Software Design Document

MapReduce Project2

**Team 4**Aravinda KR, Russell Warfield, Jacob Wansor, Trevor Engstrom

**Table of Contents**

**Table of Contents ii**

**1.** **Introduction 1**

1.1 Purpose 1

1.2 Scope 1

1.3 Conventions 1

**2.** **Design Considerations 1**

2.1 System Environment 1

**3.** **Use Cases 1**

**4.** **Architecture** 2

4.1 Overview 3

4.2 Subsystem Components (Class Diagram) 4

**5.** **Flow Chart 5**

**6. Sequence Diagram 6**

**7.** **Unit Test Specification 7**

**8.** **User Interface 8**

**9.** **References 10**

# Introduction

## Purpose

This architecture document describes the objectives, requirements, design and verification of the Architecture for Map Reduce – Phase 2.

## Scope

This document is for the developer implementing the architecture to ensure compliance with software requirements for Map Reduce – Phase 2.

## Conventions

None.

# Design Considerations

All design considerations are as per the Phase 2 Requirements.

## System Environment

The MapReduce application runs on Windows environment. Visual Studio with C++17 and above configuration required. The program requires space for an output file.

# 

# Use Cases

The primary use case for this standalone command-line program is to take a directory of text files as an input and then output a single text file that contains a list of all words in the input files and how many times they are used. The program incorporates two explicitly loaded dynamic link libraries to handle the map and reduce portions of the program.

A diagram of a map

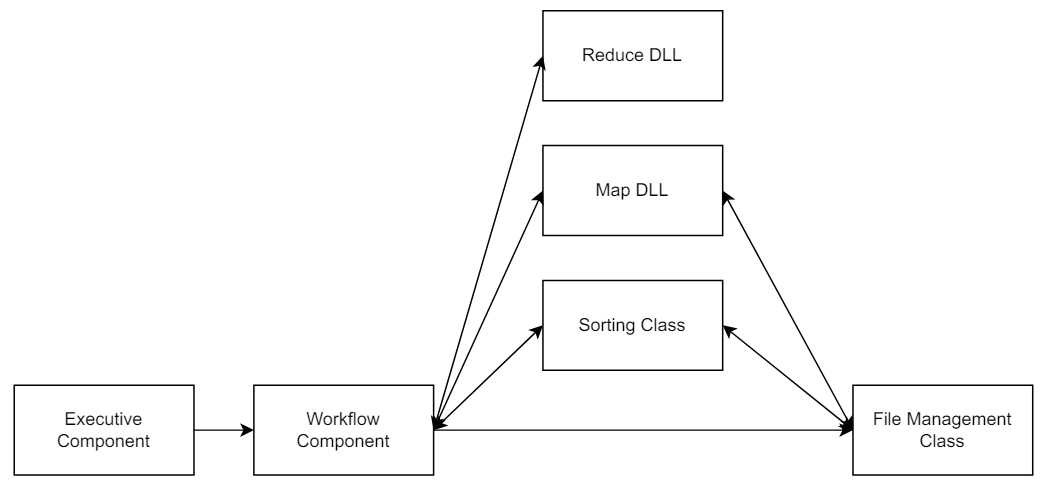
Description automatically generated

*Figure 1 - Use case Diagram.*

# 

# Architecture

The following block diagram provides the system architecture.

**

*Figure 2 - Overall Software Architecture for MapReduce – Phase2*

## Overview

Following is the list of various layers and subcomponents of the system,

* File Management Class – Handles file I/O operations. It provides an interface for the user to manipulate the directory structure, read files, and write files.
* Map Library – Dynamic link library that reads the words and creates the Vector references based on the number of words. Provides two interfaces *mapFunc()* and *exportFunc()* for users to interact with the map derived class. The *mapFunc()* function will tokenize the value parameter into distinct words. Then, it will put it in the format of (“the”, 1), (“a”, 1), etc…

The *exportFunc()* function is for buffering output in memory and to periodically write to disk.

Exports createMapper() and destoryMapper() functions to create and destroy map objects using the \_\_declspec(dllexport) keyword.

* Sorting Class – Sorts and aggregates values in the map. This is the intermediate step before the data is passed to the Reducer class.
* Reduce Library – Dynamic Link Library that takes the aggregated data and sums all the values of the iterator. Provides *reduceFunc()* interface for users to interact with the reduce derived class. The *exportFunc()* internal function is responsible for writing reduced output to a file.

Exports createReducer() and destoryReducer() functions to create and destroy reduce objects using the \_\_declspec(dllexport) keyword.

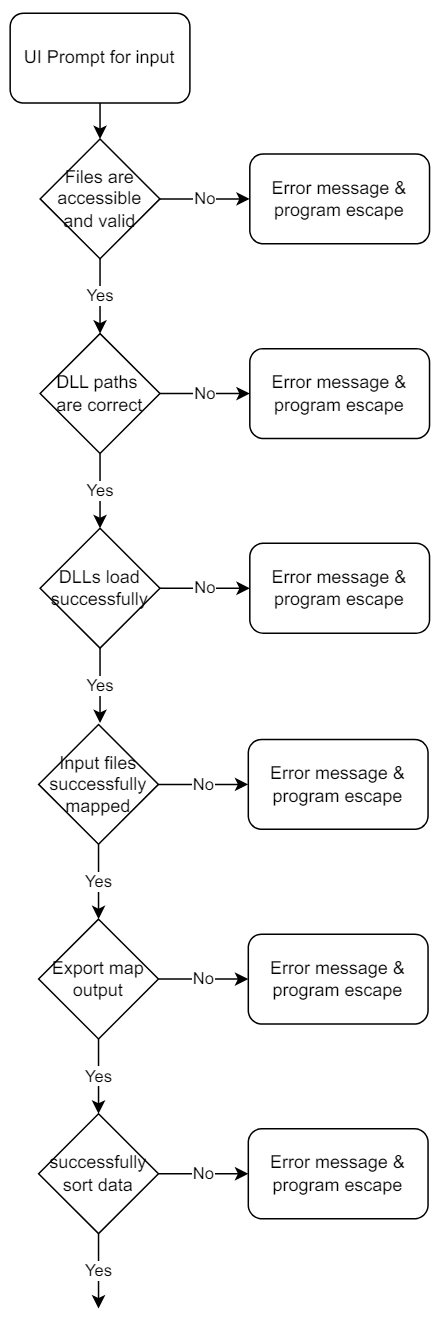
* MapReduceInterfaces - Defines the interface classes for Map and Reduce to be used as generic templates. This allows for any Map and Reduce DLLs that adhere to this interface to call from the Workflow component. This is achieved by defining MapBase and ReduceBase base classes for map and reduce derived classes respectively. Workflow invokes map and reduce methods through these base classes.
* Workflow Component – The workflow component is what the program will use to determine the order of calls. It first loads the Map and Reduce DLLs into the program based on the arguments provided at runtime. The workflow component is responsible for keeping a synchronous chain of events and manages the File Management, Mapper, Reducer and Sorter classes. Each method called by the workflow component will return a value to determine success, which will allow the workflow component to call the next method in the chain.
* Executive Component – Takes the user provided arguments as input, output and temp directories and parses them. These directories are passed into the Workflow Component if valid.

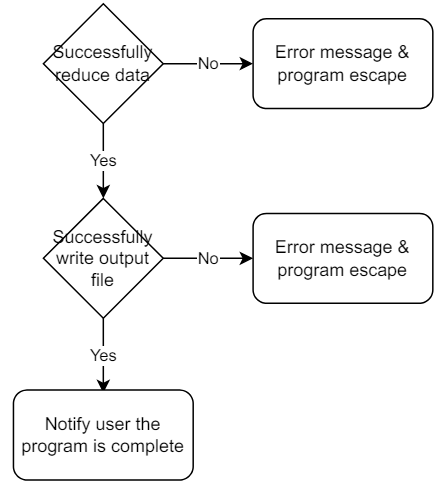
## Subsystem Components (Class Diagram)

## 

*Figure 3 - Class Diagram*

# Flow Chart





*Figure 4 - Flow chart*

# Sequence Diagram

# A diagram of a computer Description automatically generated

*Figure 5 - Map Reduce Sequence Diagram*

# Unit Test Specification

| **Test Case-1** | **Validate MapReduce with valid inputs** |
| --- | --- |
| Test Precondition | 1. Valid directory with input files 2. Valid directory to hold output files. 3. Valid temporary directory to hold intermediate output files |
| Test Procedure | 1. Run the MapReduce program. 2. Provide input, output and temp directories path via the command line. 3. Press ENTER to run the MapReduce operation |
| Pass/Fail Criteria | 1. The intermediate data will be written to the temporary directory. 2. An empty file SUCCESS will be written out in the output directory. 3. MapReduce result will be written to the output directory |

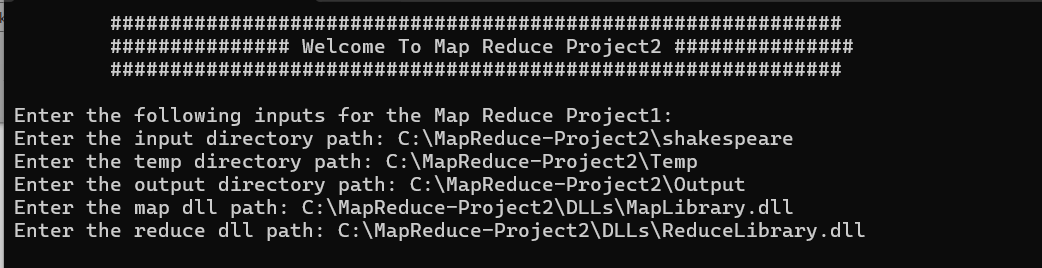
| **Test Case-2** | **Validate MapReduce with invalid directory** |
| --- | --- |
| Test Precondition | 1. Valid directory to hold output files. 2. Valid temporary directory to hold intermediate output files |
| Test Procedure | 1. Run the MapReduce program. 2. Provide invalid input directory and valid output and temp directories path via the command line. 3. Press ENTER to run the MapReduce operation |
| Pass/Fail Criteria | MapReduce function will return error |

| **Test Case-3** | **Validate MapReduce with an empty input directory** |
| --- | --- |
| Test Precondition | 1. Valid directory to hold output files. 2. Valid temporary directory to hold intermediate output files |
| Test Procedure | 1. Run the MapReduce program. 2. Provide an empty input directory and valid output and temp directories path via the command line. 3. Press ENTER to run the MapReduce operation |
| Pass/Fail Criteria | MapReduce function will return error |
| **Test Case-4** | **Validate MapReduce with a known input file for word count** |
| Test Precondition | 1. Valid directory with input file of known word counts 2. Valid directory to hold output files. 3. Valid temporary directory to hold intermediate output files |
| Test Procedure | 1. Run the MapReduce program. 2. Provide an empty input directory and valid output and temp directories path via the command line. 3. Press ENTER to run the MapReduce operation |
| Pass/Fail Criteria | MapReduce function will return output file with correct wordcount |
| **Test Case-5** | **Validate MapReduce with invalid DLL location** |
| Test precondition | 1. Valid directory to hold output files. 2. Valid temporary directory to hold intermediate output files |
| Test Procedure | 1. Run the MapReduce program. 2. Provide valid input, output and temp directories path via the command line. 3. Input invalid DLL location for map DLL 4. Press ENTER to run the MapReduce operation |
| Pass/Fail Criteria | 1. Executive component will return error |

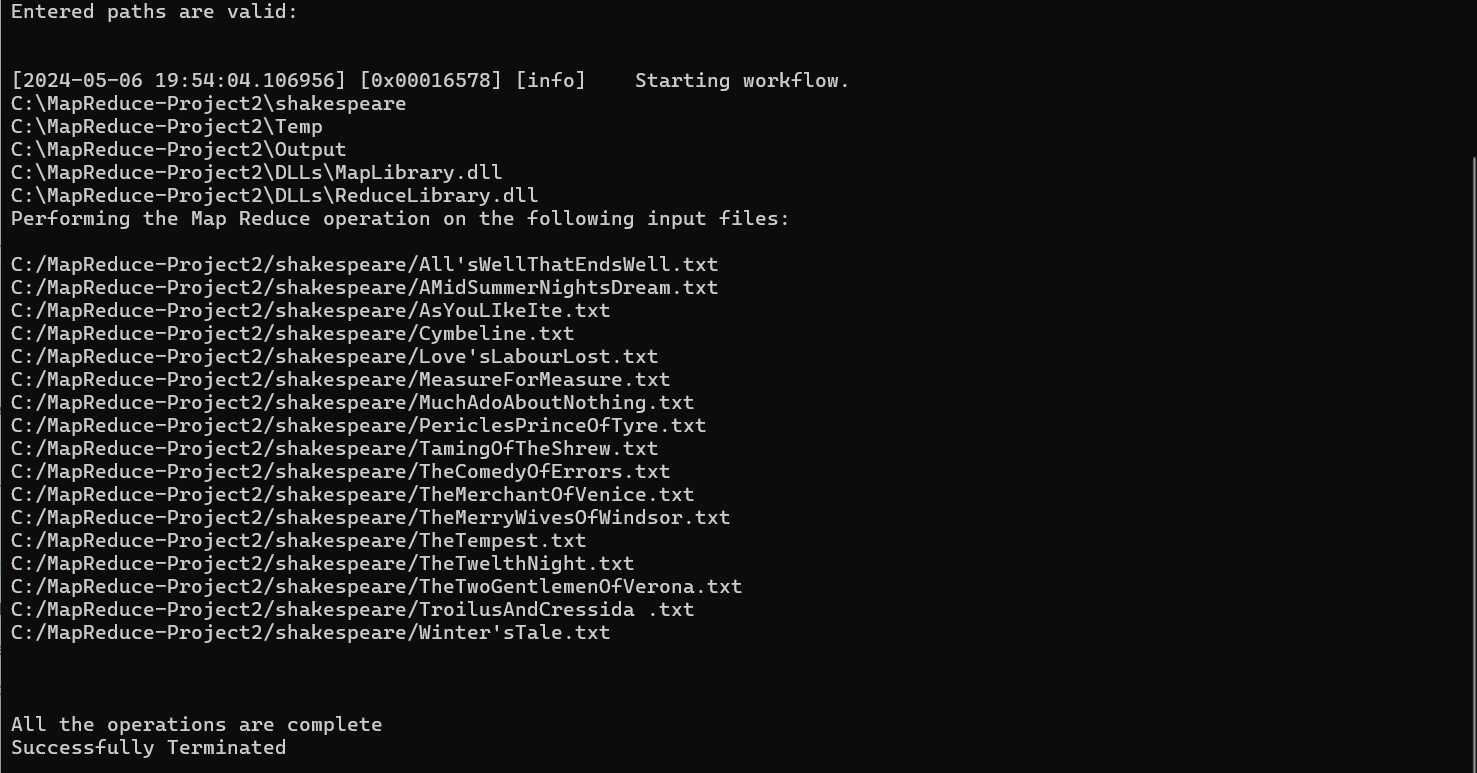
# 

# User Interface

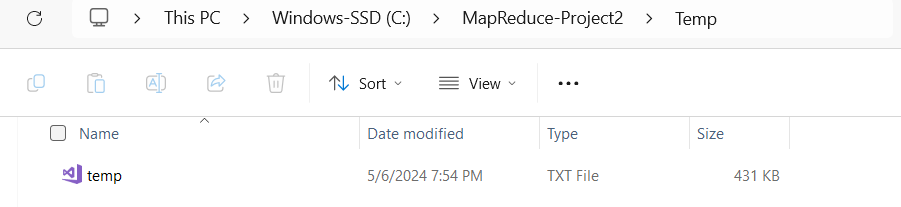
The following screenshot captures user input,



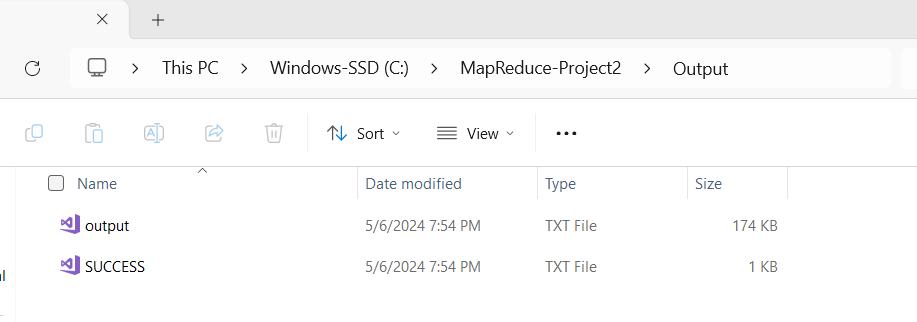
Execution flow,



Temp directory,



Output directory,



# 

# References

1. *CSE687\_Project\_Phase\_1.doc.*
2. [*https://learn.microsoft.com/*](https://learn.microsoft.com/) *for builtin APIs*