**Project Summary**

Through this project, we will be evaluating the 4 KWIC Index architectures (Shared Data, Abstract Data Type, Implicit Invocation, and Pipe and Filter) and try to decide which architecture will be the best choice to use for developing a KWIC index generation tool for an online course.

The created KWIC index generation tool takes a group of HTML files as its input that represent the online notes from one lecture where the titles of each page in the group will be indexed with couple assumptions; firstly, each lecture will have a maximum of 100 pages, and secondly, if a page has no title, the tool will run for that specific page and generate a title for it so it can be used in the bigger/general title generation process. The tool will be running once a week with a new lecture’s worth of HTML pages where it will add the new index entries to the existing index created from previous runs assuming that there’s only one lecture occurring per week with the goal of developing this tool to run for more than one lecture per week. The output of the indexing tool will be HTML webpage that displays all the index entries from the previous weeks plus the new index entry from the current week in a list form sorted ascendingly by lectures’ dates. That HTML webpage will also override any existing webpage output created from previous runs/generations. Lastly, the indexing tool will initially be executed in a Windows 10 PC platform, however, the future goal is to be able to run this tool in other platforms like Linux and macOS.

**Evaluation Criteria**

In my opinion, nowadays, most technology tools’ infrastructure should support scalability, reuse, and change in all forms like algorithm, functionality, etc. because of how fast technology keeps developing which makes clients/customers frequently ask for new requirements to make use of that technology development. Accordingly, I’m going to evaluate the 4 KWIC Index architectures by considering several factors concerning the initial use of the system but also how an architecture would adapt to future changes.

Firstly, the architecture should allow the change in algorithm and function for cases like:

* Switching between indexing just the page title and indexing all words on the page in case of page title doesn’t exist.
* Having to run the tool for only one lecture versus multiple lectures per week

Secondly, the architecture should also be efficient performance and time wise where the process does not take long time to generate the output HTML page and does not consume a lot of hardware resources like CPU, RAM, and data storage inefficiently.

In addition, I believe the architecture should have a good maintainability quality where the tool can easily be repaired when any part of the process fails due to new changes.

Lastly, I think the architecture should support portability so it would allow the tool to run not only on Windows 10, but also on other platforms like Linux or macOS which will save time and mental overhead for anyone involved in moving new versions of the tool across different environments.

That is, we are going to evaluate the 4 architectures based on the following criteria: change in algorithm and function, performance, maintainability, and portability.

**4 Architectures Evaluation**

Now that we have our set of criteria ready, we are going to discuss the strengths and weaknesses of each architecture with respect to each of the previously mentioned criterion.

* *Shared Data:* this architecture is probably the best architecture in terms of performance because the data is communicated between the system components through shared storage where all computations also share that same storage which represents both time and hardware efficiency. Besides, I believe that having a shared storage and connected components would make portability easier because then all you need to do is to focus on handling only the data storage with its components from platform to another instead of worrying about moving each isolated component. However, this shared storage logic will badly affect the ability to easily make any changes in the algorithm or function of that shared storage as it will affect all the components using that storage. In addition, I believe maintainability will be an issue with this architecture because if any failure occurred to that shared storage, it would affect all components using it which makes the maintainer applies the repairs/fixes to all components instead of just one piece that caused the issue.
* *Abstract Data Type:*
* *Implicit Invocation:*
* *Pipe and Filter:*

**Best Architecture**

**Conclusion**