4+1 View Model

Dovetail Journaling
Software Architecture Documentation

Author: Emily Ramanna

Date: August 7, 2021

Version: 1.2

Table of Contents

1. Introduction

1.1 Project Purpose

The purpose of the Dovetail Journaling software is to give people an alternative to types of social media where friend recommendations are made through topics or categories that a user must explicitly select. With Dovetail Journaling the user is meant to write brief journal entries about their thoughts and current life events, and then the software will recommend other users to get in touch with based on similarities in the topics and sentiments (e.g. positive, negative, neutral) expressed in both users' journal entries. This adds a social element to the traditionally internal and reflective activity of journaling.

From the developer's point of view, this project is an opportunity to learn about graph databases, as this fits the use case for a recommendation engine, and natural language processing.

1.2 Project Audience

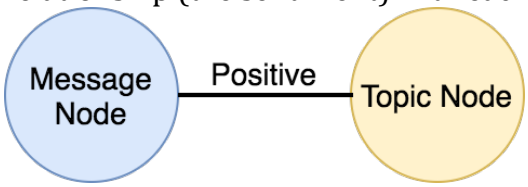
The target audience of this software will be adults (defined by the age of majority within the user's residential jurisdiction) who are already comfortable using social media platforms and with others directly contacting them This software is primarily meant for those who are casually looking to meet new people/friends.

1.3 Technologies Used in Project

Technology	Role	Source
.NET Core Framework	Development Platform (is cross-platform to aid in portability)	https://dotnet.microsoft.com/
C#	Language that will be used to implement the console application	https://docs.microsoft.com/en-us/dotnet/csharp/
Redis	Nosql database technology used for data storage	https://redis.io/
Redis Labs	Cloud database-as-a-service provider	https://redislabs.com/
RedisGraph	Graph database module that runs on a Redis database instance	https://oss.redislabs.com/redisgraph/
MeaningCloud's Sentiment Analysis API	Third party API that will perform the natural language processing analysis of the journal entries to determine	https://learn.meaningcloud.com/developer/sentiment-analysis/2.1/doc/what-is-sentiment-analysis

	what topics were written about and what the sentiment regarding those topics were	
RedisGraphDotNet Client	Used by the console application to send commands to the RedisGraph database module	https://www.nuget.org/packages/RedisGraphDotNet.Client/
RestSharp	Used by the console application to send HTTP requests to the third party API	https://restsharp.dev/
Cypher Query Language/Redis Graph Commands	The RedisGraph commands are based on the Cypher Query language, the commands are used to execute actions on a RedisGraph database	https://oss.redislabs.com/redisgraph/commands/

2. Requirements

Functional Requirements	
R01	A user shall be able to create, maintain, and login to an account that has a profile with the following pieces of information 1) unique username, 2) password, and 3) single piece of contact information.
R02	Journal Entry Management: • R02.1 - A user shall be able to create journal entries that the application will timestamp and save to a graph database. • R02.2 - A user shall be able to specify a date and view all of their own journal entries made on that date along with the identified topics and their sentiments on those topics for each journal entry. This data will be retrieved from a graph database. • R02.3 - A user shall be able to delete a journal entry by specifying a date, viewing their own journal entries made on that date, and then selecting one to delete from the graph database.
R03	The application shall leverage MeaningCloud's Sentiment Analysis API to determine the top-level topics and related sentiments contained in each journal entry. The application shall then save the journal entry's relationship (the sentiment) with each identified topic to a graph database.  *Figure 1: Showing how the sentiment (positive in this case) is the relationship between the message and a particular topic.*

R04	A user shall be able to request to see a list of other users who expressed the same sentiment about a topic during the past week. The user shall then be able to select one of the listed other users and be provided with a GUID (Globally Unique Identifier) and the selected user's piece of contact information. The intention is that they will contact the other user and include the GUID in the message.
R05	A user shall be able to view a list of all the other users that they have associated GUIDs with. This is so that they will be able to verify any external messages they receive contain a GUID that originated from this application.
R06	The application shall make use of a configuration file to read in database and API keys, credentials, and other information required for access.
R07	The application shall record log messages to a database.
Nonfunctional Requirements	
R08	Reliability Requirement: The ROCOF (Rate of Occurrence of Failure) for identifying at least one topic and sentiment per journal entry should be less than 15%. i.e. At least one topic and sentiment should be identified by the Sentiment Analysis API more than 85% of the time. Measured by testing and analyzing the running system: (number of message nodes in the database without a relationship to a topic node) / (total number of message nodes) * 100 should be less than 15%. Please note: That while this may seem to be measuring solely the reliability of the Sentiment Analysis API, the developers can customize the usage of the API (through custom dictionaries/groupings), so if there is an issue with analyzing journal entries, it is possible that it can be fixed by the internal development team.
R09	Performance Requirement: Generating a list of users that expressed the same sentiment about a topic should take 6 seconds or less 95% of the time. Measured by testing and analyzing the running system: log the amount of time it takes between the request and receiving the data each time this list is requested, then verify that (number of requests that took longer than 6 seconds) / (total number of requests) * 100 is less than or equal to 5%.

3. Scenario Viewpoint

Concern: Foster greater understanding of the system's core functionalities.

Stakeholders: This viewpoint can be useful to all stakeholders, but end-users will benefit from it especially.

Modelling Technique: UML Use Case Diagram

Primarily, a user can manage their account and their journal entries. In addition, the core functionalities involve retrieving contact information for users with similar journal entries and viewing generated GUIDs associated with the contact information being retrieved.

Aside from creating an account, all other use cases require the user to be logged in first. This is not explicitly shown in the use case diagram for readability reasons.

Please note: The use case diagram was constructed using the IBM Rational Software Architect Standard, which means that the directionality of 'include' use cases go from the base use case to the inclusion use case (IBM Rational Software Architect., n.d.). The inclusion use case can be a complete use case in and of itself. For example, in Figure 2 the 'include' relationship is shown to be directed from 'Delete Journal Entry' to 'View Journal Entry' because the 'Delete Journal Entry' use case will always include the behaviours of the 'View Journal Entry' use case.

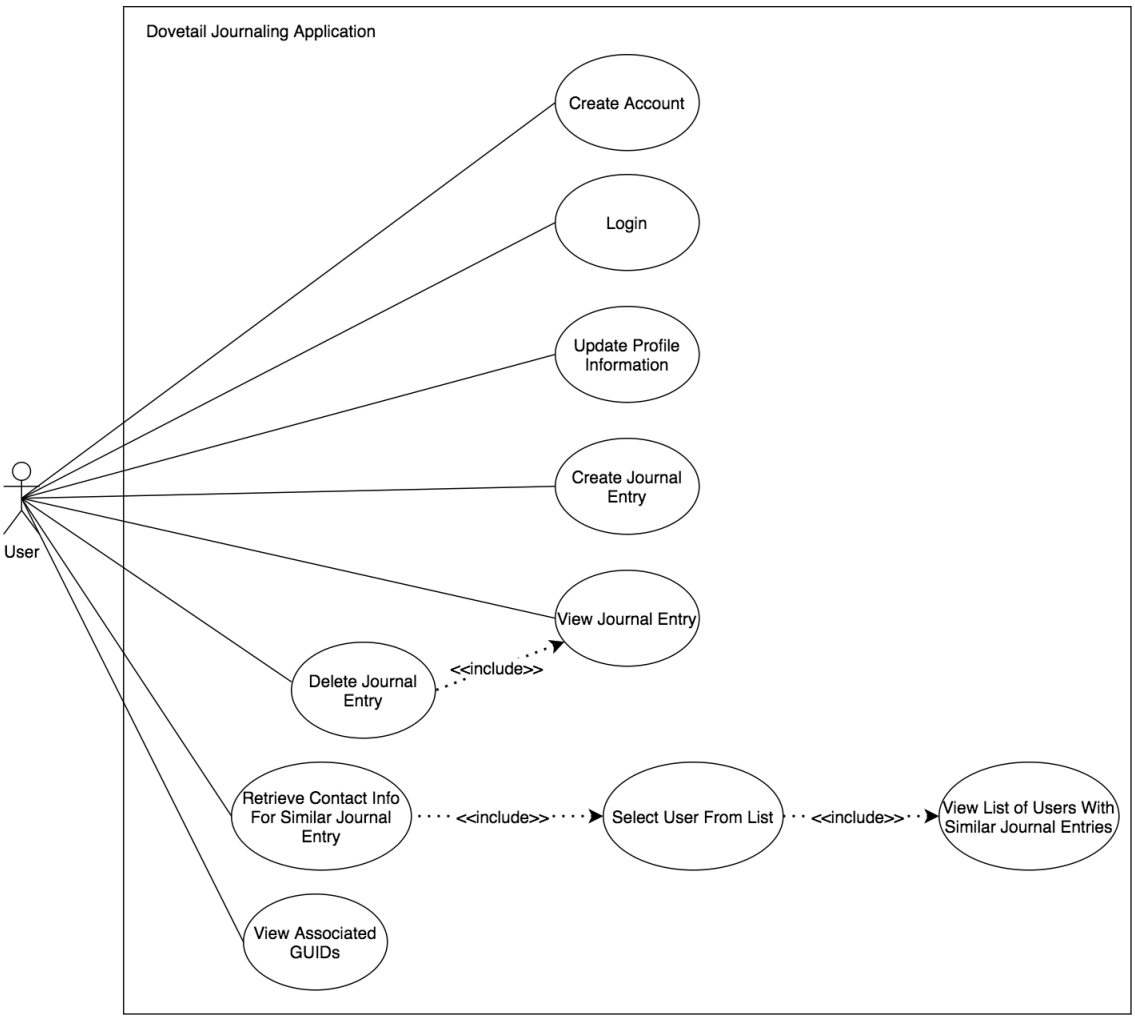


Figure 2: Use Case Diagram (Scenario Viewpoint)

4. Physical Viewpoint

Concern: Displaying which software runs on which hardware and showing which communication protocols and modules are used.

Stakeholders: Software architect/developers, deployment team/application support (they may need to know for troubleshooting errors regarding communication protocols or missing or corrupted files in the communication modules)

Modelling Technique: UML Deployment Diagram

The console application runs on the client device that has .NET Core installed. Since .NET Core is cross-platform, the operating system of the client device has not been specified. The console application makes use of the RestSharp module to make HTTP requests to the MeaningCloud Sentiment Analysis API and uses the

RedisGraphDotNet Client module to send commands to the Redis database over TCP port 16263.

The Redis database, hosted by Redis Labs (leveraging Amazon's EC2 Elastic Compute Lab) has the RedisGraph module to allow for the use of a graph nosql database.

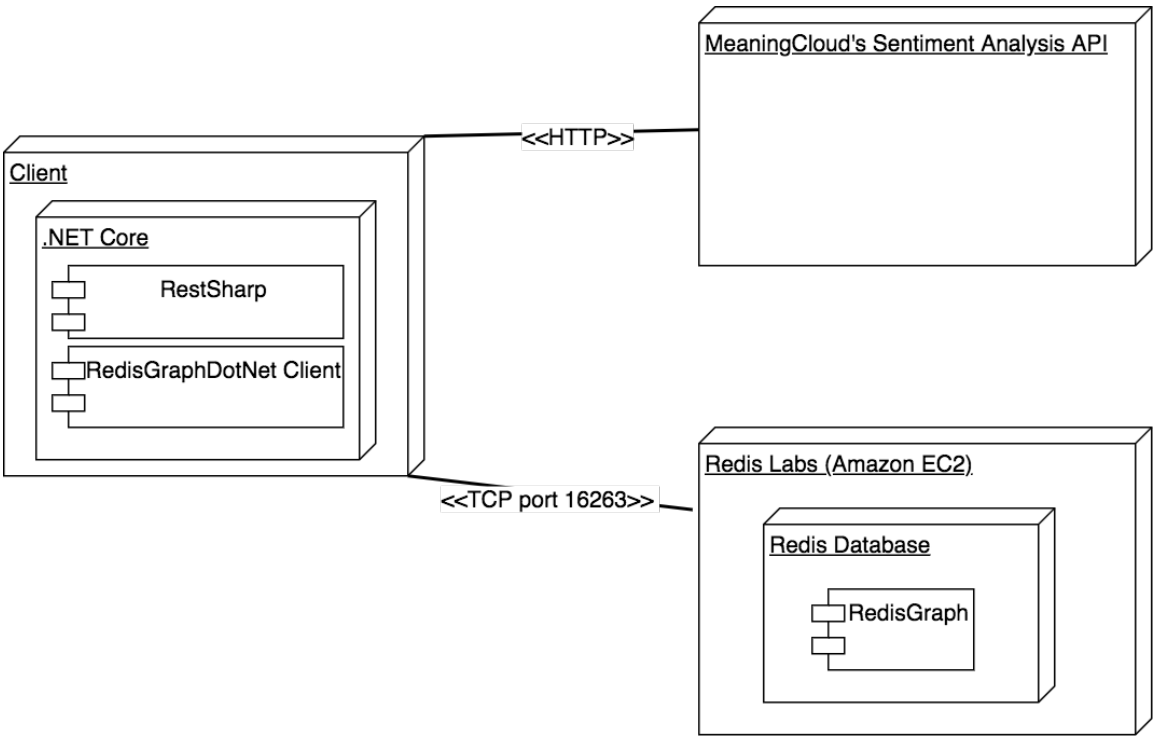


Figure 3: Deployment Diagram (Physical Viewpoint)

5. Development Viewpoint

Concern: Software module organization

Stakeholders: Software developers, project manager (for allocating development resources), testing team (for decisions regarding automated testing)

Modelling Technique: UML Class Diagram

The Model-View-Controller (MVC) pattern handles the user interaction in this application. The controller and the view depend on the models, but the models are independent of the controller and the view. The main model in this application is the UserModel, it is the model that the Controller has an instance of, and it returns the

other domain models (TopicModel, UserContactModel, JournalEntryModel, JournalEntryListModel) through its methods. This could be considered an example of the anemic domain model anti-pattern, because the majority of the other domain models simply allow for access to data. However, this was done to uphold the single responsibility principle and to mimic object-relational mapping for the database nodes/entities to make the domain models easier to grasp.

Conventionally, the model in MVC handles data persistence too. This application makes use of the repository pattern (Microsoft Docs, 2018) to separate the model from a specific database implementation (e.g. MySQL, RedisGraph, etc.). The UserModel only needs to know that it has a subclass of the IUserRepository interface that it can use to access persisted data. The repository pattern is combined with the proxy pattern (Refactoring Guru, n.d.) to take care of logging functionalities. So, UserModel will have an instance of GraphDatabaseUserRepositoryProxy, which it will use to handle data persistence.

The composite pattern is used to allow for the application to treat a single journal entry and a list of journal entries the same way. This could be a violation of YAGNI (You Aren't Gonna Need It), because it could always be a list, even if it only contained one journal entry. However, for future development there could be further business logic related to journal entries (e.g. calculating how many times a sentiment was expressed, which could be done for a single entry or for multiple). For the meantime, this pattern facilitates easier summarization and display of the data through the ToString() method. The composite model was not used for TopicModel, because it makes the most logical sense that a journal entry will always have a list of these, even if that list is empty or only has one TopicModel.

Similar to the reasoning for the use of the repository pattern, the analysis service will also be made available through an interface (IAnalysisService) in order to keep the details regarding which specific API service is used separate from the UserModel.

All classes related to the analysis service and the repository pattern will also be considered part of the Model layer because they will contain domain logic, which fits with the definition of the Model layer.

The AppConfigData singleton is used to read a configuration file and keep those configuration details in a consistent, read-only state that can be accessed by the Controller, or subclasses of the IUserRepository and IAnalysisService interfaces or other classes. Although constructors were not included in Figure 4 for the sake of readability, the Controller will have its IUserRepository and IAnalysisService subclass instances passed to it through its constructor. Inside of the Controller's constructor, it will pass both of those instances to the UserModel through setters (auto-properties in C#). In the future, which subclass instances to pass could be determined through the configuration file.

There will also be a couple of helper/utility classes that will not need to be instantiated. PasswordHelper will contain methods related to salting and hashing passwords. StringHelper will contain methods related to escaping/sanitizing input. For greater readability, these classes have been omitted from the class diagram.

Design Pattern Summary

Pattern	**Role**	**Classes**
Singleton	Contains read-only configuration data.	AppConfigData
Composite	Allows for the remainder of the application to treat one journal entry the same as multiple entries.	IJournalEntryComponent, JournalEntryModel, JournalEntryListModel
Proxy (Refactoring Guru, n.d.)	Responsible for logging related to database use.	IUserRepository, GraphDatabaseUserRepository, GraphDatabaseUserRepositoryProxy
Repository (Microsoft Docs, 2018)	Encapsulates logic for data access, separates model from specific database implementation.	IUserRepository, GraphDatabaseUserRepository, UserModel
Model-View-Controller	Handles user interaction.	Controller, View, Model Layer: • UserModel • IJournalEntryComponent • JournalEntryModel • JournalEntryListModel • TopicModel • UserContactModel • IUserRepository • GraphDatabaseUserRepository • GraphDatabaseUserRepositoryProxy • IAnalysisService • MeaningCloudAnalysisService

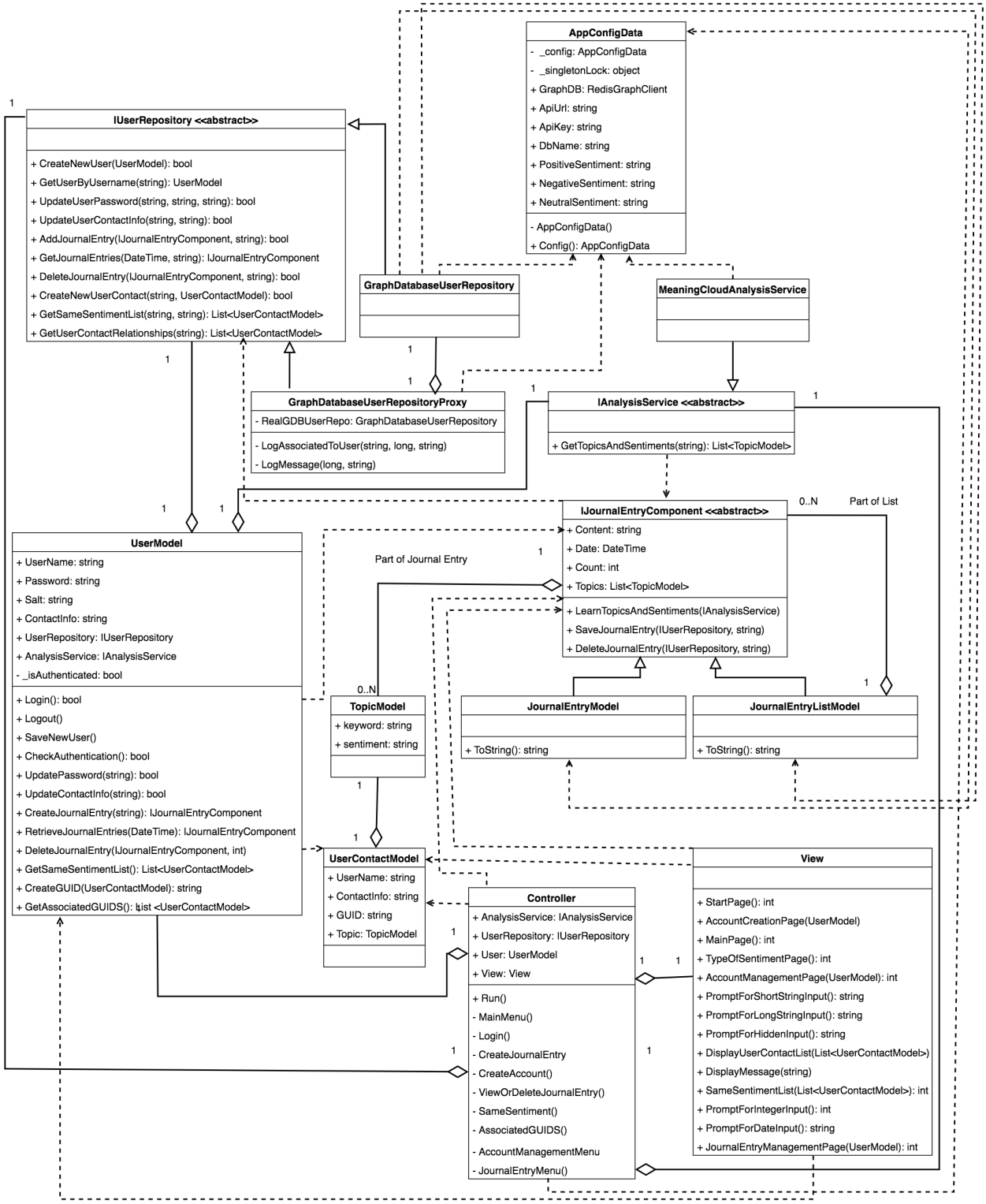


Figure 4: Class Diagram (Development Viewpoint)

6. Logical Viewpoint

Concern: Supporting the functional requirements that are available to end-users

Stakeholders: End-users, application testing team (ensuring that all end-user functionality is tested), software architect

Modelling Technique: UML Activity Diagram

The console application will present a sequence of prompts and menus that will enable end-users to perform the use cases presented in Figure 2.

There are 4 major menus:

- Start Menu – presents the options:
  - create an account,
  - login, and
  - quit the application
- Main Menu – once the user is logged in, this menu's options are:
  - logout,
  - access the account management menu, and
  - access the journal entry menu
- Account Management Menu – presents the options:
  - go back to the main menu,
  - change password,
  - change piece of contact info
- Journal Entry Menu – presents the options:
  - go back to the main menu,
  - create journal entry,
  - view journal entry,
  - delete journal entry,
  - view a list of users with similar sentiment, and
  - view previously generated/associated GUIDs

Users will select options from the menus by entering a number that is associated with the option. All input and output between the user and the application will be text entered from the keyboard and displayed on the screen. Users will return to the menu they selected the activity from once that activity is complete. The progression of menus and activities is shown in Figure 5.

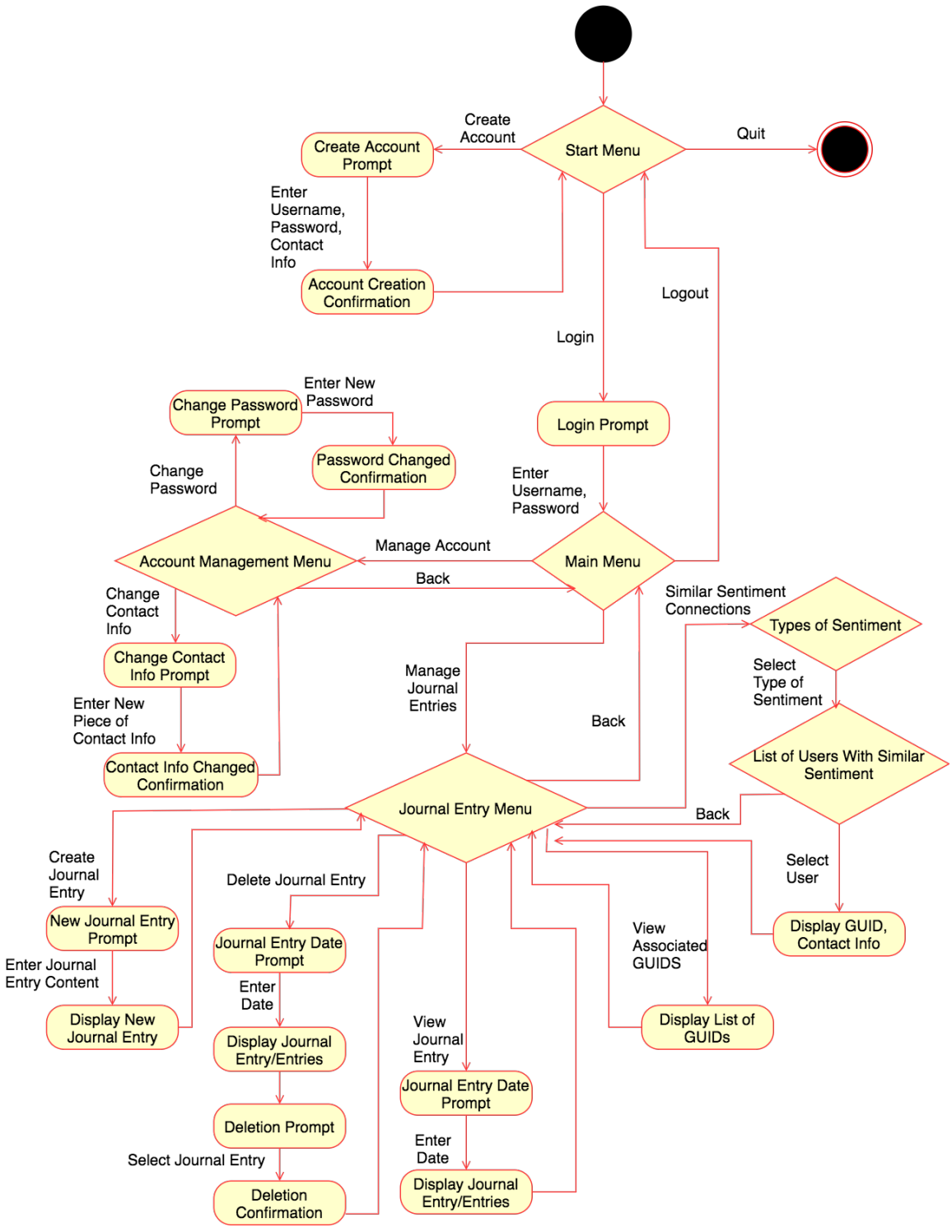


Figure 5: Activity Diagram (Logical Viewpoint)

7. Process Viewpoint

Concern: Behaviour and communication at runtime

Stakeholders: Software architect, software developers, budget personnel (see where usage of paid services occurs)

Modelling Technique: UML Sequence Diagram

Figure 6 depicts the communication that happens between the application and the Sentiment Analysis API, and between the application and the Redis Labs database, once a user has decided to create a new journal entry. The application requests the service that extracts topics and sentiments from the Sentiment Analysis API. The application then parses this data and makes a request to the Redis Labs database to store the journal entry, the extracted topics, and the sentiment as a relationship between the journal entry node and the topic node. Finally, the Redis Labs database will send back a response regarding whether the journal entry data was successfully saved.

The communication between the application and the Redis Labs database is noteworthy because it only needs to occur once per journal entry saved, even though multiple nodes need to be created. This is possible through careful use and combination of the CREATE, MERGE, and MATCH RedisGraph commands. However, if this becomes too cryptic, having multiple communications between the application and the Redis Labs database to save the journal entry and each individual topic may be preferable.

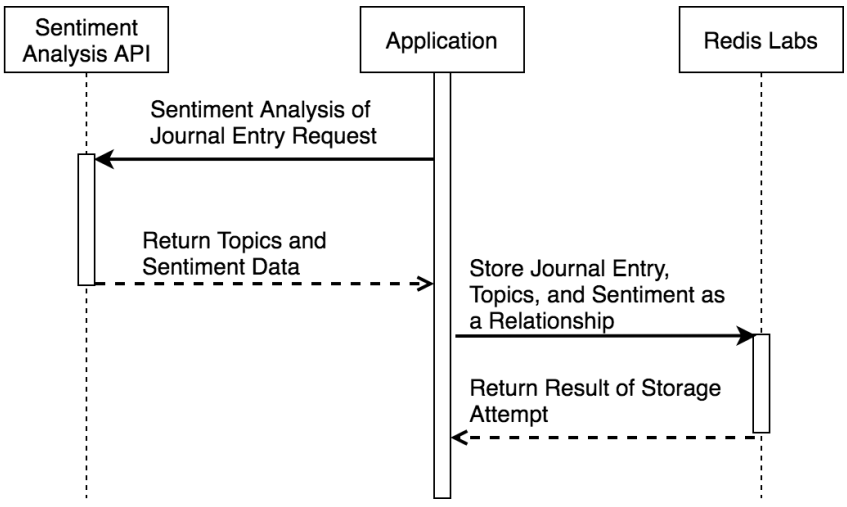


Figure 6: Journal Entry Creation Communication Sequence Diagram (Process Viewpoint)

Figure 7 depicts a generic sequence of communications that will occur when the user selects an option that requires access to the database. The View will pass along the selection to the Controller. The Controller will then call the appropriate method of the UserModel. The UserModel knows that it has a subclass of IUserRepository that it can use to access the data stored in the database, it has a GraphDatabaseUserRepositoryProxy (ProxyUserRepo) in this case. The ProxyUserRepo will perform any processing related to logging (e.g. starting a timer) prior to calling the corresponding method in the RealUserRepo. The RealUserRepo is what will communicate with the Redis Labs database, and it will get a result back based on the command that it issued. The ProxyUserRepo will log to the Redis Labs database and will receive back data regarding whether the logging was successful. Next, the ProxyUserRepo will return the data it received from the RealUserRepo to the UserModel, which will then return to the Controller.

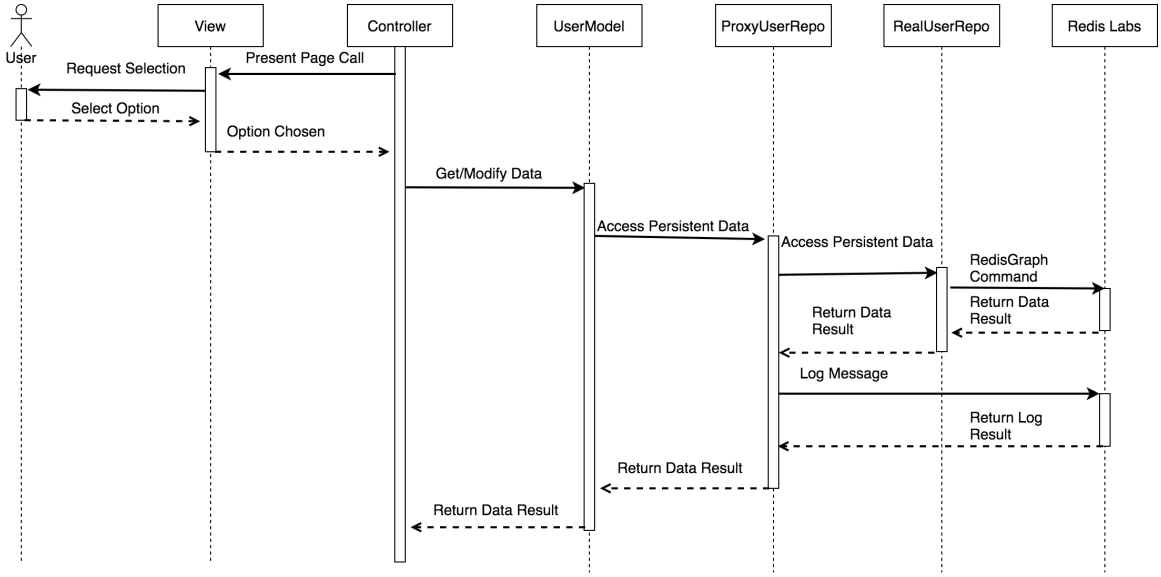


Figure 7: Generic Persistent Data Access Sequence Diagram (Process Viewpoint)

8. Graph Database Details

Types of Nodes:

- User
  - Stores information about a user
- JournalEntry
  - Stores the content and timestamp for a journal entry
- Topic
  - Stores a keyword or phrase that constitutes a topic
- UserConnection
  - Stores a GUID, topic, and sentiment that 2 users had
- LogMessage
  - Stores a string containing a log message

Types of Relationships:

- Sentiment
  - Relationship between a JournalEntry and a Topic (can be positive, negative, or neutral)
- Author
  - Relationship between a User and a JournalEntry
- Similar
  - Relationship between a User and a UserConnection
- Activity
  - Relationship between a User and a LogMessage

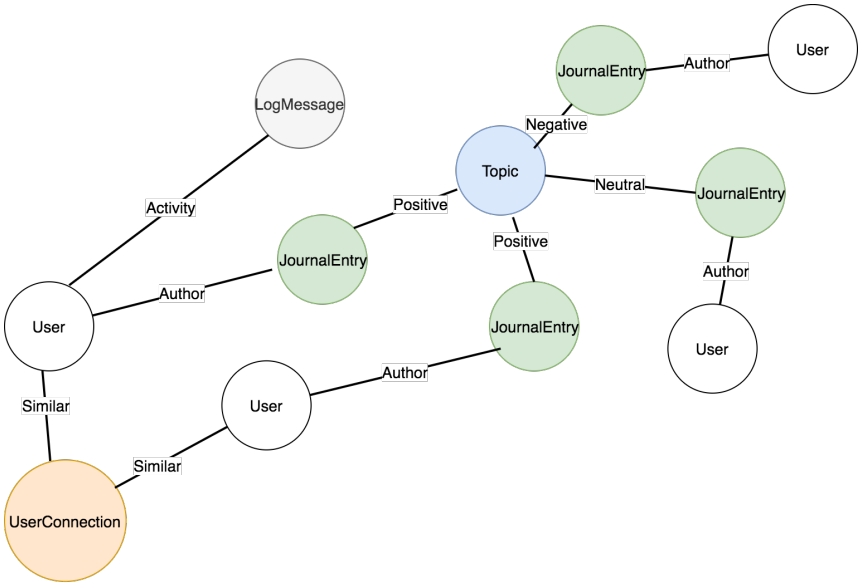


Figure 8: Sample Graph Database Structure

9. Configuration Data and Starting Point

The application will require a json file, named “config.json”, containing configuration data.

Example of required json file with key names:

```
{
"databaseHost": "redis-16263.c273.us-east-1-2.ec2.cloud.redislabs.com",
"databasePort": 16263,
"databasePassword": "kT2ygstJVpJBASntruPq18ihhyyBgTcp",
"databaseName": "projectGraph",
"positiveSentiment": "p",
"negativeSentiment": "n",
"neutralSentiment":  "none",
"APIURL": "https://api.meaningcloud.com/sentiment-2.1?",
"APIKey": "ef3686e753dfd83dbc9p4e45d8cc7cc8"
}
```

The “Main” method of the program is found in the file “Program.cs”.

10. Additional References

IBM Rational Software Architect. (n.d.). *Include relationships*. Retrieved June 26 2021, from https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=diagrams-include-relationships

Microsoft Docs. (2018). *Design the infrastructure persistence layer*. Retrieved July 11 2021, from https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design

Refactoring Guru. (n.d.). *Proxy*. Retrieved July 10, 2021, from https://refactoring.guru/design-patterns/proxy