CosmoSurf

KWIC Lexical Sort System

SE 6362 – Advance Software Architecture and Design (Fall 2024) - [Prof. Lawrence Chung]

Team 3 : CosmoSurf

Team 3 Members:

- Pooja Ramesh Korake
- Shreyas Gaikwad
- Nikhil Hegde

Phase 1

- Introduction
- Functional Requirement
- Non – Functional Requirement
- Architecture
- Key Design Concepts
- Flow Execution
- Conclusion
- Question & Answer

Tools:

Programming Language: Java

Front-End Tools: HTML, CSS, JavaScript (Fetch API)

Back-End Tools: Spring Boot

Database: SQL

Development Tools: VSCode, Maven

JSON: Data interchange format for communication between front-end and back-end

Introduction

- The aim is to build a search engine using KWIC (Key Word in Context) system.
- The KWIC Lexical Sort System generates all possible circular shifts of a sentence and sorts them lexicographically.
- The system implements a KWIC system with a focus on circular shifts and lexicographical sorting.
- The circular shifts involve shifting words, where the first word moves to the end in each shift.
- After generating these shifts, the system sorts them in lexicographical order and displays the results.

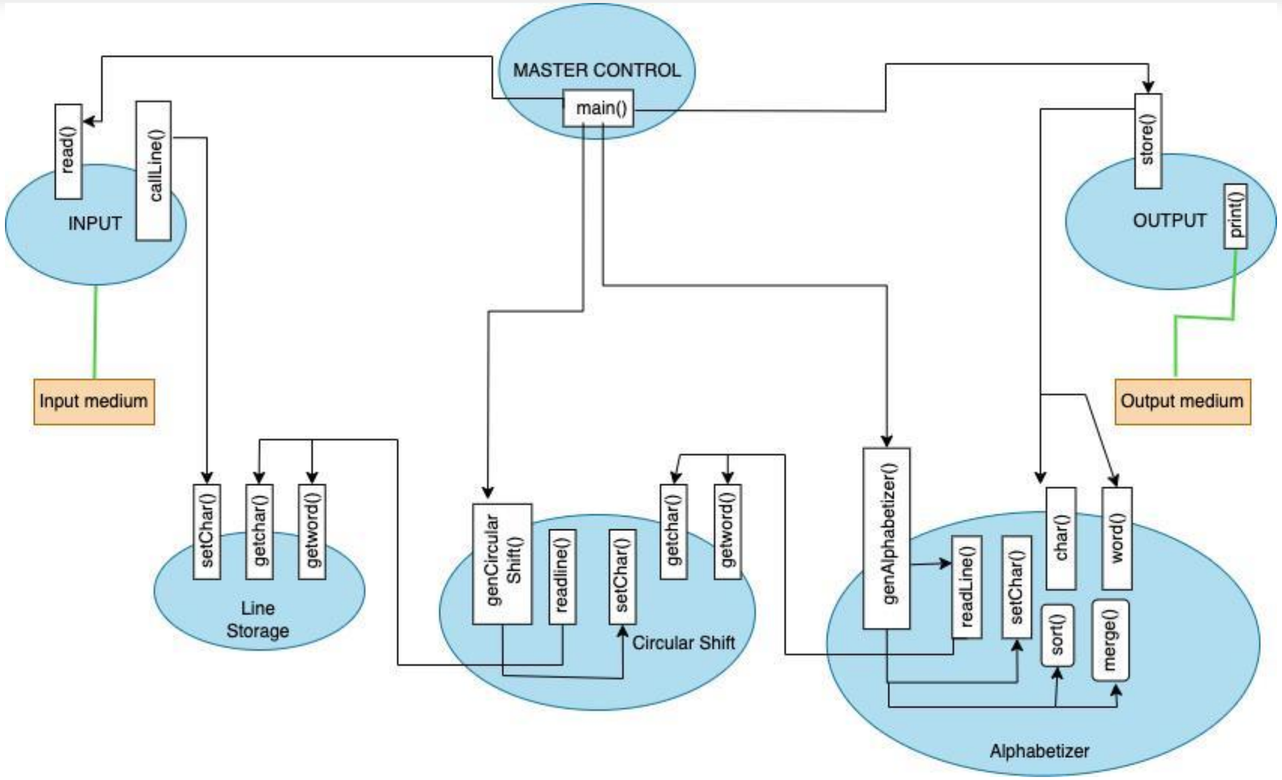
Functional Requirements:

ID	Description
FR001	The KWIC system shall accept the input entered by keyboard.
FR002	The KWIC system shall accept the ordered set of lines from the user.
FR003	The KWIC system shall accept input characters supported by ASCII character set.
FR004	The System shall circularly shift each word in a line until the original line produced.
FR005	The KWIC system shall be able to add additional modules without any issues.
FR006	The KWIC system shall output the circular lines within 10 seconds.

Non – Functional Requirements:

ID	Description
NFR001	The KWIC system shall be user friendly.
NFR002	The KWIC system shall be responsive.
NFR003	The KWIC system shall be good performance.
NFR004	The KWIC system should be enhanceable.
NFR005	The KWIC system should be portable.

Abstract Architecture



Key Design Concepts : Circular Shift

Circular Shift Generation :

- Shifting Logic : The `generateCircularShifts` method takes a input sentence, splits it into words, and systematically generates circular shifts by reordering the words. It appends each circular shift to a list of `KWICEntry` objects.

- Modular Arithmetic: The circular nature of the shifts is achieved using modular arithmetic (`j % words.length`) to loop back to the beginning of the word array when necessary.

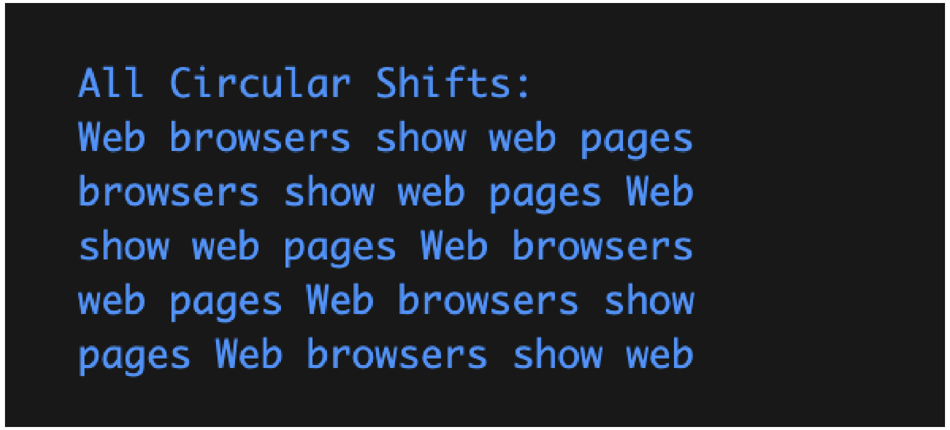
Flow of Execution(Circular Shift)

1. The input sentence "Web browsers display web pages" is passed to the ‘generateCircularShifts()’ method.
2. Shift Generation : For each word in the sentence, a new circular shift is created by rotating the words in the sentence. This continues until all possible shifts are generated.
   - Split this input string at “<space>” delimiter and store it in an array called ‘words’
   - words = [web, browsers, display, web, pages]
0 1 2 3 4
   - Outer Loop (for (int i = 0; i < words.length; i++)):
     - This loop iterates over each word in the input sentence. The variable i represents the starting index for the circular shift.
   - Inner Loop (for (int j = i; j < i + words.length; j++)):
     - This loop constructs the shifted sentence starting from the word at index i. It runs ‘i+words.length’ times to include all words in the circular shift.
     - The modulus operator is used to generate circular shifts

Circular Shift execution with example:

words[]=	Web	browsers	show	web	pages
indices	0	1	2	3	4

	i						
Iteration 1	0	j	0	1	2	3	4
		j%5	0	1	2	3	4
		words[j%5]	Web	browsers	show	web	pages
Iteration 2	1	j	1	2	3	4	5
		j%5	1	2	3	4	0
		words[j%5]	browsers	show	web	pages	Web
Iteration 3	2	j	2	3	4	5	6
		j%5	2	3	4	0	1
		words[j%5]	show	web	pages	Web	browsers
Iteration 4	3	j	3	4	5	6	7
		j%5	3	4	0	1	2
		words[j%5]	web	pages	Web	browsers	show
Iteration 5	4	j	4	5	6	7	8
		j%5	4	0	1	2	3
		words[j%5]	pages	Web	browsers	show	web



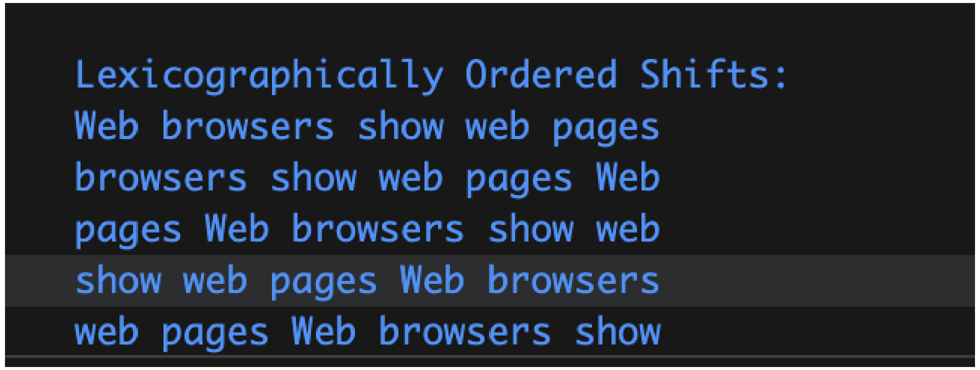
```
All Circular Shifts:
Web browsers show web pages
browsers show web pages Web
show web pages Web browsers
web pages Web browsers show
pages Web browsers show web
```

Key Design Concepts : Lexicographical Sort

Lexicographical Sorting :

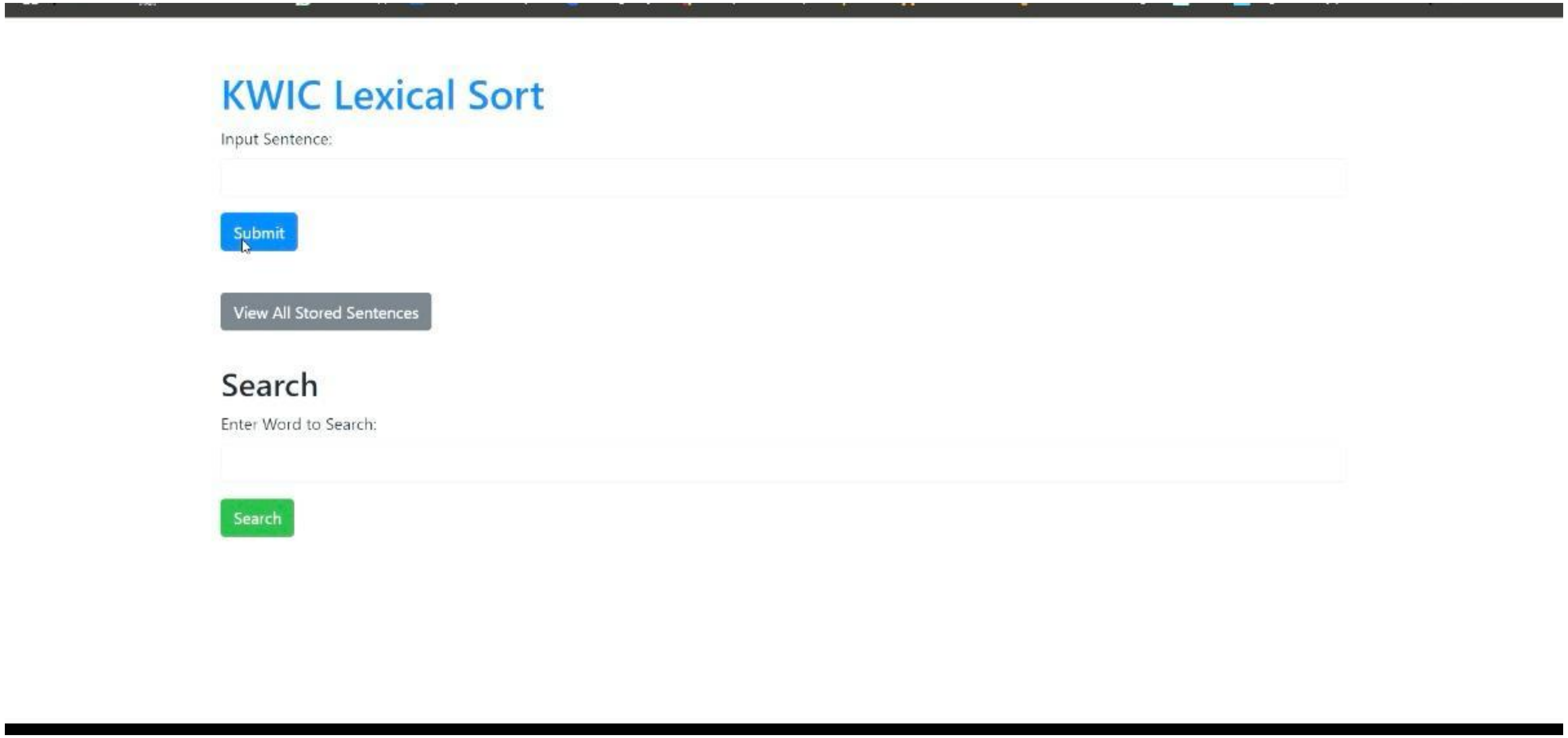
1. Sorting : Sorting with Collections.sort() : Once all circular shifts are generated, the system uses `Collections.sort()` to sort the list of `KWICEntry` objects. Since `KWICEntry` implements `Comparable`, this sort is lexicographically based on the shifted sentences.

2. Output : The system outputs both the unsorted and sorted lists of circular shifts.



```
Lexicographically Ordered Shifts:
Web browsers show web pages
browsers show web pages Web
pages Web browsers show web
show web pages Web browsers
web pages Web browsers show
```

Frontend :



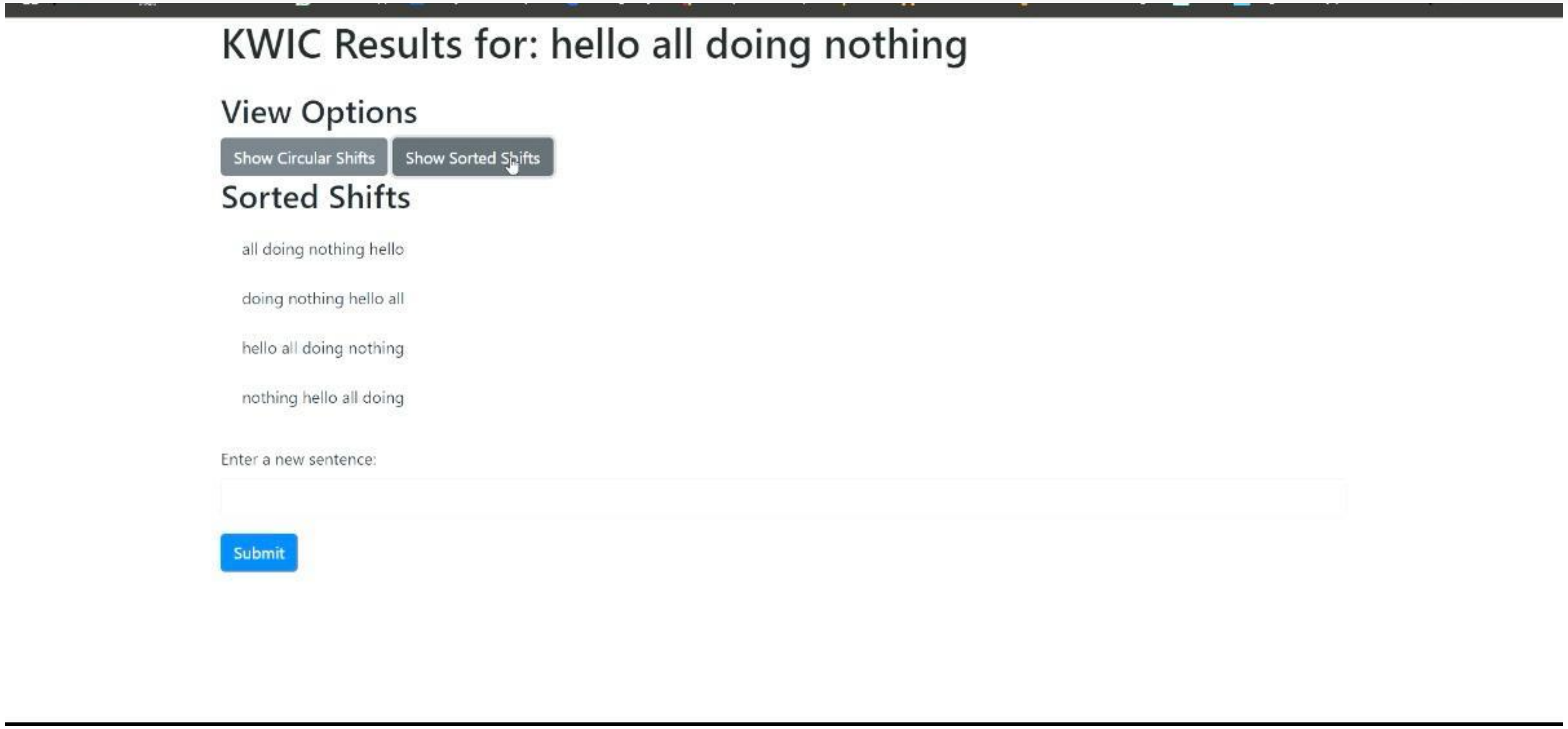
KWIC Lexical Sort

Input Sentence:

Search

Enter Word to Search:

Output :



KWIC Results for: hello all doing nothing

View Options

Sorted Shifts

all doing nothing hello

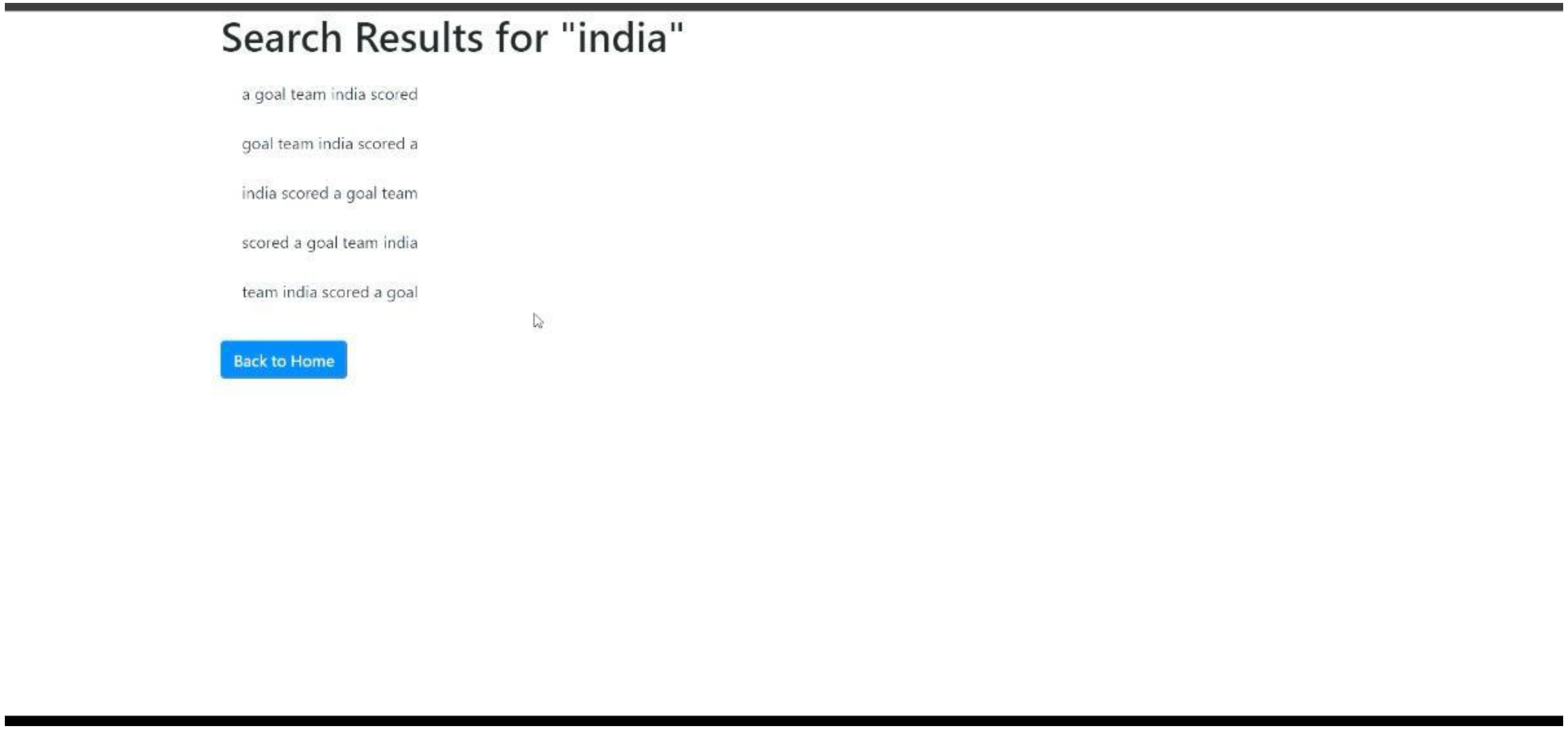
doing nothing hello all

hello all doing nothing

nothing hello all doing

Enter a new sentence:

Search output :



Search Results for "india"

a goal team india scored

goal team india scored a

india scored a goal team

scored a goal team india

team india scored a goal

Conclusion :

- The KWIC Lexical Sort system is a modular and efficient implementation of generating circular shifts and sorting them lexicographically.
- By encapsulating the logic for each shift in the `KWICEntry` class and leveraging Java’s built-in sorting mechanisms, the system ensures clean and maintainable code.
- Future extensions can be easily incorporated by modifying or adding functionality to the core methods.

THANK YOU !

ANY QUESTION ?

Back-up slides for Q&A

How circular shift we executed:

1.Initialization:
Let’s understand this with the following example
Input string: “Web browsers display web pages”
Split this input string at “<space>” delimiter and store it in an array called ‘words’

words = [web, browsers, display, web, pages]
0 1 2 3 4

1.Appending Words:

1. shiftedSentence.append(words[j % words.length]).append(" ");
2. Here, words[j % words.length] is used to wrap around the array of words:

   If j exceeds the length of words, j % words.length gives the appropriate index to loop back to the start of the array.
3. Each word is appended to shiftedSentence, followed by a space. This effectively builds the shifted version of the input sentence.

For the circular shift generation, the modulus (%) operator is used.

This % operator can be used as indices of the array of ‘words’ to generate the circular shift for the ‘input string’

Here we have used 2 for loops,

> **The outer ‘for’ loop with the variable ‘i’, that iterates from 0 to the ‘words.length’.**
>
> **An inner ‘for’ loop with the variable ‘j’, that iterates from ‘j=i’ to ‘j< i+words.length’**

“The modulus operator generates circular numbers shown below

Numbers=0,1,2,3,4,…,N,N+1,N+2,N+3,……

Numbers % N =:

0%N = 0, 1%N= 1,

0,1,2,3,4,…,N-1,0,1,2,3,…,N-1,0,1,2,…”

Iteration 1:

```
    i=0
        j=0, words [0%words.length] = words[0]
        J=1, words[1%words.length]=words[1]
        .
        .
        .
        J= n-1, words[n%words.lenth]=0
```