

|  |
| --- |
|  |
| Technical Documentation |
| Michael Boge, Ian Mckay, Aron Hardy-Bardsley, Shiwei Zhang |

Preamble

The following documentation outlines the technical information regarding the design and development of the News Feeder system. This information is intended for other developers and those interested in the technical properties of the system. This information is preliminary and subject to change.

Table of Contents

[Preamble 1](#_Toc326303955)

[Table of Contents 2](#_Toc326303956)

[Vision 4](#_Toc326303957)

[Introduction 4](#_Toc326303958)

[Glossary 4](#_Toc326303959)

[Problem Statement 5](#_Toc326303960)

[Position Statement 6](#_Toc326303961)

[System Overview 7](#_Toc326303962)

[System Structure 8](#_Toc326303963)

[Assumptions and Dependencies 9](#_Toc326303964)

[Users 10](#_Toc326303965)

[User Environment 11](#_Toc326303966)

[Scope & Limitations 12](#_Toc326303967)

[Competitors and Alternatives 14](#_Toc326303968)

[Documentation Requirements 15](#_Toc326303969)

[Requirements 16](#_Toc326303970)

[Priorities 16](#_Toc326303971)

[Requirements Summary 16](#_Toc326303972)

[Functional Requirements 18](#_Toc326303973)

[Non-Functional Requirements 25](#_Toc326303974)

[Operational Requirements 27](#_Toc326303975)

[Technologies 28](#_Toc326303976)

[Software Technologies 28](#_Toc326303977)

[Hardware Specifications 29](#_Toc326303978)

[Development Tools 29](#_Toc326303979)

[Development Iterations 30](#_Toc326303980)

[Test Planning 31](#_Toc326303981)

[System Architecture 32](#_Toc326303982)

[Repository Design 32](#_Toc326303983)

[Sub System Architecture 33](#_Toc326303984)

[System Interaction 37](#_Toc326303985)

[Website Design 38](#_Toc326303986)

[Domain Registration 38](#_Toc326303987)

[Hosts Purchasing 38](#_Toc326303988)

[Social Media 39](#_Toc326303989)

[Application Design 40](#_Toc326303990)

[Global User Interface Design 40](#_Toc326303991)

[Sheet Layouts 42](#_Toc326303992)

[Search for Feeds 43](#_Toc326303993)

[Use Cases 45](#_Toc326303994)

[Front-End Use Cases 45](#_Toc326303995)

[Back-End Use Cases 51](#_Toc326303996)

[Web Service API Use Case Diagram 54](#_Toc326303997)

[Downloader Application Use Case Diagram 55](#_Toc326303998)

[Database Design 58](#_Toc326303999)

[Data Dictionary 58](#_Toc326304000)

[Entity Relationship Diagram 62](#_Toc326304001)

[Interactions 63](#_Toc326304002)

[Back-End Protocol 63](#_Toc326304003)

[Algorithms and Logic 71](#_Toc326304004)

[Back-End Logic 71](#_Toc326304005)

[Usage of third-party libraries 72](#_Toc326304006)

[libcurl 72](#_Toc326304007)

[RapidXML 72](#_Toc326304008)

[SimplePie 72](#_Toc326304009)

[MySQL++ 72](#_Toc326304010)

[SWIG 72](#_Toc326304011)

[Front-End Class Documentation 73](#_Toc326304012)

[Front-End Class Diagram 73](#_Toc326304013)

[Front-End Class Descriptions 74](#_Toc326304014)

[Back-End Class Documentation 75](#_Toc326304015)

[Back-End Class Diagram 75](#_Toc326304016)

[Back-End Class Descriptions 76](#_Toc326304017)

[Back-End Exception Hierarchy 77](#_Toc326304018)

[Shared Database Library Class Documentation 78](#_Toc326304019)

[Shared Database Library Class Diagram 78](#_Toc326304020)

[Downloader Application Class Documentation 82](#_Toc326304021)

[Downloader Application Class Diagram 82](#_Toc326304022)

[Web Service API Class Documentation 83](#_Toc326304023)

[Web Service API Class Diagram 83](#_Toc326304024)

Vision

# Introduction

News Feeder is a news feed aggregator service, which compiles content from multiple online content sources into a common news sheet format. It is intended for use by anyone in the general public that wishes to view regularly updating content in a simple manner.

# Glossary

**Item/Article –** An individual piece of content. This includes articles from RSS feeds or items from other types of content sources (such as an email, forum thread, weather forecast, etc).

**Feed –** A collection of items/articles all related to the same content source (i.e. the same site or RSS feed).

**Sheet –** A grouping method for feeds. It displays feeds on the website as views the user has created.

**Sheet Layout –** The view used to display feeds and content associated with the sheet. An individual sheet may have many of these views. When referring to a sheet and the way it displays its content, we are implicitly referring to its default view.

**Front-end –** The web interface of the system, where users gain access to and interact with the system.

**Back-end –** An application which runs a continuous queue and crawls/parses content sources for articles to add to the database.

# Problem Statement

|  |  |
| --- | --- |
| **The problem of** | retrieving and viewing content from multiple online content sources that are updated regularly |
| **affects** | any member of the general public who wishes to stay up to date with all this information; |
| **the impact of which is** | as they have to visit each location for its relative content which is time consuming and it is easy to miss information, particularly for regularly updated content. |
| **A successful solution would be** | a service which crawls content sources specified by a user and retrieves content to display in a practical and suitable format. Meanwhile, requiring as little interaction/configuration from the user as possible to continue retrieving this content. |

# Position Statement

|  |  |
| --- | --- |
| **For** | any members of the general public |
| **who** | wish to view regularly updated online content (such as RSS feeds) without needing to manually visit each content source. |
| **News Feeder is a** | online news aggregation service |
| **that** | compiles content from multiple online content sources into a common news sheet format. |
| **Unlike** | the alternative of accessing each content source for its relative news manually, and at regular intervals (to ensure no news is missed), |
| **our product** | automatically crawls the content source and retrieves new content as it becomes available, and displays it in an attractive, organised and customisable format. This ensures no content is missed and it is easily accessible for the user via an online interface. |

# System Overview

News Feeder is a news feed aggregation service, which compiles content from multiple online content sources. Its main purpose is to provide a single location and format to view news content instead of users having to access multiple websites for their specific news.

News Feeder allows users to specify which type of articles they wish to view, based on a number of criteria, including; location, category, author, publishing site and specific feed URLs.

The system crawls feeds which match the specified criteria and stores the relevant articles/items and their content. The articles/items can come from a number of different content sources (such as RSS, forums and social networks) and their content can be of any media type, including; text, images, video and geo-locational information.

The user can then view the stored articles and the article content on their sheets. The layout of the sheet will be in a standard news sheet format by default, but the system also provides a simple drag and drop interface for creating custom user layouts for sheets. Sheets will allow users to set filtering options to ensure certain types of content are not displayed to inappropriate audiences (such as pornographic material to minors).

Users can choose to subscribe to the News Feeder service for a monthly fee. This provides them with a number of benefits over standard users, such as access to the downloader application, customisable sheet layouts, increased feed/sheet limits and increased priority for content retrieval.

The system provides access for administrators to manage the News Feeder service. Administrators can manage users, feeds and items, as well as managing and monitoring back-end system processes (such as starting and stopping the crawler).

Access to this functionality is available via a web-interface. Users may also access News Feeder content offline via the use of a downloader application.

# System Structure

The service’s structure can be broken down into 6 main sub-systems:

## The Database

This is where user, sheet, feed and item information is stored. It is used by both the front-end and back-end components and acts as a repository between the two.

## The Front-End

This is the interface of the system. It consists of a number of web pages accessible by users and administrators. It provides the means by which the users and administrators interact with the system (including the back-end). This interface provides methods for users to manage and view feeds, as well as managing the layout used to display these feeds. The interface also provides methods for administrators to manage users, feeds and items, as well as allowing them to manage and monitor the back-end processes.

## **The Back-End**

This is an application that is continuously running in the background. It provides the main processing of the system. Its main role is to crawl and parse feeds and content. To perform this role this component contains a continuous queue holding references to all feeds and ordered according to crawl priority. The parsers mainly operate on RSS and HTML items, but other content sources (such as weather feeds, forums and social networks) can also be parsed for storable content.

Additional non-essential components of the service are the API web service and the downloader application. The downloader application (and other possible applications) accesses the system by using the API web service as a gateway.

## The Downloader Application

The downloader application is a client desktop application of which subscribed News Feeder users can download through the News Feeder website. The downloader application is not a primary subsystem; it is an extension of the News Feeder service and was built as an offline viewer alternative to the website.

## The Web Service

The primary purpose of the web service API is to provide an internet facing gateway to allow downloader application to communicate with the internal subsystems of the News Feeder service.

## The Shared Database Library

The Shared Database Library is a C++ library used by the Back-End and Web Service sub-systems as a standard interface to access the News Feeder database. The library itself provides methods that can be called to access and modify the database entities and records via SQL statements. This library uses the MYSQL++ library to perform standard database tasks (such as executing a query) and instead focuses on generating application specific queries and retrieving the results.

This system was added to ensure that database calls within the system are performed in a standardised way and as such will achieve the same results. It was also added to separate the database connection/query generation from these sub-systems.

As the library needs to be used by the different systems which are written in different programming languages (C++ for Back-End and Java for the Web Service), the library provides a number of wrappers/interfaces for each specific version (these are generated by SWIG, Simplified Wrapper and Interface Generator).

# Assumptions and **Dependencies**

The following assumptions and dependencies relate to the development and capabilities of the News Feeder service:

* The News Feeder system is intended to provide a service to the general public for use. It is not a product marketable to third-parties for implementation. This in turn adds the following assumptions:
  + The system will be internally based within one organisation, which uses the hardware and software required by each sub-system and module.

Each sub-system and module will have fast and secure communication methods to all other sub-systems and modules.

# Users

There are 5 different classes of users who interact with the News Feeder service:

## Standard User

The main end-users of the system, including anyone in the general public. They have registered with the service and can specify feeds they wish to view. They can set a number of user preferences and settings regarding what content they wish to display and how they wish to display it (such as content filters and sheet layouts). They are expected to have basic computer knowledge (sufficient for using a web browser).

## Subscribed User

Standard users who have subscribed to the system for a fee. As such they gain additional benefits related to their use of the system, such as access to custom sheet layouts and access to the downloader application.

## Administrator

Managers of the service. They manage a number of system features and components (such as user accounts and items) and are able to manage and monitor system processes (such as starting and stopping the crawlers). They are considered to have more in depth computer knowledge than a standard user and understand the internal workings of the system.

## Guest

Anonymous users that can register to the service in order to become standard users. Guests can access the main page, FAQ and are also able to view links sent by registered users of the system (such as a link to an article).

## Web Master

Owner of an external site that is being crawled. They access the system via a portal referenced in the crawler HTTP headers in order to specify how frequently they prefer their site to be crawled (or if they want it to be crawled at all).

# **User Environment**

All users of the service will have access to an active ADSL internet connection in order to use the service, with the exception of the Downloader Application which will only require intermittent access to retrieve content.

The users will access the service using one of the following operating systems: Windows, Linux or Mac OSX. For online service access, users will be able to use: Internet Explorer 7+, Google Chrome, Mozilla Firefox or Safari. Standard web technologies will be used by the system to ensure future browsers will be able to access the system as well.

# Scope & Limitations

This development of this service will focus on fulfilling the Base, High and Medium level requirements outlined in the Requirements section of this document.

In terms of limitations, it has a few mainly considering the online accessibility of the service for users and of the content sources:

* The ability for a content source to be crawled and parsed by the service will rely on how it displays its updates (e.g. via an RSS feed). If it does not provide these services or does not provide necessary information (such as a URL link to the article) it will not be able to be stored and used by the system.
* Information provided by the feed will be dependent on the information provided by the actual content source (e.g. we cannot show geo-location information if it is not provided in the article).
* The performance of the service for a user will be dependent on the network connection used to access it (we assume a standard ADSL network connection for the requirements defined in this document).

## Constraints

The News Feeder project and its requirements are limited by the following constraint:

* **Network -** The performance of the service will be dependent on the network connection used to access it. The requirements defined in this document assume a standard ADSL network connection.

## Cost and Pricing

The News Feeder system will be available for free to the general public. However, the system will also provide a paid subscription service, where a user will pay a small monthly fee to gain additional features of the service, such as:

* Removal of advertising from the site and sheets
* Access to the Downloader Application
* Increased limit on the number of sheets the user can create
* Ability to create custom sheet layouts

In addition to this, the API Web Service used by the downloader application can also be made public. This would allow other businesses to access News Feeder’s content for a fee.

As the system is not being marketed as a product, there will be no licencing charges for potential vendors.

## Quality Ranges

The following defined the quality ranges for the News Feeder service in terms of performance, usability and other factors:

* **Availability –** the service will be available 24 hours a day, 7 days a week, with the exception of small amounts of scheduled downtime for maintenance.
* **Usability –** the service will be easy to use for both the standard users, who are expected to have only small amounts of technical knowledge, and for the administrators, who are expected to have general knowledge of the system and its components.
* **Maintenance –** the system is able to be changed and/or configured without the need to reconstruct and redeploy the system.
* **Accessibility –** the service will provide help for the users, both in form of online help (and FAQ) and user and administrator manuals.

For additional quality ranges in terms of performance please see the Non-Functional requirements section of the Requirements document.

## Precedence and Priority

The functionality discussed in this vision document should be available in the first release of the service to the public. Any additional features of lower priority will be added after the deployment over time (provided they can be integrated with the existing system causing minimal disturbances).

For details on the priority of the functionalities and the order they are to be implemented in is discussed in the Priorities section of the Requirements document.

# Competitors and Alternatives

Aside from the alternative of manually retrieving online content from multiple sources, the system has to compete with a number of pre-existing online RSS aggregators. The following are the three most popular competitors, they all provide the same basic functionality, retrieving online content and compiling it in an easy to access location. These competitors and their main differences are:

## Google Reader

This solution’s main advantage is that it employs an interface common to other Google services (e.g. Gmail) with simple, easily understandable and powerful features. It provides a folders option to read and group subscriptions easily and also provides a simple method for sharing feed items, by email or directly via Google Reader itself. It can handle almost any type of feed and retrieves updated content quickly.

## Bloglines

This solution differentiates itself from the others by providing the ability to support the publishing of any feed items (such as clippings or posts) to a personal blog. It allows for blogs to be created directly from this service, allowing both feed and blog management in the same interface. Bloglines does not provide the social media integration and tagging available the other competitors. Additionally this solution provides no OPML export functionality and updates its content the slowest out of these solutions.

## Newsgator

This solution provides both a desktop client and web version, which sync together to retrieve updates and changes. It allows easy migrations to the service by allowing feed subscriptions to be imported using OPML. The disadvantages are that it does not allow feed reordering and the desktop client is Windows only.

Overall, these competitors all perform similar functions with some advantages and disadvantages. News Feeder is designed to implement as many of the successful features of these competitors as possible. One area in which the News Feeder service differs from these competitors is the way the content is displayed. These competitors display the feeds in standard list/inbox type formats, whereas News Feeder will display them in fully customisable layouts, reminiscent of standard news sheets. Users will be able to design the layout format by a simple drag and drop interface. Another difference is that the content displayed in News Feeder. Whilst the other aggregators only display what they can pull from the RSS feed, News Feeder uses Patch and match to query the HTML page of the article and parse this for content. This means News Feeder will show the full article as well as any related content like images and geo-locational information.

# Documentation Requirements

The following documents are required for the full system release:

* **Technical Documentation –** This describes the structure of the system and its development process, to enable future developers to make changes to the service.
* **Standard User Manual –** This is provided as a soft-copy from the service website and provides step-by-step processes for how a user can use the service (such as feed management, sheet management, registration, etc).
* **Administrator User Manual –** This provides step-by-step processes in regards to performing actions specific to administrators in the system (such as managing users and starting/stopping modules). It will be provided as both a hard and soft copy to the administrators of the system.
* **Downloader Application User Manual –** This provides step-by-step instructions for using the downloader application (in terms of downloading sheets, setting an update schedule, etc).
* **Online Help –** This is provided both explicitly, via a set of FAQ’s accessible directly through the online service, and implicitly, via use of standard icons and a standard style interface.

Requirements

# Priorities

The priorities used in the requirements have the following meanings:

**Base -** Requirements that must be met before any high level requirements can be implemented. These will definitely be delivered.

**High -** Requirements which are necessary to fulfil the core functionality of the system. These will definitely be delivered.

**Medium -** Requirements that provide additional functionality for the service. We will attempt to deliver all of these requirements.

**Low -** Requirements that are the stretch goals for the project. These will only be implemented if time presents itself.

# Requirements Summary

|  |  |  |
| --- | --- | --- |
| **ID** | **Requirement** | **Priority** |
| F1.1 | Front-end Database Interaction | Base |
| F2.1 | Login | Base |
| F2.2 | Manage Users | Base |
| F3.1 | Add Feeds | Base |
| F10.1 | Back-end Database Interaction | Base |

|  |  |  |
| --- | --- | --- |
| **ID** | **Requirement** | **Priority** |
| F2.3 | Account Registration | High |
| F3.2 | Search For Feeds To Add | High |
| F3.3 | View Sheets | High |
| F3.4 | Additional Media - Images | High |
| F4.1 | Manage Sheets | High |
| F4.2 | Manage Sheet Feeds | High |
| F5.1 | Back-end Monitoring | High |
| F11.1 | Crawl RSS Feeds | High |
| F11.2 | Crawl Priority | High |
| F12.1 | Parse RSS Feeds | High |
| F12.2 | Advanced Parsing Techniques | High |
| F13.1 | Logging | High |
| N1 | Accessibility | High |
| N4 | Performance (Front-end) | High |
| N5 | Performance (Back-end) | High |
| N7 | Stability (Back-end) | High |
| N8 | Maintainability | High |
| N9 | Sustainability | High |

|  |  |  |
| --- | --- | --- |
| **ID** | **Requirement** | **Priority** |
| F2.4 | User Profile/Settings Configuration | Medium |
| F2.5 | User Subscriptions | Medium |
| F2.6 | Subscription Management | Medium |
| F3.5 | Printing Support | Medium |
| F3.6 | Additional Media - Video | Medium |
| F3.7 | Additional Media - Maps | Medium |
| F4.3 | System Defined Sheet Layouts | Medium |
| F4.4 | Custom Sheet Layouts | Medium |
| F5.2 | Manage Back-end | Medium |
| F5.3 | User Statistics | Medium |
| F5.4 | Abuse Reporting | Medium |
| F6.1 | Content Filtering | Medium |
| F6.2 | Content Blacklisting | Medium |
| F6.3 | Bad Word Censoring | Medium |
| F7.1 | Web Security | Medium |
| F8.1 | Downloader Application | Medium |
| F9.1 | Recommended Feeds | Medium |
| F9.2 | User Sheet Update Notifications | Medium |
| F9.3 | Mobile Support | Medium |
| F12.3 | Retrieve Additional Media | Medium |
| F12.5 | Tag/Keywords Generation (Categories) | Medium |
| F12.7 | Multiple Content Sources | Medium |
| F14.1 | Email Notifications | Medium |
| F15.2 | API Web Service | Medium |
| N2 | Ease of Use (User) | Medium |
| N3 | Look and Feel (Aesthetics) | Medium |
| N6 | Portability (downloader application) | Medium |
| N10 | Help | Medium |
| N11 | Scalability | Medium |

|  |  |  |
| --- | --- | --- |
| **ID** | **Requirement** | **Priority** |
| F8.2 | Mobile Application(s) | Low |
| F9.4 | Social Network Authentication | Low |
| F12.4 | Geo-location Retrieval | Low |
| F12.6 | RSS and HTML Parsing Configurations | Low |
| F13.2 | Auditing | Low |
| F14.2 | SMS Messaging | Low |
| F15.1 | Image Storage (Thumbnails) | Low |
| F15.3 | Learning Algorithms | Low |

# Functional Requirements

## Front-end

#### F1 – Database Interaction

|  |  |  |
| --- | --- | --- |
| **ID:** F1.1 | **Requirement:** Front-end Database Interaction | **Priority:** Base |
| **Description:** The front-end is able to retrieve and store information about feeds, users and sheets in a database. | | |

#### F2 - User Management/Login System

|  |  |  |
| --- | --- | --- |
| **ID:** F2.1 | **Requirement:** Login | **Priority:** Base |
| **Description:** The system will provide users and administrators with the ability to login to the system via the website by providing a username and password. After logging in, users are to be able to access account specific information (such as their feeds and sheets) and functionality. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F2.2 | **Requirement:** Manage Users | **Priority:** Base |
| **Description:** The system will allow administrators to add/remove/edit users from the system for administrative purposes (such as removing users who have been inactive for too long). The editing ability also includes resetting user passwords/password attempts. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F2.3 | **Requirement:** Account Registration | **Priority:** High |
| **Description:** The system will enable guests to register an account by providing an email address, username, appropriate password and other relevant user information. All account information will be validated on registration, such as checking for unique usernames. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F2.4 | **Requirement:** User Profile/Settings Configuration | **Priority:** Medium |
| **Description:** Using the web interface, users will be able to modify their profile information (email address, name, avatar image, etc) and personal settings, such as how much priority they put on a particular feed, how often they wish to receive notifications. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F2.5 | **Requirement:** User Subscriptions | **Priority:** Medium |
| **Description:** The system will allow users to subscribe/unsubscribe to the news feeder service to gain additional benefits such as access to the downloader application, customisable sheet layouts, increased Feed/Sheet limits and increased priority for content retrieval. Subscriptions will be recurring on a monthly basis for a fee. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F2.6 | **Requirement:** Subscription Management | **Priority:** Medium |
| **Description:** The system will allow administrators to create/edit/delete/review user subscriptions via the web interface. | | |

#### F3- Feed Management

|  |  |  |
| --- | --- | --- |
| **ID:** F3.1 | **Requirement:** Add Feeds | **Priority:** Base |
| **Description:** The system will provide users with the ability to add feeds to their sheets. The feeds to add can be newly created, pre-existing user feeds or default feeds. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F3.2 | **Requirement:** Search For Feeds To Add | **Priority:** High |
| **Description:** The system will provide the users with the ability to search for feeds either currently in the database or from the internet (using a search API) matching specified search criteria (such as keywords, author and location). The resulting feed(s) can then be added to the user’s sheet(s). | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F3.3 | **Requirement:** View Sheets | **Priority:** High |
| **Description:** The website will display sheet content in a suitable web format for the user to read. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F3.4 | **Requirement:** Additional Media - Images | **Priority:** High |
| **Description:** The front-end is able to display image content that was included in the original item in standard HTML format. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F3.5 | **Requirement:** Printing Support | **Priority:** Medium |
| **Description:** The system will provide users with the ability to print off their sheets and individual feed items in a readable format similar to that of the site. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F3.6 | **Requirement:** Additional Media - Video | **Priority:** Medium |
| **Description:** The front-end is able to display video content that was included in the original item in standard HTML format. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F3.7 | **Requirement:** Additional Media - Maps | **Priority:** Medium |
| **Description:** The front-end is able to display geo-locational content that was included in the original item. The content will be displayed using the relevant map API. | | |

#### F4 - Sheet Management

|  |  |  |
| --- | --- | --- |
| **ID:** F4.1 | **Requirement:** Manage Sheets | **Priority:** High |
| **Description:** The system provides users with the ability to create sheets which contain a number of different feeds. Users will be able to set the sheet layout from a predetermined list (1 column, 2 column, and 3 column layout). Users will be able to delete their sheets. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F4.2 | **Requirement:** Manage Sheet Feeds | **Priority:** High |
| **Description:** The system will allow users to add and remove feeds from their sheets. The will also be able to edit a feed’s information in relation to how it is displayed on the sheet, such as what content (title, description, body, images) of the feed will display or how many items to show from that feed. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F4.3 | **Requirement:** System Defined Sheet Layouts | **Priority:** Medium |
| **Description:** The system will provide a set of pre-defined layouts which the user will have access to when creating sheets and sheet views. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F4.4 | **Requirement:** Custom Sheet Layouts | **Priority:** Medium |
| **Description:** The system will enable users to create their own custom sheet layouts via a drag and drop interface. Users will then be able to drag and drop feeds of their choice into the layout. | | |

#### F5 - Administration

|  |  |  |
| --- | --- | --- |
| **ID:** F5.1 | **Requirement:** Back-end Monitoring | **Priority:** High |
| **Description:** The front-end web interface allows administrators to retrieve statistics on the back-end (such as crawler, thread and database statistics). Administrators will also be able to monitor the current back-end load. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F5.2 | **Requirement:** Manage Back-end | **Priority:** Medium |
| **Description:** From the front-end web interface, administrators will be able to start and stop crawling as needed. Administrators will also be able to change the back-end configuration information such as the number of threads available to each part of the back-end system. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F5.3 | **Requirement:** User Statistics | **Priority:** Medium |
| **Description:** The system will display basic user statistics (such as how many users, how many user feeds, etc) on the home page of the website. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F5.4 | **Requirement:** Abuse Reporting | **Priority:** Medium |
| **Description:** The system will allow webmasters of content sources to adjust the frequency that the system crawls their sites (including the ability to disable all crawling), to ensure it does not exhaust their resources. This is done via a portal referenced in the crawler HTTP headers. | | |

#### F6 - Content Filtering

|  |  |  |
| --- | --- | --- |
| **ID:** F6.1 | **Requirement:** Content Filtering | **Priority:** Medium |
| **Description:** The system will allow users to specify categories/keywords which they do not want displayed in their sheets. This can be used to filter out inappropriate material (such as pornography). | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F6.2 | **Requirement:** Content Blacklisting | **Priority:** Medium |
| **Description:** The system will enable administrators will be able to add specific sites/content sources to a blacklist that the system will ignore. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F6.3 | **Requirement:** Bad Word Censoring | **Priority:** Medium |
| **Description:** Offensive words (listed in the database) will be censored in content displayed to the user. This will be a user preference. | | |

#### F7 - Security

|  |  |  |
| --- | --- | --- |
| **ID:** F7.1 | **Requirement:** Web Security | **Priority:** Medium |
| **Description:** The front-end will use SSL for secure communications when communicating important information with the user (such as logins). This will be a user preference. | | |

#### F8 - Additional Applications

|  |  |  |
| --- | --- | --- |
| **ID:** F8.1 | **Requirement:** Downloader Application | **Priority:** Medium |
| **Description:** The service will provide subscribed users with access to a downloader application. This application will provide a method for viewing sheets offline, either by providing PDFs or an offline version of the content. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F8.2 | **Requirement:** Mobile Application(s) | **Priority:** Low |
| **Description:** Specific applications will be developed for mobile devices to provide access to and use the system. | | |

#### F9 - Miscellaneous

|  |  |  |
| --- | --- | --- |
| **ID:** F9.1 | **Requirement:** Recommended Feeds | **Priority:** Medium |
| **Description:** The system will be able to suggest relevant feeds the user may be interested in. This will be based on the feeds the current user is and has been subscribed to, and any relations between these feeds (such as keywords, author, location, source, etc). | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F9.2 | **Requirement:** User Sheet Update Notifications | **Priority:** Medium |
| **Description:** The website will display the number of new items available in a content area on a sheet for each user. The number of sheets which have new content available will be displayed in the global menu area. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F9.3 | **Requirement:** Mobile Support | **Priority:** Medium |
| **Description:** The web site will support mobile access. All pages and content will be able to be viewed on mobile devices (without issues and in the correct format, including sheet layout). | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F9.4 | **Requirement:** Social Network Authentication | **Priority:** Low |
| **Description:** The front-end will integrate with social network authentication providers (such as Facebook, Twitter) to provide access via users’ respective logons. | | |

## Back-end

#### F10 – Database Interaction

|  |  |  |
| --- | --- | --- |
| **ID:** F10.1 | **Requirement:** Back-end Database Interaction | **Priority:** Base |
| **Description:** The back-end is able to retrieve and store information about feeds and crawled/parsed content in a database. | | |

#### F11 - Crawling

|  |  |  |
| --- | --- | --- |
| **ID:** F11.1 | **Requirement:** Crawl RSS Feeds | **Priority:** High |
| **Description:** Given an RSS feed, the back-end will crawl it and retrieve items for parsing into content. It will continue to crawl the feed for new content on a regular basis. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F11.2 | **Requirement:** Crawl Priority | **Priority:** High |
| **Description:** Based on a number of feed statistics the crawler will dynamically arrange the feeds so that feeds of higher priority are crawled more regularly than others. The priority will be based on properties such as; the average amount of time between content updates, the number of subscribers and feed type (new, user defined, system defined, etc). | | |

#### F12 - Parsing

|  |  |  |
| --- | --- | --- |
| **ID:** F12.1 | **Requirement:** Parse RSS Feeds | **Priority:** High |
| **Description:** Given RSS XML input the back-end will extract information (such as feed title, author and content). | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F12.2 | **Requirement:** Advanced Parsing Techniques | **Priority:** High |
| **Description:** If an article’s full content is not provided in the RSS feed, the back-end is able to apply techniques which will retrieve the full article content. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F12.3 | **Requirement:** Retrieve Additional Media | **Priority:** Medium |
| **Description:** The back-end is able to retrieve non-text elements (such as images and videos) that are related to the article by parsing the actual HTML page it is displayed on. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F12.4 | **Requirement:** Geo-location Retrieval | **Priority:** Low |
| **Description:** Given an article the parser will attempt to determine locations relevant to the item (this will work by using any available Maps API references in the RSS, or by looking for place names in the body). | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F12.5 | **Requirement:** Tag/Keywords Generation (Categories) | **Priority:** Medium |
| **Description:** The parser will be able to retrieve/generate tags/keywords for an item, to assist in the searching of items. These will be generated using the attributes, such as the description, if not directly available in the RSS feed. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F12.6 | **Requirement:** RSS and HTML Parsing Configurations | **Priority:** Low |
| **Description:** The system will allow administrators to configure the back-end to parse items differently based on heuristics within the RSS/HTML (e.g. meta-tags identifying creation with WordPress) using configuration files. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F12.7 | **Requirement:** Multiple Content Sources | **Priority:** Medium |
| **Description:** The back-end will be able to crawl and parse content from sources other than RSS (such as weather, email, social networks, forums, etc). | | |

#### F13 - Logging

|  |  |  |
| --- | --- | --- |
| **ID:** F13.1 | **Requirement:** Logging | **Priority:** High |
| **Description:** The back-end will provide error logging and optional verbose logging. These will be output to log files which are accessible on the back-end server. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F13.2 | **Requirement:** Auditing | **Priority:** Low |
| **Description:** The system will provide logs of users’ interactions with the system in the form of audit trace logs. | | |

#### F14 - Notifications

|  |  |  |
| --- | --- | --- |
| **ID:** F14.1 | **Requirement:** Email Notifications | **Priority:** Medium |
| **Description:** Email notifications may be sent when new content is found for a user’s sheet. Alternatively, it may be configured to send ‘batch emails’ which consist of all updates over a period of time. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F14.2 | **Requirement:** SMS Messaging | **Priority:** Low |
| **Description:** The system will provide update notifications to users when new content becomes available via SMS messaging. | | |

#### F15 - Miscellaneous

|  |  |  |
| --- | --- | --- |
| **ID:** F15.1 | **Requirement:** Image Storage (Thumbnails) | **Priority:** Low |
| **Description:** Images will be resized and stored in the database to reduce bandwidth usage in the user’s browser, prevent linking to missing images and also prevent possible HTTPS issues. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F15.2 | **Requirement:** API Web Service | **Priority:** Medium |
| **Description:** The system will provide an interface for the downloader application and potentially, other 3rd party applications. Its purpose is to act as a gateway for external systems to access content. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** F15.3 | **Requirement:** Learning Algorithms | **Priority:** Low |
| **Description:** The system will be able to gradually learn the best method for parsing particular content sources. This will be based on what the system has learnt and parsing configurations from administrators. | | |

# Non-Functional Requirements

|  |  |  |
| --- | --- | --- |
| **ID:** N1 | **Requirement:** Accessibility | **Priority:** High |
| **Description:** The system will be accessible via any modern browser (Chrome, Firefox, Safari, IE7+) on any modern Operating System (Windows, Linux, Mac). | | |

|  |  |  |
| --- | --- | --- |
| **ID:** N2 | **Requirement:** Ease of Use (User) | **Priority:** Medium |
| **Description:** Users will be able to learn to use the features of the system within a short time period (e.g.10 mins), without any help/training external to the site. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** N3 | **Requirement:** Look and Feel (Aesthetics) | **Priority:** Medium |
| **Description:** The front-end interface should be intuitive, consistent and familiar to users, such that they can easily navigate and use the site. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** N4 | **Requirement:** Performance (Front-end)\* | **Priority:** High |
| **Description:** The front-end sub-system will adhere to the following performance metrics:   * Page Load – All pages should load (excluding media, such as images, video, maps) in under 2 seconds. * Searching for News Feeds – First results should be returned within 5 seconds. * Crawling Feeds – First results should be available within 1 minute. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** N5 | **Requirement:** Performance (Back-end) | **Priority:** High |
| **Description:** The back-end sub-system will adhere to the following performance metrics:   * Responding to Front-end – Calls will on average take 1 second to respond. * Crawling and Parsing a Feed Item – Crawling for a feed item (including media, patch and match) will be completed within 5 seconds. * Queuing – One queue item will not be in the queue longer than its expected update time without being crawled. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** N6 | **Requirement:** Portability (downloader application) | **Priority:** Medium |
| **Description:** The downloader application will be available on all modern platforms (Windows, Linux, Mac OSX) | | |

|  |  |  |
| --- | --- | --- |
| **ID:** N7 | **Requirement:** Stability (Back-end) | **Priority:** High |
| **Description:** The back-end is able to handle invalid inputs and malicious SQL without error. Any errors which occur will be handled cleanly, as will service stops and restarts. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** N8 | **Requirement:** Maintainability | **Priority:** High |
| **Description:** The source code of all subsystems will conform to code styling standards and implement good design principles, which will allow for future upgrades and maintenance without requiring a system overhaul. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** N9 | **Requirement:** Sustainability | **Priority:** High |
| **Description:** The system will have mechanisms in place to maintain itself without the need for external interaction (e.g. the system will clear the logs automatically after a set period/size limit). | | |

|  |  |  |
| --- | --- | --- |
| **ID:** N10 | **Requirement:** Help | **Priority:** Medium |
| **Description:** The site provides help and FAQ for the user, both directly (i.e. showing a help page and tutorial) and indirectly (i.e. descriptions like tooltips). | | |

|  |  |  |
| --- | --- | --- |
| **ID:** N11 | **Requirement:** Scalability | **Priority:** Medium |
| **Description:** There can be an unlimited number of users, sheets, feeds and items. The system will be able to cope with increased numbers without requiring changes to the internal subsystem (i.e. changes that require re-compiling/deployment). | | |

**\*** *Please note that these performance requirements exclude possible network limitations*

# Operational Requirements

To install the News Feeder Downloader application you must meet the following requirements:

* Web Browser (for initial download)
* Runs on Microsoft Windows 2000 or greater
* Runs on Linux 2.6.x kernel
* Runs on Mac OS X
* Support for 32-bit and 64-bit systems
* Working network connection
* Printer (if printing)

To use the News Feeder web service you must meet the following requirements:

* Web Browser
* Screen resolution of 420x380 or greater
* Working network connection
* Printer (if printing)

Technologies

# Software Technologies

**PHP:** PHP is an open source web technology which is used for server side processing in a web environment. PHP is the primary language used in the News Feeder website and is used to retrieve and display data to the user as well as persist data to the News Feeder database. PHP was chosen as the primary language as it is well documented, stable and provides the functionality needed by the News Feeder. This functionality includes (but is not limited to):

* Network Sockets Support
* MySQL Support
* Session/Cookie support

*Technology Reference*:<http://php.net/>

**C++:** C++ has been used to develop the News Feeder Back-End crawler. The crawler required a language that supports abstract functionality like classes, templates and threading. The Back-End does not need to be cross platform as it will sit on a server with a controlled environment. All members of the team are fluent in C++.

*Technology Reference*: <http://cplusplus.com/>

**Java:** Java is the language used to develop the News Feeder Downloader Application which allows a user to download content and then view it while offline. Java is used because it is cross-platform and so the Downloader Application is not limited to a single audience.

*Technology Reference*: <http://java.com/>

**OPML:** OPML is an XML mark-up used with both the News Feeder service and other related services to store information about a collection of RSS/ATOM feeds.

**SQL:** SQL (Structured Query Language) is used to communicate to the database server (MYSQL). SQL is used by both the Front-End and Back-End.

**HTML/JavaScript:** The HTML and JavaScript scripting languages are used to render content on the users browser. HTML version 4.0.1 is used, it was decided that HTML 5 was not to be used as it lacks support. JavaScript is used to enhance the users experience by enabling dynamic content to be rendered on the page. Dynamic features of the Front-End Website include:

* Drag and Drop functionality
* Ajax powered searches and content loading
* Dynamic client side form validation

# Hardware Specifications

## Front-End Host

**Host:** VMPort

**Location:** Kansas City, US

**Virtualization Type:** OpenVZ

**RAM:** 1GB

**Disk:** 60GB

**Bandwith:** 600GB

**Monthly Price:** 4.42 GBP (~$7 AUD)

## Back-End Host

**Host:** BuyVM

**Location:** San Jose, USA

**Virtualization Type:** OpenVZ

**RAM:** 1GB (2GB Burst)

**Disk:** 60GB

**Bandwith:** 3TB

**Monthly Price:** $12.95 US (~$13 AUD)

## Database Host

**Host:** JustHost.com

**Location:** Chicago, US

**Virtualization Type:** Standard Web Host

**Disk:** Unlimited (subject to TOS)

**Bandwidth:** Unlimited (subject to TOS)

**Monthly Price:** $2.95 AUD

# Development Tools

The following tools were used during the development of the News Feeder system:

* Subversion repository from Assembla
* phpMyAdmin
* FileZilla FTP Client
* NetBeans IDE
* PuTTY
* Notepad++
* Xcode 4
* IntelliJ IDEA

Development Iterations

The News Feeder service is to be developed using a combination of the throw-away prototype and iterative development models.

Initially the system will be developed as a throw-away prototype where each sub-system and its features are implemented. This enables development to start early so all concepts and technologies can be tested for feasibility (for example, testing SWIG is viable before designing the Shared Database Library). The prototyping phases will each focus on adding particular functionality, namely:

* Phase 1 (Prototype) – In this phase each sub-system (most importantly the front-end and back-end) will be developed separately and not integrated. Its focuses are:
  + RSS feed parsing
  + Crawling for RSS feeds
  + Site login
  + Sheet management
  + Database connections (to all sub-systems)
* Phase 2 (Prototype) – At the end of this phase (after each sub-system has been developed it will be integrated). Its focuses are:
  + Queue management
  + Threading
  + Administration Service
  + Layouts
  + Downloader Application
  + Shared Database Library

After these initial prototyping phases, we will have determined what sections are feasible so we will finalise our requirements. The prototypes will be discarded and the development will start from the finalised design details. The development process will now follow the standard pattern of implementing functional requirements of progressively lower priority with each iteration. The iterations will implement the following functionality:

1. Base level requirements
2. High level requirements
3. Medium level requirements
4. Low level requirements (this iteration is non-essential, it will only be performed if there is available time).

Test Planning

Throughout the development process the system and its sub-systems will be tested. During development, each sub-system will be individually tested. Each sub-system will have standard white-box and black-box testing performed on it. In addition to these, each sub-system will have its own specific test cases and methods (based on the sub-system type and technologies used):

* **Front-End –** The front-end will focus on black-box testing and usability testing. It will use Selenium to perform the black-box testing and its usability test cases will focus on possible user input and errors. The front-end will also be load and performance tested, to ensure it meets the performance and response requirements.
* **Back-End –** The back-end’s testing will focus on unit testing using CPP-Unit and by code-coverage tests. Each individual module (such as the crawler, the parser or the queue) will be individually unit tested before it is integrated with the other modules. Load and stress tests will be performed on specific modules, notably the crawler and queue (due to the large number of items that will be involved in production use).
* **Web-Service –** The web service will be load tested, to ensure it can cope with the number of requests coming from the downloader application.
* **Downloader Application –** The downloader application itself will be unit tested via J-Unit. Usability testing will also be performed in regards to the user interactions with the application.
* **Shared Database Library –** The database library will be white-box tested to ensure its performance is satisfactory. It will also be load tested considerably, as it has to handle a large number of requests from all other sub-systems.

As each sub-system is further developed and new versions are created (both minor and major), it will be regression tested to confirm previous functionality is not lost as more is added.

When major versions of the sub-systems are completed (such as at the end of the iteration cycles), they will be integrated into the service as a whole. As these parts are added/updated, integration testing will be performed on the whole service. Additional sub-systems will not be added until the current system has passed its current integration tests.

System Architecture

# Repository Design

The News Feeder system is designed using the repository architecture. With all data being stored in a central database (with the shared database library acting as an interface into it) and all subsystems and modules being used to retrieve, add or modify data in the database.

The main reason this decision was made was because all sub-systems use the same shared data (i.e. the feed and item information). The use of the repository architecture means that there is no need to duplicate the storage of this data and also that any modifications made by one sub-system will propagate up to all other sub-systems (this is particularly important when the backend updates feed information, as the front-end wishes to show as up to date information as possible).

This architecture was also chosen as it supports support our development method, with the breaking up of the system into sub-systems. This means that during development changes can be made to any one sub-system without affecting the development of the others. This is also true in terms of maintenance upon release, as it reduces the amount of downtime for the system (as only the sub-system in question would have to be taken down).

The service also uses the 3-tier architecture to a degree, with the 3 sections:

* **Client** – The UI interface in the web-browser and applications the user will interact directly with on their home machine.
* **Servers** – The servers running within the service architecture that will perform all of the logic of the service.
* **Database** – Where all content is stored, the servers interface into this using the shared database library.

# Sub System Architecture

The service’s structure can be broken down into 6 main sub-systems:

## The Database

This is where user, sheet, feed and item information is stored. It is used by all other modules, where access is provided using the shared database library.

## The Front-End

This is the interface of the system. It consists of a number of web pages accessible by users and administrators. It provides the means by which the users and administrators interact with the system (including the back-end). This interface provides methods for users to manage and view feeds, as well as managing the layout used to display these feeds. The interface also provides methods for administrators to manage users, feeds and items, as well as allowing them to manage and monitor the back-end processes.

The Front-End is developed using the Model View Controller pattern in order to use a standardized way of accessing data and integrate with other systems such as the Shared Database Library.

## **The Back-End**

The Back-End does not use conventional design patterns, instead it uses a pattern named the "independent module pattern".

### Independent Module Pattern

Unlike the traditional module pattern, a module is not a Singleton but a class inherited from a module prototype which consists of common components and interfaces. All modules are managed by a master class and can be dynamically loaded or unloaded as required, so they can run in their own threads.

This pattern provides the following advantages:

1. It allows for multiple instances of a module to exist at any one time.
2. It allows for efficient data sharing between modules.
3. Running the modules in threads is less performance heavy (particularly in memory).
4. Modules can be easily managed (such as started and stopped)

This pattern has one disadvantage, namely if a module incurs a fatal error, causing that process to crash, all modules are affected.

## The Downloader Application

### Dependencies

The downloader application directly depends on the web service API. All communication between the downloader application and other News Feeder subsystems is channelled through the Web Service API.

There are no subsystems which depend on the downloader application.

### Portability

To allow for the largest audience possible, the downloader application is written in Java. Java provides a platform independent approach and will allow the downloader application to be used on Windows, Linux, Mac and many more platforms.

### Data Architecture

As previously stated, the downloader application is available for offline use. This requirement implies the use of a secondary offline data source. A secondary data source which will be stored on the client’s machine will only hold the information it requires to display the content to the user based on their view. It will provide a mechanism to clean the data and maintain it by downloading the most up to date data from the Web Service API.

### Application Design

The primary design pattern used in the design and development of the downloader application is the Model View Controller pattern. The application will also have a background process which will run while the process sits in the task tray. The background process will perform data maintenance tasks on the secondary data source and

## The Web Service

### Dependencies

The web service API directly depends on the News Feeder Shared Database subsystem. The web service API does not directly communicate with any other subsystems.

The only subsystem which depends on the web service API is the downloader application. Without the web service, the downloader application would be useless.

It is also important to mention here that the web service may be released to a public audience, allowing third party developers to leverage the News Feeder Service. Any third party applications developed which leverage the News Feeder Service would then depend on the web service API.

### Web Service Abstraction Layer

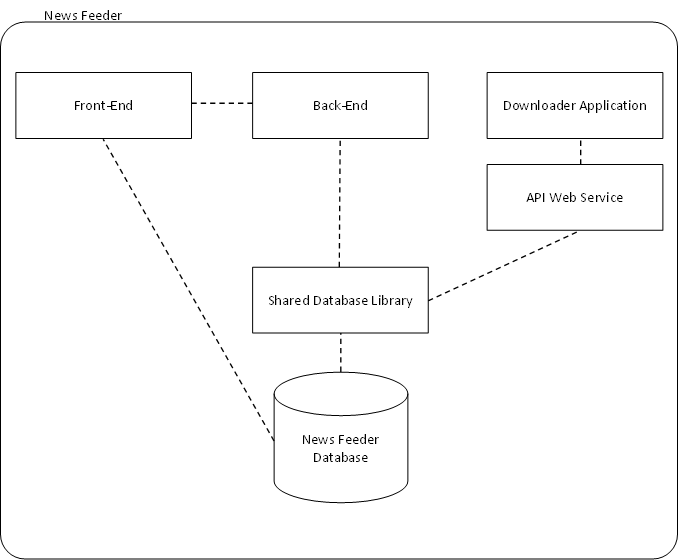
The creation of the web service itself was an architectural decision made primarily to prevent direct database connectivity from the downloader application. However, having a web service also has other advantages to the overall architecture of the system including:

* Enhanced security - Preventing external direct access to the database server will prevent attacks on the database
* Enhanced Control – Only allow the downloader application and other third party applications to execute a controlled set of queries/commands. If the downloader application was discontinued, or a security breach was detected, the web service could be shut down without affecting the core service.
* Public API – The web service may allow third party application developers to leverage the News Feeder Service.
* Load balancing / separation / scalability – If the system is required to be expanded the URL used for the web service can be load balanced to multiple servers to handle external requests, those servers can then utilize their own database cluster separate to the front end.

## The Shared Database Library

The Shared Database Library is designed using the Model View Controller pattern. Where the models are classes representing of the database records that are to be altered/retrieved and the controllers are the classes called in order to modify these models. The views in relation to this library are the other sub-systems which access it (namely the front-end, back-end and API web service) as they are using the library to retrieve the information from the models for use.

The Model View Controller pattern was chosen because the shared database library is designed to act as a standardised interface used by other sub-systems. As such, each sub-system will view and interact with the data in different ways. This pattern enables the other sub-systems (acting as views) to interact with the database simply by using the library’s associated controller classes and then processing the returned information as needed. The additional advantage of choosing this pattern is that any major changes regarding database access/interaction (such as changing to another database type or changing the schema) means the changes only have to be made once, in this library, rather than on all of the other subsystems.



# System Interaction

## Sub-systems and the Database

The all sub-systems need to communicate to the database in order to store and retrieve information.

For both the back-end and web service, this access is provided via the shared database library, which acts as an interface for the interactions between these sub-systems and the database.

The sub-systems instantiate controller objects from the library and call methods relating to retrieving or storing the entity types associated with each controller. The controller classes then construct SQL queries based on the inputs from the sub-systems.

The shared database library will then attempt to execute these queries on the database, by using connections and methods provided by the MySQL++ library. The results from these queries will be returned to the shared database library, which will then transform them into suitable responses (i.e. a basic response or entities to return) which will be sent back to the sub-system.

The front-end was initially also going to use this, however due to the prototype nature of our development, most of the front-ends database calls were already implemented via another API before the shared library was finished. Rather than change all working code for the front-end we decided it was best to leave it as is. Also, due to the closely integrated nature of PHP and MySQL there was no real need to provide a library for this sub-system.

## Downloader Application to API Web Service

The downloader application communicates to the API web service to obtain information from the News Feeder database. The API web service can be seen as a gateway to the internal systems from the internet, the API web service provides a limited set of functionality targeted at what the downloader application requires.

The downloader application is required to authentication before retrieving any information from the API web service by providing a username and password of a subscribed user who has access to the downloader application functionality.

In the future, if the API web service is released to the public, the security and functional implementation details may be changed.

## Front-End to Back-End

A communication link between the Front-End and Back-End, this is the ‘Administration Service’ which provides methods for the administrators to interact with the back-end (and its configuration) via the web interface on the front-end.

This connection is provided using sockets between the two sub-systems and a custom protocol, which is further explained in the ‘Back-End Protocol’ section later in this document.

Website Design

The News Feeder company site is a very informative place in the system. It gives users, or potential users, of the system a brief look into the inner workings of the team and the project. When designing the company website, emphasis was put into making a clean yet informative website where potential clients may be swayed by the professionalism.

# Domain Registration

The team members decided to register a domain name fairly quickly after initial testing. We found that using IP addresses and ports to navigate through our website was tedious and time wasting. The domain www.News Feeder.com was not available due to it being a parked domain; however the new .CO top-level domain was available. Since the TLD was a retro new look we decided to obtain the domain [www.News Feeder.co](http://www.newsfeeder.co). Through the use of our domain registrars in-build DNS manager we also had access to the creation of subdomains (A records) and mail exchanges (MX records).

# Hosts Purchasing

The purchasing of hosts proved to be a difficult issue for our team to decide on. In addition to the purchase of a domain and hosting for our company site, we needed a host powerful enough to also support the project itself. After looking on the market for a while we purchased the following host:

**Host:** VMPort

**Location:** Kansas City, US

**Virtualization Type:** OpenVZ

**RAM:** 1GB

**Disk:** 60GB

**Bandwidth:** 600GB

**Price:** 4.42 GBP (~$7 AUD)

# Social Media

The use of social media becomes a great marketing tool as they become free advertising. The social media sites, Twitter and Facebook, are among the most popular social media services online and hence the creation of accounts on these was decidedly necessary. Here are screenshots from our social media pages:

## Twitter



## Facebook



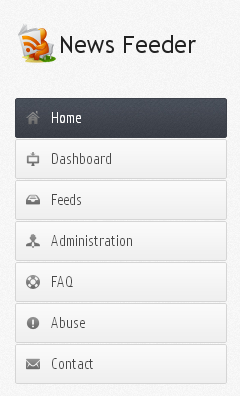
Application Design

# Global User Interface Design

## Header



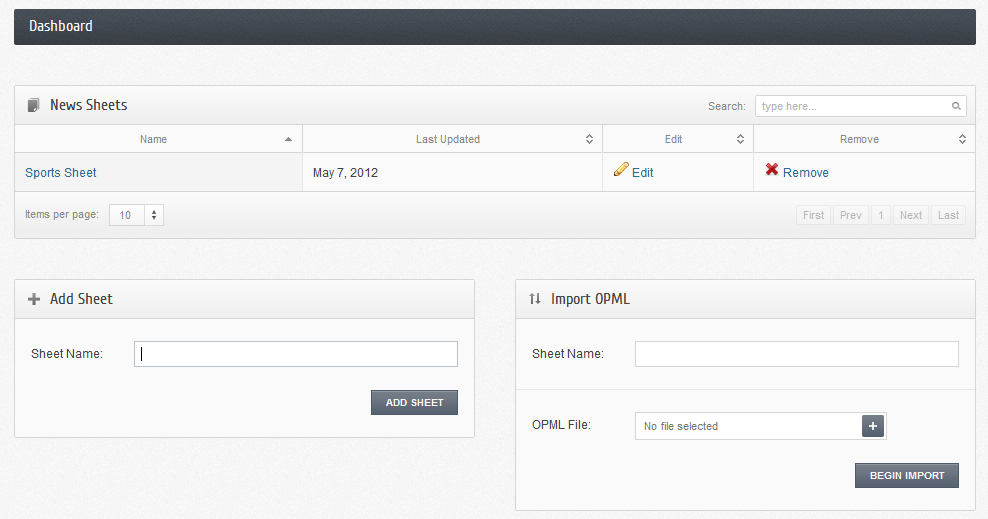
The dark grey navigation bar at the top of the page provides users with a personalized greeting, along with their profile picture to the left and some quick links on the right-hand side. The **Profile** link, to allow users to change settings like name ad e-mail address; the **Messages** link, to give users a quick way to see new updates and changes to sheets; the **Settings** link, to allow users to set global options such as SSL settings; and the **Logout** link for users to sign out of the system.



## Side Bar

The left pane features the side bar. This navigation pane allows users to navigate the various static content of the page and the dynamic content that the dashboard creates. The **Home** link brings users to the index of the website, which features information and statistics about the website as a whole. The **Dashboard** link allows signed in users to navigate their sheets and related options. The **FAQ** link directs users to a page of Frequently Asked Questions about the service. The **Contact** link allows users to provide feedback on the service.

## Main Content



The content of each page is broken up into “widgets” with a wide heading at the top of the content section. These widgets group content to allow easy separation of forms and information. These widgets may span either the full length of the page or be split into left and right sides simultaneously.

## Footer



The dark grey pane at the bottom of each page is a copyright text with a link to the company website.

## Miscellaneous

### Top of Page Button

When navigating long pages that span greater than the length of the screen a small button on the bottom right of the page appears. This follows the users scrolling to stay seemingly stationary.

# Sheet Layouts

## Old Method

This section describes how custom sheet layouts were originally intended to be constructed.

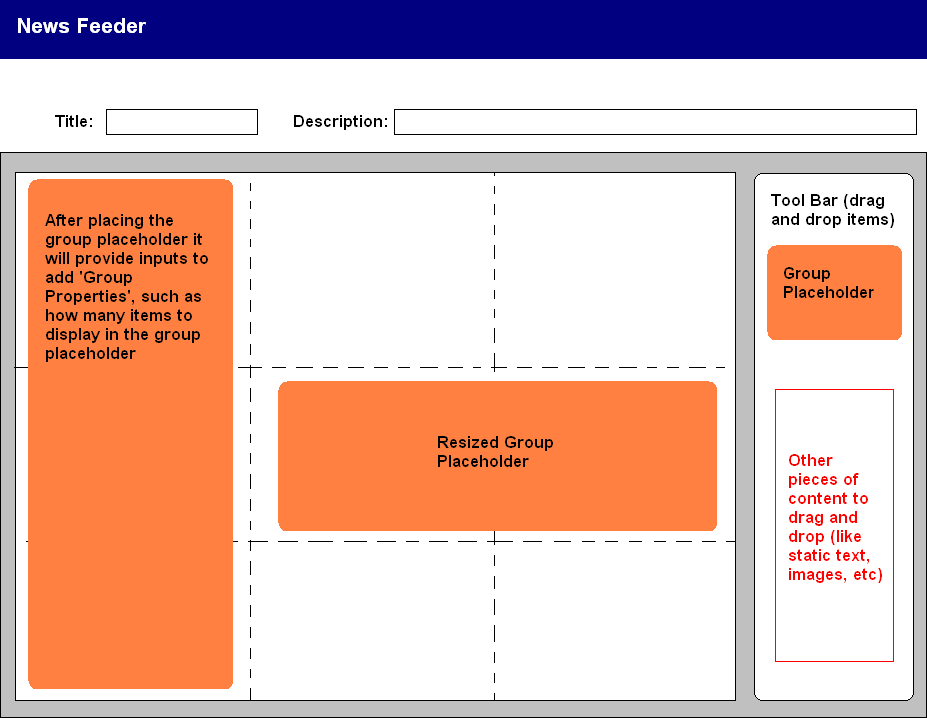
Users can set the style of a sheet (e.g. 2 column or 3 column layout) upon that sheet’s creation; they can either select from a small number of predefined feeds or design their own custom sheet layouts. Custom sheet layouts are designed and managed using 3 main elements:

### Layout

This is the initial design phase, here the user will select how many rows and columns they wish to have on their sheet.

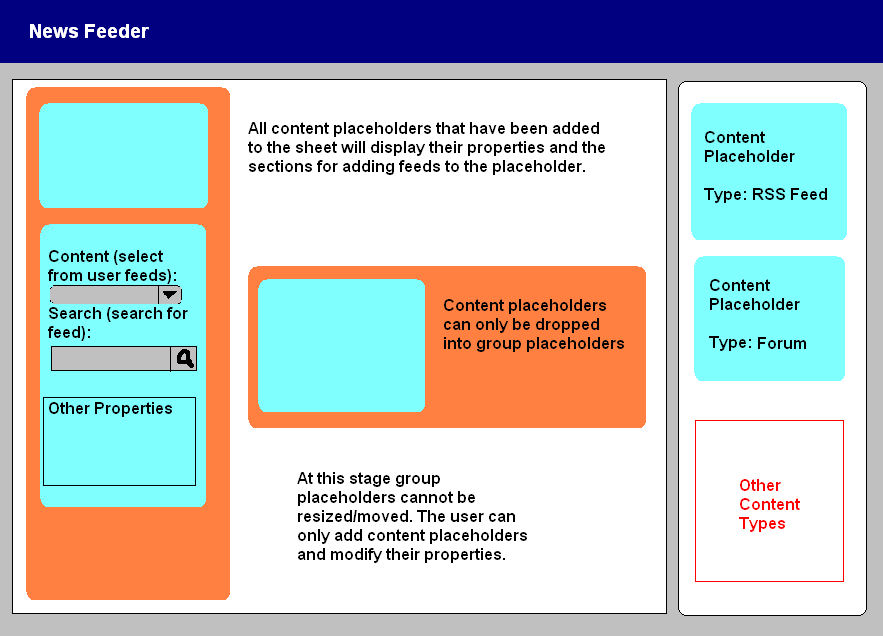
### Group Placeholders

In this phase, the user will be able to click and drag elements called ‘group placeholders’ onto the layout previously created. These group placeholders define the areas where the content should be positioned on the sheet. The placeholders can also be resized to span multiple columns/rows defined by the layout.



### Feed Placeholders

The final phase of the design, here the user clicks and drags elements called ‘feed placeholders’ into the group placeholders available on the sheet. These feed placeholders are the locations content is shown when the sheet is displayed. The feed placeholders come in a number of types, depending on what type of content they are to display (such as RSS feed, weather). After positioning the feed placeholders a user will be able to select the feeds they wish to display inside it (by using the built-in feed search). They can also set a number of properties about how the feeds will be displayed (such as are images show, how many items from each feed).



Based on the layout used for a sheet, when the sheet is rendered it uses the positions of each element to place the content. It uses the layout to generate percentages to see how much room content should take up (e.g. if there are 2 columns, each will take up 50% of the sheet width).

Only administrators and subscribed users will be able to access the first 2 levels of the design process (layout and group placeholders), all other users will only be able to position the feed placeholders within pre-defined layouts.

## New Method

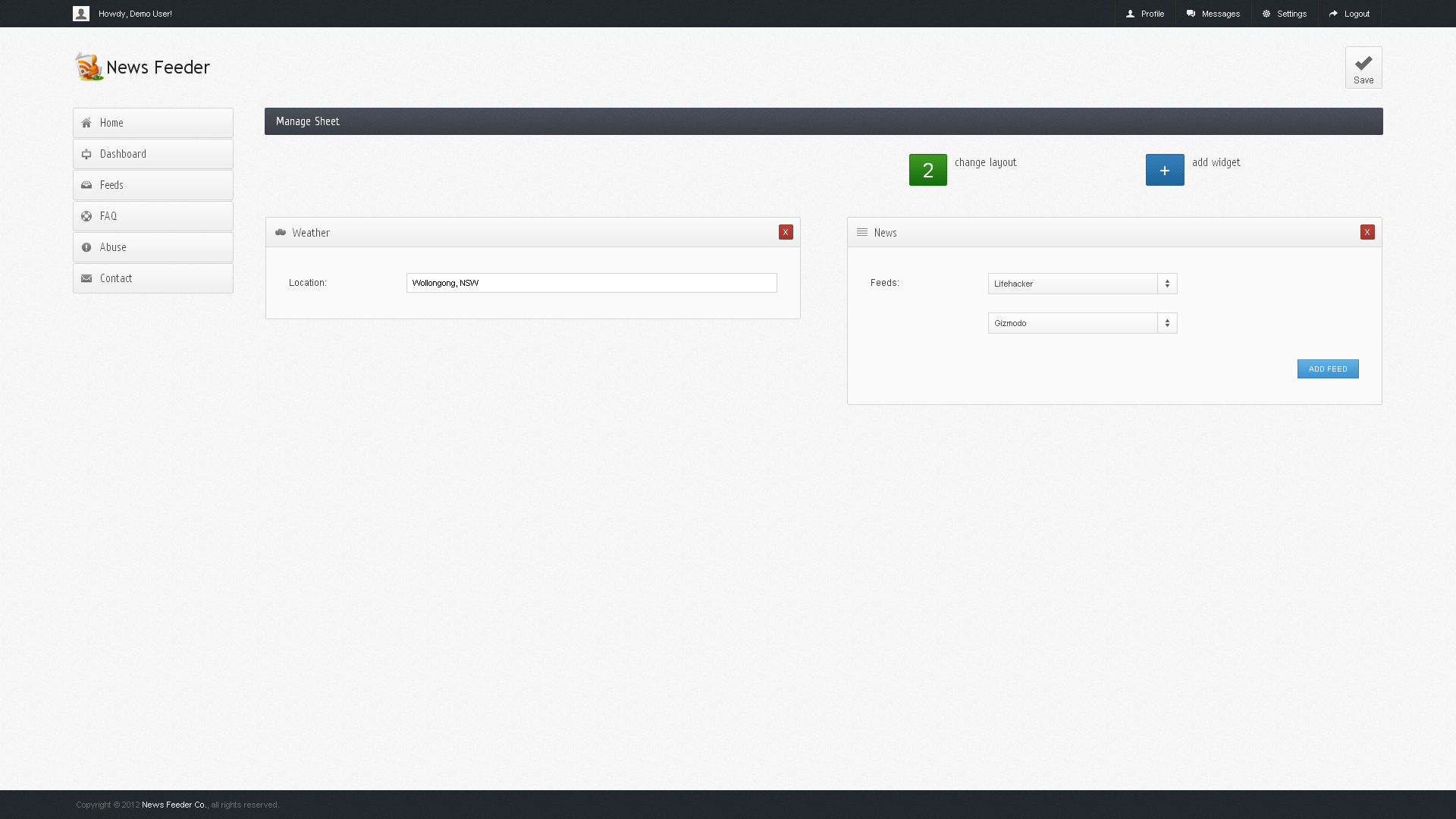
This section describes the method used by the completed system to store the custom layouts.

As opposed to the previous method, this way was simplified greatly and uses only one main element, the Content Placeholder. The main reasons for this simplification were:

* We eliminated the need for Group Placeholders by allowing multiple feeds to be added into a single Content Placeholder. This is done by selecting a feed from a drop down box (with search available) that includes all user specified feeds and pre-defined feeds. Addition drop-down boxes for more feeds can be added by clicking the “Add Feed” button.
* We eliminated the need for Layouts by the removal of rows. This was done as most formats for news sheets are simple column based layouts, and it is unlikely that users would want anything drastically different. Also, this eliminated the problem of it being difficult to determine how high a placeholder would be once filled with content and compensating for that to prevent large areas of whitespace.

In addition to this we also changed the way the user adds the Content Placeholders to the page. Instead of dragging from a side-bar, the user gets a dialog which asks for what type of placeholder and it is then added to the sheet.

Other than these changes, the interface still performs the same way as the previous method, allowing users to move placeholders via drag and drop.



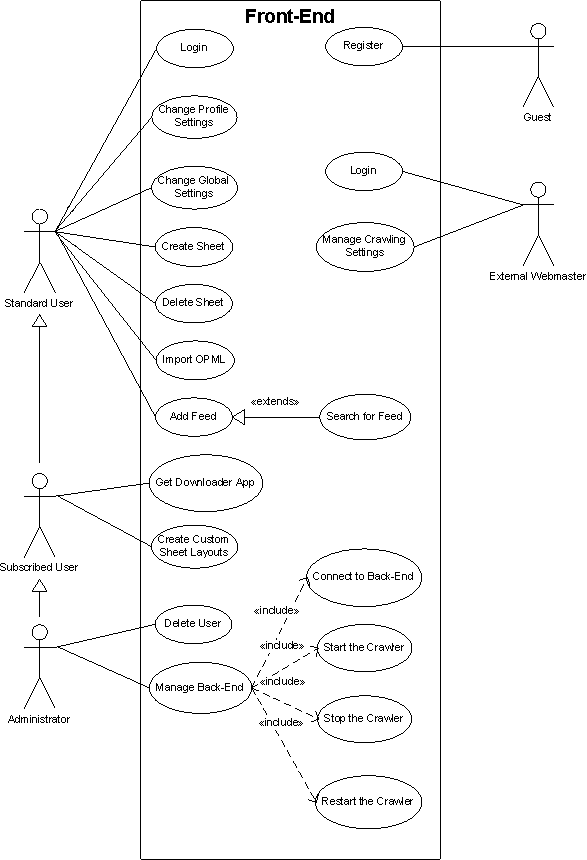
# Search for Feeds

The search for feeds functionality will be provided through an interface which implements a memcache server. When a user requests a search for a particular term to be performed, a separate process begins searching for RSS feeds using Microsoft’s Bing API. When this process obtains results they are stored in the memcache which will subsequently be displayed to the user without them having to reload the page.

Use Cases

# Front-End Use Cases

## Use Case Diagram



## Use Case Scenarios

The following are the scenarios we defined for the above uses cases:

|  |  |
| --- | --- |
| **Use Case Name** | Register |
| **Primary Actor** | Front-End |
| **Summary** | To create a user on the system |
| **Pre-Condition** | N/A |
| **Normal Flow of Events** | 1. Front-End presents a registration form to the user 2. The user submits the form with their details 3. The Front-End presents the user with the login form |
| **Extensions** | Register Fail |
| **Post-Conditions** | N/A |

|  |  |
| --- | --- |
| **Use Case Name** | Login |
| **Primary Actor** | Front-End |
| **Summary** | To authenticate a user to the system |
| **Pre-Condition** | The user has registered an account on the system |
| **Normal Flow of Events** | 1. Front-End presents a login form to the user 2. The user submits the form with their username and password 3. The Front-End presents the user with their dashboard |
| **Extensions** | Login Fail |
| **Post-Conditions** | Authenticated Front-End |

|  |  |
| --- | --- |
| **Use Case Name** | Change Profile Settings |
| **Primary Actor** | Front-End |
| **Summary** | To change details stored in the users profile |
| **Pre-Condition** | The user has logged in to the system |
| **Normal Flow of Events** | 1. The user clicks on the **Profile** link 2. Front-End presents a form to enter updated user details 3. The user submits the form with updated details 4. The system updates the details on the system 5. The user is presented with a success message in their dashboard |
| **Extensions** | N/A |
| **Post-Conditions** | N/A |

|  |  |
| --- | --- |
| **Use Case Name** | Change Global Settings |
| **Primary Actor** | Front-End |
| **Summary** | To change details stored in the users global settings |
| **Pre-Condition** | The user has logged in to the system |
| **Normal Flow of Events** | 1. The user clicks on the **Settings** link 2. Front-End presents a form to enter new settings 3. The user submits the form with updated settings 4. The system updates the settings on the system 5. The user is presented with a success message in their dashboard |
| **Extensions** | N/A |
| **Post-Conditions** | N/A |

|  |  |
| --- | --- |
| **Use Case Name** | Create Sheet |
| **Primary Actor** | Front-End |
| **Summary** | To add a sheet to a user’s account |
| **Pre-Condition** | The user has logged in to the system AND there is no sheet with the same name |
| **Normal Flow of Events** | 1. Front-End presents a field to enter the new sheet name 2. The user submits the new sheet name to the system 3. The system creates a new sheet with the name entered 4. The user is presented with a success message in their dashboard |
| **Extensions** | Create Sheet Fail |
| **Post-Conditions** | N/A |

|  |  |
| --- | --- |
| **Use Case Name** | Delete Sheet |
| **Primary Actor** | Front-End |
| **Summary** | To remove a sheet on a user’s account |
| **Pre-Condition** | The user has logged in to the system AND there exists the sheet to delete |
| **Normal Flow of Events** | 1. Front-End presents a table of sheets to the user 2. The user clicks the **Delete** link in the same row as the sheet they are deleting 3. The Front-End removes the sheet from the users account 4. The user is presented with a success message |
| **Extensions** | N/A |
| **Post-Conditions** | N/A |

|  |  |
| --- | --- |
| **Use Case Name** | Import OPML |
| **Primary Actor** | Front-End |
| **Summary** | To import OPML into a sheet |
| **Pre-Condition** | The user has logged in to the system AND there is no sheet with the same name |
| **Normal Flow of Events** | 1. Front-End presents a form to the user 2. The user submits the new sheet name and OPML file to the system 3. The system creates a new sheet with the name entered and the feeds within the OPML file 4. The user is presented with a success message in their dashboard |
| **Extensions** | N/A |
| **Post-Conditions** | N/A |

|  |  |
| --- | --- |
| **Use Case Name** | Add Feed (Predefined) |
| **Primary Actor** | Front-End |
| **Summary** | To add a predefined feed to a sheet |
| **Pre-Condition** | The user has logged in to the system AND there exists the sheet being modified |
| **Normal Flow of Events** | 1. Front-End presents the user with a list of existing feeds 2. The user clicks on the next empty space in the list 3. The system presents the user with a list of feeds to subscribe to 4. The user picks a feed and the feed is added to the list (the user may repeat from 2) 5. The user clicks the **Confirm** button |
| **Extensions** | N/A |
| **Post-Conditions** | N/A |

|  |  |
| --- | --- |
| **Use Case Name** | Add Feed (From Search) |
| **Primary Actor** | Front-End |
| **Summary** | To add a new feed to a sheet |
| **Pre-Condition** | The user has logged in to the system AND there exists the sheet being modified |
| **Normal Flow of Events** | 1. Front-End presents the user with a query box 2. The user submits a query term with the query box 3. The system presents the user with a list of feeds to subscribe to 4. The user picks a feed and the feed is added to the sheet |
| **Extensions** | N/A |
| **Post-Conditions** | N/A |

|  |  |
| --- | --- |
| **Use Case Name** | Delete User |
| **Primary Actor** | Front-End |
| **Summary** | To delete a user from the system |
| **Pre-Condition** | The administrator has logged in to the system AND there exists the user being deleted |
| **Normal Flow of Events** | 1. Front-End presents the administrator with a list of existing users 2. The administrator clicks on the **Delete** link next to the users username 3. The system will present the administrator with a success message |
| **Extensions** | N/A |
| **Post-Conditions** | N/A |

|  |  |
| --- | --- |
| **Use Case Name** | Connect to Back-End |
| **Primary Actor** | Front-End |
| **Summary** | To connect to the back-end system and retrieve the crawlers status |
| **Pre-Condition** | The administrator has logged in to the system |
| **Normal Flow of Events** | 1. Front-End presents the administrator with a **Connect** button 2. The administrator clicks the **Connect** button 3. The system presents the administrator with the crawlers status and buttons relating to it |
| **Extensions** | N/A |
| **Post-Conditions** | N/A |

|  |  |
| --- | --- |
| **Use Case Name** | Start the Crawler |
| **Primary Actor** | Front-End |
| **Summary** | To connect to the back-end system and start the crawler |
| **Pre-Condition** | The administrator has logged in to the system AND connected to the back-end |
| **Normal Flow of Events** | 1. The system has presented the user with a **Start Crawler** button 2. The administrator clicks the **Start Crawler** button 3. The system provides confirmation the crawler started successfully |
| **Extensions** | N/A |
| **Post-Conditions** | N/A |

|  |  |
| --- | --- |
| **Use Case Name** | Stop the Crawler |
| **Primary Actor** | Front-End |
| **Summary** | To connect to the back-end system and stop the crawler |
| **Pre-Condition** | The administrator has logged in to the system AND connected to the back-end |
| **Normal Flow of Events** | 1. The system has presented the user with a **Stop Crawler** button 2. The administrator clicks the **Stop Crawler** button 3. The system provides confirmation the crawler stopped successfully |
| **Extensions** | N/A |
| **Post-Conditions** | N/A |

|  |  |
| --- | --- |
| **Use Case Name** | Restart the Crawler |
| **Primary Actor** | Front-End |
| **Summary** | To connect to the back-end system and restart the crawler |
| **Pre-Condition** | The administrator has logged in to the system AND connected to the back-end |
| **Normal Flow of Events** | 1. The system has presented the user with a **Restart Crawler** button 2. The administrator clicks the **Restart Crawler** button 3. The system provides confirmation the crawler restarted successfully |
| **Extensions** | N/A |
| **Post-Conditions** | N/A |

# Back-End Use Cases

## Use Case Diagram



## Use Case Scenarios

The following are the scenarios we defined for the above uses cases:

|  |  |
| --- | --- |
| **Use Case Name** | Authenticate |
| **Primary Actor** | Front-End |
| **Summary** | To authenticate whether in-coming client is valid Front-End |
| **Pre-Condition** | The Front-End has a valid pair of username and password |
| **Normal Flow of Events** | 1. Front-End sends username to the Back-End 2. Front-End receive prompt form the Back-End 3. Front-End sends password to the Back-End 4. Front-End receive “success” message |
| **Extensions** | Fail |
| **Post-Conditions** | Authenticated Front-End |
| **Use Case Name** | Fail |
| **Primary Actor** | Front-End |
| **Summary** | To receive the reason of failure |
| **Pre-Condition** | Execution of any use cases |
| **Normal Flow of Events** | 1. Front-End receives fail message from the Back-End |
| **Extensions** | N/A |
| **Post-Conditions** | Disconnected by the Back-End |

|  |  |
| --- | --- |
| **Use Case Name** | Check back-end uptime |
| **Primary Actor** | Front-End |
| **Summary** | To get the uptime of back-end since started |
| **Pre-Condition** | Authenticated Front-End |
| **Normal Flow of Events** | 1. Front-End sends “status uptime” to the Back-End 2. Front-End receives back-end uptime |
| **Extensions** | N/A |
| **Post-Conditions** | N/A |

|  |  |
| --- | --- |
| **Use Case Name** | Check AdminService status |
| **Primary Actor** | Front-End |
| **Summary** | To get the status of AdminService module at the Back-End |
| **Pre-Condition** | Authenticated Front-End |
| **Normal Flow of Events** | 1. Front-End sends “status” to the Back-End 2. Front-End receives status of all modules including AdminService module at the Back-End |
| **Extensions** | N/A |
| **Post-Conditions** | N/A |

|  |  |
| --- | --- |
| **Use Case Name** | Check crawler status |
| **Primary Actor** | Front-End |
| **Summary** | To get the status of Crawler module at the Back-End |
| **Pre-Condition** | Authenticated Front-End |
| **Normal Flow of Events** | 1. Front-End sends “status crawler” to the Back-End 2. Front-End receives status of Crawler module at the Back-End |
| **Extensions** | N/A |
| **Post-Conditions** | N/A |

|  |  |
| --- | --- |
| **Use Case Name** | Start crawler |
| **Primary Actor** | Front-End |
| **Summary** | To start the crawler module at the Back-End |
| **Pre-Condition** | Authenticated Front-End and stopped crawler module |
| **Normal Flow of Events** | 1. Front-End sends “crawler start” to the Back-End 2. Front-End receive “success” message |
| **Extensions** | Fail |
| **Post-Conditions** | Running crawler module |
| **Use Case Name** | Stop crawler |
| **Primary Actor** | Front-End |
| **Summary** | To stop the crawler module at the Back-End |
| **Pre-Condition** | Authenticated Front-End and running crawler module |
| **Normal Flow of Events** | 1. Front-End sends “crawler stop” to the Back-End 2. Front-End receive “success” message |
| **Extensions** | Fail |
| **Post-Conditions** | Stopped crawler module |

|  |  |
| --- | --- |
| **Use Case Name** | Get config attribute value |
| **Primary Actor** | Front-End |
| **Summary** | To get the value of an attribute in the configuration of the Back-End |
| **Pre-Condition** | Authenticated Front-End |
| **Normal Flow of Events** | 1. Front-End sends “config get” along with configuration sector and attribute name to the Back-End 2. Front-End receive “success” message 3. Front-End sends “transfer” to the Back-End 4. Front-End receive the value of requested attribute |
| **Extensions** | Fail |
| **Post-Conditions** | N/A |

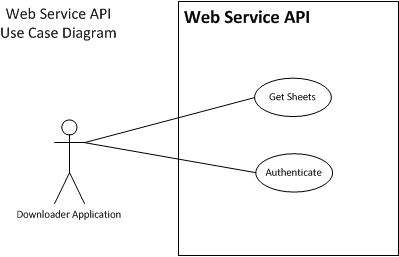
|  |  |
| --- | --- |
| **Use Case Name** | Change config attribute value |
| **Primary Actor** | Front-End |
| **Summary** | To add or change the value of an attribute in the configuration of the Back-End |
| **Pre-Condition** | Authenticated Front-End |
| **Normal Flow of Events** | 1. Front-End sends “config add” along with configuration sector name, attribute name and value to the Back-End 2. Front-End receive “success” message |
| **Extensions** | Fail |
| **Post-Conditions** | N/A |

|  |  |
| --- | --- |
| **Use Case Name** | Read config from file |
| **Primary Actor** | Front-End |
| **Summary** | To read configuration of the Back-End from a file local to the Back-End |
| **Pre-Condition** | Authenticated Front-End |
| **Normal Flow of Events** | 1. Front-End sends “config load” along with the file name to the Back-End 2. Front-End receive “success” message |
| **Extensions** | Fail |
| **Post-Conditions** | N/A |

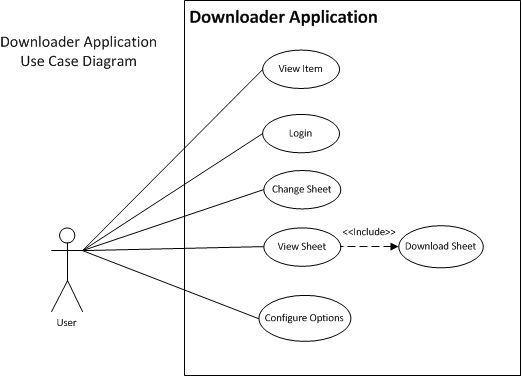
|  |  |
| --- | --- |
| **Use Case Name** | Write config from file |
| **Primary Actor** | Front-End |
| **Summary** | To write configuration of the Back-End to a file local to the Back-End |
| **Pre-Condition** | Authenticated Front-End |
| **Normal Flow of Events** | 1. Front-End sends “config save” along with the file name to the Back-End 2. Front-End receive “success” message |
| **Extensions** | Fail |
| **Post-Conditions** | N/A |

|  |  |
| --- | --- |
| **Use Case Name** | Quit |
| **Primary Actor** | Front-End |
| **Summary** | To disconnect the connection with the Back-End actively |
| **Pre-Condition** | Authenticated Front-End |
| **Normal Flow of Events** | 1. Front-End sends “quit” to the Back-End |
| **Extensions** | N/A |
| **Post-Conditions** | N/A |

# Web Service API Use Case Diagram



# Downloader Application Use Case Diagram



|  |  |
| --- | --- |
| **Use Case Name** | View Item |
| **Primary Actor** | User |
| **Summary** | To view a specific item in the downloader application |
| **Pre-Condition** | Login  View Sheet |
| **Normal Flow of Events** | 1. Click on the title of a specific item on the screen |
| **Extensions** | 1.1 The item was not found, error |
| **Post-Conditions** | 1. The item view window opens and displays detailed information about the item |

|  |  |
| --- | --- |
| **Use Case Name** | Login |
| **Primary Actor** | User |
| **Summary** | Login into the downloader application to view sheets/items and download them for offline use. |
| **Pre-Condition** | N/A |
| **Normal Flow of Events** | 1. Open the downloader application  2. Enter the username and password  3. Select the Login button |
| **Extensions** | 3.1 The username and password are incorrect, an error is displayed to the user |
| **Post-Conditions** | 1. The user is logged into the downloader application  2. The user is presented with the view sheet screen |

|  |  |
| --- | --- |
| **Use Case Name** | Change Sheet |
| **Primary Actor** | User |
| **Summary** | To change the current sheet view to a different sheet. |
| **Pre-Condition** | Login |
| **Normal Flow of Events** | 1. Select the desired sheet from the dropdown |
| **Extensions** | 1.1 A database error occurs and the sheet is not found or not cached for offline use, an error is presented to the user |
| **Post-Conditions** | 1. The sheet view changes to the selected sheet |

|  |  |
| --- | --- |
| **Use Case Name** | View Sheet |
| **Primary Actor** | User |
| **Summary** | To view the current selected sheet |
| **Pre-Condition** | Login |
| **Normal Flow of Events** | 1. The user selects the news feeder icon in the task tray |
| **Extensions** | N/A |
| **Post-Conditions** | 1. The view sheet screen of the downloader application shows and the current sheet is visible to the user |

|  |  |
| --- | --- |
| **Use Case Name** | Configure Options |
| **Primary Actor** | User |
| **Summary** | To configure options available to the user regarding the behaviour of the downloader application |
| **Pre-Condition** | Login |
| **Normal Flow of Events** | 1. The user selects the options button from the task tray or from the downloader application screen  2. The user alters the options as per their preferences  3. The user selects the save button |
| **Extensions** | 3.1 A validation error occurs on the options and the user is presented with an error message  2.1 The user selects the cancel button to cancel the editing of the options |
| **Post-Conditions** | 1. The options screen is closed |

|  |  |
| --- | --- |
| **Use Case Name** | Download Sheet |
| **Primary Actor** | User |
| **Summary** | To download a sheet in a PDF format |
| **Pre-Condition** | Login |
| **Normal Flow of Events** | 1. The user selects the PDF icon on their sheet  2. The user selects where they would like to save the PDF |
| **Extensions** | N/A |
| **Post-Conditions** | 1. A PDF containing detailed information about the items in the current sheet is downloaded into the directory the user specified. |

Database Design

# Data Dictionary

## Table: censor

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Primary Key** | **Null?** | **Default Value** | **Description** |
| id | INTEGER | ✔ | NO | None | Unique identifier |
| word | TEXT |  | NO | None | Word that must be censored |

## Table: comments

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Primary Key** | **Null?** | **Default Value** | **Description** |
| id | INTEGER | ✔ | NO | None | Unique identifier |
| itemid | INTEGER |  | NO | None | The item the comment resides in |
| author | TEXT |  | NO | None | The author of the comment |
| via | TEXT |  | YES | NULL | The site where the comment was made |
| date | TIMESTAMP |  | YES | NULL | The date the comment was published |
| comment | TEXT |  | NO | None | The comment text |
| avatar | MEDIUM BLOB |  | YES | NULL | The comment authors avatar |

## Table: cph

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Primary Key** | **Null?** | **Default Value** | **Description** |
| id | INTEGER | ✔ | NO | None | Unique identifier |
| name | TEXT |  | NO | None | The name of the content placeholder |
| columns | INTEGER |  | NO | None | The number of columns the content placeholder spans |
| groupphid | INTEGER |  | YES | NULL | The ID of the group placeholder it belongs to |

## Table: cph\_feeds

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Primary Key** | **Null?** | **Default Value** | **Description** |
| id | INTEGER | ✔ | NO | None | Unique identifier |
| cphid | INTEGER |  | NO | None | The ID of the content placeholder it belongs to |
| feedid | INTEGER |  | NO | None | The ID of the sheet the feeds reside in |
| censor | INTEGER |  | NO | 0 | The level of censoring to apply |

## Table:cph\_weather

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Primary Key** | **Null?** | **Default Value** | **Description** |
| id | INTEGER | ✔ | NO | None | Unique identifier |
| cphid | INTEGER |  | NO | None | The ID of the content placeholder it belongs to |
| location | VARCHAR |  | NO | None | The location to retrieve weather from |

## Table: feeds

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Primary Key** | **Null?** | **Default Value** | **Description** |
| id | INTEGER | ✔ | NO | None | Unique identifier |
| url | TEXT |  | NO | None | Word that must be censored |
| name | TEXT |  | NO | None | The name of the feed |
| frequency | INTEGER |  | NO | 0 | Update frequency (mins) |
| lastupdate | TIMESTAMP |  | YES | NULL | Time of last update |
| category | TEXT |  | YES | NULL | Category of feed |
| type | INTEGER |  | NO | None | Integer representing feed type (RSS, ATOM etc.) |

## Table: images

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Primary Key** | **Null?** | **Default Value** | **Description** |
| id | INTEGER | ✔ | NO | None | Unique identifier |
| itemid | INTEGER |  | NO | None | The item the image resides in |
| image | LARGE BLOB |  | NO | None | The binary image |

## Table: items

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Primary Key** | **Null?** | **Default Value** | **Description** |
| id | INTEGER | ✔ | NO | None | Unique identifier |
| feedid | TEXT |  | NO | None | Word that must be censored |
| title | TEXT |  | NO | None | The title of the article |
| url | TEXT |  | NO | None | The URL of the original |
| content | TEXT |  | NO | None | The article text |
| postdate | TIMESTAMP |  | YES | NULL | The publish date of the article |
| author | TEXT |  | YES | NULL | The authors name |
| geolocation | TEXT |  | YES | NULL | The co-ordinates of the article |

## Table: notifications

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Primary Key** | **Null?** | **Default Value** | **Description** |
| id | INTEGER | ✔ | NO | None | Unique identifier |
| username | TEXT |  | NO | None | Username the notification is for |
| sheetid | INTEGER |  | NO | None | The ID of the sheet that was updated |

## Table: session

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Primary Key** | **Null?** | **Default Value** | **Description** |
| id | INTEGER | ✔ | NO | None | Unique identifier |
| username | TEXT |  | NO | None | The username of the users session |
| key | TEXT |  | NO | None | A unique session identifier |
| time | TIMESTAMP |  | NO | None | When the session was created |

## Table: sheets

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Primary Key** | **Null?** | **Default Value** | **Description** |
| id | INTEGER | ✔ | NO | None | Unique identifier |
| layoutid | INTEGER |  | NO | None | The layout of the sheet |
| name | TEXT |  | NO | None | Word that must be censored |
| username | TEXT |  | NO | None | The username of the creator of the sheet |
| updated | TIMESTAMP |  | YES | NULL | The time of last update |

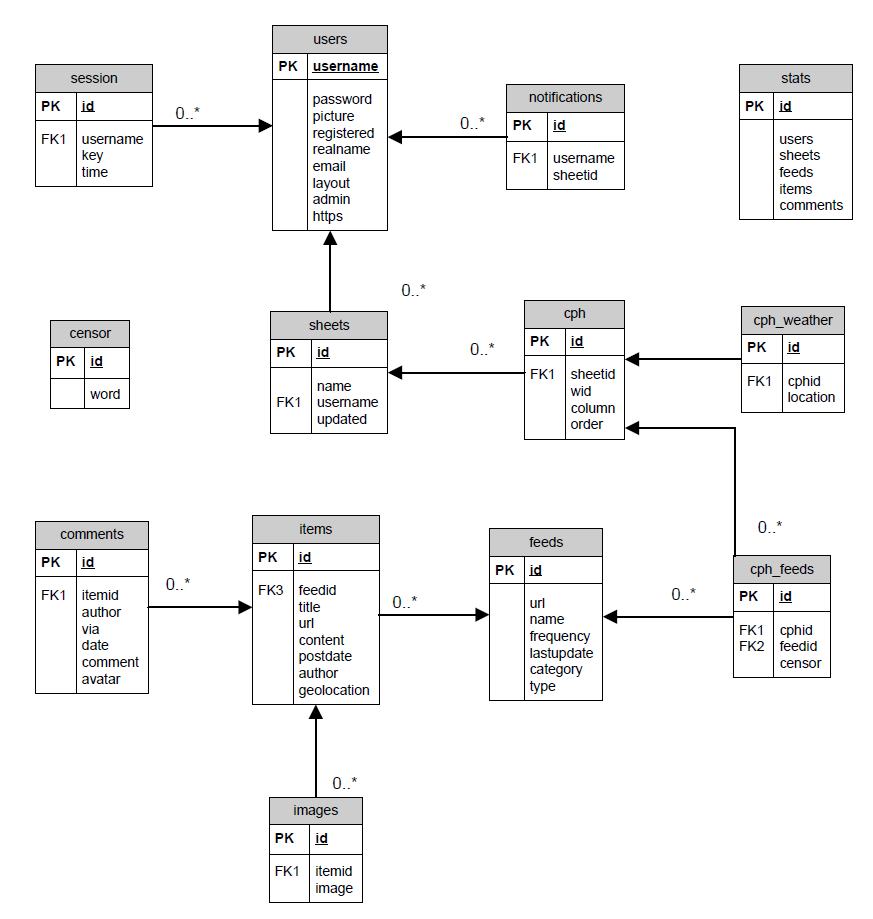
## Table: stats

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Primary Key** | **Null?** | **Default Value** | **Description** |
| id | INTEGER | ✔ | NO | None | Unique identifier |
| users | INTEGER |  | NO | None | Number of users in the system |
| sheets | INTEGER |  | NO | None | Number of sheets in the system |
| feeds | INTEGER |  | NO | None | Number of feeds in the system |
| items | INTEGER |  | NO | None | Number of items in the system |
| comments | INTEGER |  | NO | None | Number of comments in the system |

## Table: users

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Primary Key** | **Null?** | **Default Value** | **Description** |
| username | TEXT | ✔ | NO | None | The username of the user |
| password | TEXT |  | NO | None | The password of the user |
| picture | MEDIUM BLOB |  | YES | NULL | The binary image of the user |
| registered | TIMESTAMP |  | NO | None | The timestamp of when the user first created their account |
| realname | TEXT |  | NO | None | The users full name |
| email | TEXT |  | NO | None | The users e-mail address |
| layout | INTEGER |  | NO | 0 | An integer representing which interface layout the user prefers |
| admin | BOOLEAN |  | NO | False | Represents whether the user is an administrator |
| https | BOOLEAN |  | NO | True | If true, the service will force HTTPS protocol |

# Entity Relationship Diagram



0..\*

Interactions

# Back-End Protocol

This section contains defines the protocol which communicates with the front-end of the News Feeder service. Currently, the protocol is plaintext for readability, but this is a stop-gap measure as it offers less efficient and security. In the next version of this protocol will be based on binary, and in future versions, the protocol will be encapsulated by a security layer.

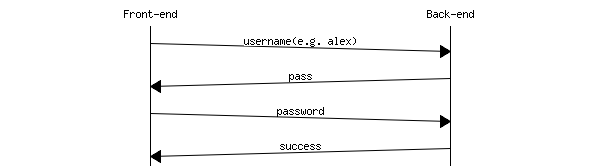
### Sockets

The back-end listens to both Unix sockets and TCP sockets as defined by the config file. The user can turn off the listening to either of these types by modifying the config file. The two different types are included because each type offers advantages based on whether or not the front-end and back-end are based on the same machine. If they are installed on the same machine Unix sockets are suggested and TCP sockets are turned off in order to prevent external attacks. On the hand, if they are installed in different servers, TCP sockets are used and Unix sockets are turned off.

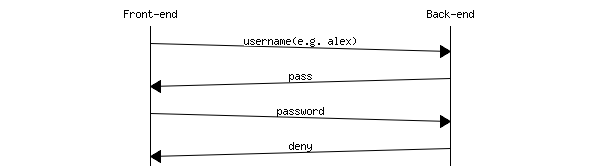
## Login

Before the front-end can communicate with and control the back-end, it must first login and be authenticated. The front-end must provide a username and password for authentication.

### Success case



### Fail case

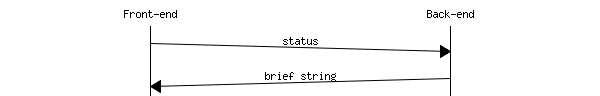


## status - Check the status

The front-end can check the status of the back-end via "status" and combine sub-commands as following:

status [sub-command]

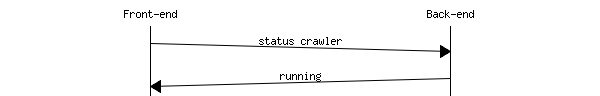
If sub-command is not provided, the back-end will send a brief string to the front-end. This string may not be parsed by the front-end but is human readable.



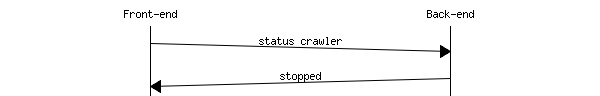
In the later versions, statistical information will be available on the front-end.

### crawler - crawler status

This sub-command "crawler" will tell whether the crawler is running or not.  
If the crawler is currently running, it will return "running":



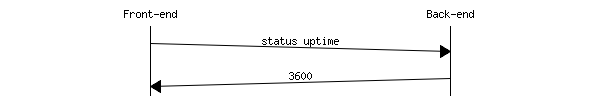
If the crawler is currently stopped, it will return "stopped":



Note: There are 4 statuses: running, stopped, stopping and starting.

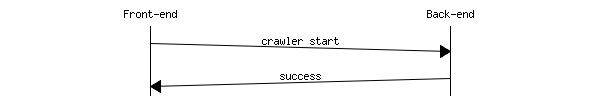
### uptime - Back-end uptime

This sub-command "uptime" will return how many seconds that the back-end has run. For example: (the server has been running for 1 hour)

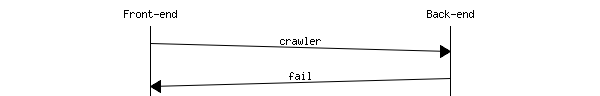


## crawler - Control the crawler

The crawler command in this version of protocol is under the assumption that the back-end is only running one crawler (however, the back-end is designed to handle multiple crawler at one time, so future versions will remove this assumption).  
Same as command status - Check the status, this command is used with sub-commands while a sub-command is required. For example:

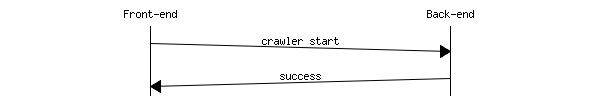


If no sub-command is attached to the command "crawler", the back-end will return a "fail":

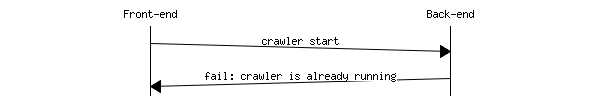


### start - Start the crawler

This sub-command "start" tells the back-end to start the crawler:



If exceptions are thrown and caught by the back-end, it will return a "fail" with the error message separated by a ": ":

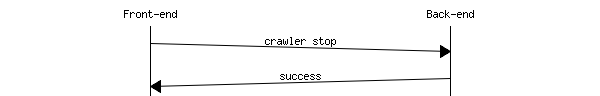


### status - Check the status of the crawler

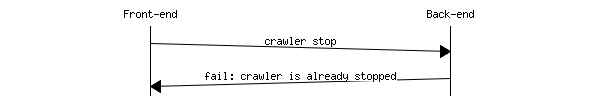
This sub-command "status" performs the same function as the command "status crawler".

### stop - Stop the crawler

This sub-command "stop" tells the back-end to stop the crawler:

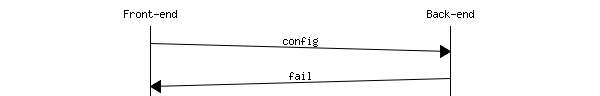


If exceptions are thrown and caught by the back-end, it will return a "fail" with error message separated by a ": ":



## config - Configure back-end

The "config" command allows the direct control of the config information at the back-end. Like crawler - Control the crawler, a sub-command is required when using this command.

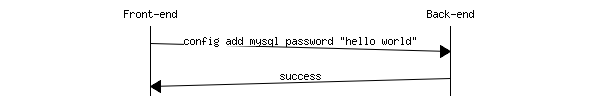


### add - Add/Change an attribute

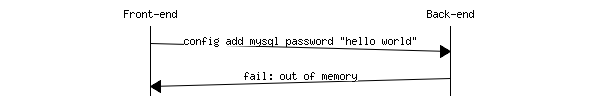
This sub-command "add" tells the back-end to add or change an attribute. The syntax for this command is:

config add sector attribute value

The sector and attribute should not contain whitespaces. It will take everything after attribute as the value of that attribute. For example:



In this example, "mysql" is parsed as sector name, "password" is parsed as attribute name, and "hello world" is parsed as its value. This command may fail (because of memory problem or others) and it will return a "fail" with error message separated by a ": ":



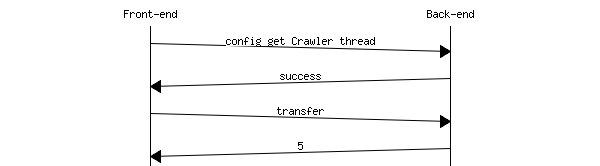
If the required sector or attribute does not exist, it will automatically add a new one. If the attribute is already there, the back-end will change it.

### get - Return an attribute

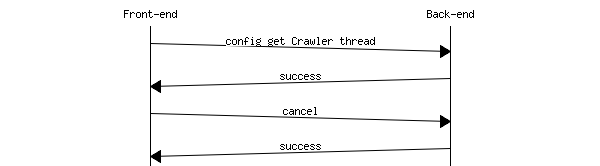
This sub-command "get" tells the back-end to get the value of an attribute. The grammar for this command is:

config get sector attribute

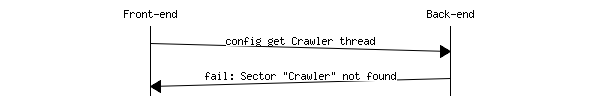
The back-end will search for this attribute as requested. If the request is successful, the back-end will return "success". Otherwise, it will return "fail", followed by an error message. On a successful request, the front-end can get the value back by sending "transfer".



Of course, the front-end can cancel this by sending "cancel" or a garbage string.



Also, if the sector or the attribute is not available:

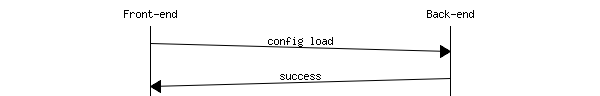


### load - Load configuration file

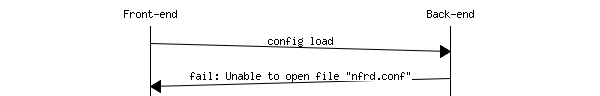
This sub-command "load" tells the back-end to load all configurations from a config file. The grammar for this command is:

config load [file]

Where "file" is optional. Note: The default config file is "nfrd.conf" which is hard coded in the nfrd program. It will return "success" on success or "fail" followed by error message on failure.  
Success case:



Fail case:

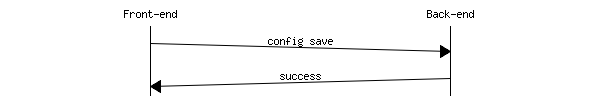


### save - Save configuration file

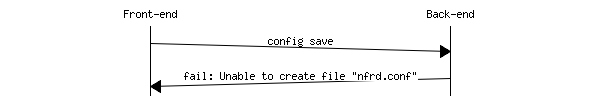
This sub-command "save" tells the back-end to save all configurations to a config file. The grammar for this command is:

config save [file]

Where "file" is optional. Note: The default config file is "nfrd.conf" which is hard coded in the nfrd program. It will return "success" on success or "fail" followed by error message on failure.  
Success case:

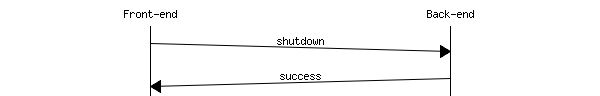


Fail case:

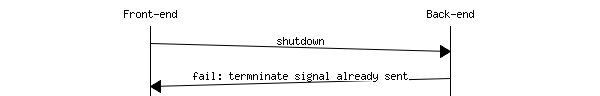


## shutdown - Shut down the Back-End

Shut down the Back-End process in a safe way. Stopping all modules and unloading them so it can then terminate itself. Success case:



Fail case:



Warning:

This command should be used for maintenance only.

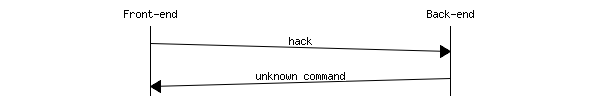
## quit - Close connection

After all communication with the back-end, the front-end should say good-bye, which is "quit" in this case, to the back-end to close the connection between them.

msc_inline_mscgraph_25

## Unknown command handling

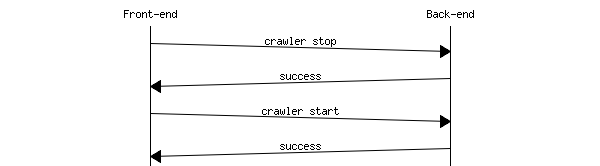
If a command is not supported by the back-end, a message of "unknown command" will be returned.



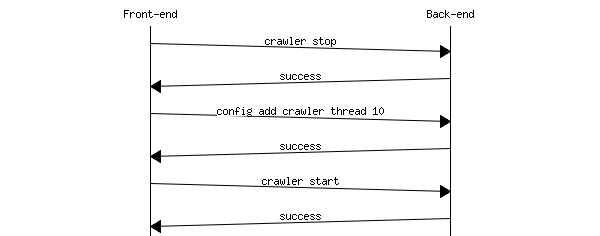
## Examples

Here are some examples for advanced operations.

### Restart the crawler



### Change the number of threads used by the crawler



Algorithms and Logic

# Back-End Logic

## **Overall**

Back-End can be divided into two parts: Master and modules. Master works as a manager of all modules. It loads modules according to the configuration file and can also dynamically start, stop, load, unload modules without restarting the Back-End. The modules are independent by design.

Parse Feed  
The Back-End dynamically using different parser; parses the sources obtained from the URL given by the Crawler according to the source type. After parsing, items are generated and stored in the database by Crawler module via NFDB Shared library.

Patch-Match Algorithm  
*Alias: Match and Patch*. As the name of the algorithm, it involves two phases: Match and Patch.

Matching Phase  
When parsing the web feed (RSS feed or Atom feed), if a feed file does not contain the full text of the content, the Back-End will trace into the original URL of the content and fetch the HTML page using libcurl. The Back-End then has three tries to find the full content and match the brief content (raw matching string) in the feed file to the contents source. If it can be found, the Back-End will obtain the extra article content.

The three tries to find the full content are using different algorithms to generate the matching string. Each try is based on the previous try.

1. Initial try
   1. Get the longest text from the brief content (an html source code).  
      For example:  
      <p>short</p><p>longest</p> 🡪 longest
   2. Remove the tailing dots, as following.  
      For example  
      This is matching string generation… 🡪 This is matching string generation
2. Second try: use the longest line in the current matching string.

For example:

This is a normal string.

This is the longest string.

This is a longer string.

Note: The matching string to be processed has three lines.

The result matching string will be “This is the longest string”

1. Third try: Divide the matching string and use the mid part.

Detail algorithm is base off the following equations

Where sizetarget is the size of the processed matching string, sizeoriginal is the size of the matching string in the second try. The SIZEMODIFIER is a constant value, currently 3.

Patching Phase  
After matching the full content (or use the original content if Match fails), the parser starts to extract information.

1. Extract images and make them into 100x100 thumbnails. The URLs of the images are converted to be absolute rather than relative.
2. Extract geographical location

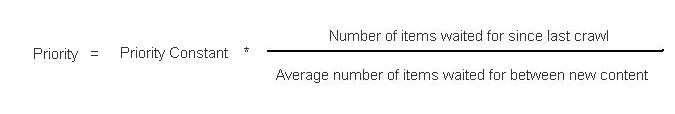
Advanced parsing work flow (activity diagram)  
A work flow of how advance parsing works (FeedXParser).



Statistics  
This module to periodically update the statistics information. It sleeps while idle.

Crawler Queuing  
The back-end uses a priority queue (i.e. a heap) in order to store the feeds to be crawled. This queue is run continuously with feeds being taken off, to be crawled and parsed, then added back onto the end of the queue.

The priority used for the queuing is based off the following algorithm:



This algorithm is re-calculated every time an item (or batch of items) is taken off the queue. As the number of wait increases, the priority increases meaning it is more likely to be chosen. Also, feeds with less time between new content (i.e. feeds that are updated more quickly) will increase in priority much more quickly than other feeds.

The priority constant is a value that will always be greater than 1, with this value being based off a number of feed factors (such as the number of people subscribed and the type of feed). It is only calculated on the initial queue start and at spaced, regular intervals.

Usage of third-party libraries

# libcurl

libcurl is a free and easy-to-use client-side URL transfer library under a MIT/X derivate license. This library is used for easy fetching file from the Internet at Back-end. For example: Fetch RSS files from a website and pass it to a RSS parser.

# gdLibrary

gdLibrary, a.ka. libgd2, is an open source library for the dynamic creation of images. It is used by the Back-end for generating thumbnails for the images crawled.

# RapidXML

RapidXML is an attempt to create the fastest XML parser possible, while retaining useability, portability and reasonable W3C compatibility written in C++. It is used as an XML parser for the Back-End. For example: Parse RSS feed items from a RSS file obtained using libcurl.

# htmlcxx

htmlcxx is an open source C++ library that converts a HTML document into a DOM hierarchy so the developer can easily access the children and parent of each node. It is used by the parser in the Back-End to perform patch and match.

# SimplePie

SimplePie is a PHP class that allows RSS and ATOM feeds to be parsed in PHP. The class includes functions to process data within the feeds and obtain it in an appropriate way. For example: Attempt to parse a feed and detect whether it is valid or not.

# MySQL++

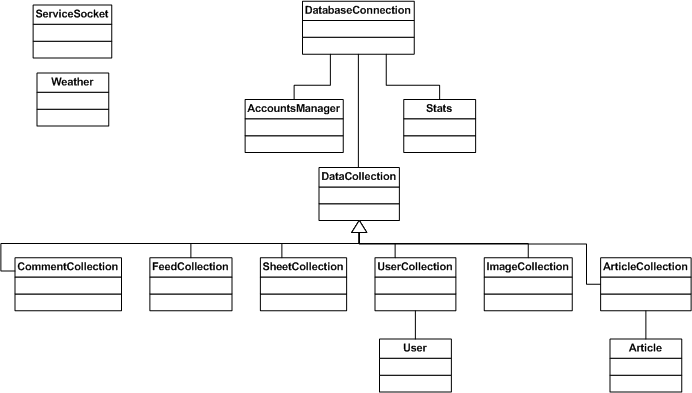
MySQL++ is a C++ wrapper for MySQL’s C API that is designed around the same principles as a standard C++ library. It is by the Shared Database Library to interact with the MySQL database with a C++ interface.

# SWIG

SWIG (Simplified Wrapper and Interface Generator) is an open source software tool used to connect C or C++ libraries with other programming and scripting languages. It is used to construct the interfaces for the shared database library, so it can be used by the Back-End (C++) and Web Service (Java).

Front-End Class Documentation

# Front-End Class Diagram



# Front-End Class Descriptions

|  |  |
| --- | --- |
| **Full name of the class** | **Brief Description** |
| AccountsManager | Provides methods for the creation and authentication of user accounts |
| Article | Provides storage class with getters for a single article |
| ArticleCollection | Provides a storage class for Articles in a particular sheet |
| CommentCollection | Provides a storage class for Comments on a particluar item |
| DatabaseConnection | A database controller with methods for each SQL related query |
| DataCollection | Provides a generic storage class with functions for extended classes |
| FeedCollection | Provides a storage class for Feeds in a particular sheet |
| ImageCollection | Provides a storage class for Images in a particular article |
| ServiceSocket | Connects to the Back-End controlling service |
| SheetCollection | Provides a storage class for Sheets that belong to a user |
| Stats | Generates front page statistics |
| User | Identifies a particular user of the service |
| UserCollection | Provides a storage class for all users of the system for administration purposes |
| Weather | Calls the Google Weather API for weather data |

Back-End Class Documentation

# Back-End Class Diagram



# Back-End Class Descriptions

## Namespace nfrd

Contains all classes unique to News Feeder Refresh Daemon

|  |  |
| --- | --- |
| **Full name of the class** | **Brief Description** |
| nfrd::Master | Manage, contain and access all components of nfrd |

## Namespace nfrd::config

Contains classes related to configuration

|  |  |
| --- | --- |
| **Full name of the class** | **Brief Description** |
| nfrd::config::ConfigSector | A part of ConfigManager (as a container) |
| nfrd::config::ConfigManager | Manages config files (core class).  Note: The config is stored in the memory. |

## Namespace nfrd::log

Contains classes related to log

|  |  |
| --- | --- |
| **Full name of the class** | **Brief Description** |
| nfrd::log::LogManager | Manage logs, writing logs to the local file. |

## Namespace nfrd::misc

Contains classes that commonly used in other classes

|  |  |
| --- | --- |
| **Full name of the class** | **Brief Description** |
| nfrd::misc::Utility | Contains all utility functions |
| nfrd::misc::DateTime | A class to store date and time |
| nfrd::misc::Image | A class to store image and process image |
| nfrd::misc::Image::File | A class to represent an image file (as an ownner) |

## Namespace nfrd::module

Contains classes that control different modules

|  |  |
| --- | --- |
| **Full name of the class** | **Brief Description** |
| nfrd::module::AdminService | Manage sockets talk to the front end and interacts with other components |
| nfrd::module::AdminServiceThread | Handle sockets talk to the front end and interacts with other components |
| nfrd::module::Crawler | The main class representing the crawler module The responsibilities of this class:   * Management of priority queue * Management of worker threads * Responsible for maintaining priority queue health |
| nfrd::module::CrawlerThread | A worker class representing a thread.  This class will poll the priority queue for more work and then execute the work. |
| nfrd::module::FeedPriorityQueue | Implements a queueing/threading model. The model was designed to  sort and prioritise the feeds in the background while providing a  fast way of delegating work to background feeds. So to summarise  the responsibilities of this class:   * Proritising all feeds for processing * Delegating work to threads (in a consumer/producer type model) |
| nfrd::module::Module | A generalised module interface class, providing all the interfaces of a module that start or stop. Since it is an abstract class, all pure virtual functions have to be implemented by the subclasses. |
| nfrd::module::QueueItem | A queue item used by Crawler |
| nfrd::module::Statistics | Periodly record and update the statistics information. |

## Namespace nfrd::parser

Contains classes related to all kinds of feed parsers.

|  |  |
| --- | --- |
| **Full name of the class** | **Brief Description** |
| nfrd::parser::AtomItem | A class to store details of an item obtained by the AtomParser |
| nfrd::parser::AtomParser | A parser to parse Atom feeds, using standard: <http://tools.ietf.org/html/rfc4287> |
| nfrd::parser::FeedItem | A class to store details of an item obtained by the FeedParser |
| nfrd::parser::FeedParser | A parser to parse web feeds |
| nfrd::parser::FeedXParser | A parser to parse web feeds with Patch and Match feature. This parser will automatically identify feed type (RSS or Atom) |
| nfrd::parser::Item | A class to store details of an item obtained by the Parser.  The relationship with Parser is many-to-one, that Parser can have many Item. The attribute can be added to this class at anytime and it will not make subclasses unable to compile. Instead, all subclasses will automatical derive new attributes. |
| nfrd::parser::Parser | A generalised parser interface class, providing all the interfaces of a class that reads resource from an URL and parse it into a list of Item. Since it is an abstract class, all pure virtual functions have to be implemented by the subclasses. |
| nfrd::parser::RSSItem | A class to store details of an item obtained by the RSSParser |
| nfrd::parser::RSSParser | A parser to parse RSS feeds, using standard: <http://cyber.law.harvard.edu/rss/rss.html> |

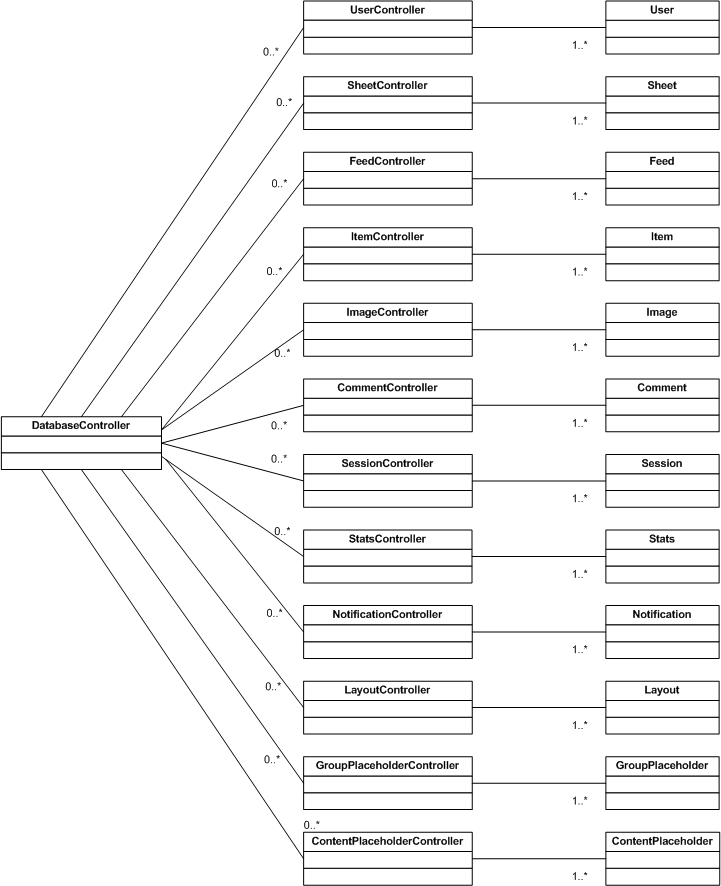
# Back-End Exception Hierarchy



|  |  |
| --- | --- |
| **Full name of the class** | **Brief Description** |
| nfrd::module::ModuleException | General exception for module |
| nfrd::config::ConfigException | General exception for config |
| nfrd::config::IOException | Input/Output exception for config |
| nfrd::config::ItemNotFound | Sector or attribute not found |
| nfrd::parser::ParserException | General exception for parse |
| nfrd::parser::HasNoValue | Has no value exception |
| nfrd::parser::InvalidSource | Invalid source exception |
| nfrd::log::LogException | General exception for log |
| nfrd::log::IOException | Input/Output exception for log |
| nfrd::misc::ImageException | General exception for Image |

Shared Database Library Class Documentation

# Shared Database Library Class Diagram



The shared database library is designed in the model view controller pattern; as such it has the following types of classes:

## Entities

These classes are simply structures that represent the tables in the database. They consist of a number of private member variables that relate to the columns in the tables, as well as accessor/mutator methods to operate on these variables.

|  |  |
| --- | --- |
| **Full name of the class** | **Brief Description** |
| User | Represents the users (and administrators) who access the system. |
| Sheet | Represents a sheet/view which displays the feeds in the system. |
| Feed | Represents a news (or other content) feed in the system. |
| Item | Represents a news item or piece of content from a feed |
| Image | Represents an image for a particular item. |
| Comment | Represents a comment made by a user on a particular item. |
| Session | Represents one of the of the currently active user sessions. |
| Stat | Represents statistics logged by the Back-End. |
| Notification | Represents the notifications shown to the user. |
| Layout | Represents the layout of a sheet in terms of rows and columns. |
| GroupPlaceholder | Represents an area where content placeholders can be placed on a sheet. |
| ContentPlaceholder | Represents a placeholder where content can be |

## Controllers

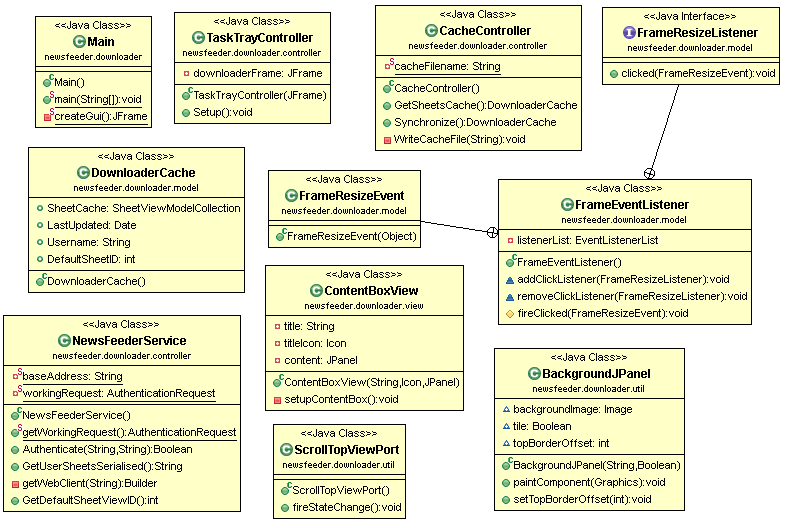
These classes are the interfaces used by the other sub-systems to interact with the database. They consist of a number of methods which are used to find, modify, delete and create entities. Each controller interacts solely with its related entity class. They return entities (individually or in collections) or responses based upon which of these actions they are performing.

|  |  |
| --- | --- |
| **Full name of the class** | **Brief Description** |
| UserController | Called to create, update, delete or find users. It also includes a method which is used to authenticate the user. |
| SheetController | Called to create, update, delete or find sheets. |
| FeedController | Called to create, update, delete or find feeds. |
| ItemController | Called to create, update, delete or find items. |
| ImageController | Called to create, update, delete or find items. |
| CommentController | Called to create, update, delete or find comments. |
| SessionController | Called to create, delete or find sessions. |
| StatController | Called to create, delete or find statistic records. |
| NotificationController | Called to create, delete or find user notifications. |
| LayoutController | Called to create, update, delete or find sheet layouts. |
| GroupPlaceholderController | Called to create, update, delete or find group placeholders. |
| ContentPlaceholderController | Called to create, update, delete or find content placeholders. |
| DatabaseController | Represents an individual connection to the database. Its methods are used to connect and disconnect from the database. It contains members that hold basic database information (such as the open connection, DBName, URL, etc). All other controllers make use of this class to obtain their connections. |

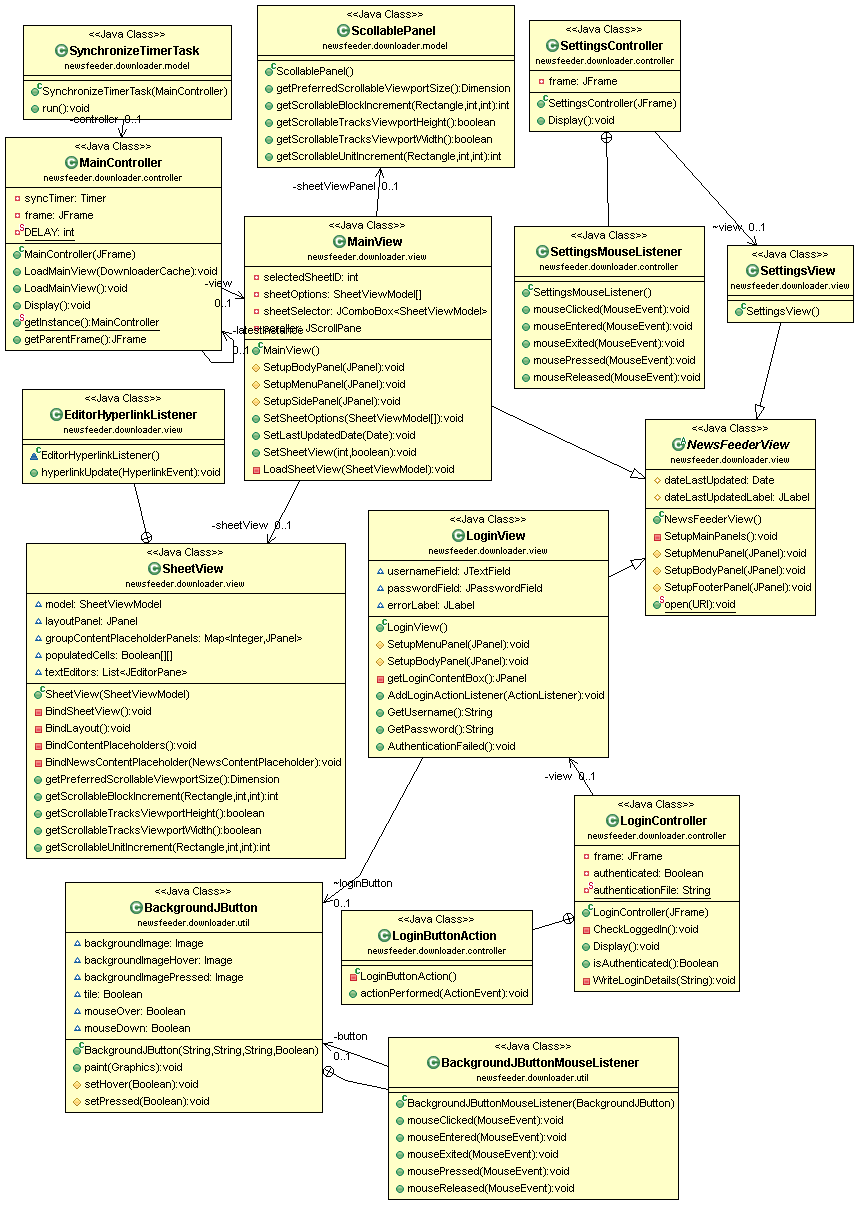
Downloader Application Class Documentation

# Downloader Application Class Diagram

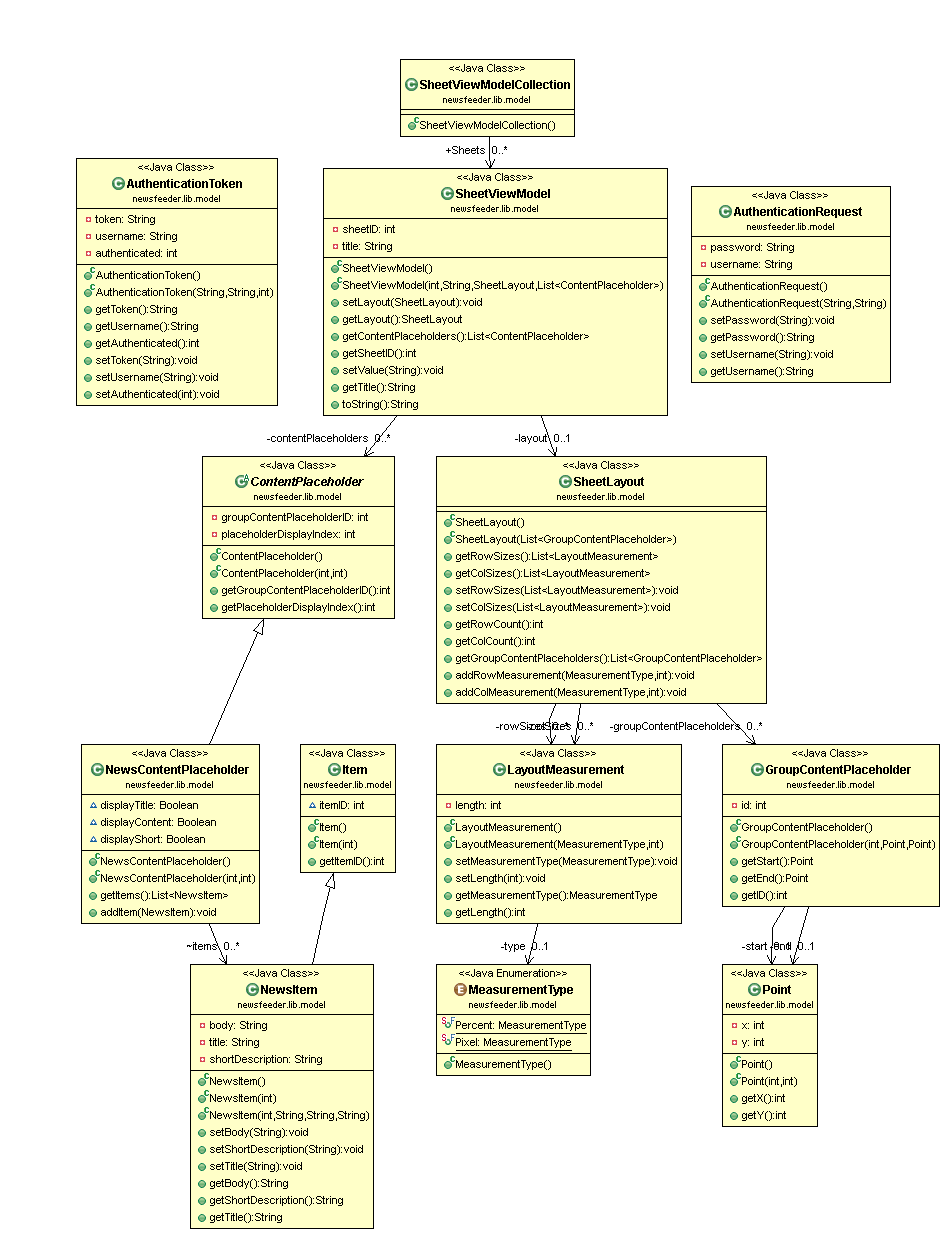
## Part 1



## Part 2



# Downloader Application Library Diagram



Web Service API Class Documentation

# Web Service API Class Diagram

*Refer to Downloader Library and Shared Database Library for shared model classes*

