COMP90015 Distributed Systems
Assignment 1: Multi-threaded Dictionary Server

Wei Ge 1074198
The University of Melbourne

April 17, 2022

1 Introduction

This report will expound on a Java Dictionary application under the client-server architecture. By using the multi-threaded and thread-safe design, the server allows the concurrent clients to query, add, delete, update or append words. Moreover, the TCP socket provides a reliable interaction between server and client with transferring JSON objects. Afterwards, all the exceptions, such as the IO exception, Null Pointer exception are well handled through exhibiting user-friendly notification. In addition, the invalid request from the client will be detected, followed by guiding the user to send the legal request.

2 Components of the System

2.1 Overall

This program contains several Classes:

- Client Component
    - Client
    - Client UI
    - Request Handlers of Client
- Server Component
    - Server
    - Server UI
    - Threads to handle request from Clients
- Database Component
    - Database

The domain model is shown in Figure 1.

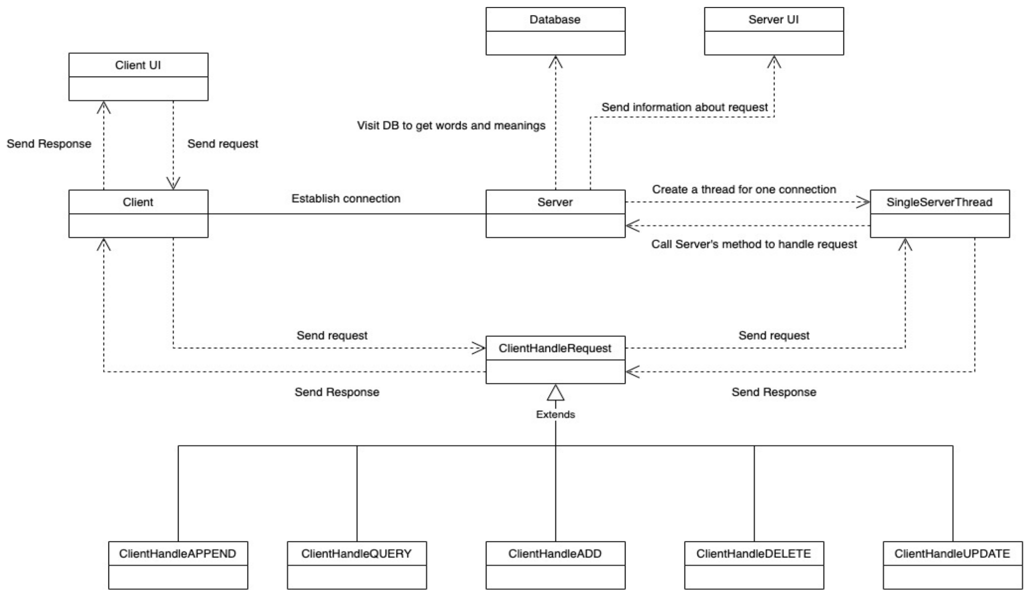


Figure 1: The System Domain Model

2.1 Database

The *Database* contains a hashmap to store the dictionary and the methods to manipulate the dictionary. The *Server* will call the API methods from *Database* to add, query, delete, update or append words to the dictionary.

2.3 Server

The *Server* creates *ServerUI* and *SingleServerThread*.

The *ServerUI* will be generated when the *Server* is invoked as a process. The information regarding client's request will be exhibited on *ServerUI* and the 'export dictionary file' function can be applied through the *ServerUI*.

Moreover, the *SingleServerThread* will be generated when the *Server* accepts a new connection request. Afterwards, the *SingleServerThread* handles all interaction between the *Server* and the corresponding client.

In addition, all methods concerning manipulation to dictionary, which be called by *SingleServerThread*, are provided by the *Server*. The *SingleServerThread* cannot feel the *Database*.

The domain model of *Server* components and *Database* is shown in Figure 2.

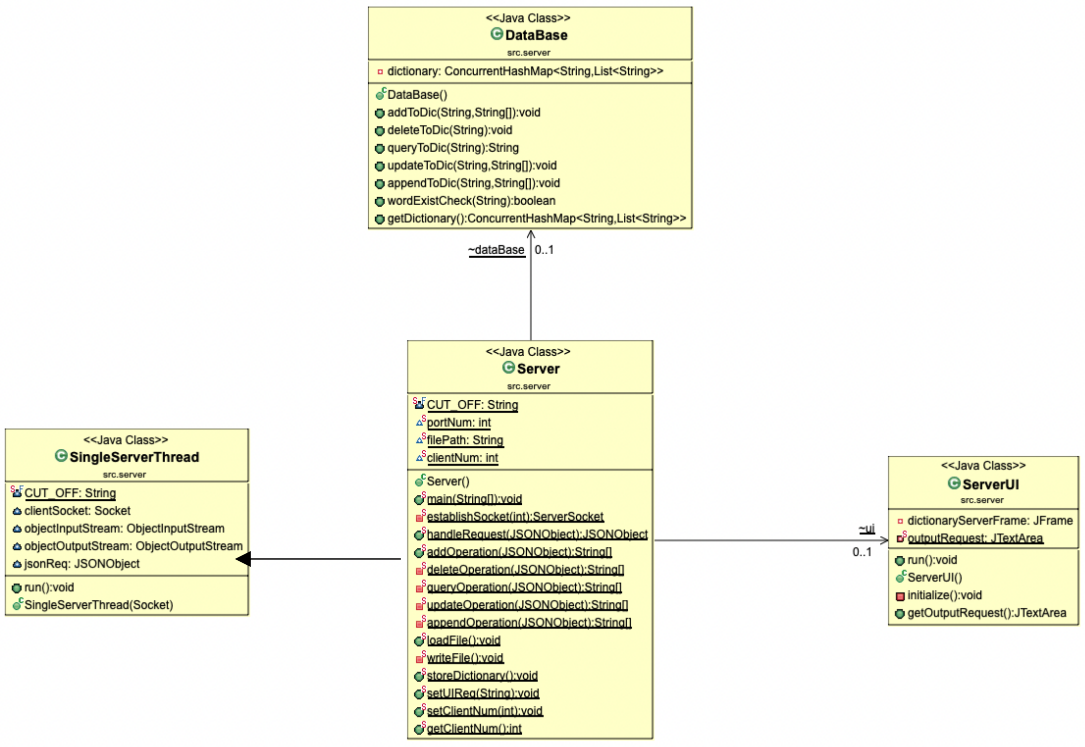

<<Java Class>>
DataBase
src.server
dictionary: ConcurrentHashMap<String,List<String>>
DataBase()
addToDic(String,String[]):void
deleteToDic(String):void
queryToDic(String):String
updateToDic(String,String[]):void
appendToDic(String,String[]):void
wordExistCheck(String):boolean
getDictionary():ConcurrentHashMap<String,List<String>>
~dataBase
0..1
<<Java Class>>
Server
src.server
CUT_OFF: String
portNum: int
filePath: String
clientNum: int
Server()
main(String[]):void
establishSocket(int):ServerSocket
handleRequest(JSONObject):JSONObject
addOperation(JSONObject):String[]
deleteOperation(JSONObject):String[]
queryOperation(JSONObject):String[]
updateOperation(JSONObject):String[]
appendOperation(JSONObject):String[]
loadFile():void
writeFile():void
storeDictionary():void
setUIReq(String):void
setClientNum(int):void
getClientNum():int
<<Java Class>>
SingleServerThread
src.server
CUT_OFF: String
clientSocket: Socket
objectInputStream: ObjectInputStream
objectOutputStream: ObjectOutputStream
jsonReq: JSONObject
run():void
SingleServerThread(Socket)
~ui
0..1
<<Java Class>>
ServerUI
src.server
dictionaryServerFrame: JFrame
outputRequest: JTextArea
run():void
ServerUI()
initialize():void
getOutputRequest():JTextArea

Figure 2: Server and Database Domain Model

2.4 Client

The *Client* creates *ClientUI* and *ClientHandleRequest.*

The *ClientUI* will be generated when the *Client* is invoked as a process. The user is going to send a request and get the expected output through this UI. Also, the *ClientUI* provides 'HELP' and 'CLEAR' function for users to get a better experience.

While the processing on the client-side is complicated, the design pattern is used to decouple the request handler. The *ClientHandleRequest* contains five children-class correspond to different requests.

The *Client* is responsible for connecting the server and providing API methods for *ClientUI* to send requests, followed by getting a response. Similarly, the *ClientUI* cannot feel the *ClientHandleRequest*.

The domain model of Client components is shown in Figure 3.

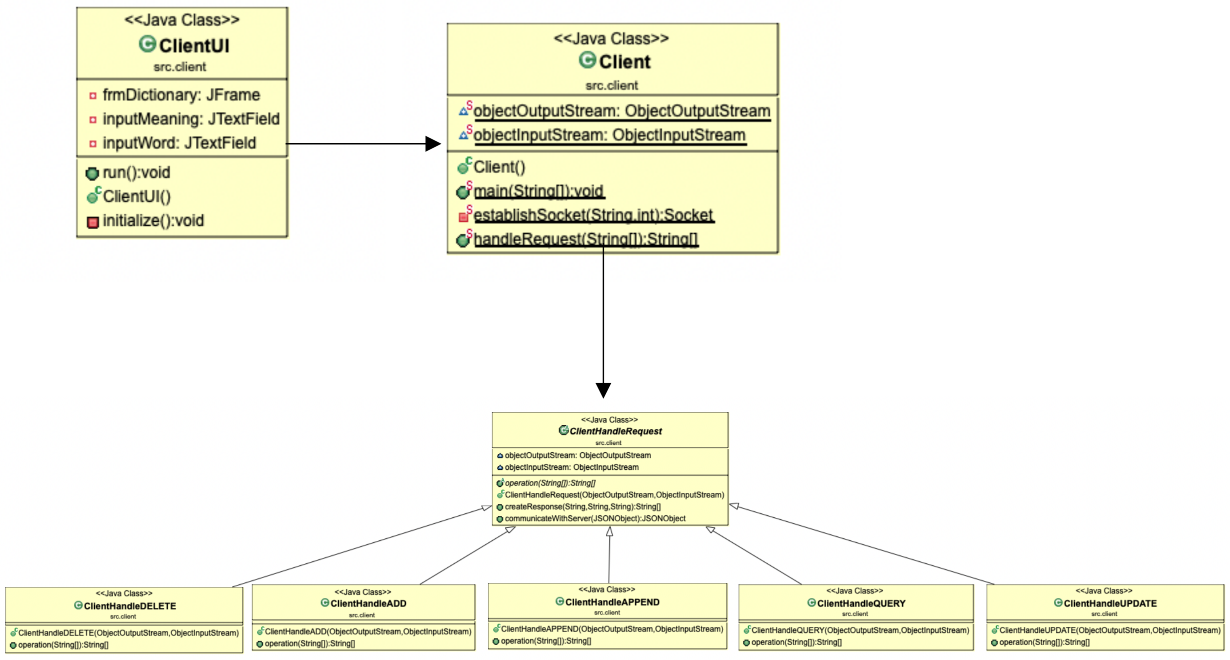

<<Java Class>>
ClientUI
src.client
frmDictionary: JFrame
inputMeaning: JTextField
inputWord: JTextField
run():void
ClientUI()
initialize():void
<<Java Class>>
Client
src.client
objectOutputStream: ObjectOutputStream
objectInputStream: ObjectInputStream
Client()
main(String[]):void
establishSocket(String,int):Socket
handleRequest(String[]):String[]
<<Java Class>>
ClientHandleRequest
src.client
objectOutputStream: ObjectOutputStream
objectInputStream: ObjectInputStream
operation(String[]):String[]
ClientHandleRequest(ObjectOutputStream,ObjectInputStream)
createResponse(String,String,String):String[]
communicateWithServer(JSONObject):JSONObject
<<Java Class>>
ClientHandleDELETE
src.client
ClientHandleDELETE(ObjectOutputStream,ObjectInputStream)
operation(String[]):String[]
<<Java Class>>
ClientHandleADD
src.client
ClientHandleADD(ObjectOutputStream,ObjectInputStream)
operation(String[]):String[]
<<Java Class>>
ClientHandleAPPEND
src.client
ClientHandleAPPEND(ObjectOutputStream,ObjectInputStream)
operation(String[]):String[]
<<Java Class>>
ClientHandleQUERY
src.client
ClientHandleQUERY(ObjectOutputStream,ObjectInputStream)
operation(String[]):String[]
<<Java Class>>
ClientHandleUPDATE
src.client
ClientHandleUPDATE(ObjectOutputStream,ObjectInputStream)
operation(String[]):String[]

Figure 3: Client Domain Model

3 Technology and design

The **TCP socket** is selected to establish the communication between client and server while it guards the communication's reliability. Afterwards, the **JSON** format is used for packaging the object to be transferred between the client and the server.

In order to transfer objects between two sides,
the **ObjectInputStream/ObjectOutputStream** is used in this project. This method uses **Serializable** to transform Object into Bit, which can be communicated between sides.

While original Java does not support JSON, the **JSONObject** package from Alibaba company is applied to generate and deal with JSON objects. To manage the dependencies, this project is built as a **maven** program.

Thread-per-connection model is built because this project is not large. The server will not face a host of clients' disconnecting and connecting thus the overhead of creating threads can be ignored if each connection creates threads.

To implement the concurrence, the Java Object **ConcurrentHashMap** is used. Different from 'Synchronized' locks methods, the ConcurrentHashMap locks the data.
The ConcurrentHashMap is constituted by segments that can be considered as sub-HashMap. Each data put into ConcurrentHashMap needs to be calculated hash value twice, the first one is to confirm which segments to stay, and the second one is to identify the segment's subregion it belongs to.
Instead of locking the whole ConcurrentHashMap for thread-unsafe operation, only the target segment will be locked. Therefore, both the impact on other segments' data can be minimized, and the concurrence can be maintained. If the 'Synchronized' is used here, all operations to any data will influence each other (one lock will block all other operations).

Furthermore, **static** will be applied to *Server* and *Client* classes to provide their API. Although static objects or static methods can share data between different Java Objects, the sharing only works in the same process's method area in JVM. While the clients and servers here are all independent processes, the threat of static can be ignored.

4 Process

4.1 Provides concurrence

The server is implemented by Thread-per-connection. While a new connection is established between the server and the client, a new thread will be generated to handle all requests from the client.

To guarantee the thread-unsafe operations, the ConcurrentHashMap is created.

4.2 Handle request

The sequence diagram, which exhibits how a request is handled, is shown in Figure 4.

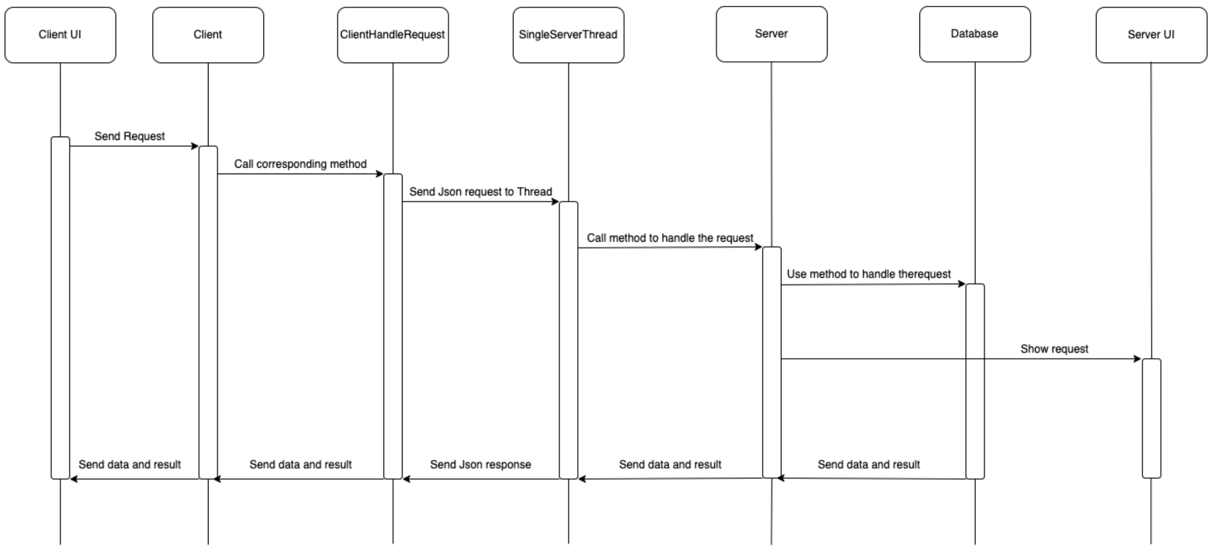


Figure 4: Sequence Diagram

While the project is related to complex information transfer, the information's structure is crucial. There are two main Java Objects used for the interactions. The first one is the JSON Object, which is used for the communication between the server and the client. The second Object is the Java String Array, which is used to transfer information inside the client-side.

All request contains two or three elements(entries):

1. The type of requests such as ADD, QUERY.
2. The word.
3. The meanings of the word if the request needs.

All response contains three elements(entries):

1. The operation is 'SUCC' or 'FAIL'.
2. The output for the client UI's content.
3. The output for the client UI's terminal.

4.2 Handle exception

While the application has users, invalid input is inevitable. This Dictionary application has a complete set of tests to ensure the program is running properly. The invalid operation and reaction are shown in Table 1.

Invalid Operation	Reply
Query, Update or Append a non-existed word. (Client Side)	Show in terminal: The word does not exist. Recommend the user to Add it.
Delete a non-existed word. (Client Side)	Show in terminal: The word does not exist.
Add an existed word. (Client Side)	Show in terminal: The word already exists. Recommend the user to query it.
Add, Update or Append without inputting meaning(s). (Client Side)	Show in terminal: Tell the user to input meanings
Operation without inputting word. (Client Side)	Show in terminal: Tell the user to input a word
Invalid port address or port number. (Client Side)	Show in terminal: Cannot connect the server. Tell the user to check several things.
Invalid port number. (Server Side)	Show in terminal: Tell the user the port number is invalid.
Port number is out of boundary. (Server Side)	Show in terminal: Tell the user the port number is invalid and exhibits the right boundary of port number.
Can not find the dictionary file. (Server Side)	Show in terminal: Tell the user the file is not found, and a new file will be generated.
The file can not be read because of the unsupported encoding or format. (Server Side)	Show in terminal: Tell the user the file can not be load.

Table 1: Invalid Operation and Reaction

While the 'missing dictionary file' error will not break the original dictionary file thus the system can continue to work. Other errors which may hurt the data such as 'can not read the dictionary file' will break the work until the user fix it.

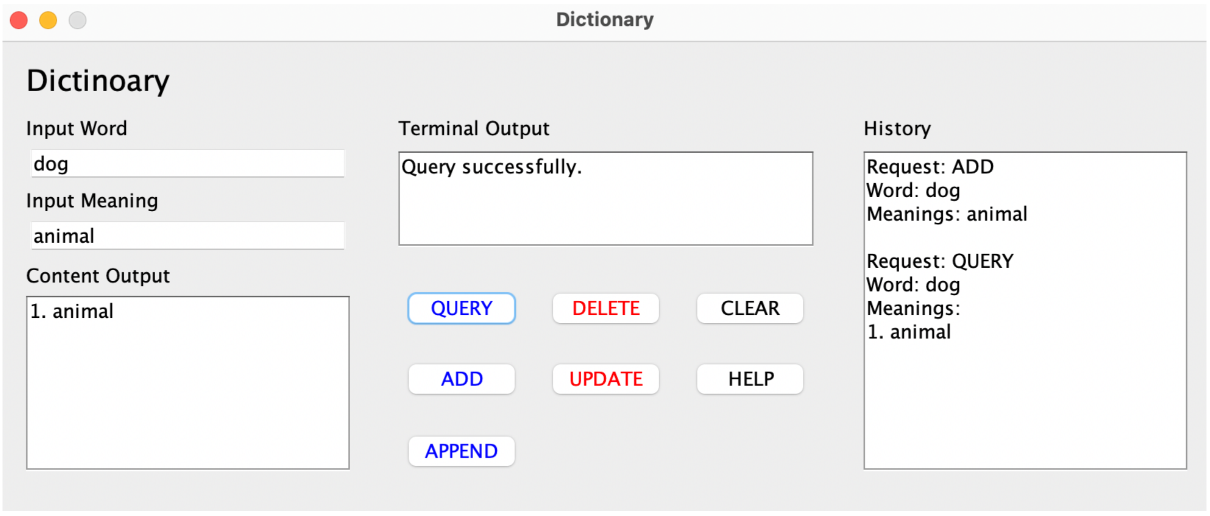
Moreover, the system error also occurs while this program is client-server. The system error and related reaction is shown in Table 2.

Exception	Reaction
JSON Object can not be loaded.	Show in terminal: Tell the users the response can not be loaded. Let them contact the developer.
The connection was suddenly lost.	Show in terminal: Tell the users the connection is failed, let them check the network or contact the server's administrator.

Table 2: System Exception and Reaction

4.4 The UIs

The demo of Client UI is shown in Figure 5.



```
dog
animal
1. animal
```

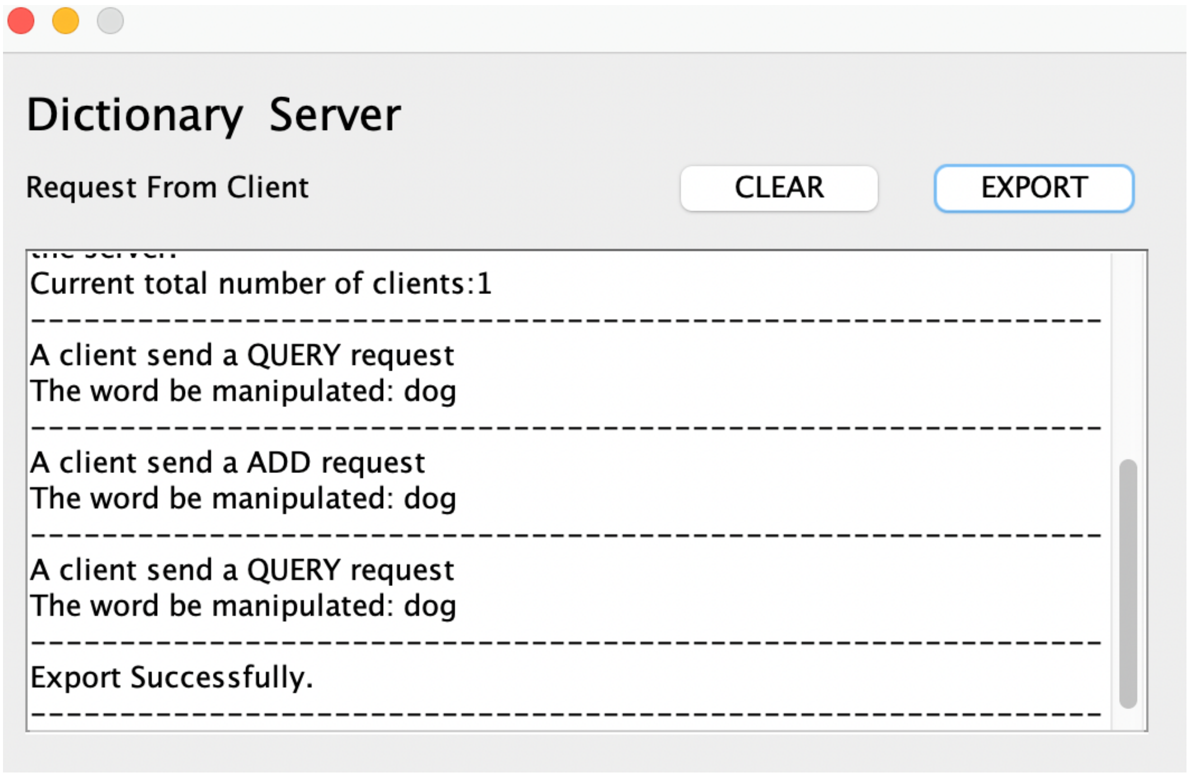
```
Query successfully.
```

```
Request: ADD
Word: dog
Meanings: animal

Request: QUERY
Word: dog
Meanings:
1. animal
```

Figure 5: Client UI demo

The demo of Server UI is shown in Figure 6.



```
Current total number of clients:1
---------------------------------------------------------------
A client send a QUERY request
The word be manipulated: dog
---------------------------------------------------------------
A client send a ADD request
The word be manipulated: dog
---------------------------------------------------------------
A client send a QUERY request
The word be manipulated: dog
---------------------------------------------------------------
Export Successfully.
---------------------------------------------------------------
```

Figure 6: Server UI demo

5 Analysis

The advantages of selecting Thread-per-connection, ConcurrentHashMap and TCP protocol have been pointed in the Technology and design part. In addition, this system provides a proper interaction with users on detailed notifications of errors. All possible operation errors and system errors are recorded. Moreover, there is a backup function for the client, and bad operations can be reversed.

About the coding, the system is separated into three parts: The client, the server and the database. Different types of functions are grouped clearly. Afterwards, if the system is low coupling, further extensions will be easily made. Furthermore, all code and methods are well commented.

However, there is still a lack of this system. Although Thread-per-connection can hold this project's low number of interactions, the extension for further implementation is not outstanding. In order to serve large amounts of clients, the Thread-pool is suitable to be deployed because it can avoid the overhead of creating and destroying.

6. Creativity work

6.1 Client Side

- The Client's Function is improved. The user can append meaning(s) to existed word.
- The Client UI will back up all operations and data in the history area. The data-hurting operations such as Delete and Update can be withdrawn.
- The Client UI provides a good document to guide the user, and the CLEAR button is positioned to clear the window.

6.2 Server Side

- The Server UI is implemented. The Server administrator can check the requests and situations of Clients.

7 Excellence elements

7.1 Implementation

- Instead of meeting the minimum requirements, more functions are implemented after deliberate design.
- The user-friendly notifications of errors are provided to show the pellucid error and recommendation. Moreover, the user can check the guide.

7.2. Report

- Tables and Figures are shown, including System’s structure, Domain Model and Sequence Diagram.
- All advantages of the design choices and the further improvement that can be implemented are all discussed in the ‘Technology and design’ and ‘Analysis’ parts.