HIGH LEVEL DESIGN

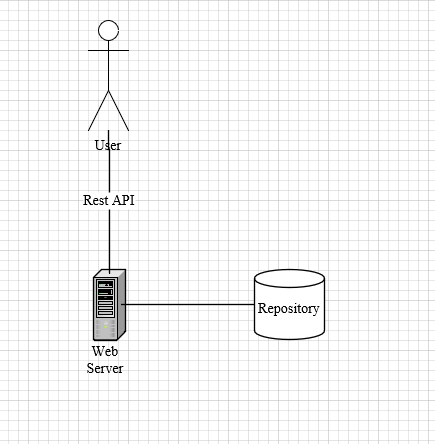
Word statistics

# Requirements

1. A 'word counter' endpoint
   * Receives a text input and counts the number of appearances for each word in the input.
   * The endpoint should not return data (but you may return success status, etc).
   * The endpoint should be able to accept the input in 3 ways:
     + A simple string sent in the request.
     + A file path (the contents of the file will be used as input).
     + A URL (the data returned from the URL will be used as input).
   * The input may be very large (up to tens of gigabytes).
   * The results (the number of appearances of each word) should be persisted, to be used by the ‘word statistics’ service.
2. A 'word statistics' endpoint
   * Receives a word and returns the number of times the word appeared so far (in all previous inputs)

# Application Architecture

This design is a “**strait forward**” solution. In this design and in particular persistence in memory implementation there no scaling option. To add a scalability Persistence must be implemented by some redistributable solution like Centralized cache or DB. Centralized file system is a bad option, the multiple number of servers might compete on this resource and there be application errors to access file



# Web application architecture

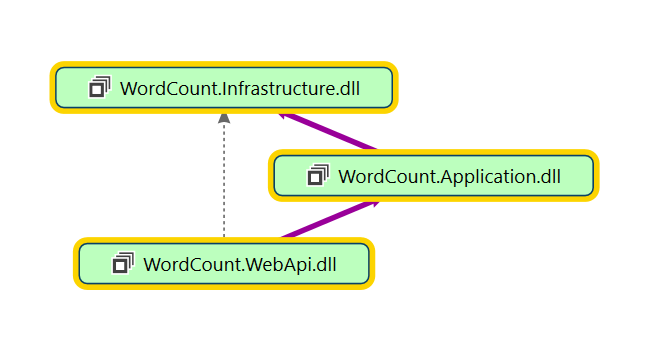
Web application based on .Net core 3.1 (LTS). Application have 2 end point

* word/analyze – this end point receives a different type of inputs
  + plain text
  + file path with data
  + url to service that return data

the end point should analyze data and store in for future use

* word/statistics – this end point receive an word and return number of occurrences of that word

## Dependencies



## Modules Responsibility

* WordCount.WebApi – this component is a controller it receives user request and return response. It should check/validate user input but it doesn’t have any business logic.
* WordCount.Application – this is a core module of application. It provides an API to any consumer such WebApi or some other consumer that might use it functionality

This module encapsulates business logic and should not be changed if no changes was made to business logic. The API (Public) of this module also shouldn’t be changed if BL didn’t changed

* WordCount.Infrastructure – this module responsible provide an API and functionality and implementation for any Infrastructure that might be required such as consume external services (REST, WCF), implement a repository interface (Memory, File, Cache, DB)

## Logic flow

As we can see the in all cases the logic flow is a same the difference between is just data source.



So when user submit data for analysis we can run all types with in a same request.



# Assumptions

1. The URL data processing is should be in GET method, it a default method if not specified
2. Exception handling handled by **ErrorHandler**.cs middleware in WebApi component, thus code doesn’t contain a try catch blocks

This is a simplest way to handle exception without crashing the client.

The correct way is to handle each operation by it exception type

For example if HTTP get/post made to some resource then response code should be checked

# Architecture changes to make system scalable

In my opinion the statistics not requires to be a real time, thus we can process/analyze data and aggregate statistics. To achieve this, the design should be event driven, here is the list of changes should be made

1. Add external service that process data
2. Remove from “existing application” data analyze/process. Leave only query to repository for statistics end point
3. On Request instead of analyzing data within request, fire/publish event
4. Persistence should be implemented using DB

## This is how a new architecture will be look like

