

COMP90015 Distributed System Assignment 1

Student Name: YU-WEI LIN

Student ID: 1537312

I. Introduction:

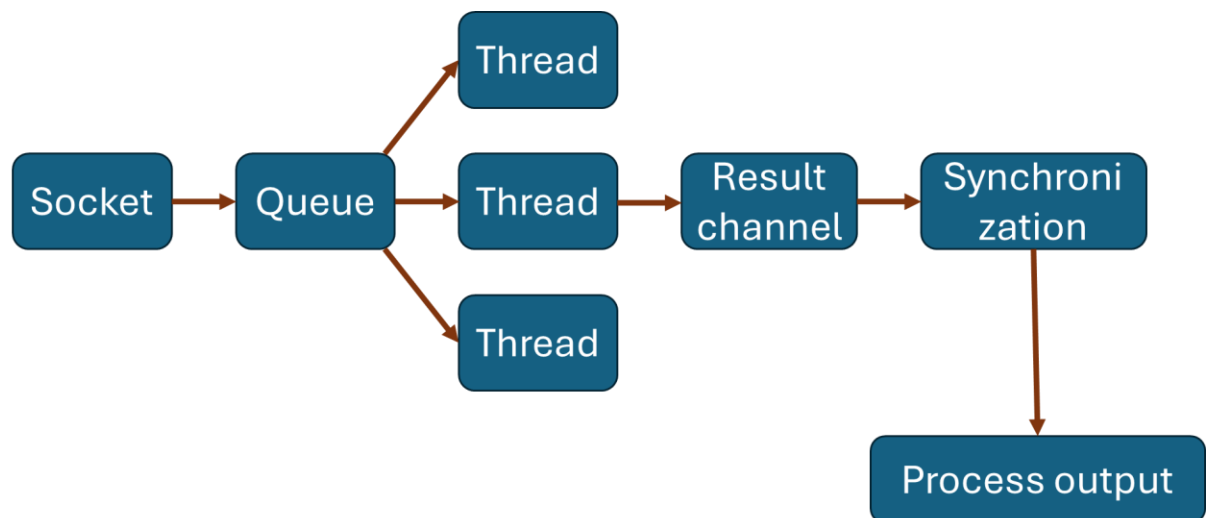
This project is mainly for inspiring students to develop their first multithreads communication socket, which should be scalable to handle multiple connections from client side. Additionally, building a GUI interface via Java Swing to make client-user more comfortable when using the dictionary.

II. Project Overview

In this section, the perspective of this project overview would be provided, including the how the multi-thread model works and model of server/ client.

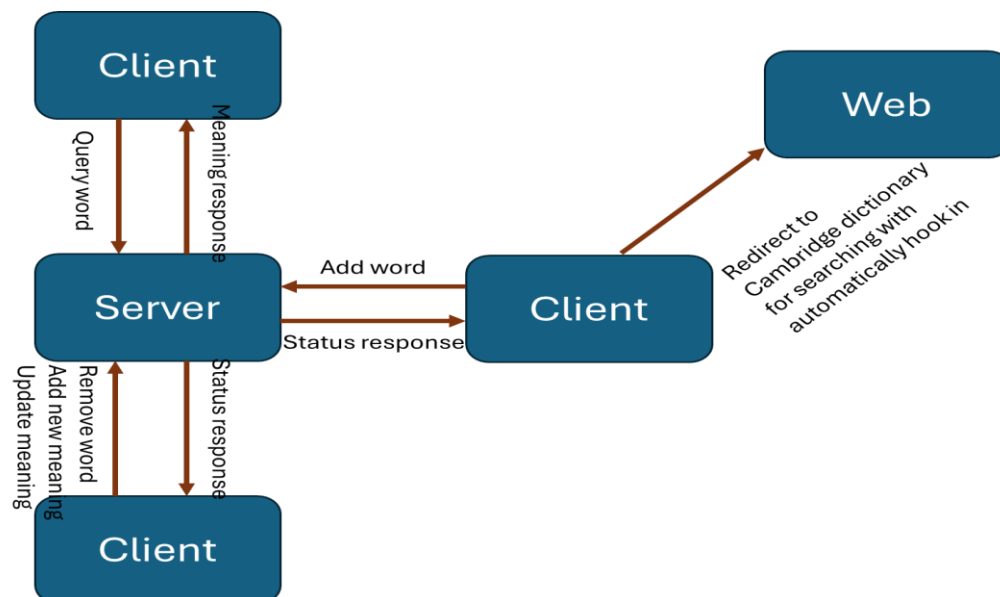
A. Multi-thread Model

Worker pool, Thread-per-request, Thread-per-connection and Thread-per-project are four common architectures for building a multi-thread socket. In this project, worker pool is the architecture which be used for handling multiple threads requests from client, processing data, and so on. The tasks from client side are all placed in a queue, which allows the system to execute numerous stuffs simultaneously. Image that in a company project team, whenever evolving requirements comes from stakeholders, the product owner would assign the task to one of the group or members instead of hiring a new team or new employee which can optimise company's effectiveness communication overhead by leveraging the current team's capabilities, similar to how a worker pool reuses threads to handle multiple tasks without creating new threads for each request.. Via reusing the existing thread, when a thread is idle it might trigger a "work stealing" mechanism which can increase the efficiency when handling the multi-thread requests.



B. Server/ Client Model

The system of model implemented in this prohect is Server/ Client Model allowing centralised control over the authentication and security. Moreover, this model is suitable for establishing the connection when clients query the central database and backup all the data to store on the server side as well, which is aligned with the project target.



With combining Server/ Client model and workers pool, allowing a centralized architecture that offers strong benefits in terms of efficiency, resource management, scalability, and fault tolerance.

III. Implementation functional requirements

The implementation of this project would be demonstrated below:

A. Server/ Client

Server:

Offering a execution pool for addressing multiple-client connections and request simultaneously, then transform the dictionary into json format.

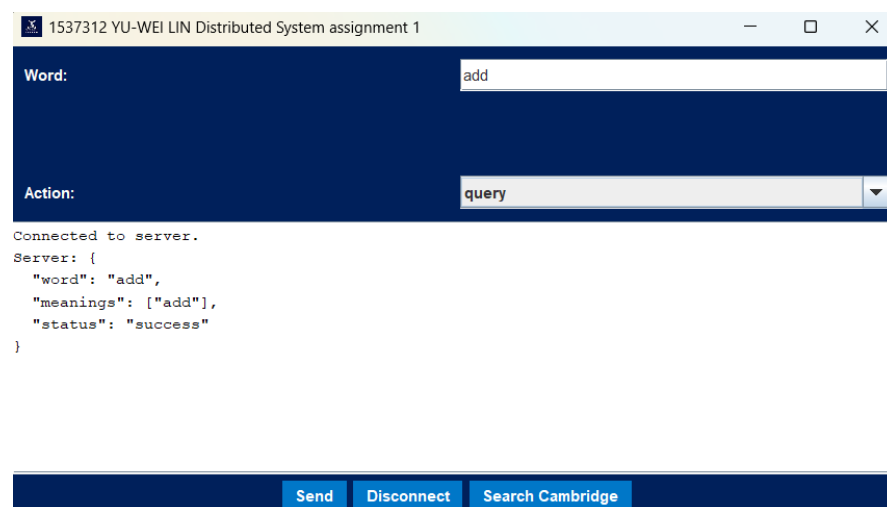
Client:

Allowing query, add, remove a word and get the status response from server side, supporting add new meaning and update meaning on existing words as well. Furthermore, client can choose to use disconnect to terminate and button which would redirect to a website for search the meaning of the word which would be automatically hooked in.



B. Function display

1. Query a word



2. Add a word

1537312 YU-WEI LIN Distributed System assignment 1

Word:

delulu

New Meaning:

believing things that are not real or true

Action:

add

Connected to server.
Server: {
 "message": "Duplicate word.",
 "word": "delulu",
 "status": "error"
}

Send

Disconnect

Search Cambridge

3. Remove a word

1537312 YU-WEI LIN Distributed System assignment 1

Word:

Action:

remove

Connected to server.
Server: {
 "message": "Word removed.",
 "word": "hi",
 "status": "success"
}

Send

Disconnect

Search Cambridge

4. Add new meaning/ Update meaning

1537312 YU-WEI LIN Distributed System assignment 1

Word:

delulu

New Meaning:

delulu is solulu

Action:

addmeaning

Connected to server.
Server: {
 "message": "Meaning added.",
 "word": "delulu",
 "status": "success"
}

Send

Disconnect

Search Cambridge

1537312 YU-WEI LIN Distributed System assignment 1

Word:

delulu

Action:

query

```
"message": "Meaning added.",
"word": "delulu",
"status": "success"
}
Server: {
  "word": "delulu",
  "meanings": [
    "delulu is solulu",
    "believing things that are not real or true"
  ],
  "status": "success"
}
```

Send

Disconnect

Search Cambridge

1537312 YU-WEI LIN Distributed System assignment 1

Word:

delulu

Existing Meaning:

delulu is solulu

New Meaning:

delulu is not solulu, Labubu is solulu

Action:

updatemeaning

Connected to server.

```
Server: {
  "message": "Meaning updated.",
  "word": "delulu",
  "status": "success"
}
```

Send

Disconnect

Search Cambridge

1537312 YU-WEI LIN Distributed System assignment 1

Word:

Action:

query

Connected to server.

```
Server: {
  "message": "Meaning updated.",
  "word": "delulu",
  "status": "success"
}
Server: {
  "word": "delulu",
  "meanings": [
    "believing things that are not real or true",
    "delulu is not solulu, Labubu is solulu"
  ]
}
```

Send

Disconnect

Search Cambridge

C. Thread Security

In this project, common dictionary thread safety was a very crucial aspect since it gets frequently updated and accessed by many clients. The following actions were taken to address the requirement:

1. Represent the internal dictionary structure using ConcurrentHashMap ensuring thread-safe concurrent access can be given to it without using any explicit synchronizations.
2. For having thread-safe inner sets on multiple insertions and deletions at the same time, implementing ConcurrentHashMap.newKeySet() for the inner sets representing definitions of a word.
3. As for saving files, the saveDictionary() method must be synchronized to allow one thread to write on the JSON file at a time.
4. Concurrent read-only operations as word queries can be allowed without locks to enhance performance.

This structure provides an equilibrium between thread safety and operational efficiency and manages to reduce contention on locks and has no sacrifices in terms of data consistency.

IV. Excellence and Creativity

A. Excellence

1. “ExecutorService” for managing multiple client connections concurrently

Utilizing a cached thread pool lets server dynamically increase and decrease the number of threads depending on the current usage by clients. Thus, server resource utilization is optimized.

```
// Multithread holding
// Create a scalable pool to handle incoming client connection.
ExecutorService pool = Executors.newCachedThreadPool();
// Socket for listening specific port, and pass to handleClient method once connection established
try (ServerSocket server = ServerSocketFactory.getDefault().createServerSocket(port)) {
    System.out.println("Server is listening on port " + port);
    while (true) {
        Socket client = server.accept();
        pool.execute(() -> handleClient(client));
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

2. Comprehensive Input Validation and Robust Error Handling

Return specific error messages, helping clients quickly identify the root cause of issues, which enhances both usability and debugging

3. Data persistence and input validation

Implementing `Runtime.getRuntime().addShutdownHook()` to automatically save the dictionary when the server shuts down, increasing the user trust and system stability.

```
// Ensuring when the connection shut down, the content would be saved in the dictionary automatically
Runtime.getRuntime().addShutdownHook(new Thread(() -> {
    saveDictionary();
    System.out.println("Dictionary saved. Server shutting down.");
}));
```

Provides detailed error messages for invalid inputs that user errors would not cause system crashes but instead result in clear explanations.

```
if (word == null || word.isEmpty() || !word.matches(regex: "[a-zA-Z]*")) {
    return new JSONObject().put("status", "error").put("message", "Invalid word. Only alphabetic characters are allowed.").toString();
}
```

4. Extensible JSON Format:

The easy-to-expanding data structure using `JSONObject` and `JSONArray` for storing meanings (e.g., examples, parts of speech to come) shows foresight and extensibility for future additions.

B. Creativity

1. Embedded error avoidance from GUI interface

Before the client takes the action, the server side would detect if the word in the word field existing or not, then deciding which kind of actions could be showed up in the toggle box. For example, if word is not existed, the toggle box would only show up “query, add, and remove”; once the word is existing, it would display two more actions “addmeaning and updatemeaning” in the toggle box.

1537312 YU-WEI LIN Distributed System assignment 1

Word:

remove

Action:

query

query

add

remove

Connected to server.

Send

Disconnect

Search Cambridge

1537312 YU-WEI LIN Distributed System assignment 1

Word:

delulu

Action:

query

query

add

remove

addmeaning

updatemeaning

Connected to server.

Send

Disconnect

Search Cambridge

1537312 YU-WEI LIN Distributed System assignment 1

Word:

delulu

Existing Meaning:

New Meaning:

Action:

updatemeaning

Connected to server.

1537312 YU-WEI LIN Distributed System assignment 1

Word:

delulu

New Meaning:

Action:

addmeaning

Connected to server.

Send

Disconnect

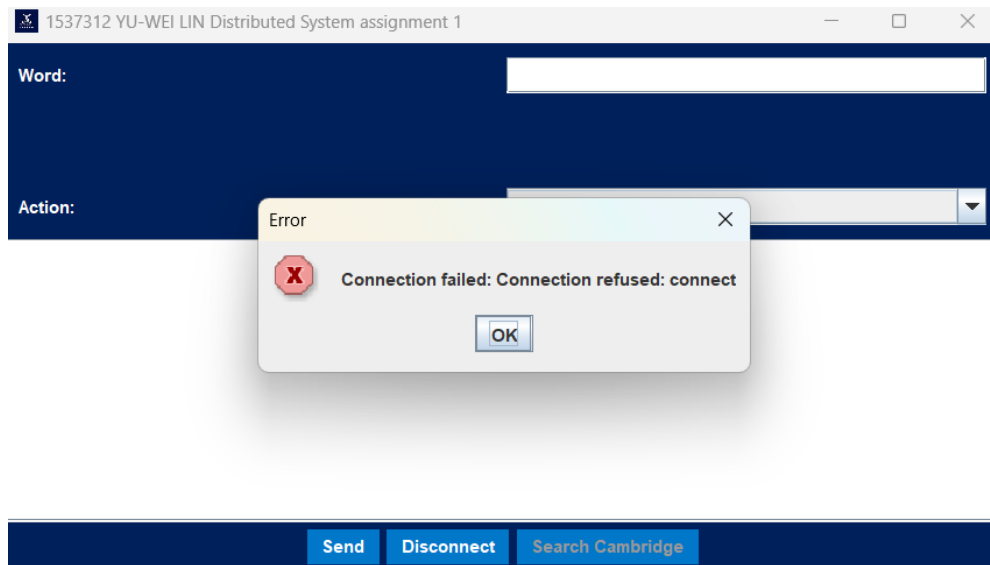
Search Cambridge

Send

Disconnect

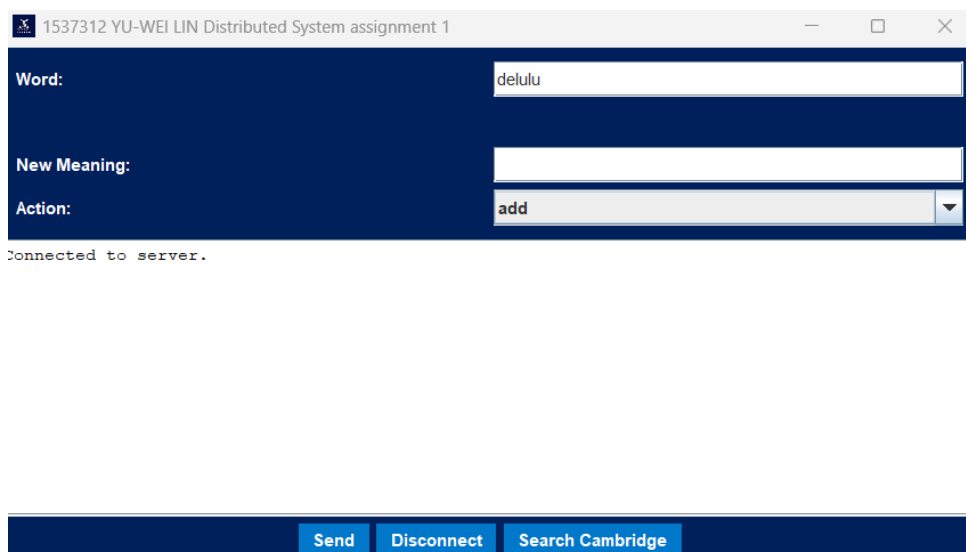
Search Cambridge

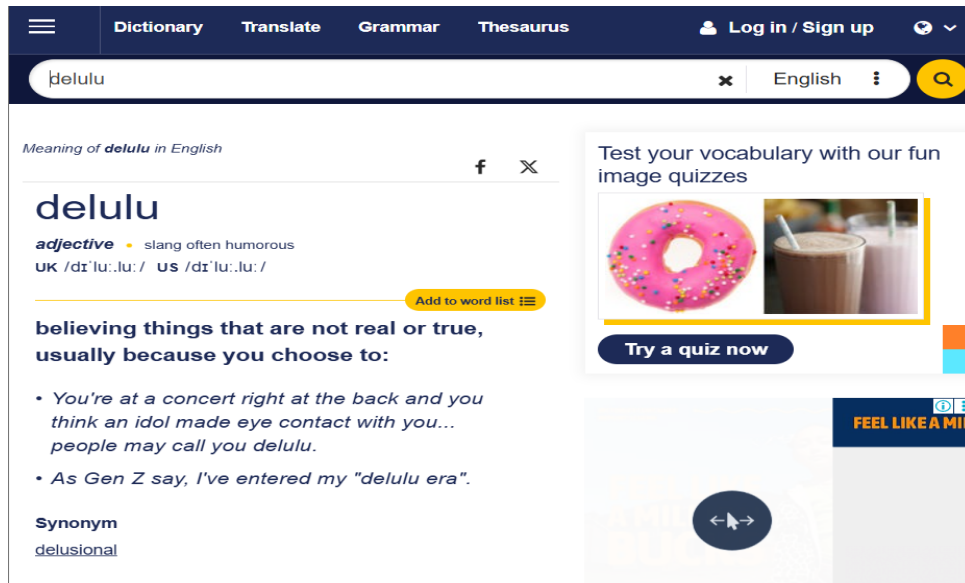
This approach that are displayed for a client are therefore relevant for the purpose of reducing unnecessary or erroneous requests besides improving the interaction with the client.



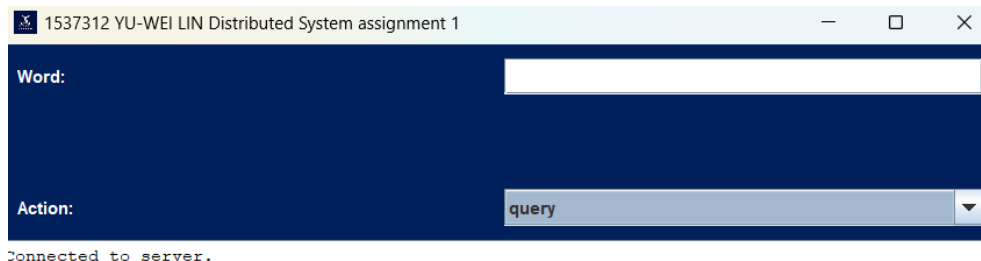
2. External website redirection

Whenever clients want to add a new word, add new meaning or update meaning and so on, if they don't understand the meaning of the word, they can click on the "Search Cambridge" button", which would redirect you to Cambridge Dictionary and hook in the word from the text field into the search query automatically to save client's time.





Additionally, this button is only enabled when the word field is not empty, preventing unnecessary redirections and ensuring intentional user interaction. This thoughtful integration enhances user experience by providing convenient, real-time reference support.



v. Conclusion

This project demonstrates Ingenious functional implementation and design with thoughtful features like input validation, error handling, dynamic UI feedback, and external dictionary integration. Future enhancements could focus on GUI aesthetics and scalability for high-load environments.