# SWEN30006 Project 2

Tagging Data

Semester 2 - 2015

### The Task

Your task is to develop an application using **Ruby on Rails**, called **The Digest**,that allows users to sign up to your service and view news articles relevant to their interests. This application will form the basis of your Project 3 service, which will extend this application to automate the process of emailing users a weekly digest.

This project will build on the work you have done in Project 1, using the scrapers you have already developed, in addition to new scrapers, to ingest data from a variety of sources and persist this in your database. You will then need to tag this data based on certain attributes.

Finally, you will provide the facilities for Users to sign up to your service and register their interests. Users should then be able to view articles of interest to them.

You should ensure you have completed Workshop 2 before beginning this project, as this workshop will provide a significant amount of assistance for completing Project Two. We will be providing a solution to Project One with this project specification and a solution to Workshop 2 on Monday the 7th of September 2015.

### 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).

## Requirements

We will now outline the requirements of the features you must implement in this project. Note, some aspects have been deliberately open to interpretation.

### Scraping and Storing Data

You are required to scrape news article data from at least **six** news sources for this application, so that you have sufficient data for tagging purposes. You may extend this number to include

as many sources as you like. While you are not required to use the code developed for Project One, it is strongly recommended that you use the importers and news articles that you wrote for Project One as a guide to designing your model layer, which consists of both the database and the model classes.

In this project you will need to store all of your News Article data in your Rails model layer (ie. the database). Unlike Project One, you will not be required to subclass your News Article class. Subclassing model classes that are stored in the database requires that the Articles all share the same data structure (and thus are only extended with additional or overriding methods), or, using multi-table inheritance. This is not an easy task to implement and therefore **not** required.

Instead, you should consider what an appropriate model layer structure would be for your News Article classes. There are a number of design decisions you can make that will allow for different degrees of flexibility in your model, and we encourage you to think about the different ways of achieving this before implementing it. You should consider what attributes are model objects in their own right (for example a news article's source) and what are simply attributes (for example a title).

Regardless of what structure you choose to store the data in, you must persist the following information at a minimum for each article:

- The source
- The title
- The date of publication
- The summary
- The author
- Associated Images
- The link to the full article

It should be noted that to tag articles with accuracy for your users, the more data the better. You will not be penalised if you extend on this data to provide more accurate tagging, or a better user experience.

You are required to include, a button that manually triggers a scrape of all sources for any new data since the last scrape. When scraping data, you must ensure that you do not save articles that you have already scraped. That is, if you already have an article in the database, do not save it again. This should be done by either using validations, or looking for existing duplicate articles in the database when scraping.

It is up to you to decide how much similarity is required to be considered a duplicate article, but you must at least use the Article title or date as a determining factor.

### Sources

As in Project One you are required to scrape data from a number of News Sources. We will be requiring that you scrape at least one of the below JSON sources and two of the RSS Feeds.

Therefore, we will need a variety of news sources to be able to deliver a comprehensive weekly digest. We will be requiring that you scrape at least **one JSON API and two**

**RSS sources**. The remaining four sources can be in any format and from any site provided that the content is appropriate and in English.

> **Beware** if you pick your own sources, the content must be valid News Articles, not discussion board posts or twitter feeds. Further, not all sources are of the same calibre, some will include malformed data that will make your project difficult. Therefore, you should check your sources thoroughly before including them in your project. We will not be assisting in dealing with data from poorly chosen sources.

### RSS Feeds

Below is a selection of RSS feeds for local news sources. You may pick any of the RSS feeds offered by these news articles.

- 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 Sources

For sources that require a search term, you should pick a term that is relevant to your interests. The content of the news articles is not important at this stage.

- The New York Times
  - Instructions: http://developer.nytimes.com/
  - Description: The New York Times offers a wide range of news apis to search articles, though it is primarily focussed on US news so will not have much local content.
- The Guardian
  - Instructions: http://open-platform.theguardian.com/access/
  - Description: Similar to the New York Times, The Guardian offers API access to all of their content from 1999 onward. They provide free access to their developer api for up to 12 calls a second.

### Tagging Data

People often wish to tag their interests and use those tags to find interest of them.

You are required to tag your News Articles so that Users can find all Article's in a certain category and similar articles. There are two design considerations that need to be made:

1. How you determine the tags relevant to an Article.

2. How you persist that data.

To persist tag data we suggest the use of the 'acts-as-taggable-on' gem. This gem creates the appropriate database tables and adds a number of methods that make it easier for us to provide tagging functionality to users. We recommend you view the documentation here: https://github.com/mbleigh/acts-as-taggable-on. You **do not** have to use this gem if you do not want to, however it will likely be simpler than writing a similar implementation yourself.

To determine what tags are appropriate for each given Article, you should look first to the content available in your data's source. A number of the provided data sources will include a 'category' or 'section' field that offers a good starting point. From there, you should review the data provided by each of your sources and see if there are any more identifying features that could contribute meaningful tags. You might, for example, decide to tag all proper nouns as meaningful attributes (using regular expressions to extract them from the summary). You do not need to demonstrate very complex logic here, but all articles imported into your system must have some form of tags.

You should be tagging data as it is imported, that is, you should modify your importers from Project 1 to tag the article before it is saved. It is important to have tags, but we will not be looking carefully at the tags you are generating in this project. As long as you are using a sensible method to generate the tags, and that you are including tags for all article, you will pass this criteria.

### User Signup

You are required to create the functionality to allow Users to sign up to your site and log in securely (using username and password). This will require the creation of a User model, which is persisted in the database. To do this, we suggest you use the inbuilt 'has-secure-password' functionality offered by Rails 4. This is the same technique that was used in Workshop 2 - Wordgram, to create the user authenticated. Other than this, your User model should have the following attributes:

- first_name
- last_name
- bio
- email
- username
- password_digest
- interests

You are required to include validations for, at a minimum, first_name, last_name, email, username and passwords. These validations should validate the presence of these attributes, and that the values provided are sensible (that is, that the email field is actually an email, that the first_name is alphabetic).

You **must not** store user passwords in plain text. Instead you must store a password digest in the user model.

User interests should be an array of categories or tags that a user is interested in. You must also provide the functionality for User's to delete their accounts.

### Authentication and Access Control

You must implement session based authentication in your Application, as demonstrated in **Workshop 2 - Wordgram**. You can define either a Model or Helper Module to accomplish this, but you must demonstrate good object oriented design considerations when making your decision. You **must not** save the session to your database.

Users must be able to sign in using their **username** and **password**. The User shall only be allowed to enter if the username corresponds to an actual User in the database and the hashed result of the password argument matches the password digest stored in the database for that User.

All routes in this Application, with the exception of the login page and the sign up page, **must be authenticated.**

Users should remain logged in, even if the server is turned off and on again within a short period of time.You must also provide the functionality for a user to log out. If a User logs out, that User's session should be deleted. User's should not be able to modify their Account details if they are not logged in. At **no point in time** should a User be able to modify another User's account.

You should return appropriate HTTP status codes (403 or 401) when a user fails to authenticate or tries to access a resource that they do not have the right to access (either because they are not logged in or it is not their account).

### User Interface

We have deliberately not included a strict requirement on user interface design, this is not a Graphic Design subject and we will therefore not be marking you on how *pretty* your Application is. That being said, you will be assessed on the control flow of your Application. We suggest that you make use of a frontend framework like Bootstrap (www.getbootstrap.com) to handle the UI and employ their standard UI elements. For an example on how you can successfully integrate Bootstrap with your Rails Application, please take a look at the Workshop Two *Wordgram* Skeleton.

Your application must provide, at a minimum, the following user accessible pages:

- Sign In Page
- Sign Up Page
- Edit Account Page
- News Articles Index
- All News Articles that match a User's interests
- News Article Detail Page showing all information for a given Article.

### Submission Instructions

Project submission will be enabled through the LMS. We will require you upload a zip file of your Rails Application. We will then test your program by running the rails server and verifying behaviour. We will be running your program like so:

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

Should your application require additional commands in order to set up, you **must** include a README file in your project detailing the additional commands. If we cannot run your application with these commands, and you have not provided a read me detailing how to run your application, you will lose marks.

### Implementation Checklist

This list should serve as a guide as to a suggested order of implementation. You do not need to follow this order, but you will require all features on this list to get full marks.

- News Articles are created in the model layer
- News Importers are created in the model layer.
- News Articles are from **6** sources are persisted in the database.
  - News Articles have the appropriate relationships to other model layer objects.
  - News Articles are unique in the databse, duplicates are not saved.
- Users can sign up and edit their profile
- Users can request fresh articles.
- Users can sign in with email and password
  - Passwords are not stored in plain text in the database
- Routes that require authentication only allow a user in when signed in.
- Users can save their interests in the system.
- Users can view articles only relevant to their interests

### Marks

This project will account for 10 marks out of the total 100 available for this subject. These will be broken down as follows:

| Criterion | Percentage |
|---|---|
| User Account Creation and Updates | 1% |
| User Account authentication | 1% |
| News Articles, Tags and Interests are Persisted in the database | 1% |
| Articles are Tagged appropriately and Users can save their preferences | 2% |
| Users can view Articles that are relevant to their preferences | 1% |
| Users can trigger a manual scrape | 1% |
| Code Quality | 3% |

We also reserve the right to award or deduct marks for clever or poor implementations on a case by case basis outside of the prescribed marking scheme.

The code quality will be judged on how well you have adapted to Ruby style. We expect to see good variable names, well commented functions, inline comments for complicated code. We also expect good object oriented design principles and functional decomposition. Output correctness will be automatically verified against expected results, therefore it is important your program outputs the correct format.

We also expect that you have followed the Rails framework conventions, that all logic is appropriately separated according to the MVC Pattern and that you have applied appropriate validations, security considerations and performance considerations.

To assist with your code quality, we suggest you review the following ruby style guide: https://github.com/bbatsov/ruby-style-guide. We will be using this style guide to form the basis of our style marking.

## Submission Date

This project is due at **11:59 p.m. on the 16th of September.** Any late submissions will incur a 1 mark penalty per day unless you have supporting documents. If you have any issues with submission, please email Mat at mathew.blair@unimelb.edu.au, before the submission date.