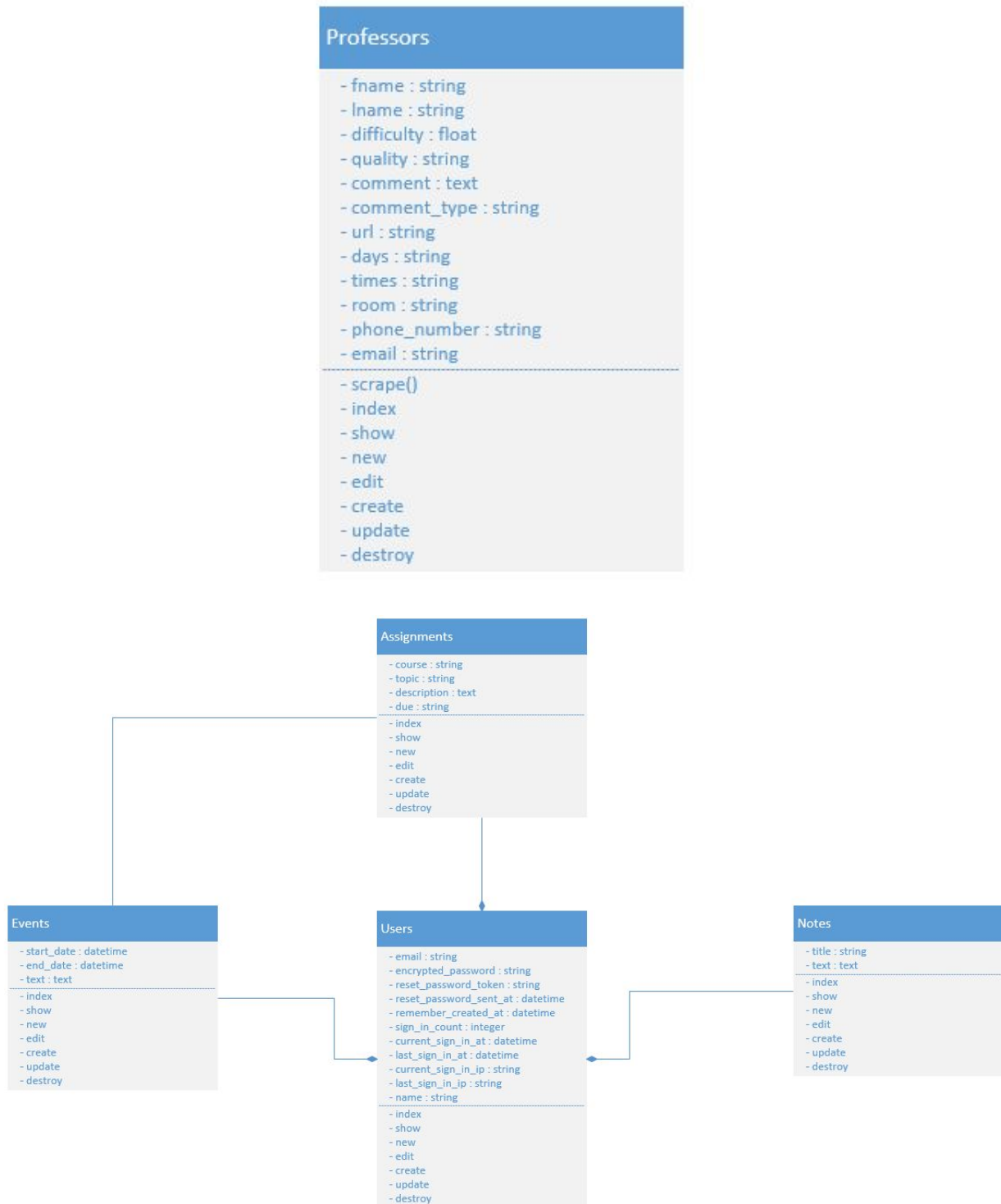
87     page = Nokogiri::HTML(open(page_url))
88   rescue Errno::ETIMEDOUT, Timeout::Error
89     @exceptions << page_url
90     return nil
91   end
92
93   # Grab course ID e.g. "CMPE 030"
94   page.css("h3").each do |h3|
95     case(department)
96     when "CMPE"
97       if h3.text[0..3] == "CMPE"
98         course_info['ID'] = h3.text
99       end
100    when "SE"
101      if h3.text[0..1] == "SE"
102        course_info['ID'] = h3.text
103      end
104    end
105  end
106
107  # Scrape course information and add to dictionary
108  page.css("table > tr").each do |tr|
109    data = []
110    tr.css("td").each do |td|
111      data.push(td.text)
112    end
113    case data[0]
114    when "Schedule"
115      course_info['Schedule'] = data[2]
116    when "Title"
117      course_info['Title'] = data[2]
118    when "Section"
119      course_info['Section'] = data[2]
120    when "Code"
121      course_info['Code'] = data[2]
122    when "Units"
123      course_info['Units'] = data[2]
124    when "Type"
125      course_info['Type'] = data[2]
126    when "Enrollment"
127      course_info['Enrollment'] = data[2]
128    when "Days"
129      course_info['Days'] = data[2]
130
131      when "Time"
132        course_info['Time'] = data[2]
133      when "Dates"
134        course_info['Dates'] = data[2]
135      when "Location"
136        course_info['Location'] = data[2]
137      when "Instructor"
138        course_info['Instructor'] = data[2]
139      end
140    end
141    return course_info
142  end
143
144 end

```

MVC Diagram

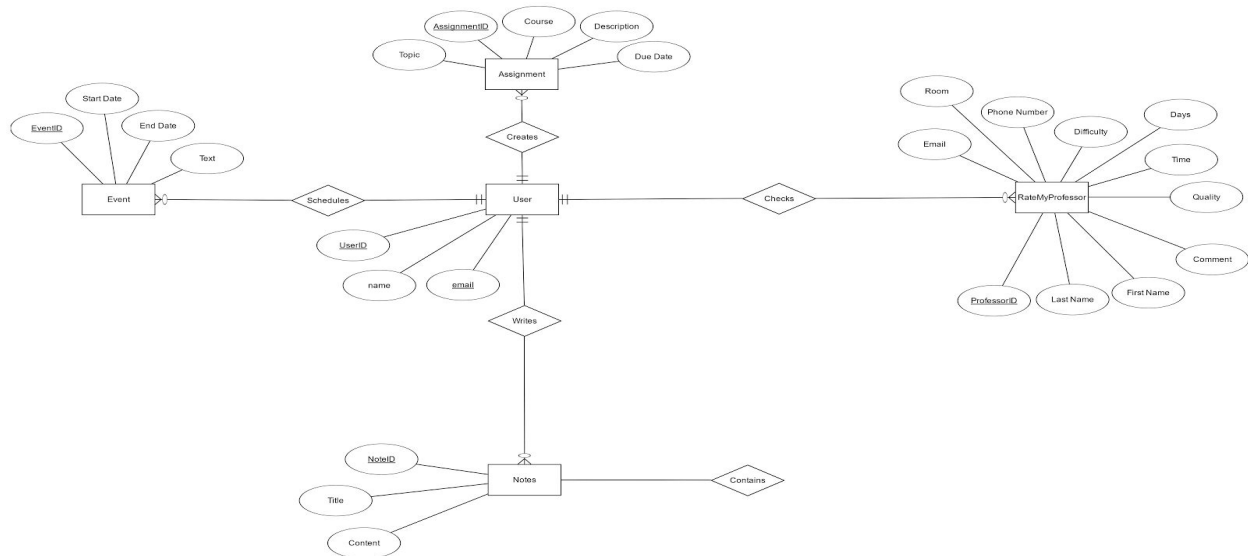


UML Class Diagrams



Data Models

Conceptual Data Model - ER Diagram



Logical Data Model - Relational Schema

