# WordLadder

Challenge to implement a Word Ladder game solver

## Scope

The project wordladder is an implementation of a solver for the word ladder problem.

The Word ladder challenge consists of finding a sequence of words that differ from the siblings by just one letter in the same position.

The solver should return one of shortest lists or conclude that is not possible to resolve given the parameters (start word, end word, word List).

The purpose of this project was to propose and efficient and correct implementation for the solver using modern framework (net Core 3.1)

and programming concepts using appropriate design patterns to promote a good architecture design for testability and scalability.

## Technology

The project was written in C# using Microsoft .Net Core 3.1 framework and is a Console application.

## Code structure/Architecture

The WordLadder project consist of three projects plus a Test project:

### 1. WordLadder -

Console App that hosts a BackgroundWorker service that runs the actual processor.

### 2. WordLadder.Models -

Contain classes for type definition only used in the other two projects.

### 3. WordLadder.Services -

Interface declarations and service implementation for injected services.

Here we define the services:

- CommandLinePayloadLoader - Loads the arguments passed by the user and creates a JobPayloadCommand that will define the search parameters.

- WordLadderProcessor - The class that contains the Word Ladder Algorithm.

- WordListRepository - Abstracts the data acquisition layer. Reads the word list from a txt file and make the lines available to the app.

-

## The Algorithm

The algorithm starts by filtering the dictionary into a subset containing words with the same length of the start word.

It adopts a recursive behavior in which for each word starting with the "Start Word", a set of regex expressions are generated that allow for

the matching of words from the dictionary with only one, in same position, letter of difference.

Then for each of the successor words a successor set is generated until the final word is found among the successors or is not possible to find

new successors. In this case the algorithm concludes that does not exist a solution given the start, end and dictionary given.

The algorithm assures to find the best sequence, that is, the minimum number of words by applying a bread-first approach in the word successor generation.

This means that at each iteration, we only generate the grand-Children of a node if there are no more siblings of that word without an attempt to generate successors.

## Final notes

This project was a good challenge that allow me to consider dispar software concerns.

It was possible to include different technics and technologies, but I believe that while it is something of academic exercise

it is also important to restrict the use of the technologies minding the final objective and not throw them into the project just because.

I could do it.