Word Evaluator Research

VocabVersus

Thomas van der Molen

|  |  |
| --- | --- |
| **Project Information** | |
| Project members | Thomas van der Molen |
| Project Name | VocabVersus |

Table of Contents

[Topic 3](#_Toc132230965)

[Goal 3](#_Toc132230966)

[Summary 3](#_Toc132230967)

[Questions 3](#_Toc132230968)

[Word Set Storage 4](#_Toc132230969)

[Persistence 4](#_Toc132230970)

[Storage Options 4](#_Toc132230971)

[Relational Database 4](#_Toc132230972)

[Loosely Structured Database 5](#_Toc132230973)

[Word evaluation 5](#_Toc132230974)

[Database Query 5](#_Toc132230975)

[Apache Lucene 5](#_Toc132230976)

[Cloud Search 6](#_Toc132230977)

[Evaluator Performance 7](#_Toc132230978)

[SQL Database 7](#_Toc132230979)

[Storage 7](#_Toc132230980)

[Execution Time 7](#_Toc132230981)

[Apache Lucene 8](#_Toc132230982)

[Storage 8](#_Toc132230983)

[Execution Time 8](#_Toc132230984)

# Topic

Vocab Versus requires a service where **correct words can be stored** and later be checked against to **see if a word submitted by a player is correct**.

For this service, large amounts of data in the form of word lists (which contains all correct words for a given topic e.g. English, periodic table, etc.) have to be persistently stored to be re-used in games. Secondly, the service will have to be able to look through a given word list and find if a word submitted by a user is contained within the list.

This whole process will have to happen with many player submitions and return the result in real-time, as to give the player feedback on their submission.

# Goal

The goal of this research will be to explore the different methods of **storing large amounts of text** in the form of word lists, with the ability to (very) **quickly search** through the text and find a match for a given word.

# Summary

Shortly summarize the findings here

# Questions

There are multiple questions that will guide the research, these questions are based on the topics explained in the [introduction to this topic](#_Topic_1).

|  |  |
| --- | --- |
| **Main Question** | |
| What is the best method of storing and evaluating large amounts of grouped words to use for a game of Vocab Versus | |
| **Sub Questions** | **Research Methods** |
| What storage solutions exist for storing large amounts of textual data | [Literature study](https://ictresearchmethods.nl/Literature_study)  [Community research](https://ictresearchmethods.nl/Community_research) |
| What specialized methods exist for finding matching parts of text within a large amount of text | [Available product analysis](https://ictresearchmethods.nl/Available_product_analysis) |
| What combination of persistent storage and word matching technique gives the most optimal execution time | [Prototyping](https://ictresearchmethods.nl/Prototyping)  [Benchmark test](https://ictresearchmethods.nl/Benchmark_test) |

# Word Set Storage

The VocabVersus game relies on word sets to validate that a given word submission by a player is a valid word. This word set must be stored somewhere so it can be referenced in the future during a game.

## Persistence

Due to the same word set being allowed to be used in many different games, the data will have to be stored persistently and cannot be coupled, for example to a single game instance.

Due to the large nature of a word set and the ability to use different word sets depending on the game rules, the data size and storage capacity will have to be kept in mind when determining how to persistently store these datasets.

## Storage Options

For this research, database solutions will be discussed as they are the most prominent and accessible method of persistently storing data, the specific database technologies that will be explored are based on the most popular database technologies as explored in the [2022 stackoverflow developer survey](https://survey.stackoverflow.co/2022/#section-most-loved-dreaded-and-wanted-databases).

### Relational Database

[Relational databases](https://www.ibm.com/topics/relational-databases) are a common practice method for storing persistent data, as it allows for large amounts of data to be stored in a predefined structure.

There are many different structured relational databases, such as:

* [MSSQL](https://www.microsoft.com/en-us/sql-server/sql-server-2019)
* [MySQL](https://www.mysql.com/)
* [SQLite](https://sqlite.org/index.html)
* [PostgreSQL](https://www.postgresql.org/)

Most of these databases offer similar functionality based on the SQL query language and therefore also have similar [benefits and tradeoffs](https://databasetown.com/relational-database-benefits-and-limitations/#:~:text=The%20main%20benefits%20of%20using,issue%20of%20speed%20can%20arise.).

|  |  |
| --- | --- |
| **Benefits** | **Limitations** |
| Fast query processing | Harder scalability |
| Data consistency | Complex relational structures |

One other large limitation that is often brought up for SQL databases, is that performance may decrease drastically as datasets grow however, as shortly introduced in [this blog article](https://blog.greglow.com/2018/02/05/sql-server-big-databases-really-slower/) this can often be remedied with modern database technologies via methods such as [indexing or compressing information](https://www.mssqltips.com/sqlservertip/7309/improve-sql-server-performance-when-querying-very-large-log-tables/).

Storing word sets in a structured SQL database could look like the ERD displayed below:

Text

Description automatically generated

### Loosely Structured Database

Loosely structured databases, more often referred to as [NoSQL databases](https://www.ibm.com/topics/nosql-databases#:~:text=NoSQL%2C%20also%20referred%20to%20as,structures%20found%20in%20relational%20databases.), are created to be support more flexible data structures then conventional [SQL based databases](#_Relational_Database). NoSQL databases have several different types, with the 4 primarily used ones being:

* Document
* Key-value
* Column-oriented
* Graph

For a clearer understanding of these 4 types I highly recommend the [introduction of these types by MongoDB](https://www.mongodb.com/scale/types-of-nosql-databases).

|  |  |
| --- | --- |
| **Benefits** | **Limitations** |
| Data flexibility | Less extensible querying |
| Significant modern growth support | No guaranteed data consistency |

# Word evaluation

Besides storing the word sets, word sets will have to be searched through to find a match with a player submitted word, this can be done in many ways with the primary goal being to find a match as fast as possible.

## Database Query

When using an [SQL based database](#_Relational_Database) to store word sets, the SQL language can be used to make queries directly to the database.

An SQL query to find the first occurrence of a word inside a wordlist, utilizing the database structure as [presented during the relational database](#_Relational_Database) could look like the following.

SELECT **COUNT(1)**

FROM **words**

WHERE **setId = wordsetid** AND **word = playersubmittion**;

This query will return either 0 if no instance was found or 1 if a match was found. An issue with this approach, however is that with normal behavior all word with the given wordsetid will be looked through individually until a match is found, this process can however be improved by using [indexing methods](https://www.sqlshack.com/top-five-considerations-for-sql-server-index-design/) on the word value.

One security risk to keep in mind when generating a database query with user input, is to ensure no [SQL injection](https://portswigger.net/web-security/sql-injection) happens by sanitizing the request for which there are many different techniques; one common method is by escaping any input thereby ensuring it is not read as valid code/queries.

## Apache Lucene

[Apache Lucene](https://lucene.apache.org/) is a specialized library for text search, search engines such as [ElasticSearch](https://www.elastic.co/elasticsearch/) and [Solr](https://solr.apache.org/) are based on Apache Lucene.

Lucene works similar to a [NoSQL document based database](#_Loosely_Structured_Database), with the ability to serialize stored text specifically for future search, speeding up this process significantly along with other built-in features such as [fuzzy search](https://www.ibm.com/docs/en/informix-servers/12.10?topic=modifiers-fuzzy-searches).

Due to Lucene being an open source java based library, it has been ported to many other languages and frameworks such as [.NET](https://lucenenet.apache.org/), [python](https://lucene.apache.org/pylucene/), [etc](https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=119540992).

## Cloud Search

There are several enterprise level cloud solutions built specifically for text search, while these could solutions might not be as versatile as solutions previously mentioned, the easy of use along with the reliability of large scale cloud solutions could be a valuable trade off.

While there are many text search cloud solutions some popular ones include: [Azure Cognitive Search](https://azure.microsoft.com/en-us/products/search/?ef_id=_k_Cj0KCQjwxMmhBhDJARIsANFGOStIUCOOE99ZIdHGoTfI7hNZkJXyeNsWVsp4-Pm0QsnRaGXG6pUYs20aAm4iEALw_wcB_k_&OCID=AIDcmmy4pl1olr_SEM_k_Cj0KCQjwxMmhBhDJARIsANFGOStIUCOOE99ZIdHGoTfI7hNZkJXyeNsWVsp4-Pm0QsnRaGXG6pUYs20aAm4iEALw_wcB_k_&gclid=Cj0KCQjwxMmhBhDJARIsANFGOStIUCOOE99ZIdHGoTfI7hNZkJXyeNsWVsp4-Pm0QsnRaGXG6pUYs20aAm4iEALw_wcB). [Amazon CloudSearch](https://aws.amazon.com/cloudsearch/) and [Algolia](https://www.algolia.com/).

While these cloud search engines are often very powerful in applications such as in-website search or general web indexing, they often sadly do not provide much flexibility for specific data list search.

# Evaluator Performance

Explain that a couple of solutions will be built and compared via Postman response time to see which is the fastest

Say smth about that the projects tested are setup as ASP.NET applications and the data used (link to the EnglishWordsList that is used)

## SQL Database

Link to the used project

### Storage

Insert using the insert WebAPI



|  |  |  |
| --- | --- | --- |
| Default MSSQL database | | |
| **1 million records** | **2 million records** | **3 million records** |
| 322,34 MB | 441,90 MB | 561,65 MB |
| Indexed MSSQL database | | |
| **1 million records** | **2 million records** | **3 million records** |
| 629,41 MB | 866,91 MB | 1.102,97 MB |

How expensive are different options in terms of storage capacity required

### Execution Time

Explain that the execution time is a measurement of the full RESTfull API request time using the average of 10 requests with a data size of 1,1 million records split into 4 word sets

Request word evaluation via WebAPI



|  |  |  |
| --- | --- | --- |
| 1 million total records, 4 word sets, 10 requests | | |
| Default MSSQL database | | |
| **Starting Record** | **Middle Record** | **Last Record** |
| 4,0 ms | 71,5 ms | 141,0 ms |
| Indexed MSSQL database | | |
| **Starting Record** | **Middle Record** | **Last Record** |
| 4,5 ms | 4,1 ms | 3,5 ms |

## Apache Lucene

### Storage

### Execution Time