SWEN30006 Project 3

Putting It All Together

23rd September 2015

An Introduction to the Project

The **aim** of the project is to create a Rails application, called **The Digest**. In doing so, you will be required to design **The Digest** by developing a *Software Architecture*, *Class Models* and their associated *Sequence Diagrams*.

There are be two submissions for this project:

- **Part A** - will be a design document that specifies your system architecture and UML design models, and
- **Part B** - will be the actual implementation of your design together with reflection of your design and implementation.

Unlike the first two projects, the third project is to be completed in teams of 3. You are required to choose a team, and then to submit your team by Friday, the 25th of September. If you cannot find a team to join then please submit your name and student number by the 25th of September and you will be allocated to a team.

> **On Plagiarism**
>
> We take plagiarism very seriously in this subject. You are not permitted to submit the work of others under your own name. This is an **individual** project. More information can be found here: (https://academichonesty.unimelb.edu.au/advice.html).

Project Overview

The Digest is an application that allows subscribers to register, nominate interests, browse news articles, and receive a digest of news articles by email.

Your application must be a **Rails Application** written in **Ruby**.

Requirements

The following is a list of requirements for **The Digest**. These requirements are given in the style of a textual requirements specification.

The requirements are broken down into sections that describe the features that **The Digest** must deliver. The requirements are not *complete* (see the lecture 3 on requirements analysis for the meaning of *complete*) and you may make reasonable assumptions where this is the case. Where you make an assumption about the requirements, you must state the assumption that you made as a part of your design submission.

Section 1 - Registering Subscribers

Overview A subscriber must register an account with the news service application before they can use the application. Once registered and logged in, subscribers can:

- Create a list of *interests*. These *interests* are used to filter stored articles (see *Section 2* below).
- Perform searches on the stored articles (see *Section 4* below).
- Subscribe to have a digest of news articles sent to their nominated email address (see *Section 5* below).
- Edit their personal details.

[**S1**] As a minimum, subscribers must to provide the following personal details to register:

- A *username*;
- The subscriber's first name and their family name;
- A password for the account (minimum 8 character length); and
- An email address to which news digests and any other correspondence will be sent.

[**S2**] Subscribers must be able to edit their *personal details*, with the exception of their *username*, which must not be editable.

[**S3**] Subscribers must be able to edit their news *interests*. When a subscriber registers an account, they should initially have no news *interests*.

[**S4**] Subscribers must be able to register to have a *News Digest* sent to their email and de-register to no longer receive *News Digest* emails.

[**S5**] Each subscriber may only register a single email address with the application.

Section 2 - Scraping and Viewing Articles

Overview

The application must scrape articles from news feeds. Once scraped, these articles can be viewed by subscribers when they are logged in. A special *administrator* request can be sent to the application using the URL path */admin/scrape/* to run the scraping process.

[**A1**] News articles must be scraped and stored from both JSON feeds and RSS feeds as follows:

RSS Feeds

News articles from all the following RSS feeds must be scraped and stored:

- The ABC: http://www.abc.net.au/services/rss/
- The Age: http://www.theage.com.au/rssheadlines
- The Herald Sun: http://www.heraldsun.com.au/help/rss
- The Sydney Morning Herald: http://www.smh.com.au/rssheadlines
- SBS: http://www.sbs.com.au/news/rss-list

JSON Feeds

News articles from all the following JSON feeds must be scraped and stored:

- The New York Times http://developer.nytimes.com/
- The Guardian http://open-platform.theguardian.com/access/

[**A2**] Duplicate news articles found during scraping must not both be stored. Only one of the duplicates should be stored. Two articles are considered to be *duplicates* if they have the same title.

[**A3**] All of the news feeds must be scraped when the URL path /admin/scrape/ is requested. This URL must not require a login.

[**A4**] Subscribers must be able to list the stored articles relevant to their *interests*. The articles must be sorted by article publishing date and time (most recent first) when displayed.

[**A5**] Subscribers must be able to list all of the articles stored in the application, in pages of 10 articles at a time, sorted by article publishing date and time (most recent first).

Section 3 - News Tags

Overview

Tags are used for a number of functions including searching, sorting and compiling weekly email news digests. All articles must be processed once only to automatically add tags.

Tagging is a feature of *The Digest* that is deliberately left open-ended.

[**T1**] Every article must have tags that are generated using five different methods of tag generation. Tags must be obtained from an article's fields, for example, the article title and the article description.

[**T2**] Two of the five tag generation methods required by **T1** must derive tags for the *semantic* content of an article.

The 'semantic' content of an article include concepts like *article sentiment*, *article keywords* and *article topic*. Tags used in project two such as *article categories* from news feeds and *nouns* can also be used to tag articles in this project.

Note: *A list of gems and API's that can be used to generate tags, and examples of how to use the resources, are given in the accompanying resources file.*

[**T3**] The scraping module and the tagging module must be independent. That is, the scraping code for each source must not call the tagging code and the tagging code must not call the scraping code.

[**T4**] The processing performed to generate tags must be the same for all article sources.

Section 4 - Searching for News Articles

Overview

Subscribers must be able search the news articles stored by **The Digest**. A search is composed of a combination of keywords that can be found in an article's attributes.

[**F1**] A subscriber must be able to use the web interface to search stored news articles. A *search* is composed of one or more keywords entered by the subscriber.

[**F2**] An article *matches a keyword* if the keyword occurs as: - One of the article *tags*; or - One of the words in the *article title*; or - One of the words in the *article description*; or - A substring of the the *article source*. An article *matches a search* if and only if the article matches all of the keywords in the search.

[**F3**] Every article returned by a search has a weighting. The weighting of a search keyword in an article returned by the search is determined by where the keyword appears. If the keyword appears:

- In the article tags then assign the keyword a weight of 4.0

- In the words in the article title then assign the keyword a weight of 3.0
- In the words in the article description then assign the keyword a weight of 2.0
- In the article source then assign the keyword a weight of 1.0.

The weighting of the keyword is the sum of all of the places that it appears. Each appearance is only counted once. For example, if a keyword appears in both the title and description of an article, that keyword has a weighting of 5.0. If a keyword appears three times in the description and nowhere else then the weighting of the keyword is still only 2.0.

The weight of an article returned by a keyword search is the sum of the weights of all of the keywords in the search.

[**F4**] Articles returned by a search must be presented in order of descending weights (see **F3** above).

[**F5**] Where articles matching the search have the same weighting, those articles must be sorted by the published date and time (most recent first).

Section 5 - Compiling and Emailing a News Digest

Overview

Subscribers must be able to request that a news digest of articles matching their *interests* be sent to their registered email address. A special *administrator* request can be sent to the application using the request *GET /admin/email/* that emails **all** subscribers that requested a news digest by email.

A news digest is composed of up to ten articles matching the subscriber's *interests*. Matching articles are ordered in the digest by published date and time (most recent first).

[**E1**] A subscriber must be able to request that an email news digest be sent to their registered email address from the web interface.

[**E2**] An email news digest must be sent to every registered subscriber that requested emailed digests when the URL path /admin/email/ is requested. This URL will be used for testing and must *not* require a login.

[**E3**] Articles that have been sent to the subscriber in a previous news digest should not be included again in newly requested news digests.

[**E4**] If there are no articles matching the subscriber's *interests* or all matching articles have already been included in news digests, the subscriber must be sent an email advising that no articles matching their *interests* are available.

[**E5**] The email must be sent to the subscriber's email address, using the *Mandril* mailing service from Mailchimp, using the Ruby gem (available at https://rubygems.org/gems/mandrill-api). The API documentation and examples of using Mandril can be found at https://mandrill.zendesk.com/hc/en-us/articles/205582357-Using-the-Mandrill-Ruby-Gem.NDR

Relating Tags To Interests

It is up to you to decide what a 'relevant' article is for a user's interests. For example, if I say as a user I'm interested in Coffee then you may return me only articles tagged as "Coffee" or you may return me articles tagged as "coffee","Coffee","cafe","coffee machine", or "brunch". It is part of your design challenge to decide what approach you will take here, and how you will choose to implement it.

Software Design - Submission 1

Your first task is to create a *design* for **The Digest**. You will need to analyse the requirements above, group the requirements into modules, determine how requirements can be implemented and if any code from projects one and two can be re-used, and then specify your design using UML.

For your initial software design you will be required to submit 4 design documents, collated into **one** PDF. They are as follows:

- **A Component Diagram**
  - Your component diagram should specify the software architecture for your application and should include all of the major top-level components (Models, Views, and Controllers) as well as the sub-components inside each of these.
  - Your component diagram should also specify all interfaces between the components and the methods available on them
- **Class Diagrams**
  - Detailed class diagrams of **all** models and controllers.
- **Sequence Diagrams**
  - You need to produce three sequence diagrams that specify how your design will implement the following core functions:
    * The *scraping* of articles;
    * The *tagging* of articles; and
    * The *compiling* and *emailing* of a news digest.
  - These must be low-level sequence diagrams, showing how you intend to implement the features above in your class model.
- **A Short Design Rationale**
  - Give a short introduction to your design, specifying your control structure (centralised, distributed) and any assumptions that you have made.

In your design you should apply the design theory that you have learn so far in class, whilst also following the rails convention of thicker models and thinner controllers. That is, most of the 'business' logic should be in the models and the controllers should act as facilitators.

All of your design models **must** be provided in a single PDF document. Please do not submit multiple files, visio documents, word documents or diagrams in other formats.

Implementation - Submission 2

For your implementation you will be building a Rails Application based on the design your team has produced for submission 1. This Rails application should be entirely self contained and must use an 'sqlite' database.

It is **strongly** recommended that you use *Git* for version control, for example, using the free private repositories available on bitbucket (https://bitbucket.org/). Using a version control system makes it much easier to work in a team environment on a Rails project. If you do use so, please provide us with access to the repository by adding Mat (on github: matblair, on bitbucket: mblair).

Your code will be marked on quality and maintainability, therefore you should endeavour to make follow good object oriented practices and Ruby code style wherever possible. Gems like 'rubocop' may help with this process.

Along with your Rails application, you are also required to submit, as a group the following:

- A Reflection on your Design and Implementation, including:

  - What changes, if any, were made to your original design, along with updated design documentation
  - Aspects you found challenging
  - Your thoughts on the value of spending time and effort developing a thorough design prior to implementation.

- A README on how to install and use your Rails application, regardless of whether or not it differs from the standard procedure outlined in submission requirements below.

Important As an individual we are also asking you to submit a short (up to 400 words) explanation outlining the contributions of your team members and yourself to the final project along with any troubles that were faced in development that you believe we may need to hear about (this is not required if you have used Git as we can see the commit history).

> **Note**
>
> With any group project there tends to be a few groups that run into issues that, for some reason or another, impact upon their ability to deliver the project. If any issue has arisen in your group, we want to know about it sooner, rather than later, so that there is appropriate time to fix the situation.
>
> If you do not tell us when the issue arises, but instead wait until your project demonstration, there is a lot less flexibility available in how we can handle it.

Submission Instructions

Project submission will be enabled through the LMS. We will require you upload a zip file of your rails application and PDF of your design documentation to the submission links we provide.

Software Design Submission

You should submit a single PDF file to the provided *Turnitin* link by the due date. Only one member from each group needs to submit the file.

Implementation Submission

You must submit your rails application, along with the reflection document, in a zip file to the provided LMS submission page. Only one member from each group needs to submit the file.

The rails app you provide will be run as follows:

```
bundle install
rake db:create
rake db:migrate
rails s
```

You **must** include a README file in your project detailing how to install and execute your Rails application regardless of whether or not it follows the standard procedure outlined below.

If your program does not run using the standard sequence below or using the procedure outlined in your README file then you will lose **1 mark** and we will attempt to fix. If we cannot make your rails application run at all then we will not award marks for the correctness criteria.

Marks

This project will account for 15 marks out of the total 100 available for this subject. 9 of these marks will come from the design submission, and 6 from the implementation.

Design Document Criteria

Criterion	Percentage
Component Diagram	3%
Sequence Diagrams	3%
Detailed Class Diagrams for all Controllers and Models	2%
Fitness for Requirements	1%

All UML submissions will be marked on their correctness, how well the problem has been deconstructed and cohesion with other diagrams.

The final 1% mark relates to how well your design captures and implements the project requirements. That is, does it look like your design will fulfil all requirements if it is implemented.

Implementation Criteria

Criterion	Percentage
Functional Correctness	3%
Reflection on Design Adherance	2%
Code Quality and Maintainabilty	1%

Functional correctness will be marked on your project's compliance to the requirements.

Your reflection must comment on how well your implementation adheres to your original design implementation. If you have changed your design in your final implementation, you must include a new class diagram and explanatory reasoning as to why you made those changes to receive any marks.

Code Quality will be marked on following good object oriented principles, with no unnecessary code duplication and good functional decomposition.

Submission Dates

Submission 1 is due at **11:59 p.m. on Sunday, 11th of October.** Submission 2 is due at **11:59 p.m. on Sunday 25th of October.**

Any late submissions *will incur a 1 mark per day penalty* unless you have supporting documents. If you have any issues with submission, please email Mat at mathew.blair@unimelb.edu.au, or Ed at edmundak@unimelb.edu.au before the submission date.