Cyberminer
Phase 1 Product Specification

Group: Anonymous Architects
Group Members: Trevor Celenza (txc200025), Yousef Magableh (ynm240000), Cole Oftedahl (cxo220001)
Team Leader (Phase 1): Cole Oftedahl

https://coftedahl.github.io/

Table of Contents

List of Tables

List of Figures

Revision History

Table 1: Document Revision History

Document Version Checked Out	Document Version Checked In	Date	Overview of Document Changes
0.0	1.0	09/08/2025	Created document, outlined sections. Started work on Requirement Specifications section.
1.0	1.1	09/09/2025	Added problem definition, worked on Introduction and Requirement Specifications section.
1.1	1.2	09/11/2025	Added Functional Requirements, Non-Functional Definitions, NFR SIG, and Appendix I; worked on Requirement Specifications.
1.2	1.3	09/14/2025	Added NFRs and NFRSs.
1.3	1.4	09/15/2025	Added Appendix II, revised NFRs and NFRSs, revised Section 2 Requirements Specification.
1.4	1.5	09/16/2025	Outlined Section 3 Architecture, started architecture introduction and architecture diagrams.
1.5	1.6	09/17/2025	Worked on architecture section - added system architecture, back end architecture, and KWIC Module architecture.
1.6	1.7	09/18/2025	Outline Use Case subsection in architecture section, worked on Architectural Alternatives and architecture diagrams.
1.7	1.8	09/19/2025	Updated KWIC Module Architecture diagrams, added use case diagrams.
1.8	1.9	09/20/2025	Worked on architecture traceability section.
1.9	1.10	09/21/2025	Added tabular use cases.
1.10	1.11	09/22/2025	Added to architectural alternatives, renamed system references, added a sequence diagram.
1.11	1.12	09/27/2025	Added to architectural alternatives, formatted references.
1.12	1.13	09/29/2025	Added User Manual
1.13	1.14	10/01/2025	Added Implementation section for revised architecture diagrams, updated links table.

Glossary (Definitions, Acronyms, Abbreviations)

In the glossary, terms which may appear throughout the document, but which are not commonly understood as having a particular inherent meaning, are defined for clarity.

Table 2: Glossary Definitions

Term, Acronym, or Abbreviation	**Definition**
KWIC	Key Word in Context
FR	Functional Requirement
NFR	Non-Functional Requirement
FRS	Functional Requirement Specification
NFRS	Non-Functional Requirement Specification
SDLC	Software Development Life Cycle
TTFB	Time-To-First-Byte (Used in measuring latency of webpage loading)
SIG	Softgoal Interdependency Graph
API	Application Programming Interface
GUI	Graphical User Interface

1. Introduction

This document's purpose is to convey an understanding of the Cyberminer system by describing it through the lens of each phase in the SDLC. A software product may be complex and may be worked on by many different groups over time, so a consistent understanding is essential to maintaining the system over time. Additionally, the rationale for choices made in the development of the Cyberminer system are provided here, to better understand the purpose of the system itself.

The document is organized along the traditional SDLC phases. First, the project description and preliminary overview are provided, followed by the body of the document, entailing requirements, architecture, software details, user manual, and testing plan. Appendices can be found in the document end matter.

2. Requirements Specification

1. Problem Definition

Each software system is intended to solve some problem. In order to both develop the system and to understand the purpose of the system, the problem must be defined. However, a problem cannot exist without the notion of a goal, as a problem without a goal is simply a phenomenon. The goal of this system's users is defined as finding resources and retrieving information associated with a given set of keywords. The problem is that due to the large number of resources available, finding only the resources associated with a given topic is difficult.

2. Domain Assumptions and Constraints

For a system to achieve its purpose, it must be used as intended. This includes a particular operating environment and constraints on facets of the interactions, including constraints on the users, actions, setup of the system, other systems with which the primary systems interact, and so forth. Following are listed various constraints which must be met or which are assumed to be met for the purpose of ensuring the proper operation of the Cyberminer system.

Table 3: Domain Items

Item ID	**Domain Item Description**	**Rationale for Domain Item**
D_1.	User accesses system via modern web browser with javascript enabled.	Cyberminer is a web-based system, and should only be accessed by users from the deployed front end. A modern browser ensures that all intended functionality of the system will be available for the users. Additionally, in this way the system requirements shall describe the intended webpage to be displayed on the user's end, rather than describing the HTML document structure

		which is truly to be transmitted as an outgoing message from the system, as the rendered result is more useful in understanding the purpose of the system.
D_1.1.	User has Internet access.	In addition to the constraint of using a modern web browser, the user is expected to have Internet access, otherwise requests to our system will never reach our system and thus the expected behavior will not occur. This is an issue which our system cannot address, as it is intended to be simply a deployed piece of software, and the network connectivity required for the system is expected to be met by any web user.
D_2.	User requests are in English.	Following from D_2., the user is likely able to translate the site into their native language for enhanced accessibility through an existing feature in the browser, and this streamlines the Cyberminer system to be uniform in its performance to all user groups instead of possibly providing a lower-quality result to non-English users.

3. Requirements

Requirements should detail what the system is capable of, and by which behaviors the system should be characterized. This section details the outward, visible actions of the system from the user's perspective. It is important to recognize, however, that the system alone may not satisfy all the requirements listed here. Instead, the domain constraints in tandem with the Requirement Specifications should satisfy the Requirements listed below.

1. Functional Requirements

Table 4: Functional Requirements

FR ID	**FR Description**
FR_1.	If the user navigates to the site, the search page shall be displayed.
FR_2.	If the user enters input in the search bar on the search page, the user shall see an autocomplete list populated for their search phase.
FR_3.	If the user clicks the search button on the search page and the search bar is empty, the user shall see an error message indicating to provide input in the search bar.
FR_3.1.	If the user clicks the search button on the search page and the search bar is non-empty, the user shall see a loading screen unless the user navigates to a different page.

FR_3.2.	If the user sees the loading screen, after at most 5 seconds they shall see an option to try resending the request or returning to the search page.
FR_3.3.	If the user sees results from the search, the results shall be displayed in order.
FR_4.	If the user clicks the Search tab, they shall see the search page.
FR_4.1.	If the user clicks the Search tab, the search bar will be cleared if the search tab was not already active.
FR_5.	If the user clicks the Index Page tab, they shall see the page index interface page.
FR_6.	If the user clicks the index pages button on the page index interface page and the input field is empty, the user shall see an error message indicating to provide input in the input field.
FR_6.1.	If the user clicks the index pages button on the page index interface page and the input field is non-empty, they shall see 3 windows on the page.
FR_6.1.1.	If the user sees the 3 windows on the page index interface page, the windows will update in real-time.
FRS_6.2.	If the user sees the final result set of computing the circular shifts, the store circular shifts button will be enabled.
FRS_6.2.1.	If the user clicks on the store circular shifts button on the compute shifts page and the store circular shifts button is enabled, the user shall see a response indicating whether the data was saved.

2. Non-Functional Requirements

This system has a variety of non-functional requirements which must be met in order for the system to be able to effectively provide its intended service. However, the meaning of non-functional requirements can be ambiguous or inconsistent between different people. Thus, each non-functional term shall be defined here, and later the relationships between these non-functional terms shall be explored in the NFR Softgoal Interdependency Graph.

Table 5: NFR Term Definitions

NFR Term	**Definition**
Availability	The ability of a user to access the system and utilize its features or services at any given time.
Modifiability	The ability of the system to have one component modified with minimal ripple effects in other components.

Performance	The ability of the system to respond to user engagement in real time. In other words, the responsiveness of the program is the measure used for performance.
Portability	The ability of the system to be run on a variety of different environments, providing a uniform experience to users on all environments.
Reusability	The ability of system components to be reused in other systems through the full modularization of the component, encompassing a minimization of dependencies and coupling along with a maximization of cohesion.
Understandability	The ability of an individual to understand the system through the use of documentation at all stages of the SDLC. This includes the use of requirements documentation to understand the purpose of the system, a well-described and documented architecture to understand the layout of and interactions between the system's components, and program comments and self-describing naming conventions to understand the program.
Usability	The ability of an individual to fully utilize the system's features both on their first interaction with the system and on recurring uses.

1. NFR_1. Performance [Cyberminer System]: The system should feel fast when returning and rendering results for typical inputs (≤10 lines).
2. NFR_2. Performance [Information Retrieval]: Search processing (querying/sorting) should be efficient for common inputs.
3. NFR_3. Performance [Information Storage]: Storing and retrieving computed shifts should feel instantaneous for common inputs.
4. NFR_4. Availability [Cyberminer System]: The system should be reliably accessible during demo/use.
5. NFR_5. Availability [User Pages]: User pages should degrade gracefully when the API is down (useful error + retry).
6. NFR_6. Usability [Cyberminer System]: First-time users should complete the core flow without external help.
7. NFR_7. Usability [User Page Switching]: Switching between key pages/tabs should be discoverable and straightforward.
8. NFR_8. Usability [User Home Page]: The home page should make entry points to Search and Compute Shifts obvious.
9. NFR_9. Usability [Information Retrieval]: Search/results should be straightforward with clear feedback (no silent failures).

10. NFR_10. Portability [Platform (Device + OS)]: The UI should function on common desktop device/OS combinations.
11. NFR_11. Portability [Browsers]: The UI should work on major desktop browsers without vendor-specific features; leverage Web Workers correctly.
12. NFR_12. Portability [Cyberminer System]: Standards-based Web APIs only; avoid browser-specific flags.
13. NFR_13 Reusability [Back End]: Indexing/shifting logic should be packaged for direct reuse and unit testing.
14. NFR_14. Modifiability [Back End]: Common changes (e.g., new filter/API route) should require minimal ripple effects.
15. NFR_15. Understandability [Back End]: Code should be self-descriptive with module-level docs and consistent naming.
16. NFR_16. Understandability [Cyberminer System Design]: High-level architecture should be documented and traceable to modules.
17. NFR_17. Maintainability [Cyberminer System]: Routine evolution should be straightforward (CI, tests, docs).
18. NFR_18. Responsiveness [Cyberminer System]: Provide immediate feedback during network activity; use modern server type and sane timeouts

4. Requirement Specifications

Requirement Specifications are distinct from Requirements in that they are more structured, and are written only in terms of environment and system events. Specifically, most Requirement Specifications should be in the form of “if e_v, then s_v,” where e_v is an environment event visible to the system, and s_v is a system event visible to the environment. A possible exception is, for a processing task, for the specification to be written in the form of “the system shall s_v,” which does not require detecting an external event in order to begin processing.

1. Functional Requirement Specifications

As the Requirement Specifications are more structured, as detailed above, each major component of the system shall have their own Requirement Specifications detailed here. Specifically, the Front End and Back End each will have their own Requirement Specifications, identified by the prefix “BE” for a Back End specification or “FE” for a Front End specification in the FRS_ID column.

Table 6: Functional Requirement Specifications

FRS ID	**FRS Description**

FRS_BE_1.	If the system receives a GET_CIRCULAR_SHIFTS request of the correct form, then the system shall process the request.
FRS_BE_1.1.	If the system receives a GET_CIRCULAR_SHIFTS request, the system shall process the request only if the request consists of an ordered set of lines, where each line is an ordered set of words, and each word is an ordered set of English UTF-8 encoded characters .
FRS_BE_1.2.	When processing a GET_CIRCULAR_SHIFTS request, the system shall return a list of all circular shifts of the input set in alphabetically sorted order, where a circular shift is a transformation of a line by removing the first word and appending it to the end of the line.
FRS_BE_1.2.1.	When performing circular shifts, a transformation of the line shall only be included in the return list if the transformed line does not begin with a garbage word (see Appendix 1).
FRS_BE_2.	If the system receives a STORE_REFERENCES request of the correct form, then the system shall process the request.
FRS_BE_2.1.	If the system receives a STORE_REFERENCES request, the system shall process all the lines not beginning with a preposition.
FRS_BE_2.2.	When processing a STORE_REFERENCES request, the system shall persist the request's associated data in a quickly retrievable manner.
FRS_BE_3.	If the system receives a GET_REFERENCES request, the system shall return a set of references associated with the request's data.
FRS_FE_1.	If the site is navigated to, the system shall display the search page.
FRS_FE_2.	If input is received in the search bar on the search page, the system shall generate a list of potential completed search phrases based on the input and display them.
FRS_FE_3.	If the search button on the search page is clicked, the system shall send a GET_REFERENCES request only if the search field is non-empty.
FRS_FE_3.1.	If a GET_REFERENCES request is sent, the system shall display a loading screen until a response is received or a timeout is indicated or a different page is activated.
FRS_FE_3.2.	If a GET_REFERENCES request is sent, the system shall timeout the request given there is no response within 5 seconds and the search page is still active, causing the system to display an option to try resending the request or returning to the search page.
FRS_FE_3.3.	If a response is received after sending a GET_REFERENCES request and the search page is active, the system shall display the returned set of references in order.

FRS_FE_4.	If the Search tab is clicked, the system shall display the search page.
FRS_FE_4.1.	If the Search tab is clicked, the system shall clear the search bar if the Search tab was not already active.
FRS_FE_5.	If the Compute Shifts tab is clicked, the system shall display the compute shifts page.
FRS_FE_6.	If the compute shifts button is clicked on the compute shifts page, the system shall send a GET_CIRCULAR_SHIFTS message only if the input field is non-empty.
FRS_FE_6.1.	If a GET_CIRCULAR_SHIFTS request is sent, the system shall display 3 windows on the compute shifts page, where Window 1 is titled “Line Processing,” Window 2 is titled “Circular Shifts,” and Window 3 is titled “Sorted Circular Shifts.”
FRS_FE_6.1.1.	If a GET_CIRCULAR_SHIFTS request is sent, the system shall update the 3 windows displayed in real-time according to responses received.
FRS_FE_6.2.	If a final result response is received from sending a GET_CIRCULAR_SHIFTS request, the system shall display and enable the store circular shifts button.
FRS_FE_6.2.1.	If the store circular shifts button is enabled and clicked on the compute shifts page, the system shall send a STORE_REFERENCES request, with the data corresponding to the circular shifts just processed, and display a result message indicating the success status of the operation.

2. Non-Functional Requirement Specifications

Following are NFRS’s which represent critical qualities of the system.

1. NFRS_1 (Performance [Cyberminer System]): Given a 10-line input, time-to-first-result ≤ 500 ms and full render ≤ 1500 ms on a typical dev laptop.
2. NFRS_2 (Performance [Information Retrieval]): Lexical sort + search processing completes in ≤ 800 ms for ≤ 10 lines.
3. NFRS_3 (Performance [Information Storage]): Store→fetch of computed shifts for ≤ 10 lines completes in ≤ 2.0 s end-to-end.
4. NFRS_4 (Availability [Cyberminer System]): During Phase-1 demo week, measured uptime ≥ 99.0% (best-effort log/monitor).
5. NFRS_5 (Usability [Cyberminer System]): 3/3 first-time users complete Search → Compute Shifts → Store in ≤ 2 minutes without help.
6. NFRS_6 (Usability [User Page Switching + Home]): 3/3 users find and switch to target page in ≤ 10 s from Home/Search; first-try success.

7. NFRS_7 (Usability [Information Retrieval]): Non-empty searchbar input yields autocomplete results or explicit “no matches” in ≤ 1.5 s; autocomplete median ≤ 250 ms.
8. NFRS_8 (Portability [Device+OS]): Full flow works on common smart devices and PC browsers less than 5 years old.
9. NFRS_9 (Portability [Browsers]): Full flow works on latest Chrome/Firefox/Edge; Web Workers behave consistently; no vendor flags.
10. NFRS_10 (Portability [Cyberminer System]): Only standards-based Web APIs are used (fetch/Web Workers); verified across supported browsers.
11. NFRS_11 (Reusability [Back End]): Indexer exposes a pure function `process(lines)`; unit tests achieve ≥ 90% statement coverage on Indexer.
12. NFRS_12 (Modifiability [Back End]): Adding a new filter (e.g., garbage word) or a new API route requires changes in ≤ 1 backend module and 0 frontend modules; tests remain green.
13. NFRS_13 (Understandability [Back End]): All public modules include headers (purpose, inputs/outputs, deps); linter reports 0 errors of severity “error” (warnings allowed ≤ 5).
14. NFRS_14 (Understandability [Cyberminer System Design]): Deliver 1 C&C diagram, module interface table, and naming conventions; artifacts in /docs and linked from README.
15. NFRS_15 (Maintainability [Cyberminer System]): CI pipeline (lint + unit tests) runs on every PR; branch protection enforces green checks; minor change lead time (PR open→merge) ≤ 24 hrs.
16. NFRS_16 (Responsiveness [Cyberminer System]): Show loader after 100 ms; hard timeout at 5 s with retry; server responds using a modern server stack/config with median backend processing ≤ 300 ms.
17. NFRS_17 (Responsiveness [Live Feedback / Smart Web Services]): Background operations expose progress updates (e.g., via Smart Web Services or equivalent) with status refresh ≤ 1 s when active; fallback available.

3. Operationalization of the SIG

This table maps each SIG softgoal (Figure 1, shown below) to its operationalization: where it lives in the system and how we verify it.

Table 7: NFRS-Feature Mapping

Softgoal / NFRS	Where it applies (feature from SIG)	How we verify

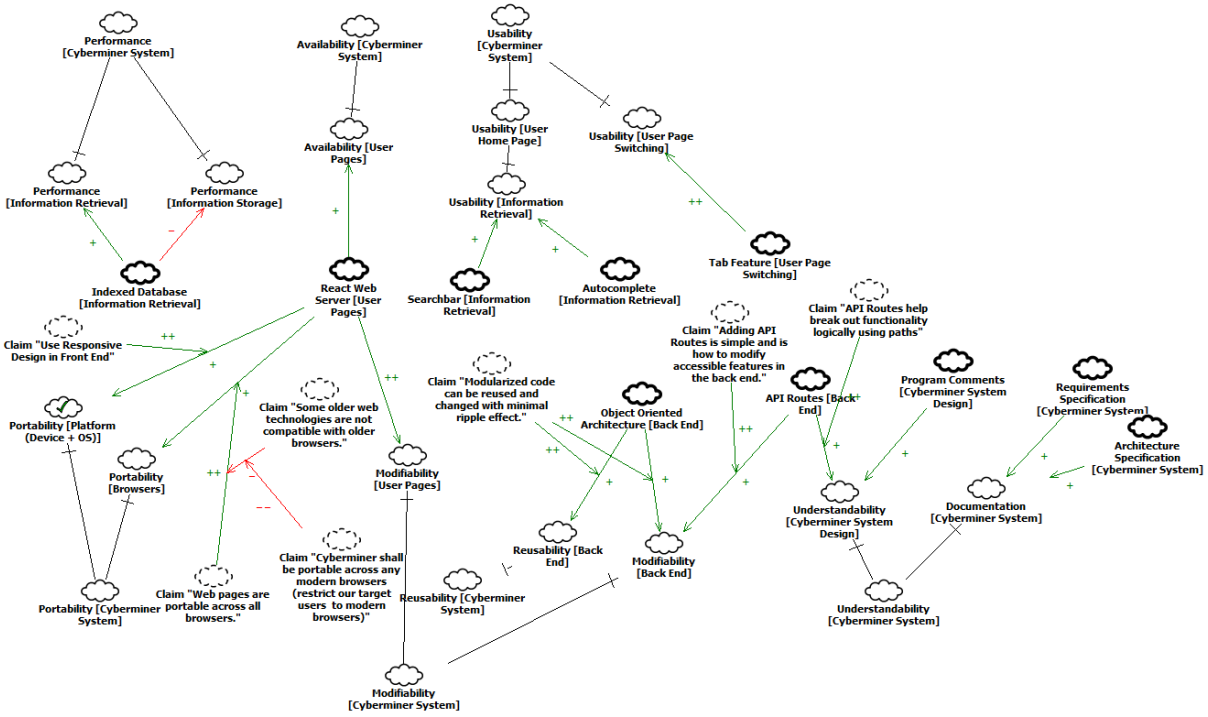
Performance (System) [NFRS_1]	Searcher (Information Retrieval), Sort (lexical)	Measure TTFB & full render on ≤10 lines; accept if ≤500/1500 ms.
Performance (IR) [NFRS_2]	Searcher (Information Retrieval)	Time sort + query; accept if ≤800 ms on ≤10 lines.
Performance (Storage) [NFRS_3]	Indexed Database; Store/Fetch	Time store→fetch; accept if ≤1.0 s end-to-end.
Availability (System) [NFRS_4]	Health check endpoint; API Routes (Back End)	Uptime monitor logs; verify /health returns 200; ≥99.0% during demo week.
Usability (System) [NFRS_5]	Whole flow	Hallway test protocol (n=3): beginner script, success ≤ 2 min, note obstacles.
Usability (Switching + Home) [NFRS_6]	Clean Page Navigation; Tab Feature (User Page Switching); User Home Page	Observe: discover target page ≤10 s; first-try success.
Usability (IR) [NFRS_7]	Autocomplete; Searcher	Autocomplete median ≤250 ms; non-empty input never yields silent failure.
Portability (Device+OS) [NFRS_8]	Use responsive design in front end	Full flow works on common smart devices and PC browsers less than 5 years old; no broken layout.
Portability (Browsers) [NFRS_9]	Web Workers across browsers; Browsers (Chrome/Firefox/Edge)	Cross-browser run; workers behave consistently; no vendor flags.
Portability (System) [NFRS_10]	API (Back End) using standards-based Web APIs	Code review: standards only; verify on 3 browsers.
Reusability (Back End) [NFRS_11]	Object Oriented Architecture (Back End); API Routes help break out functionality	Unit tests on Indexer ≥90%; package exposes pure API.

Modifiability (Back End) [NFRS_12]	Object Oriented Architecture (Back End); API Routes (Back End)	Implement new route/filter; ≤1 module changed; tests green.
Understandability (Back End) [NFRS_13]	Program Comments (Back End)	Linter run: 0 errors, ≤5 warnings; headers present.
Understandability (Design) [NFRS_14]	Documentation (Cyberminer System Design); Requirements/Architecture Specification	Artifacts exist in /docs; links present in README.
Maintainability (System) [NFRS_15]	CI/CD pipeline	PR shows green checks; measure lead time ≤24 h on minor change.
Responsiveness (System) [NFRS_16]	Timeout + Loader; Recent Web Server Type (User Pages)	Simulate slow server; loader after 100 ms; timeout 5 s; backend p50 ≤300 ms.
Responsiveness (Live Feedback) [NFRS_17]	Smart Web Services (Back End)	Task with progress emits update ≤1 s; fallback to polling if unavailable.

4. NFR Softgoal Interdependency Graph

A Softgoal Interdependency Graph, or SIG, is a notation used to model goals, operationalizations, and their relationships, along with showing rationale behind each connection. The following SIG is a diagram relating NFR softgoals and the functional operationalizations of each softgoal present in the system. Each NFR term is called an NFR softgoal because each term cannot be as concretely said to be satisfied as can a Functional Requirement term. Each NFR term is often met in a “good enough” standard, making it a softgoal rather than a hardgoal. For better viewing, a fullscreen version of the diagram is available at the following link: [L1] NFR SIG.

Figure 1: NFR SIG



3. Architecture Specification

The architecture of a system can be understood from various levels of abstraction. This section will both textually and graphically represent the system from various levels of abstraction to facilitate a complete understanding of the system, its components, and its connections.
A system's architecture consists of, at a minimum, the components and connections which comprise the system, the architectural style of the system, and the rationale behind the design of this architecture and why a particular style was used. This section will demonstrate all these items in the context of the Cyberminer system.

1. Architectural Alternatives and Rationale

The architectural alternatives section provides design options which were proposed for the high-level architecture of the system, the benefits and drawbacks associated with each option, and the decision made regarding each item.

AR_DEC_1. Logical location of the noise eliminator object within the KWIC Module.
Option 1: Place noise eliminator after circular shift object, but before the alphabetizer object.

- Benefits: Reduces computation time and space required for alphabetizer, because the number of lines to be copied and sorted is reduced, as no noisy lines will be passed.
- Drawbacks: Introduces an additional component within the KWIC Module, breaking the previously established architecture.

Option 2: Place noise eliminator after output, essentially adding the noise eliminator as a post-filter.

- Benefits: The KWIC Module architecture remains unchanged, and continues operating with no modification and thus it will not have additional errors introduced.
- Drawbacks: The KWIC Module is doing unnecessary work in alphabetizing the noisy lines, as these lines will never be used in the system.

Final Decision: Option 1 will be used - the noise eliminator will be placed as a separate object between the circular shift object and the alphabetizer object.
Rationale: The KWIC Architecture is already known, but the system is not yet implemented, so the variance in error injection from adding another object is very minimal. Additionally, by following the uniform interface established for the KWIC Module objects, the addition of the noise eliminator object can be very natural and seamless.

AR_DEC_2. Real-time Progress Monitoring Architecture

Option 1: Polling-based Progress Updates

- Benefits: Simple implementation, no complex event handling, easier debugging
- Drawbacks: Increased server load, potential delays in progress updates, inefficient resource usage

Option 2: Observer Pattern with Event-driven Updates

- Benefits: Real-time updates, efficient resource usage, loosely coupled design, supports NFRS_17 (Responsiveness)
- Drawbacks: More complex implementation, requires careful event management

Final Decision: Option 2 will be used - Observer Pattern with Event-driven Updates.
Rationale: This approach aligns with modular architecture principles and provides the real-time feedback required by FRS_FE_6.1.1 while maintaining loose coupling between the KWIC Module and external viewing components.

AR_DEC_3. Indexed Database Structure

Option 1: SQL Tables, 1 line per URL

- Benefits: Simple SQL operation to store the URLs.
- Drawbacks: More complex SQL query to retrieve all the URLs associated with a search phrase, maybe requiring a join, which would largely eliminate the performance benefits reaped from using an indexed database.

Option 2: SQL Tables, all associated URLs on 1 line

- Benefits: Simple SQL query to retrieve the URLs.
- Drawbacks: Since SQL requires a standard data format, and arrays are not supported in all SQL systems, which means storing a variable sized data item for each row is not efficient use of memory, and could cause problems if not enough

space is allocated up front for the item being stored (e.g. using a single JSON string or delimited string to store all the URLs together)

Option 3: NoSQL Data Storage

- Benefits: Dynamic data format, supports easy retrieval of large amounts of data.
- Drawbacks: Less consistent data formatting, requires specific query language for the system used.

Final Decision: Option 3 will be used - NoSQL Data Storage.

Rationale: This option will allow for straightforward storage and retrieval, given that the chosen NoSQL system is MongoDB, which the team is already familiar with using. Additionally, this type of system is optimal for the domain of this project, as the amount of data stored in the indexed database can vary largely in both overall size and size of each entry. The NoSQL approach allows easy retrieval of all URLs associated with a search string while also supporting the indexing feature to enhance search performance.

One additional architectural alternative to consider for this project was which style to use. A requirement for the project was to use the Abstract Data Type, also referred to as Object-Oriented style. Following is a table of comparisons between the Abstract Data Type style and the Shared Data style of architectures for this system.

Table 8: Architectural Styles Comparison

Quality Attributes (NFRs)	**Shared Data**	**Abstract Data**
Availability	XX	XX
Maintainability	--	++
Modifiability	--	++
Performance	+	-
Portability	XX	XX
Responsiveness	XX	XX
Reusability	--	++
Understandability	--	++
Usability	XX	XX

The table provides an overview of the architectural advantages and disadvantages, but a description of the information in it will provide a more complete understanding of the differences. First, the notation used includes + and - symbols, which represent positive and negative impacts. The use of several + or - symbols is used to indicate a greater magnitude of impact on the corresponding quality attribute. Finally, the entries with XX indicate an attribute which is not impacted by the choice of architectural style.

The qualities of availability, portability, responsiveness, and usability all correspond to qualities of the front end, and are largely determined by how the system is deployed rather than the architecture of the back end. As the focus here is on the architecture of the back end, which is what shall use the Abstract Data Type style, front end considerations should not be in this table.
The qualities of Maintainability, Modifiability, and Understandability are tightly correlated, as maintaining the system, and to an extent modifying the system, require understanding the system. The abstract data type style abstracts away complexities in favor of objects hiding data and methods performing as black boxes in the architectural design. Additionally, where the shared data style may work directly with the data, oftentimes the abstract data type style will work indirectly with the data by using method invocation to access parts of the needed data, rather than accessing all of the data at once.
The quality of Performance is better supported by the shared data type rather than the abstract data type, but with smaller magnitude than the other qualities. First, the shared data type does not need to replicate data across each object used in processing, where abstract data does copy all the data passed as method parameters, which costs more space and time spent copying data compared to shared data, where each processing element simply interacts directly with the data. Thus, shared data has the advantage in this category. However, the magnitude is small because with the speed of current processors, the impact of copying this data in the abstract data type will cause marginally worse performance compared to the shared data style, causing little impact in the overall system performance.
The quality of reusability is better supported by the abstract data type because each object can be used in other contexts with less modification than can the shared data processing elements. Shared data elements are very tightly coupled with the format of the data storage, where the abstract data type simply has an interface which must be satisfied for method calls, and will operate the same regardless of the type of external data storage.
Overall, the abstract data type style is better suited to the non-functional goals of this project than is the shared data style.

2. Architecture Description

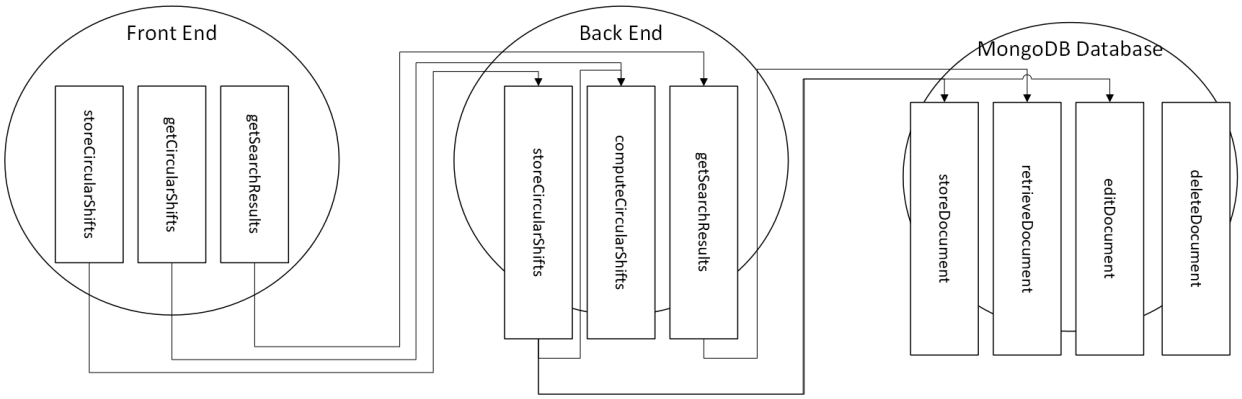
The high-level architecture of the system follows a 3-part pattern known as the MVC, or Model-View-Controller pattern. This is commonly used in web applications, as there should be a separation between the user’s interactions with the system and the data used by the system. Allowing a user direct access to the system data, or loading business logic directly onto the client’s machine without validating any information passed to the database would result in a very fragile, poor quality system. The MVC pattern allows for a robust system by ensuring the user only sees the view components, often called the front end, meant to display data and provide certain predefined interaction points with the controller component. The controller component, often called the back end, encapsulates any business logic or interaction flows of the system, which is hosted separately from the front-end so that it has a layer of protection from faults or attacks originating from the view components. Finally, the model component, often implemented as a database, is what actually stores the data, and is only ever accessed by the controller component.

The Cyberminer system follows this MVC pattern by containing a separate front and back end, as well as a database. The front end subsystem is a React server, the back end subsystem is a Node.js server, and the database is a MongoDB database.
The back end consists of modules which each encapsulate an important piece of logic which, when tied together, will provide the Cyberminer system functionality. Additionally, the back end will be designed according to an Object-Oriented, or Abstract Data Type, style. The modules found in the back end include the Database Interface Module, KWIC Module, KWIC Operation Viewer Module, Search Module, and the Save Circular Shifts Module.
The Database Interface Module provides simple methods to be called for storing in and retrieving from the database. The KWIC module provides the circular shifting functionality described in the requirements section. The KWIC Operation Viewer Module will invoke the KWIC Module and provide additional functions to allow viewing the operations of the KWIC Module in real time, allowing for the viewing progress functionality to be decoupled from the KWIC Module itself. The Search Module encapsulates the functionality to retrieve and display search results given an input search phrase. The Save Circular Shifts Module encapsulates the logic to save circular shifts passed as an argument, and handles either creating a new database entry for the circular shifts, if an entry does not yet exist, or appending the new circular shifted lines to an existing entry and saving the result in the database.

3. Architecture Diagrams

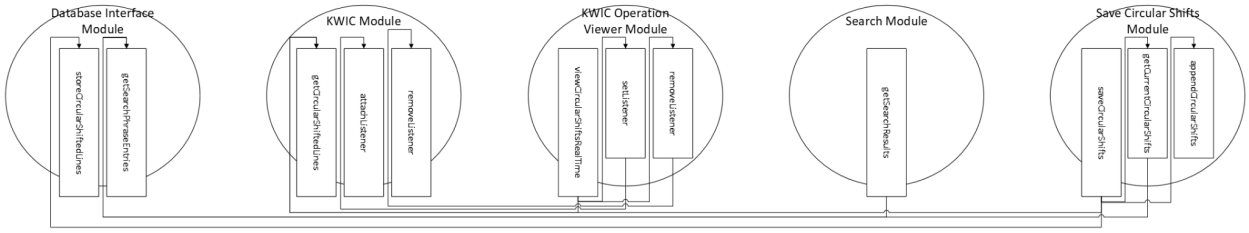
The notation used for architecture diagrams in this section is unique, but closely resembles class diagram notations. Objects shall be represented as circles, and methods within objects represented as rectangles. Method visibility is determined by whether the rectangle extrudes from the circle's perimeter or not; a method where part of the rectangle is outside the object's circular perimeter is public, and if a method's rectangle does not appear outside the object's perimeter, it is private. Method invocation is indicated by solid arrows, with the arrow pointing to the invoked method.
In both the high-level architecture of the system and the back end architecture, each major component of the system view is represented using the object notation consistent in this project, even though these components are not actually objects. This notation is used to help convey the interactions between the subsystems in a clear manner.
The high level architecture of the entire Cyberminer system is given here. The full image is available for better viewing at the following link: [L2] Cyberminer System Architecture.

Figure 2: Overall Cyberminer System Architecture



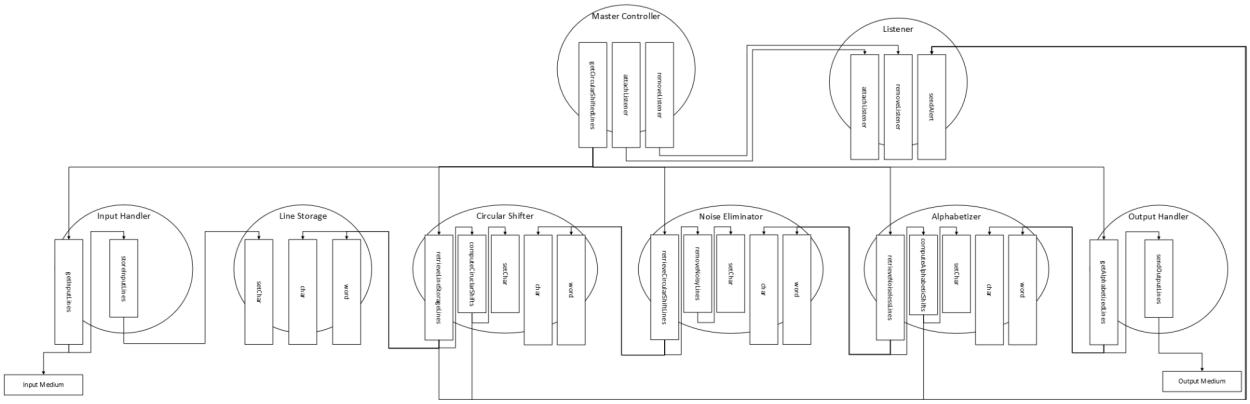
The architecture of the back end for the Cyberminer system is given here. Each method shown without an arrow pointing into the method, indicating a caller interaction, is a method which is publicly available. Such methods are shown in the back end component of the Cyberminer System Architecture diagram with caller interactions from the front end. The full image is available for better viewing at the following link: [L3] Back End Architecture.

Figure 3: Back End Architecture



Next, the high-level architecture of the circular-shift subsystem in the back end, known as the KWIC Module, is given. The three publicly available methods offered by the KWIC Module are shown in the Master Controller object, and correspond to the methods shown for the KWIC Module in the above Back End Architecture diagram. The full image is available for better viewing at the following link: [L4] KWIC Module Architecture.

Figure 4: KWIC Module Architecture



Following are several class diagrams that align with the previous diagrams, but further detail object structure in terms of methods, method parameters and parameter types, and method return types.
The overall system architecture corresponds to the Cyberminer System Class Diagram, with each main component abstracted as an object. Figure 5 full image is available for better viewing at the following link: [L5] Cyberminer System Class Diagram.

Figure 5: Cyberminer System Class Diagram

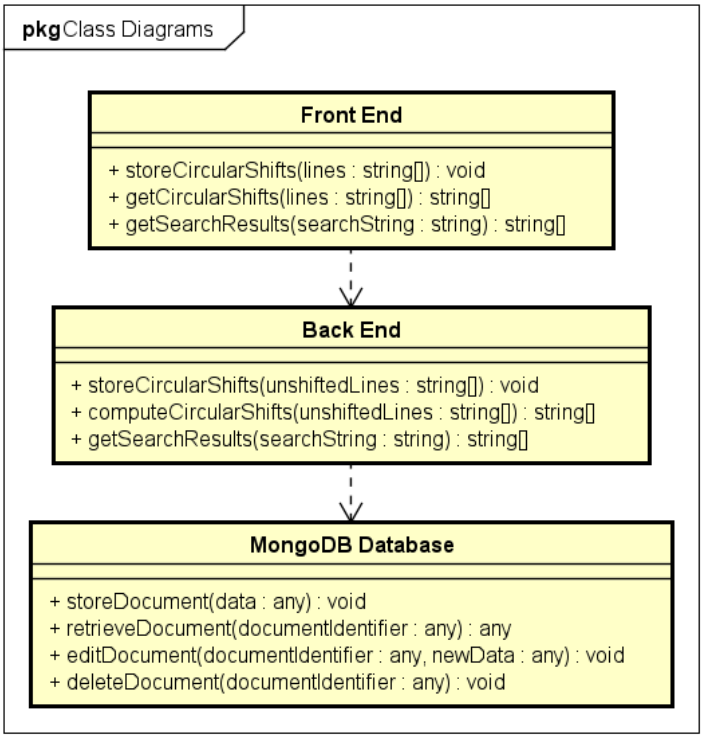


Figure 5 demonstrates the high-level MVC architecture with clear separation of concerns. The FrontEnd class handles user interactions through methods for storing shifts, retrieving lines, and managing search results. The BackEnd serves as the controller, coordinating between frontend

requests and database operations through methods like storeCircularShifts and computeCircularShifts. The MongoDB Database provides persistent storage with CRUD operations for document management. The sequential arrows show the proper MVC data flow where the frontend communicates only with the backend, which then manages all database interactions.
Figure 6 full image is available for better viewing at the following link: [L6] Back End Class Diagram.

Figure 6: Back End Class Diagram

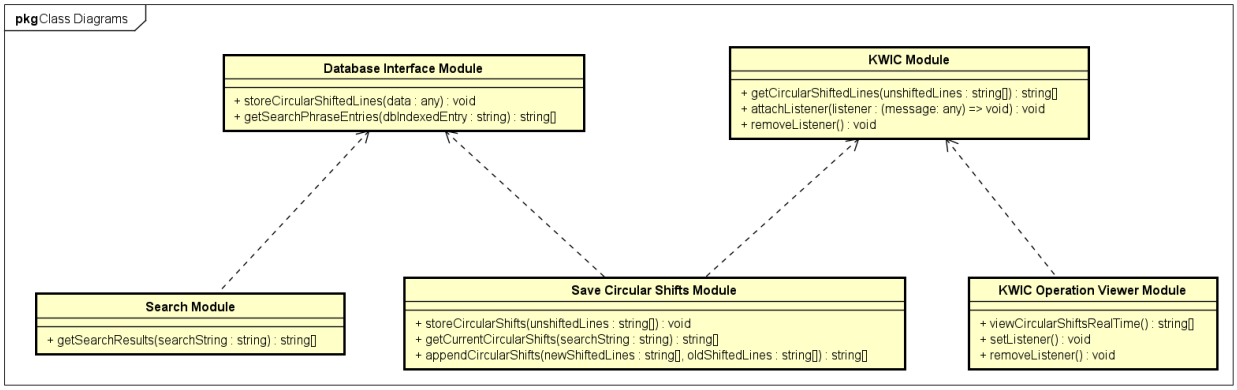


Figure 6 details the Backend's modular Object-Oriented architecture. The Database Interface Module provides clean data access methods, while the KWIC Module handles core circular shifting operations with methods like storeCircularShiftsLines and attachListener for real-time monitoring. The Search Module manages query processing independently, and the KWIC Operation Viewer Module enables progress tracking through observer pattern implementation. The Save Circular Shifts Module coordinates data persistence operations. Dotted lines indicate loose coupling between modules, supporting NFRS_12 (Modifiability) and NFRS_13 (Reusability) requirements.
Figure 7 full image is available for better viewing at the following link: [L7] KWIC Module Class Diagram.

Figure 7: KWIC Module Class Diagram

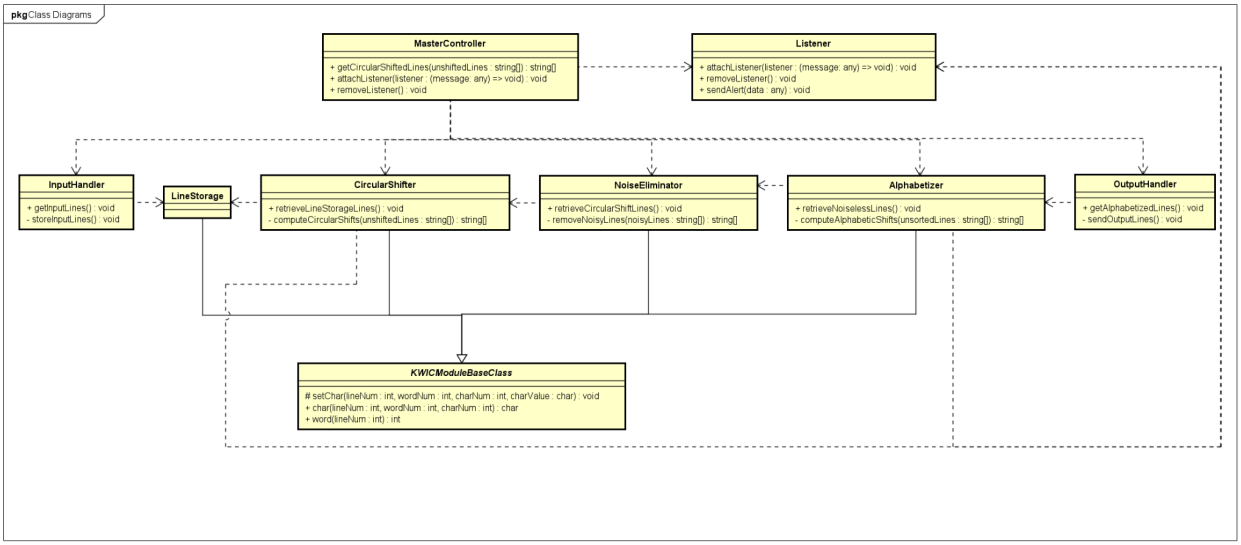


Figure 7 illustrates the internal KWIC Module processing pipeline with clear Object-Oriented design. The MasterController orchestrates operations while the Listener enables real-time progress monitoring for the observer pattern implementation. The processing flow moves sequentially: InputHandler validates input, LineStorage manages data temporarily, CircularShifter generates shifts, NoiseEliminator filters garbage words, Alphabetizer sorts results, and OutputHandler manages final output. The KWIC Module Interface at the bottom provides a clean API boundary, ensuring the module can be reused independently while maintaining loose coupling with external components.

4. Use Cases

1. Tabular Use Cases

There are three main use cases of the system based on the Requirements section. This includes the user searching the system, viewing circular shifts, and storing circular shifts.

Table 9: Tabular Use Case 1

Use Case ID	UC_1
Use Case Name	Search
Description	A user searches for resources corresponding to a particular input search phrase.
Actor	User
Precondition	User is on search screen.
Trigger	User interacts with searchbar.

Main Flow	User starts typing phrase into the searchbar input. Autocomplete suggestions appear as they type. User finishes entering search phrase and clicks the search button. After a brief loading time (during which a loading screen should appear, if it lasts long enough) the resulting page links are displayed on the screen.
Alt Flow 1	The user's search phrase returns no results - the returned screen should show a message indicating to change search phrase.
Alt Flow 2	The search request fails for some reason - a screen showing that an issue was encountered should display along with a button to return to the search screen. Upon pressing the button, the user returns to the search screen and is able to try again if desired.
Postcondition	The user is returned to the search screen and the searchbar input is cleared.
Side Effects	None

Table 10: Tabular Use Case 2

Use Case ID	UC_2
Use Case Name	View Circular Shifts
Description	The user views the circular shift processing occurring in real-time for a set of input lines they provided.
Actor	User
Precondition	User is on search screen.
Trigger	User navigates to Compute Shifts tab.
Main Flow	The user enters input lines in the input field. Once the input field is not empty, it becomes clickable. The user finishes entering the input lines and clicks the Compute Shifts button. The three windows (line currently being processed, circular shifts computed, and alphabetically sorted circular shifts) update in real-time showing the computations occurring. When all lines have been processed, the windows still show the final state of the processing, and shall remain that way until the user navigates to a different tab or clicks Compute Shifts again.
Postcondition	The user is on the Compute Shifts tab and can view the result of computing the shifts.
Side Effects	None

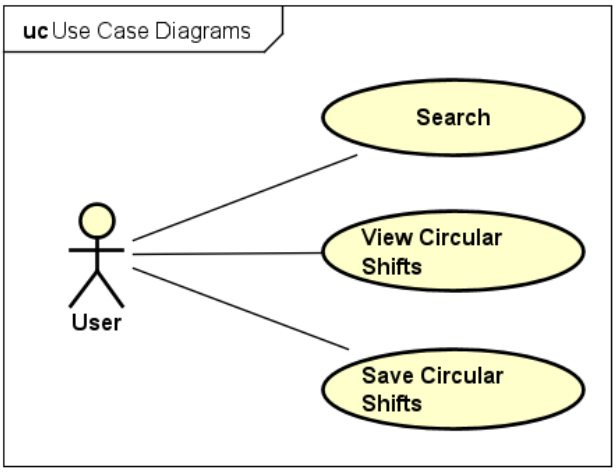
Table 11: Tabular Use Case 3

Use Case ID	UC_3
Use Case Name	Save Circular Shifts
Description	The user requests the system to compute the circular shifts for a set of input lines they provide, and then requests to store the results.
Actor	User
Precondition	User is on search screen.
Trigger	User navigates to Compute Shifts tab.
Main Flow	The user enters input lines in the input field. Once the input field is not empty, it becomes clickable. The user finishes entering the input lines and clicks the Compute Shifts button. The three windows (line currently being processed, circular shifts computed, and alphabetically sorted circular shifts) update in real-time showing the computations occurring. When all lines have been processed, the windows still show the final state of the processing, and shall remain that way until the user navigates to a different tab or clicks Compute Shifts again. After processing finishes, the Store Circular Shifts button becomes clickable. The user presses this button, and the button becomes disabled, displaying a message indicating the process is loading. Once the server finishes storing the results, the button remains unclickable but now indicates that saving was successful.
Alt Flow 1	The save functionality fails for some reason, so the button will show an option to retry saving.
Postcondition	The user is on the Compute Shifts tab with the button now showing that saving was successful. The newly computed circular shifts are saved to the database.
Side Effects	None

2. Use Case and Sequence Diagrams

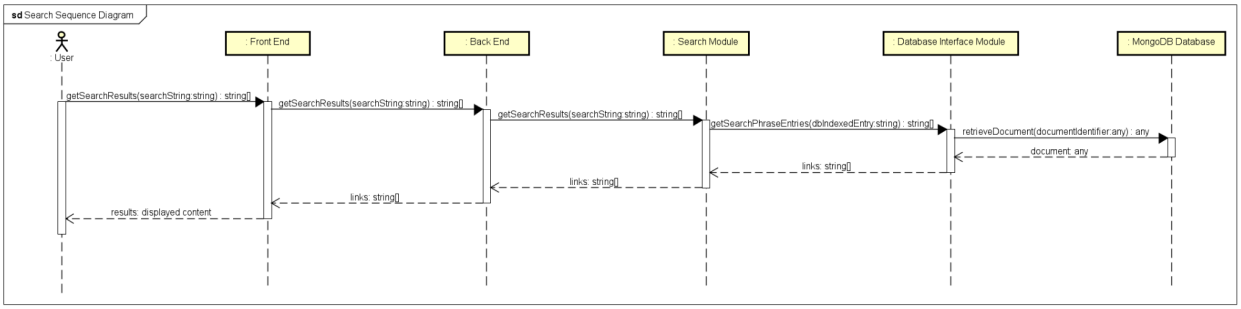
The main use cases of the system with the actor category are provided in the use case diagram. The full image is available for better viewing at the following link: [L8] Use Case Diagram.

Figure 8: Use Case Diagram



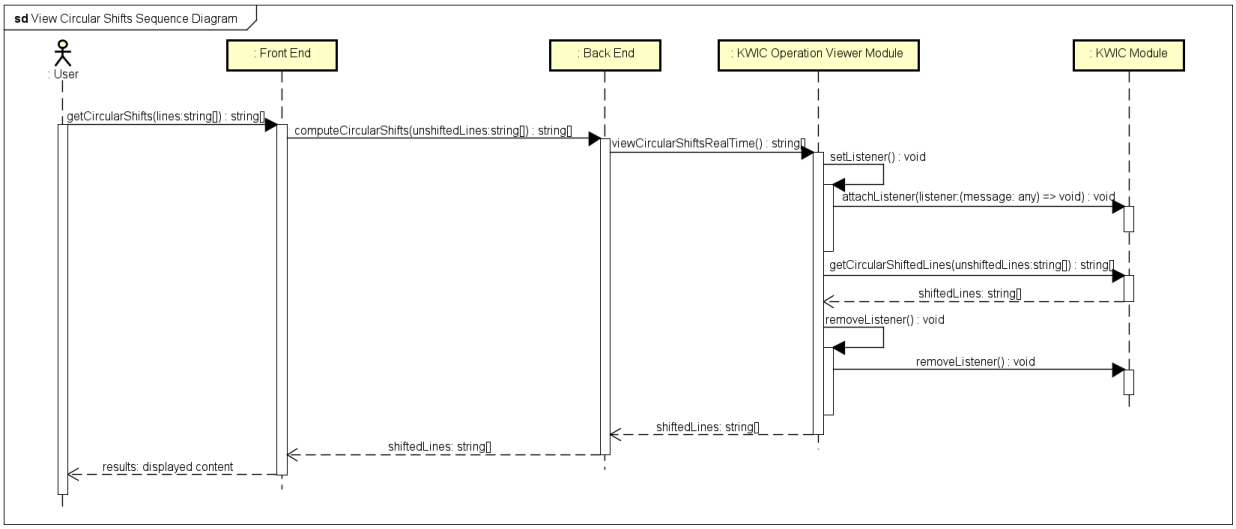
Each use case is associated with a sequence diagram to describe the interactions between system components that achieves the use case.
The search use case occurs when a user attempts to use the browser to search for a phrase, demonstrated in the following sequence diagram. The full image is available for better viewing at the following link: [L9] Search Sequence Diagram.

Figure 9: Search Sequence Diagram



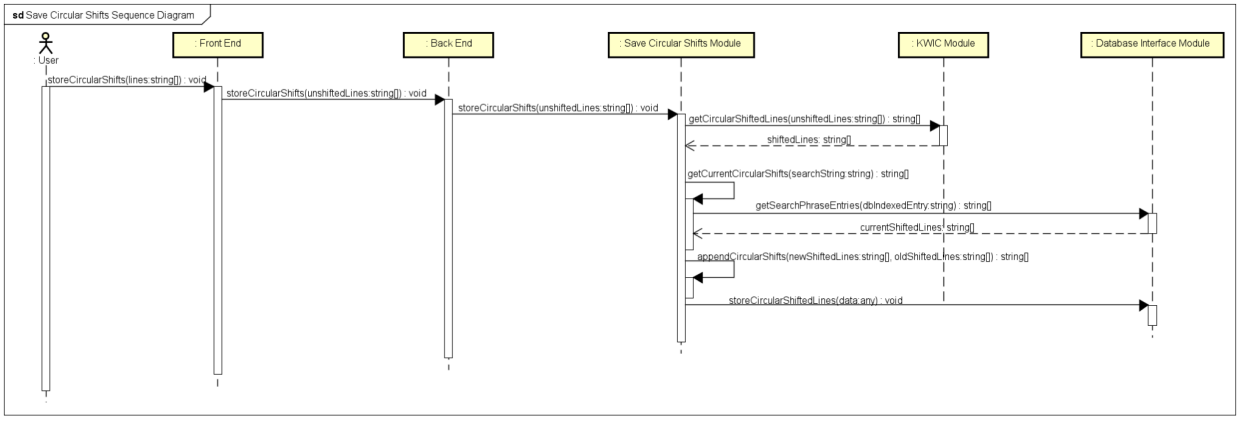
The View Circular Shifts use case occurs when a user submits a set of lines to be circularly shifted, which can be viewed in real time, demonstrated in the following sequence diagram. The full image is available for better viewing at the following link: [L10] View Circular Shifts Sequence Diagram.

Figure 10: View Circular Shifts Sequence Diagram



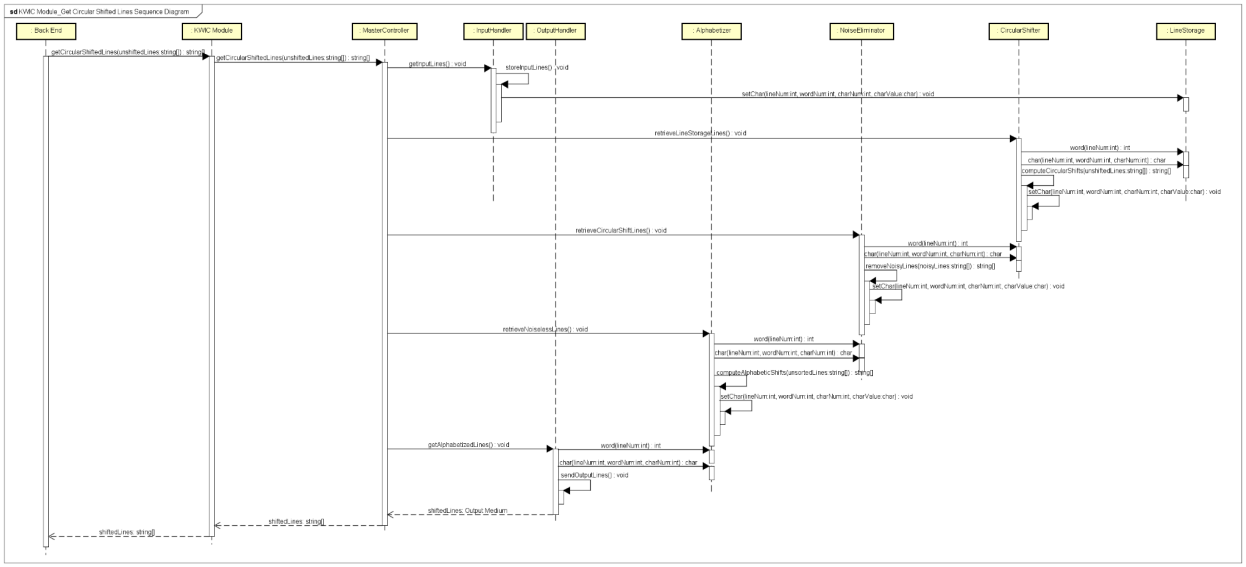
The Save Circular Shifts use case occurs when a user submits a set of lines to be circularly shifted and then saved to the database, demonstrated in the following sequence diagram. The full image is available for better viewing at the following link: [L11] Save Circular Shifts Sequence Diagram.

Figure 11: Save Circular Shifts Sequence Diagram



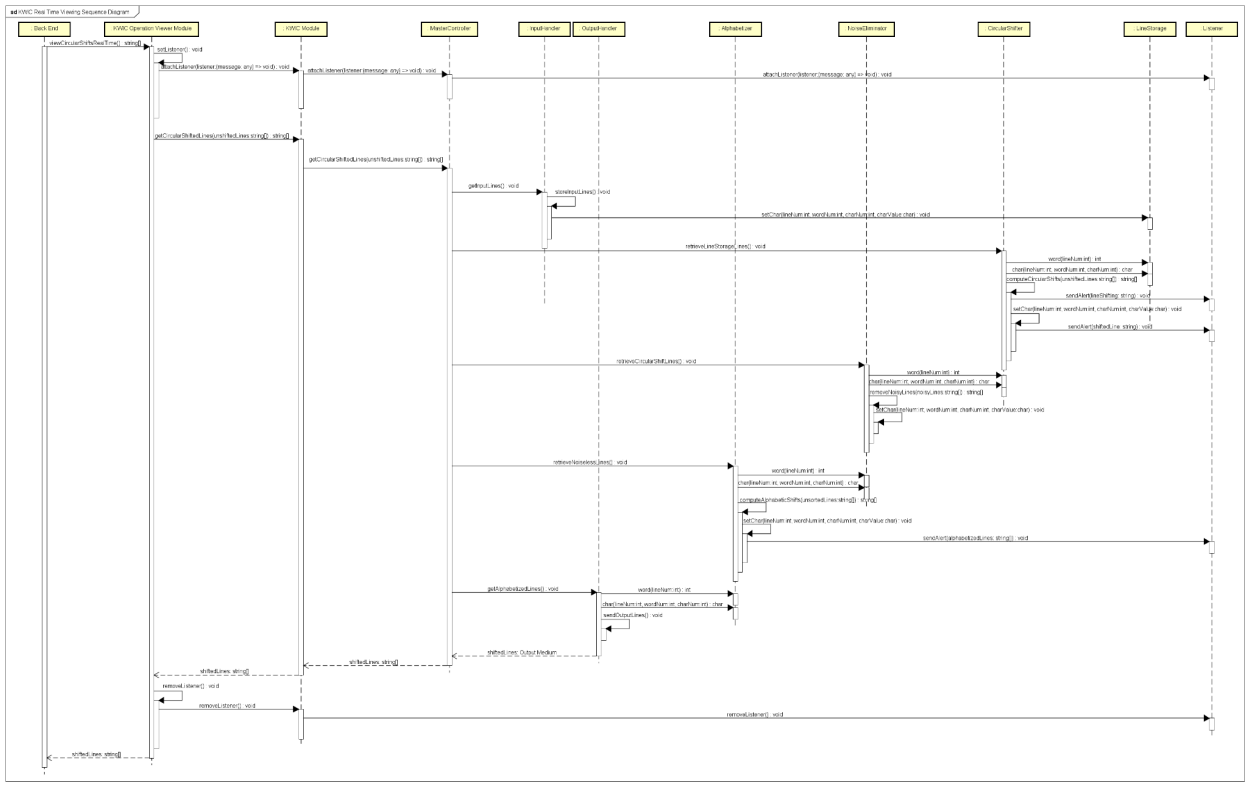
In the above diagrams, when computing the circular shifts, the KWIC Module is called for the computation step. However, the KWIC Module is not a single object - it is a module composed of objects. For clarity, the computation within the KWIC Module was omitted from the above diagrams, and abstracted to a single call to the KWIC Module. However, the internal processing of the KWIC Module is provided in the next sequence diagram, which can be substituted into the above diagrams each time the KWIC Module's public function getCircularShiftedLines is called for equivalent behavior. The full image is available for better viewing at the following link: [L12] KWIC Module Get Circular Shifted Lines Sequence Diagram.

Figure 12: KWIC Module Get Circular Shifted Lines Sequence Diagram



In addition to just computing the circular shifts, the system provides the functionality to view the operations of the KWIC Module in real time through the KWIC Operation Viewer Module. The following sequence diagram depicts the full interactions of the Back End in providing this use case. The full image is available for better viewing at the following link: [L13] KWIC Real Time Viewing Sequence Diagram.

Figure 13: KWIC Real Time Viewing Sequence Diagram



5. Traceability

Traceability involves connecting elements of each phase, both forwards and backwards, so that all components are accounted for to make sure that there are no extra items added between phases and no items are lost between phases.
This traceability section will draw traceability between the Requirements and Architecture sections, both forwards and backwards. The Functional Requirements Specifications are already identified in the Requirements section, but now the architectural components must be specified to clearly designate the traceability connections. Following is a list of architectural components, where they appeared previously in this section, and an architectural ID for clear identification of the component.

Table 12: Architectural Components

Architectural ID (AR ID)	**Architectural Component Name**	**Architectural Component Location / Description**
AR_FE	Front End	The entirety of the Front End, including all user screens.
AR_BE	Back End	The entirety of the Back End, including Routing and all functionality, designed in an OO manner and described above in the architecture diagrams.
AR_MDB	MongoDB Database	The MongoDB service which shall be connected to the Back End. Note that this is a cloud-provided service, which shall not be implemented in this project, and shall instead only be connected to.
AR_BEMOD_DBI	Database Interface Module	The Database Interface Module in the Back End, which provides an interface through which to access the necessary items from the MongoDB Database.
AR_BEMOD_KWIC	KWIC Module	The KWIC Module in the Back End, which provides the functionality to compute circular shifts.
AR_BEMOD_KWICOV	KWIC Operation Viewer Module	The KWIC Operation Viewer Module in the Back End, which allows real-time viewing of the KWIC Module’s operations from the front end.
AR_BEMOD_SE	Search Module	The Search Module in the Back End, which provides an endpoint for a user to

		be able to get search results.
AR_BEMOD_SCS	Save Circular Shifts Module	The Save Circular Shifts Module in the Back End, which saves circular shifts to the MOngoDB Database.
AR_BE_KWIC_MC	Master Controller	The Master Controller object in the KWIC Module.
AR_BE_KWIC_IH	Input Handler	The Input Handler object in the KWIC Module.
AR_BE_KWIC_LS	Line Storage	The Line Storage object in the KWIC Module.
AR_BE_KWIC_CS	Circular Shifter	The Circular Shifter object in the KWIC Module.
AR_BE_KWIC_NE	Noise Eliminator	The Noise Eliminator object in the KWIC Module.
AR_BE_KWIC_AL	Alphabetizer	The Alphabetizer object in the KWIC Module.
AR_BE_KWIC_OH	Output Handler	The Output Handler object in the KWIC Module.
AR_BE_KWIC_LI	Listener	The Listener object in the KWIC Module.

1. Forward Traceability

Forward Traceability will be drawn from the Requirements Specifications - specifically the Functional Requirements Specification - to the architectural components.

Table 13: Requirements Forward Traceability Table

FRS ID	**AR ID**
FRS_BE_1.	AR_BE AR_BEMOD_KWIC
FRS_BE_1.1.	AR_BE AR_BEMOD_KWIC
FRS_BE_1.2.	AR_BE AR_BEMOD_KWIC AR_BE_KWIC_MC AR_BE_KWIC_IH AR_BE_KWIC_LS

	AR_BE_KWIC_CS AR_BE_KWIC_AL AR_BE_KWIC_OH
FRS_BE_1.2.1.	AR_BE AR_BEMOD_KWIC AR_BE_KWIC_NE AR_BE_KWIC_LI
FRS_BE_2.	AR_BE AR_BEMOD_SCS
FRS_BE_2.1.	AR_BE AR_BEMOD_SCS
FRS_BE_2.2.	AR_BE AR_BEMOD_DBI AR_MDB
FRS_BE_3.	AR_BE AR_BEMOD_SE
FRS_FE_1.	AR_FE
FRS_FE_2.	AR_FE
FRS_FE_3.	AR_FE AR_BEMOD_SE
FRS_FE_3.1.	AR_FE
FRS_FE_3.2.	AR_FE
FRS_FE_3.3.	AR_FE
FRS_FE_4.	AR_FE
FRS_FE_4.1.	AR_FE
FRS_FE_5.	AR_FE
FRS_FE_6.	AR_FE
FRS_FE_6.1.	AR_FE
FRS_FE_6.1.1.	AR_FE AR_BEMOD_KWIC AR_BEMOD_KWICOV
FRS_FE_6.2.	AR_FE
FRS_FE_6.2.1.	AR_FE

2. Backward Traceability

Forward Traceability will be drawn from the architectural components to the Functional Requirements Specification.

Table 14: Architecture Backwards Traceability Table

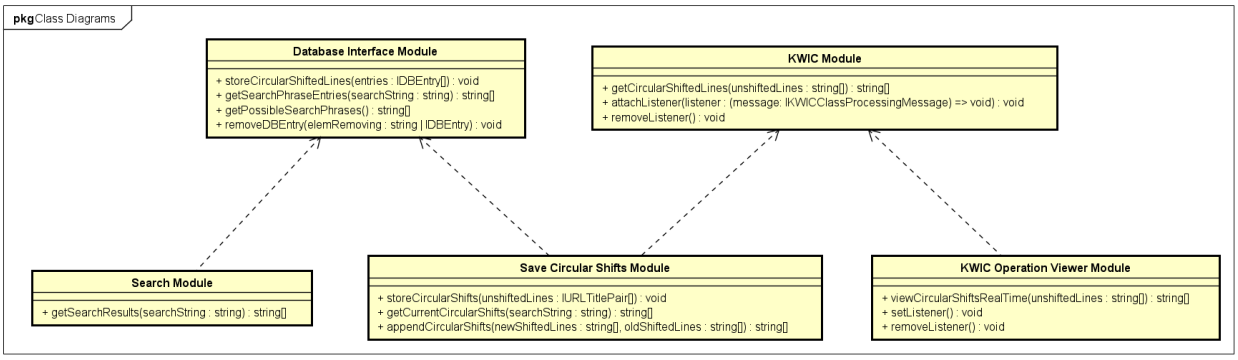
AR ID	**FRS ID**
AR_FE	FRS_FE_1. FRS_FE_2. FRS_FE_3. FRS_FE_3.1. FRS_FE_3.2. FRS_FE_3.3. FRS_FE_4. FRS_FE_4.1. FRS_FE_5. FRS_FE_6. FRS_FE_6.1. FRS_FE_6.1.1. FRS_FE_6.2. FRS_FE_6.2.1.
AR_BE	FRS_BE_1. FRS_BE_1.1. FRS_BE_1.2. FRS_BE_1.2.1. FRS_BE_2. FRS_BE_2.1. FRS_BE_2.2. FRS_BE_3.
AR_MDB	FRS_BE_2.2.
AR_BEMOD_DBI	FRS_BE_2.2.
AR_BEMOD_KWIC	FRS_BE_1. FRS_BE_1.1. FRS_BE_1.2. FRS_BE_1.2.1. FRS_FE_6.1.1.
AR_BEMOD_KWICOV	FRS_FE_6.1.1.
AR_BEMOD_SE	FRS_BE_3. FRS_FE_3.
AR_BEMOD_SCS	FRS_BE_2. FRS_BE_2.1.

AR_BE_KWIC_MC	FRS_BE_1.2.
AR_BE_KWIC_IH	FRS_BE_1.2.
AR_BE_KWIC_LS	FRS_BE_1.2.
AR_BE_KWIC_CS	FRS_BE_1.2.
AR_BE_KWIC_NE	FRS_BE_1.2.1.
AR_BE_KWIC_AL	FRS_BE_1.2.
AR_BE_KWIC_OH	FRS_BE_1.2.
AR_BE_KWIC_LI	FRS_BE_1.2.1.

4. Implementation

In implementing the system, some minor modifications to the architecture were made in order to enhance the system or clarify parameter types as needed. The revised architecture can be seen in the revised class diagrams shown below. Not all diagrams are given here again, as most changes are simply clarifying parameter types, so the sequence of interactions for each use case did not significantly change.
The back end class diagram provides an additional method for the DBInterface module for removing an entry from the database and clarified parameter types for the DBInterface, Save Circular Shifts, and KWIC Operation Viewer modules. The full image is available for better viewing at the following link: [L14] Revised Back End Class Diagram.

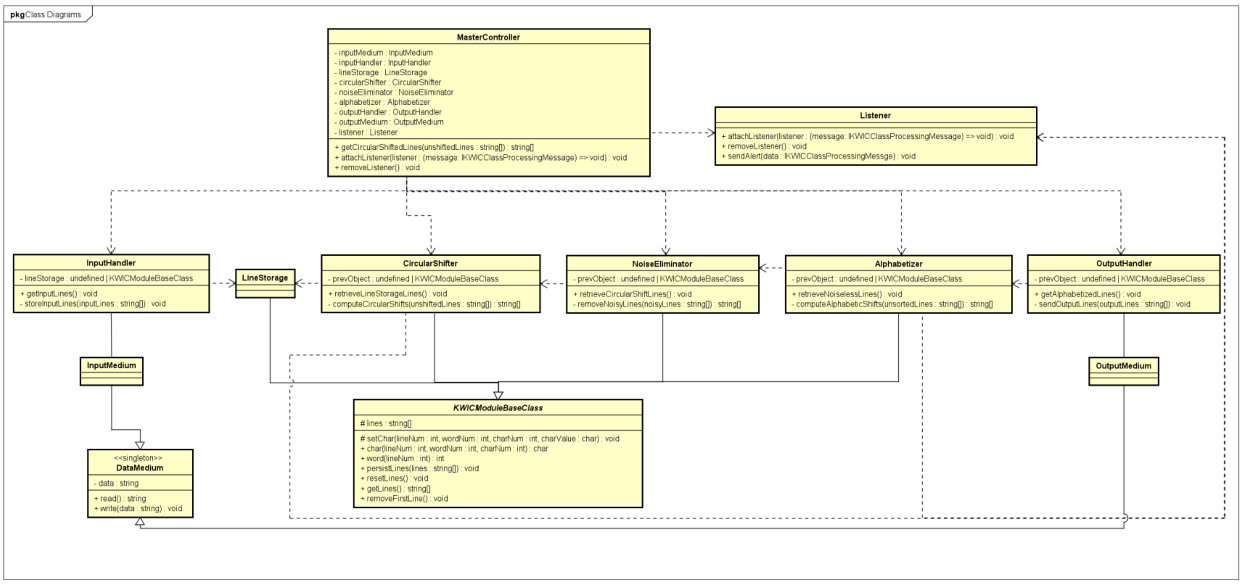
Figure 14: Revised Back End Class Diagram



In the KWIC module, several updates were made. First, each processing element needed a reference to the object from which it would retrieve stored lines, so a prevObject reference was added to these elements. The Master Controller class was updated to store all the objects needed for the processing and pass references to other objects as needed to maintain consistency of the data. The KWIC Module Base Class was updated to include additional useful functions, such as removing the first line saved in the lines variable, getting the whole content of

the lines variable at once, resetting the value of the lines variable, and persisting a set of lines using the previously established setChar method so the same logic was not implemented several times across code. Finally, data types were clarified for the Listener class. The following diagram reflects these updates in the revised KWIC Module architecture. The full image is available for better viewing at the following link: [L15] Revised KWIC Module Class Diagram.

Figure 15: Revised KWIC Module Class Diagram



5. User Manual

The system is accessible to users through the front end site, providing a graphical interface to the system's functionality. This section describes the purpose of each part of the front end and how to use the system.
When the site is requested, the home screen is displayed. All screens provide a navbar across the top to navigate between tabs by clicking on the desired tab, and the content of the tab is displayed below the navbar. There are three tabs shown in the navbar, including the Search tab, Compute Circular Shifts tab, and Save Shifts tab. The home page is the search tab. Each tab's content is described in more detail in its corresponding section below. Note that for this application, a tab is equivalent to a page, since the routing system returns the different page content, while the mechanism displays these different pages as tabs in the navbar.

Search Page

This page allows accessing the search engine functionality of the system. A search bar is the main feature of this page, allowing a user to enter a search string which will be used to access any corresponding data from the database. When a user focuses the search bar and while they are entering text, the search bar provides an autocomplete functionality, showing possible completions of the string based on current entries in the database. When the user clicks the

search button, a corresponding list of URLs from the database are displayed below the search bar.

Compute Shifts Page

This page provides live viewing of the circular shift operation in the back end, known as the KWIC Module. In storing new entries to the database, circular shifts of a string are computed in order to reference a particular site. This page provides insight into how that process works. First, there is an input bar with a button along the top of the screen, where a user can enter lines to be shifted. Upon clicking the button, the three windows below the input bar start to change in real time. As the back end processes the circular shifts, first the window displaying the line currently being processed updates, then the circular shifts window changes to display each line found by shifting the original line, and finally the sorted lines window updates to show all the sorted lines. Finally, if the user wishes to use the lines they input for the circular shifting to save an entry in the database, there is a button at the bottom of the screen labeled “Save Shifts,” which upon clicking, will navigate to the Save Shifts tab and populate the lines to shift field.

Save Shifts Page

This page provides the functionality to save entries to the database easily from a GUI. There are two input fields; one input for lines to be shifted, and one input for URLs to be associated with the shifted lines. Below these two fields is a button. Once the input fields are filled out, the button can be pressed, which causes the system to compute the shifts of the input lines, and then save the URLs with the associated lines. Note that both the number of lines and the number of URLs to save at once can be chosen by the user based on what is entered in the input fields. If only one URL is entered, it will be applied to all lines provided. However, if multiple URLs are entered, the number of URLs must equal the number of lines entered, and each URL will be saved to the database corresponding to the line entered for the URL, meaning each URL will be saved for only the circular shifts of one line entered. Once the save button is clicked and the entries are saved, the entries are now accessible on the Search tab.

Appendix I - Garbage Words

This appendix includes a list of “garbage words” which shall not be used to begin any circular shift line result. These garbage words include some prepositions and conjunctions. Generation of this list of words entailed the use of sources [2] and [3]. Following is a comprehensive list of the garbage words which shall be checked against in the system.

Table 15: Garbage Words

- And
- As
- At
- But
- By
- For
- In
- Is
- Nor
- Not
- Of
- On
- Or
- So
- Than
- The
- With
- Yet

Appendix II - Link URLs

This appendix provides the full URL of all links found in the document. In other sections of the document (barring the References section), any link is provided in shorthand notation (e.g. [L1] Link Title) for brevity and to increase readability of virtual versions of this document. For physical copies of this document, the following table is provided, listing the full URL of each link along with the page number, so web resources listed in the document can still be accessed.

Table 16: Document Links

Link Number	Link Text	Page Number of Link	Full URL
L1	NFR SIG	15	https://coftedahl.github.io/Files/Phase%201/NFR%20SIG_2.png
L2	Cyberminer System Architecture	20	https://coftedahl.github.io/Files/Phase%201/System%20Architecture.png
L3	Back End Architecture	21	https://coftedahl.github.io/Files/Phase%201/Back%20End%20Architecture.png
L4	KWIC Module Architecture	21	https://coftedahl.github.io/Files/Phase%201/KWIC%20Architecture_With%20Garbage%20Remover%20And%20Listener.png
L5	Cyberminer System Class Diagram	22	https://coftedahl.github.io/Files/Phase%201/KWIC%20System%20Class%20Diagram.png

L6	Back End Class Diagram	23	https://coftedahl.github.io/Files/Phase%201/Back%20End%20Class%20Diagram.png
L7	KWIC Module Class Diagram	23	https://coftedahl.github.io/Files/Phase%201/KWIC%20Module%20Class%20Diagram.png
L8	Use Case Diagram	26	https://coftedahl.github.io/Files/Phase%201/Use%20Case%20Diagram.png
L9	Search Sequence Diagram	27	https://coftedahl.github.io/Files/Phase%201/Search%20Sequence%20Diagram.png
L10	View Circular Shifts Sequence Diagram	27	https://coftedahl.github.io/Files/Phase%201/View%20Circular%20Shifts%20Sequence%20Diagram.png
L11	Save Circular Shifts Sequence Diagram	28	https://coftedahl.github.io/Files/Phase%201/Save%20Circular%20Shifts%20Sequence%20Diagram.png
L12	KWIC Module Get Circular Shifts Sequence Diagram	28	https://coftedahl.github.io/Files/Phase%201/KWIC%20Module_Get%20Circular%20Shifted%20Lines%20Sequence%20Diagram.png
L13	KWIC Real Time Viewing Sequence Diagram	29	https://coftedahl.github.io/Files/Phase%201/KWIC%20Real%20Time%20Viewing%20Sequence%20Diagram.png
L14	Revised Back End Class Diagram	34	https://coftedahl.github.io/Files/Phase%201/Revised_Back%20End%20Class%20Diagram.png
L15	Revised KWIC Module Class Diagram	35	https://coftedahl.github.io/Files/Phase%201/Revised_KWIC%20Module%20Class%20Diagram.png

References

[1] L. Chung. “Advanced Software Architecture and Design.” UTD Webpages. Accessed: Sep. 26, 2025. [Online.] Available: https://personal.utdallas.edu/~chung/SA/syllabus.htm
[2] “List of Conjunctions in English with Examples.” 7esl.com. Accessed: Sep. 27, 2025. [Online.] Available: https://7esl.com/conjunctions-list/
[3] “Prepositions.” Cambridge Dictionary. Accessed: Sep. 27, 2025. [Online.] Available: https://dictionary.cambridge.org/grammar/british-grammar/prepositions