News Reporter Software Architecture Documentation

June 15th 2021.

Author: Kevin Browne

Version: 1.3

Last Edit: June 15th 2021.

Table of Contents

1. Introduction

In this document we overview the software architecture for the News Reader application.

1.1. Application Overview

The News Reader console application allows a user to define and generate reports containing news story article data from news sources all over the world.

Users can define reports by selecting a source (e.g. CNN, CBC News, ABC News, etc) as well as optionally selecting search terms (both search terms limited to the title of the article, and search terms for the title and content of the article). Once a report is defined, a user can generate reports based on the report definition and write the report content to a file. Users can select from all previously defined report definitions when generating a report. Any report generated from the report definition will contain the current headlines from the given source, as well as the current search results for any search terms in the report definition.

The target audience of this application is working professionals in need of news story summaries from multiple sources.

1.2. Technologies

The following technologies are used to implement the News Reader application.

Technology	**Role**	**Sources**
Python	Implementation language for all application functionality.	https://www.python.org/
Redis	Database technology used to store application data (an in-memory, key-value database).	https://redis.io/
Redis Labs	Cloud Redis database-as-a-service provider	https://redislabs.com/
News API	Web API for accessing news article data based on sources and search terms	https://newsapi.org/
News API Python Client	Wrapper module for the News API that makes access to the API easier.	https://github.com/mattlisiv/newsapi-python

1.3. Documentation Overview

In section 2 we document the requirements for the News Reader application.

In sections 3 -7 we use the 4+1 View Model to document the News Reader application, which involves the construction of 5 views focusing on different concerns about the software architecture. We use different types of UML diagrams to document each view. You can read more about the 4+1 View Model in the original paper by Philippe Kruchten: https://people.ucalgary.ca/~far/Lectures/SENG401/PDF/4+1view-architecture.pdf.

1.4. Application Usage

The application can be run with the command **python3 main.py** from the command line. A configuration file called **config.cfg** must be present, an example of which is provided below:

```
[Database]
host=redis-13031.c92.us-east-1-3.ec2.cloud.redislabs.com
port=13031
password=5fL1AD0f5JBc58C3hNymU1NEJYRLhPpE
[NewsAPI]
apikey=570fd404cfa3402abc2f3e8fa2106d95
[Logging]
file=TRUE
database=TRUE
console=FALSE
log_filename=log.txt
```

2. Requirements

The News Reader application was built to satisfy the following requirements.

	Functional requirements
R01	Users shall be able to define a news report by providing a report name and a news source identifier, as well as optionally providing any number of 1) title-only search terms, and 2) title and article body search terms.
R02	Users shall be able to select from all report definitions to generate the report and print it to a

	file with a filename provided by the user.
R03	Report data will be retrieved from the News API to be placed into the generated report: the top headlines from the source, and the results of the relevant search terms. The report will include the following data: 1. Title and Description of any top headlines. 2. Title and Author of any title-only search term results. 3. Title and Content of any title and article body search term results.
R04	Report definitions will be stored and retrieved from a Redis Labs database.
R05	The application will log its activity to any combination of the console, file and database.
R06	A configuration file will be used to specify log, database and API settings including API keys and other credentials, a log filename, and whether to log to database, console and/or file.
	Performance requirements
R07	Reports with 3 or less title search terms and 3 or less title and article search terms should be generated and written to file in under 5 seconds (worst case performance metric). ● Measured via testing at runtime by recording and logging report generation and file write time.
	Traceability requirements
R08	At least 70% of all application methods should create a log entry at run-time. ● Measured via analysis of the implementation: (methods that include a log methods / total methods) * 100

3. Scenario View

Concerns: Understanding the central functionality of the system

Stakeholders: All stakeholders, but particularly the end user

Modelling techniques: UML Use Case Diagram

The two primary user actions are to define reports and generate reports. When defining a report, a user must provide a news source id, and then may optionally provide search terms (title only search, and searches include the article body text). See Figure 1 for the Use Case Diagram.

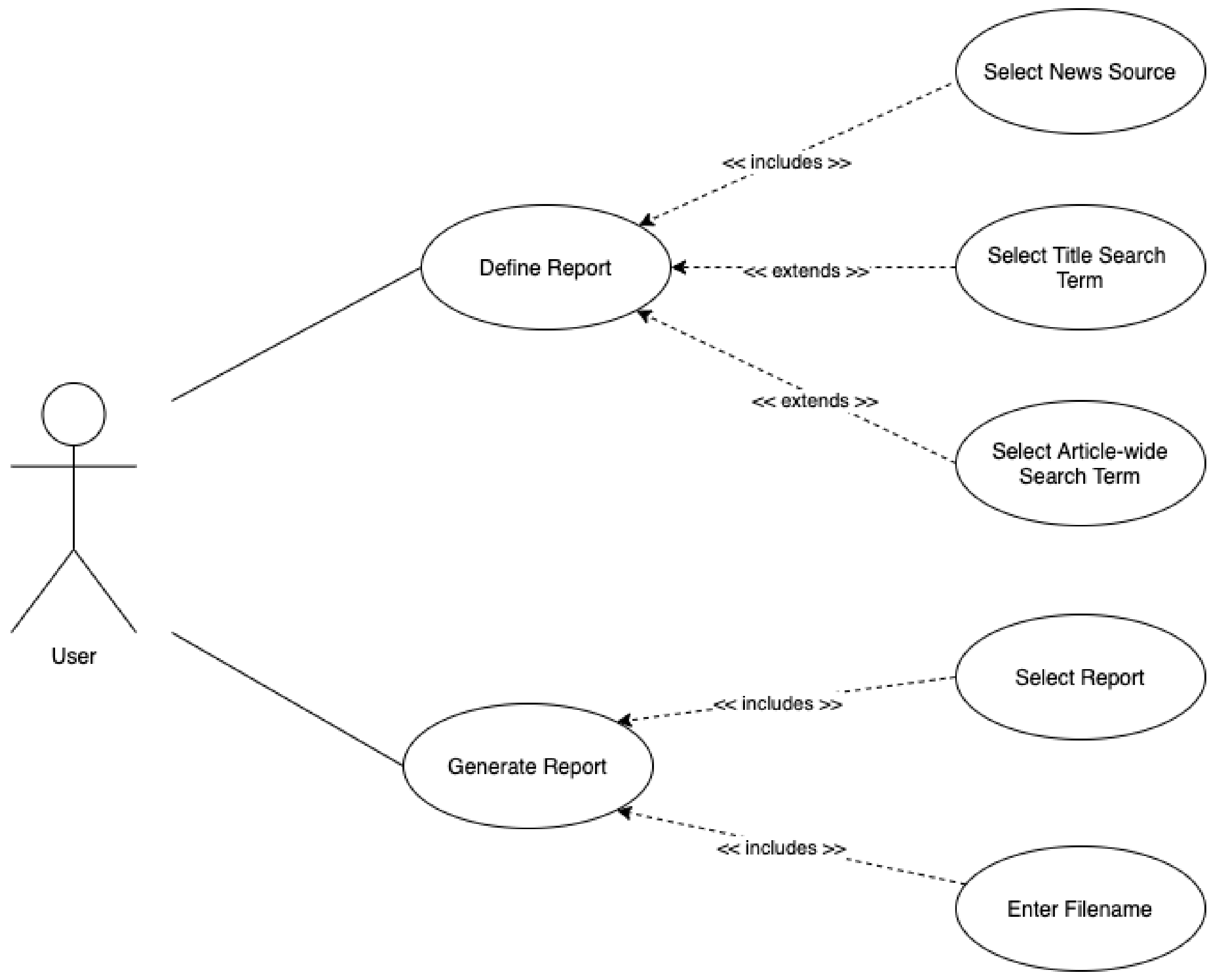

Figure 1 - Use Case Diagram (Scenario View)

4. Physical View

Concerns: Mapping of software to hardware, communication protocols and modules related to communication

Stakeholders: software architect, software developers

Modelling techniques: UML Deployment Diagram

The console application communicates with the Redis Labs database over TCP port 13031, though this is configurable via the config file if the port needs to be changed in the future. The Redis Labs database is technically hosted in Amazon's EC2 Elastic Compute Lab. The News

API is a web service made available over HTTP, and the News API Python client is used to communicate with the API. See Figure 2 for the Deployment Diagram.

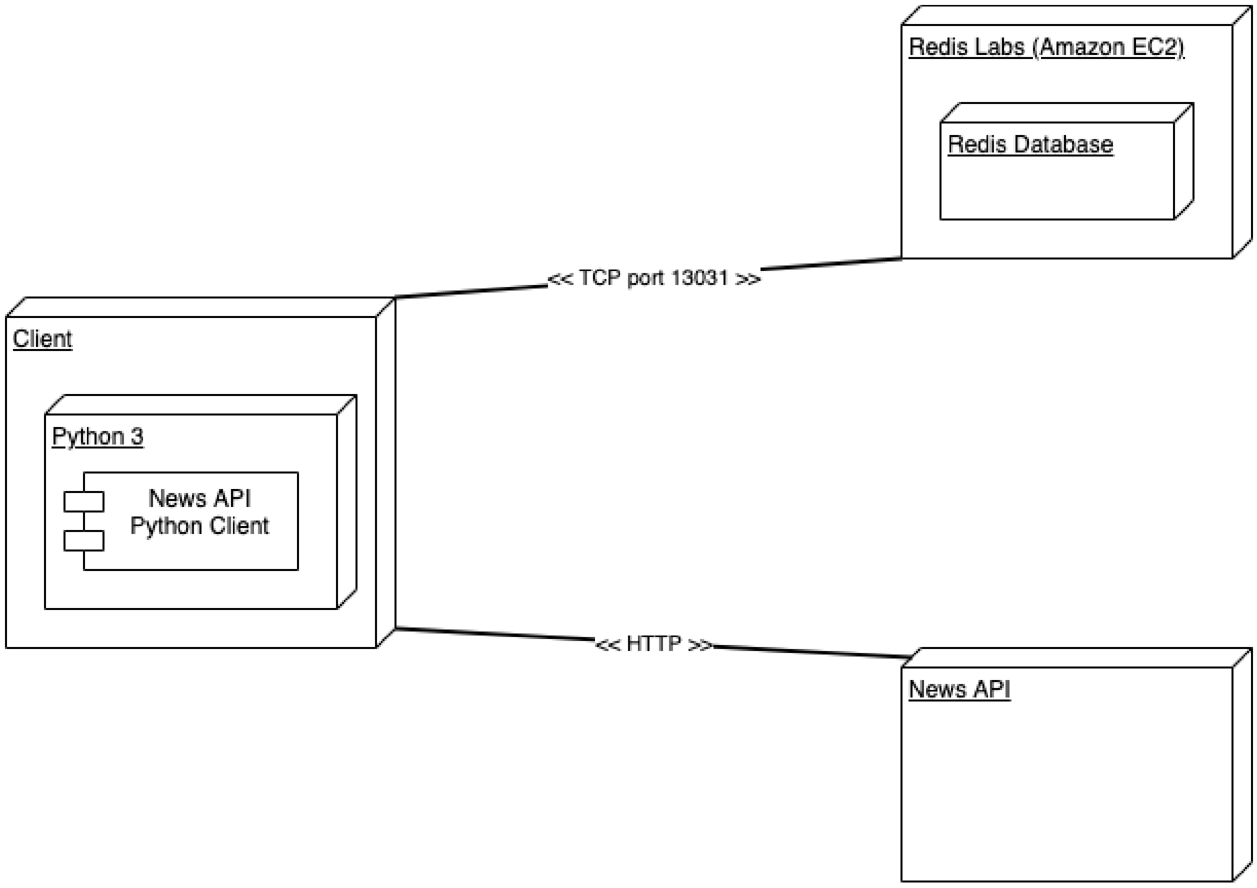

Figure 2 - Deployment Diagram (Physical View)

5. Development View

Concerns: Organization of software modules

Stakeholders: software developers, manager

Modelling techniques: UML Class Diagram

The App singleton object is used to read the configuration file as well as keep track of all read-only application state, including the database connection, News API connection, and a reference to the chain-of-loggers used by the rest of the classes to log application activity. The chain-of-loggers is implemented using a chain-of-responsibility pattern, with each logger instantiated and referencing the next logger according to whether the configuration file indicates the logger should be used. The template method pattern is used to ensure that the logic for calling the next handler can be defined once in the super class (Logger) and re-used in the subclasses without using the call super anti-pattern.

The Model-View-Controller pattern is used to handle user interaction, with the view presenting console menus and returning user input to the controller. The view has no state in terms of instance or class variables, so all of its methods are implemented as static methods to recognize and help enforce this property. The controller executes view methods to obtain user input, and calls the model functions to handle any database interactions. The controller also uses the report classes to generate reports. Report generation is implemented with the Decorator pattern, with ReportBase making up the standard minimum report, and ReportTitleSearchand ReportAllSearch handling the decoration of the report with the results of additional search terms. See Figure 3 for the UML Class Diagram.

Note that the App Singleton might be considered an instance of a "God object" anti-pattern: https://en.wikipedia.org/wiki/God_object. It is used by most objects and contains a lot of data and therefore may "do too much" and "know too much". Even by looking at the UML diagram, this is evident. We believe the use of the Singleton in this instance is justified because the data contained in the App object is **read-only data**, and the responsibility of the application is focused on application configuration. There is therefore no risk of hard to trace side-effect bugs due to global state modification: https://en.wikipedia.org/wiki/Side_effect_(computer_science).

Design Pattern Summary

Pattern	**Role**	**Classes**
Singleton	Maintains application read-only state, including connections to database, News API, and logger	App
Chain-of-responsibility	Logging to database, console and/or file	Logger, ConsoleLogger, FileLogger, DatabaseLogger
Template method	Facilitating logger subclasses without the need for call super	
Decorator	Generation of reports	Report, ReportBase, ReportExtension, ReportTitleSearch, ReportAllSearch
Model-View-Controller	User interaction	Model, View, Controller

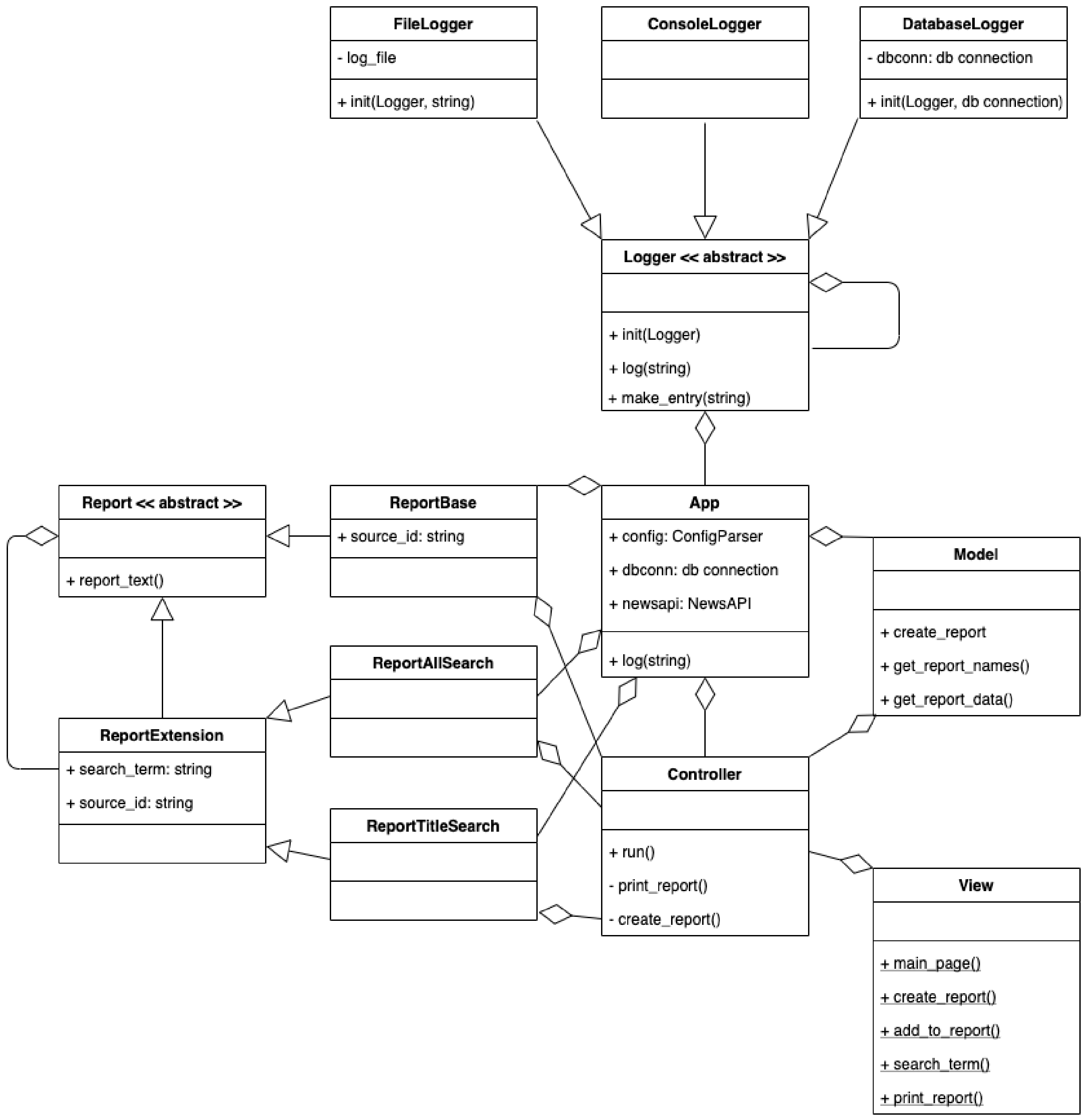

Figure 3 - Class Diagram (Development View)

6. Process View

Concerns: Runtime communication

Stakeholders: software architect

Modelling technique: UML Sequence Diagram

The most important runtime communication that occurs is between the application and the Redis Labs cloud database, and between the application and the News API. In particular, when generating a report, the report data is first requested from the Redis Labs database (the news source id, and any search terms). Then the application requests the headline data using the News API, followed by some number of requests (possibly zero) for each search term. This sequence of events is captured in the Figure 4 UML Sequence Diagram.

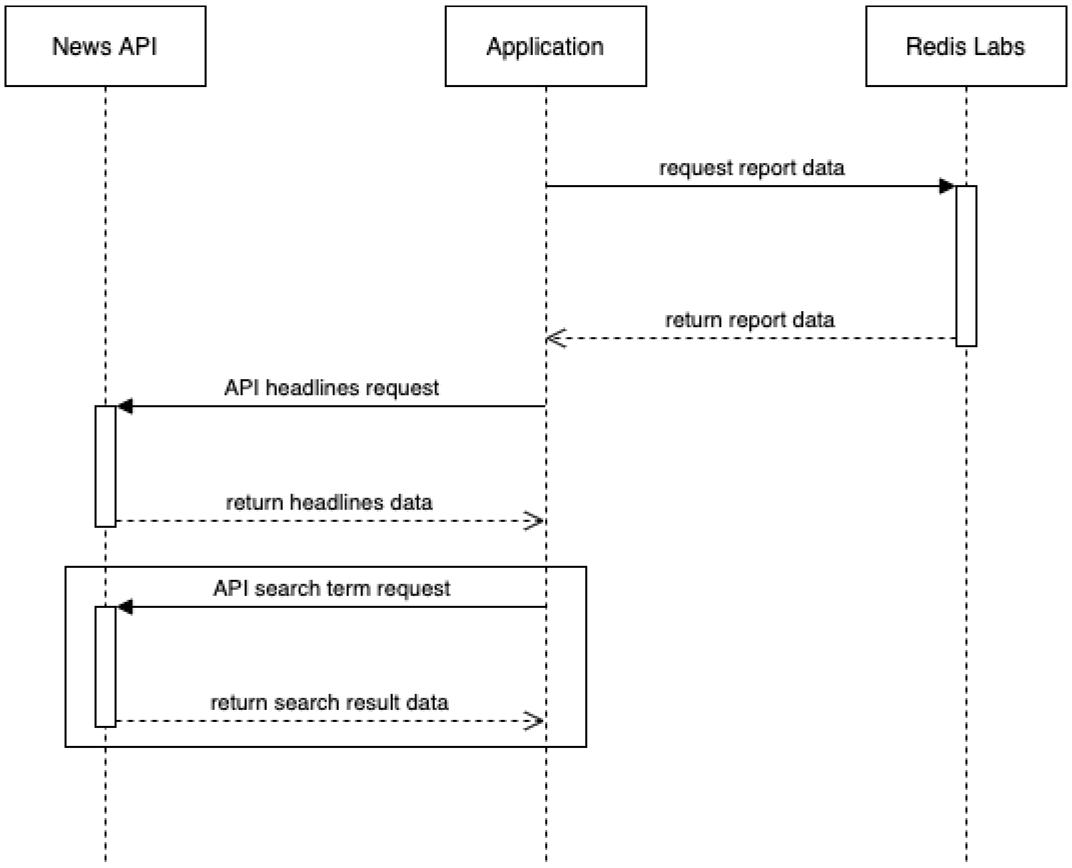

Figure 4 - Report Generation Sequence Diagram (Process View)

The MVC user interaction model involves the user being presented a view by the controller, where they can select options and/or enter data that is sent to the controller, and the controller can then carry out business logic, possibly by calling model functions. We document this sequence of communication for creating a report in Figure 5. Note that the communication expressed here has been somewhat simplified, for example the ability to enter search terms is optionally presented to the user until they decide they are done, and the model makes multiple requests to Redis Labs to store different pieces of the report data.

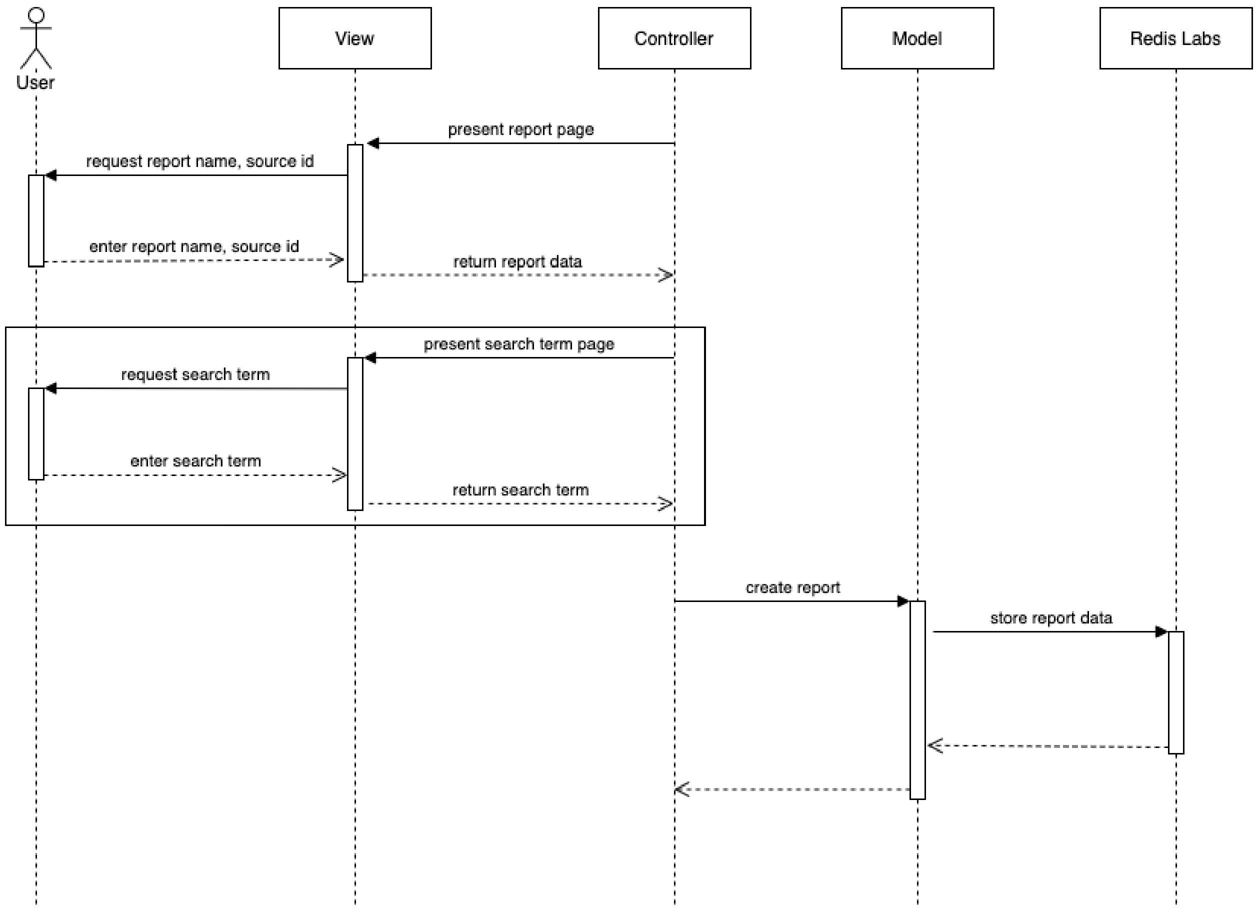

Figure 5 - MVC Report Creation Communication Sequence Diagram (Process View)

7. Logical View

Concerns: Functional requirements

Stakeholders: End user, software architect

Modelling technique: UML Activity Diagram

The use cases are realized via a series of menus and prompts that allow users to pick between printing and creating a report, selecting report options, and entering report information (the report to print, the report name, search terms, etc.). All of these menus and prompts are implemented as a console application, with text output from the application and text input from the user (with numbered options in the case of menus). The sequence of menus and options is captured in Figure 6.

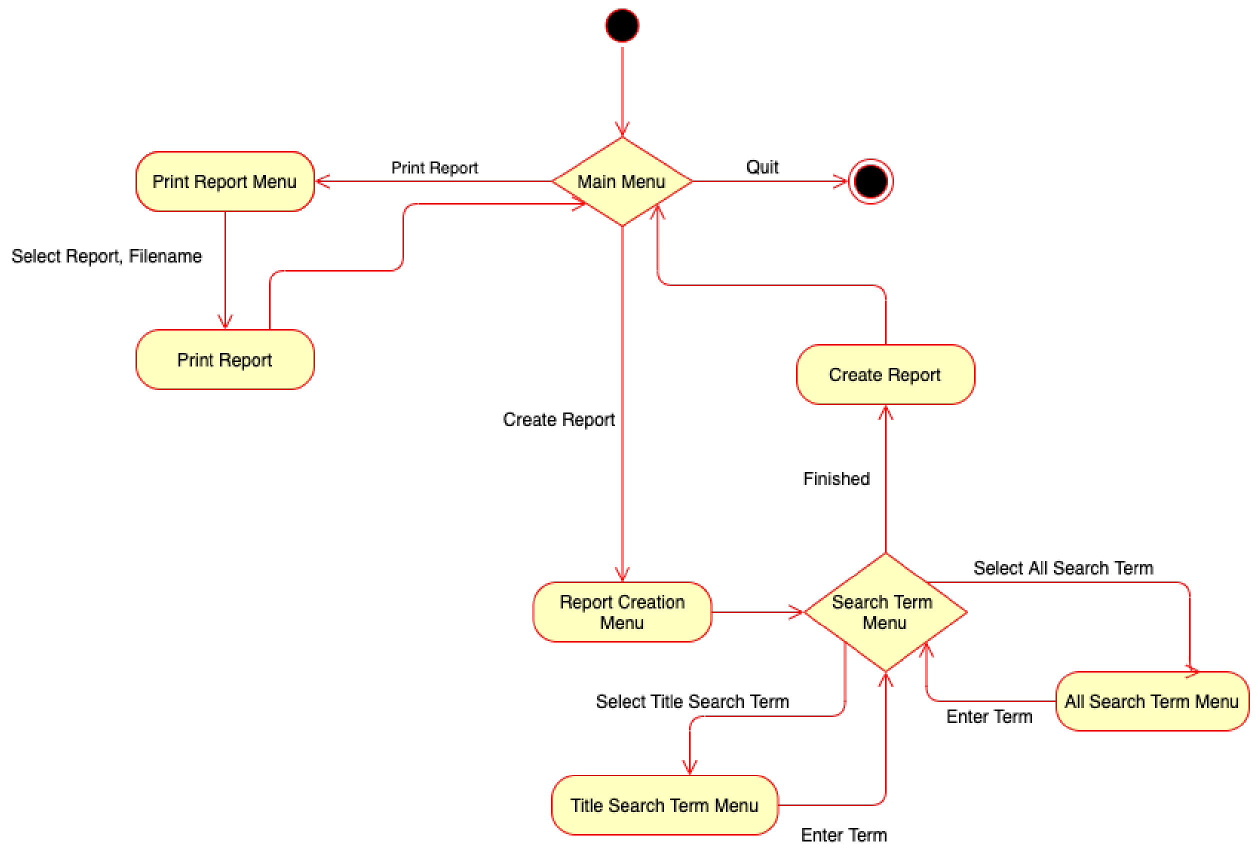

Figure 6 - Activity Diagram (Logical View)

8. Database Schema

Redis is a key-value database which stores data as keys associated with values. One possible value is a hash, which is itself a set of keys and values (however the keys are called fields in this case to distinguish them). The database schema used in News Reporter is described in the table below.

Key	Value
report:count	The total number of report definitions. Created and initialized to 1 if it does not exist and a report is created.
report:[i]	A hash containing the data for each report at report1, report2, report3, … Each hash contains: | **Field** | **Value** |

	report_name	Name of the report
	source_id	Source ID of the report.
	title_search_terms	Comma separated list of search terms for article titles.
	all_search_terms	Comma separated list of search terms for article titles and content.
log	A hash containing log entries the application has created. Each field in the hash is a timestamp for the log message, and the value is the log message content.	