**SE4433/CMSC5433 Software Architecture and Design**

**KWIC Software Architecture for a Web-based Search Engine**

**Assignment 2 by Jacob Theobald**

**Requirements Specification**

**Functional:** Microminer shall accept a list of keywords and return a list of URLs prefixed with their descriptions that contain all the given keywords. No noise word shall be given as a part of the list of keywords. Microminer shall use another software system, KWIC+ - a KWIC (Key Word In Context) index system, in order to efficiently store the URLs and the corresponding descriptions. KWIC+ shall accept an ordered set of lines, where each line consists of two parts: the descriptor part and the URL part:

* The URL part, whose syntax is:

URL ::= ‘http://’ {identifier ‘.’}+ [‘edu’ | ‘com’ | ‘org’ | ‘net’]

identifier ::= {letter|digit}+

letter ::= [‘a’ | ‘b’ | ‘c’ | … | ‘y’ | ‘z’ | ‘A’ | ‘B’ | … | ‘Y’ | ‘Z’]

digit ::= [‘1’ | ‘2’ | ‘3’ | … | ‘9’ | ‘0’]

* The descriptor part, whose syntax is:

{ , letter+}+***.***

The descriptor part of any line should be circularly shifted by repeatedly removing the first word and appending it at the end of the line. The KWIC+ index system shall output a list of all circular shifts of all lines in ascending alphabetical order <a<A<b<B<…<y<Y<z<Z, together with their corresponding URLs. No line in the output list shall start with any noise word such as “a”, “an”, “the”, “and”, “or”, “of”, “to”, “be”, “is”, “in”, “out”, “by”, “as”, “at”, “off”.

The system shall be accessible by web browser. A user should be able to type a list of zero or more lines into a text box, submit the list, and receive back the KWIC indexed results in another text box.

After using the KWIC functionality on a given input, the user should be able to search against the input terms, and have the matched lines returned if they exist.

**Non-functional:** The KWIC\* system shall be easily understandable, portable, scalable, and reusable with good performance. The KWIC\* system must also be user-friendly, responsive, and adaptive.

New types of modifications to the input lines should be able to be added easily and reordered if necessary. The resulting KWIC indexed lines should be returned to a user with broadband internet speed in under 2 seconds for a reasonable amount of lines (<2,000).

A user should be able to search against the stored KWIC shifted lines using case insensitive terms.

To supplement the requirements specification, a Use Case diagram and a Sequence diagram are provided below under the *Process View* heading.

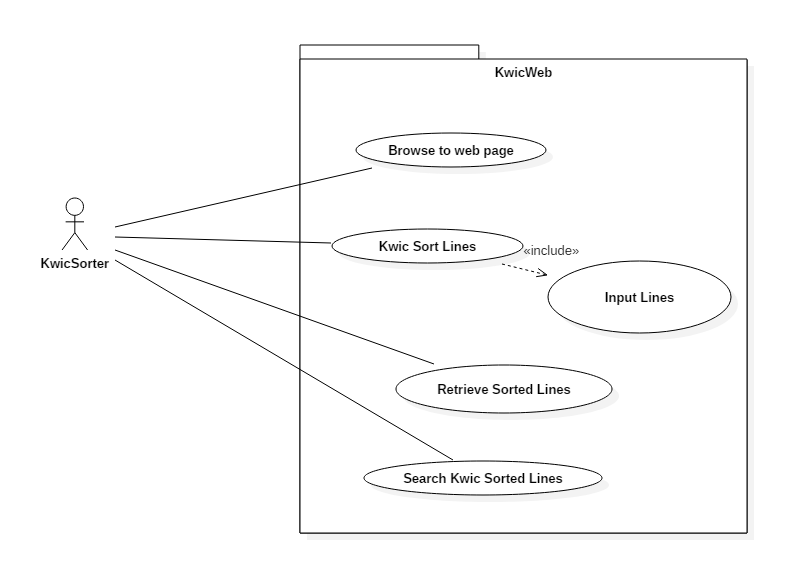
**The 4+1 View Model**

The 4+1 View Model is way of representing and describing architected software systems using four view perspectives: the logical view, the process view, the development view, and the physical view, with each playing into a final user view (the “+1” view).

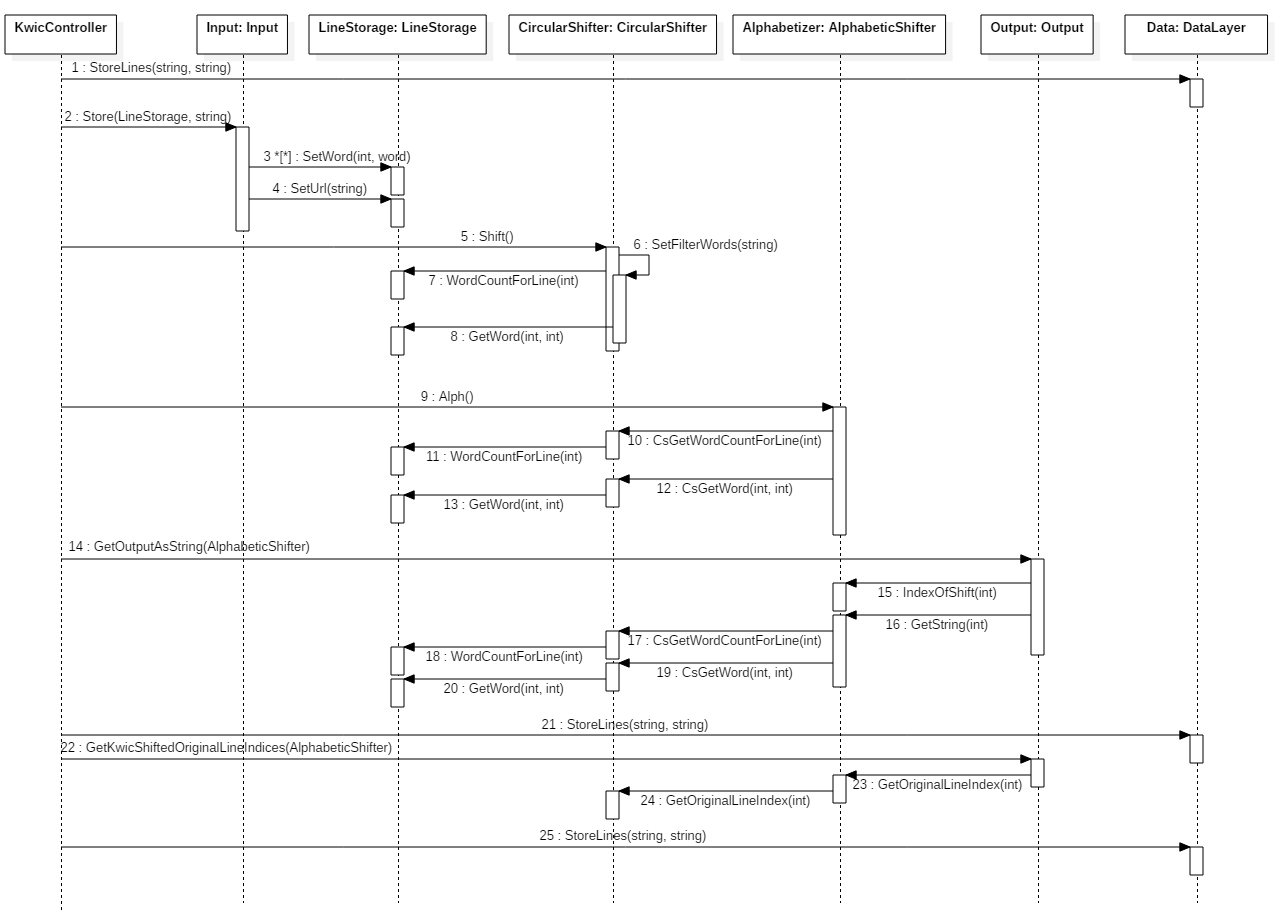
**Process View:**

• ***The process architecture*** *–* Due to scheduling conflicts and time constraints, this project was completed solo. However, the process was still completed systematically. The essential tasks were to plan out the system such that it would be performant (using shared data) but still modifiable (using an OO approach). This meant detailing how the system would component by component and deciding how those components would be self-contained. From there it was possible to begin building and testing the system using the available methods/APIs that each class offered. Adding n-tiered architecture was simple as the existing system already utilized a client server approach. The additional of a separate DataLayer .dll extended the system to having three layers.

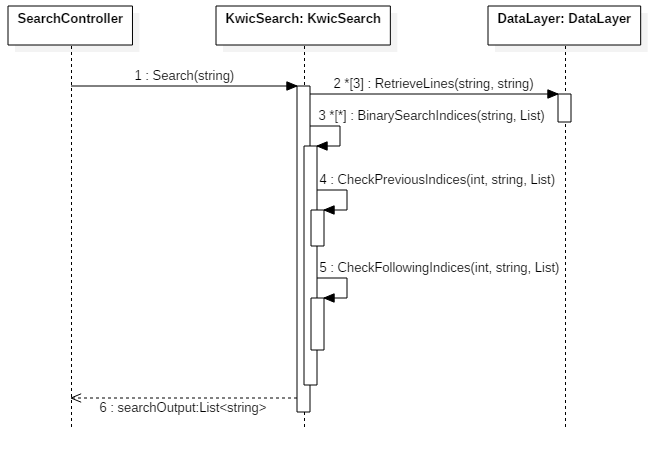
Use case for using the KwicWeb system and search functionality:



Sequence Diagram of the Kwic System sorting lines:



Sequence Diagram of the Search feature:



The first sequence diagram details how the KWIC system is handled by the controller when triggered. First, the raw input lines are saved into the DataLayer for later use by the search feature. That input is then run through each KWIC component (CircularShifter and AlphabeticShifter) with chained callbacks to make use of the shared data approach. Finally, the Output is called to get the completed output for display to the user, and to store in the DataLayer by the controller for later use by the searcher.

The search feature sequence diagram is much simpler. When a search input is given, the KwicSearch retrieves the original lines, KWIC sorted lines, and the KWIC sorted lines’ original line indices from the DataLayer. It then runs a simple binary search on each term, selects all matches before and after the binary search index, and returns the matches to the user.

**Architecture Specification and** **Implementation Specification**

This system utilizes three-tiered client-server architecture. The three tiers are 1) the client, which is made up of the website that the user uses to input their lines and search against them, 2) the webserver, which contains the controllers and functions for the Kwic Sort and search functionality, and 3) the database server, which is accessed by a DataLayer and stores various input and output files.

The KWIC system itself is essentially the same as the previous OO/Shared Data iteration, with small alterations made to store urls (or any regex match, if necessary) and the ability to get a shifted and alphabetized line’s original input line index. Each filter is a component, with the connections being the public functions available from each component class. It is constrained in that each component is somewhat coupled to the previous component in the chain, meaning that if a new filter or component is inserted, at least one other component would need to modified to account for it. The details can be seen later in the class diagram.

The KwicSearch component is a simple class that used binary searches to match the input search criteria to the stored alphabetized and shifted lines. Because it is a singular class that does not directly interact with the KWIC system in anyway, it does not have other components or constraints other than a general dependency on the DataLayer. Changes could easily be made to this class without affecting the rest of the system.

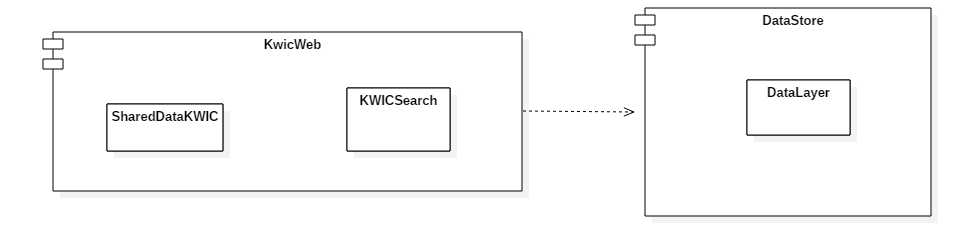
The KWICController and SearchController are responsible for handling the data storage and method calls for each subsystem and simply they use the provided public methods from each class as the connections.

The Database server/DataLayer simply presents a public api for storing and retrieving lines of files. The connections are the public methods and the constraints are that it must store items as a .txt file.

The advantages of this architectural style are numerous. Each larger system – client, webserver, and database server – are distinct from one another. Changes could be made to each on the backend that would not affect the ability of the other systems to function. For instance, I could easily replace the KWIC OO/Share Data system with Pipes and Filters, or change the data storage type from .txt files to JSON and the system would be fine. The downside is that, while not necessarily a problem for this smaller project, testing and maintainability becomes much more complex as new features are added.

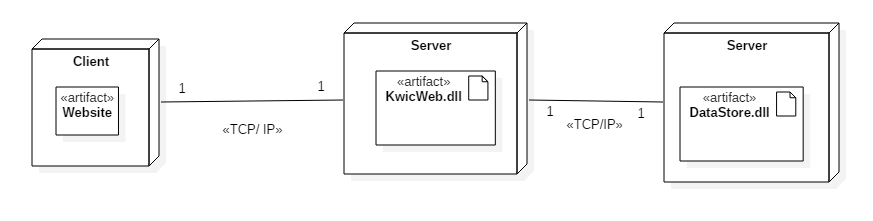
**Development View:**

Here we can see the larger components of the system and how they depend upon one another to create a system with much more advanced functionality. The client component is omitted because it is a simple HTML page, and could realistically be represented by anything capable of making web API calls.



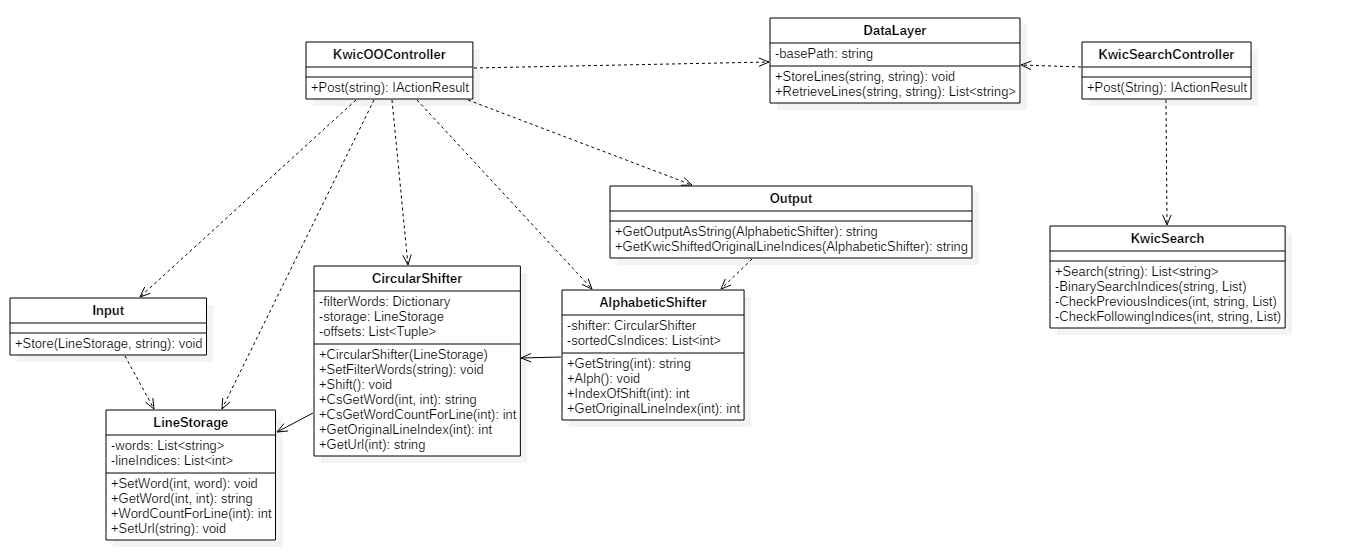
**Physical View:**

The physical view shows the three-tiered architecture. In practice, due to hardware limitations, the webserver and database server are actually running from the same location, but could easily be separated in a real-world application.



**Logical View and Design Specification:**

The system was built with a distinct OO approach in mind. Each object is self-contained and performs only the operations necessary for it. Due to the shared data nature of the system, however, each component is still reliant upon other components to perform its tasks. Thus this system has high cohesion and moderate coupling.



**User Manual**

To access the system, a use must have the appropriate version of Visual Studio and .NET frameworks installed. From there, simply open the .sln file and click run, which will open a new window in the default browser.

The user interface for the system is quite simple – an input box, output box, and search field. The user should input lines into the using the input box and save them using the “Save KWIC” button. The shifted and alphabetized results will then appear in the output box.

From there, the user can either input new lines to be shifted or search against the last input. Simply type any number of terms (case does not matter) into the search box and click “Search”. The matching results will be output to the same output box.