Multi-threaded Dictionary Server

Assignment 01

Student Name	**Jiahao Shen**
Student ID	**1381187**
Date	**04/05/2023**
Subject	**COMP90015 Distributed Systems**

Table of Contents

The problem context in which the assignment has been given.

This Java project aims to develop a scalable and reliable application for a multi-user shared dictionary by employing a distributed system and a multi-threaded server. The objective is to efficiently process concurrent requests, accommodating a growing number of users and connected devices.

The architecture should use a client-server model, using sockets for network communication and multi-threading to manage concurrent requests. The client will interact with the dictionary through a graphical user interface (GUI), which facilitates querying word meanings, as well as adding, deleting, and updating entries. the GUI can be developed using established tools and libraries, such as Swing or JavaFX.

To ensure dependable communication, the project will incorporate exception handling mechanisms and effectively manage input-output operations. This approach will enhance the overall performance and user experience, while maintaining system reliability.

A brief description of the components of the system.

The system comprises three main components: 1) Network Communication: TCP Client-Server Architecture, 2) Dictionary Service: CRUD Operations and Data Management, and 3) User Interface: JavaFX-based GUI with Custom JetBrains Theme. These components interact seamlessly to deliver a robust and user-friendly dictionary service to the end-users.

Network Communication: TCP Client-Server Architecture

Server

The server-side network communication in the dictionary application is a collaboration of three main classes (DictionaryServerApp, TCPInteractiveServer, ClientHandler), which handle the initialization and management of the server, listening for and accepting incoming client connections, and processing client requests for dictionary operations. The server efficiently manages multiple client connections using a custom thread pool, ensuring scalability and performance in a multi-threaded environment.

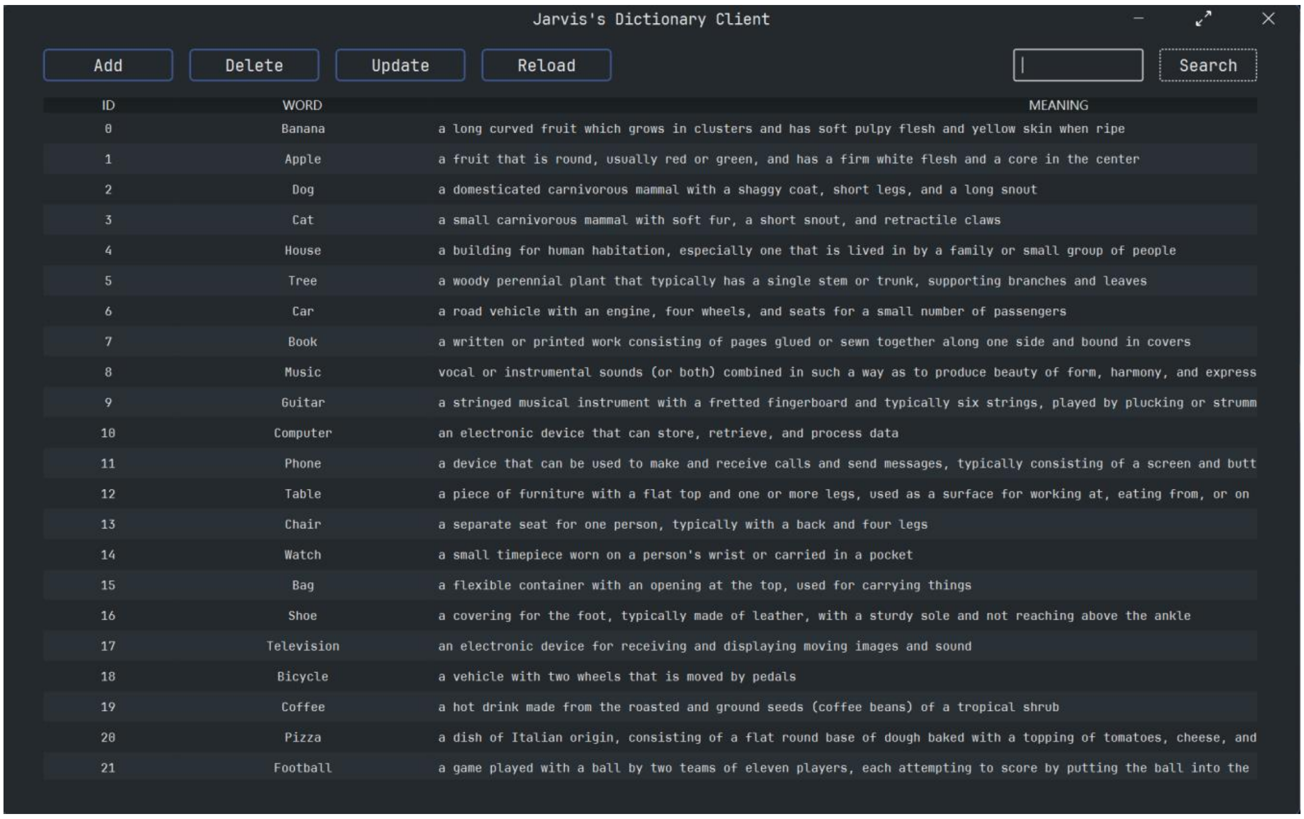
Client

The network communication on the client-side in the dictionary application is a cooperative effort among three primary classes (DictionaryClientApp, TCPInteractiveClient, MySence). These classes are responsible for the initialization and administration of the client, creating and sustaining a TCP connection to the server, and providing a graphical user interface (GUI) for user interaction. The client effectively communicates with the server by utilizing the TCPInteractiveClient class for sending commands and obtaining responses, guaranteeing a seamless and engaging user experience.

Dictionary Service: CRUD Operations and Data Management

The server-side DictionaryService class is mainly responsible for the core functionality of the dictionary application. It includes loading dictionary data, getting dictionaries, adding, deleting, updating words, and searching words. The dictionary data is stored in JSON format and is represented by a static variable named dictionaryArray. In addition, this class maintains a maxId variable for assigning unique IDs to newly added words. all methods are synchronous methods to ensure consistency of dictionary data in a multi-threaded environment.

User Interface: JavaFX-based GUI with Custom JetBrains Theme



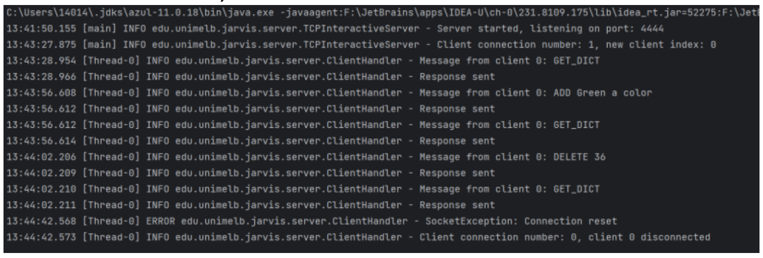
ID	WORD	MEANING
0	Banana	a long curved fruit which grows in clusters and has soft pulpy flesh and yellow skin when ripe
1	Apple	a fruit that is round, usually red or green, and has a firm white flesh and a core in the center
2	Dog	a domesticated carnivorous mammal with a shaggy coat, short legs, and a long snout
3	Cat	a small carnivorous mammal with soft fur, a short snout, and retractile claws
4	House	a building for human habitation, especially one that is lived in by a family or small group of people
5	Tree	a woody perennial plant that typically has a single stem or trunk, supporting branches and leaves
6	Car	a road vehicle with an engine, four wheels, and seats for a small number of passengers
7	Book	a written or printed work consisting of pages glued or sewn together along one side and bound in covers
8	Music	vocal or instrumental sounds (or both) combined in such a way as to produce beauty of form, harmony, and express
9	Guitar	a stringed musical instrument with a fretted fingerboard and typically six strings, played by plucking or strumm
10	Computer	an electronic device that can store, retrieve, and process data
11	Phone	a device that can be used to make and receive calls and send messages, typically consisting of a screen and butt
12	Table	a piece of furniture with a flat top and one or more legs, used as a surface for working at, eating from, or on
13	Chair	a separate seat for one person, typically with a back and four legs
14	Watch	a small timepiece worn on a person's wrist or carried in a pocket
15	Bag	a flexible container with an opening at the top, used for carrying things
16	Shoe	a covering for the foot, typically made of leather, with a sturdy sole and not reaching above the ankle
17	Television	an electronic device for receiving and displaying moving images and sound
18	Bicycle	a vehicle with two wheels that is moved by pedals
19	Coffee	a hot drink made from the roasted and ground seeds (coffee beans) of a tropical shrub
20	Pizza	a dish of Italian origin, consisting of a flat round base of dough baked with a topping of tomatoes, cheese, and
21	Football	a game played with a ball by two teams of eleven players, each attempting to score by putting the ball into the

The system's user interface is built using JavaFX, which provides a modern and customizable platform for creating rich and interactive desktop applications. The GUI is designed to be intuitive and user-friendly, allowing users to easily interact with the dictionary service.

The main window consists of a search bar for querying words, buttons for adding, updating, and deleting words, and a table for displaying the dictionary. In addition, a window pops up to inform the user of specific information in case of a failed dictionary operation or a system problem. For example, if a user searches for a word that does not exist in the dictionary, a window will pop up to let the user know that the word is not found. The GUI also includes a custom dark theme, which improves the overall appearance and enhances user experience. The theme uses the JetBrains Mono font and features a dark background color, light text color, and a white border.

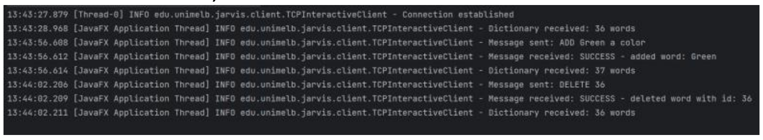
Others

I have imported the slf4j-api library in this project for printing logs, such as messages communicated between the client and the server.
On the server side,



```
C:\Users\14014\.jdks\azul-11.0.18\bin\java.exe -javaagent:F:\JetBrains\apps\IDEA-U\ch-0\231.8109.175\lib\idea_rt.jar=52275:F:\Jet
13:41:50.155 [main] INFO edu.unimelb.jarvis.server.TCPInteractiveServer - Server started, listening on port: 4444
13:43:27.875 [main] INFO edu.unimelb.jarvis.server.TCPInteractiveServer - Client connection number: 1, new client index: 0
13:43:28.954 [Thread-0] INFO edu.unimelb.jarvis.server.ClientHandler - Message from client 0: GET_DICT
13:43:28.966 [Thread-0] INFO edu.unimelb.jarvis.server.ClientHandler - Response sent
13:43:56.608 [Thread-0] INFO edu.unimelb.jarvis.server.ClientHandler - Message from client 0: ADD Green a color
13:43:56.612 [Thread-0] INFO edu.unimelb.jarvis.server.ClientHandler - Response sent
13:43:56.612 [Thread-0] INFO edu.unimelb.jarvis.server.ClientHandler - Message from client 0: GET_DICT
13:43:56.614 [Thread-0] INFO edu.unimelb.jarvis.server.ClientHandler - Response sent
13:44:02.206 [Thread-0] INFO edu.unimelb.jarvis.server.ClientHandler - Message from client 0: DELETE 36
13:44:02.209 [Thread-0] INFO edu.unimelb.jarvis.server.ClientHandler - Response sent
13:44:02.210 [Thread-0] INFO edu.unimelb.jarvis.server.ClientHandler - Message from client 0: GET_DICT
13:44:02.211 [Thread-0] INFO edu.unimelb.jarvis.server.ClientHandler - Response sent
13:44:42.568 [Thread-0] ERROR edu.unimelb.jarvis.server.ClientHandler - SocketException: Connection reset
13:44:42.573 [Thread-0] INFO edu.unimelb.jarvis.server.ClientHandler - Client connection number: 0, client 0 disconnected
```

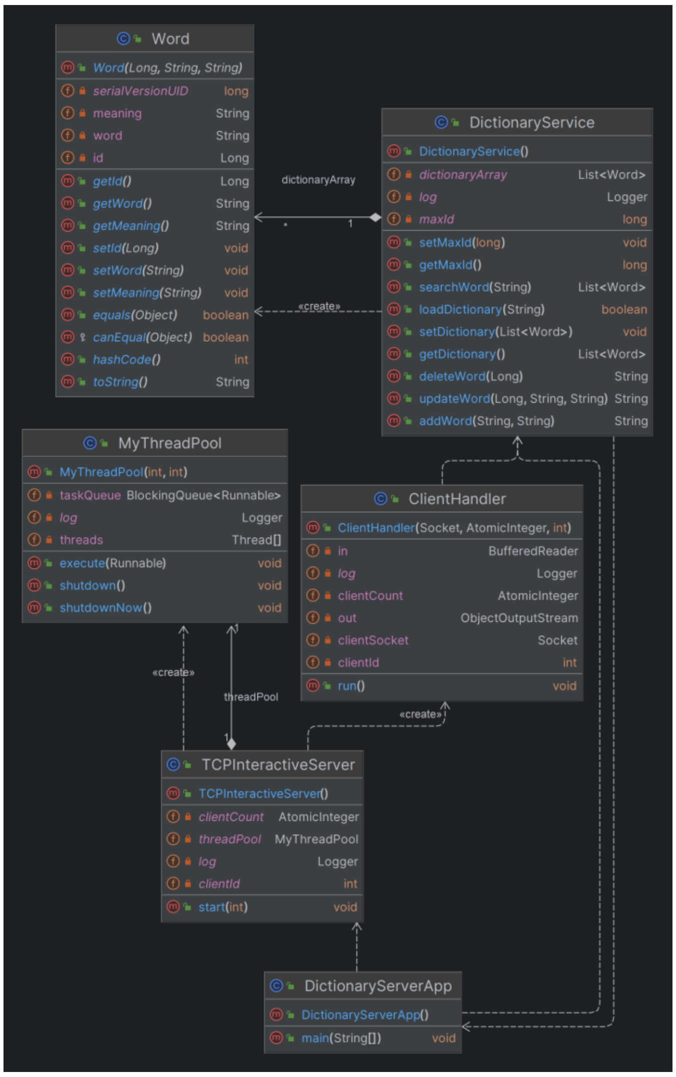
On the client side,

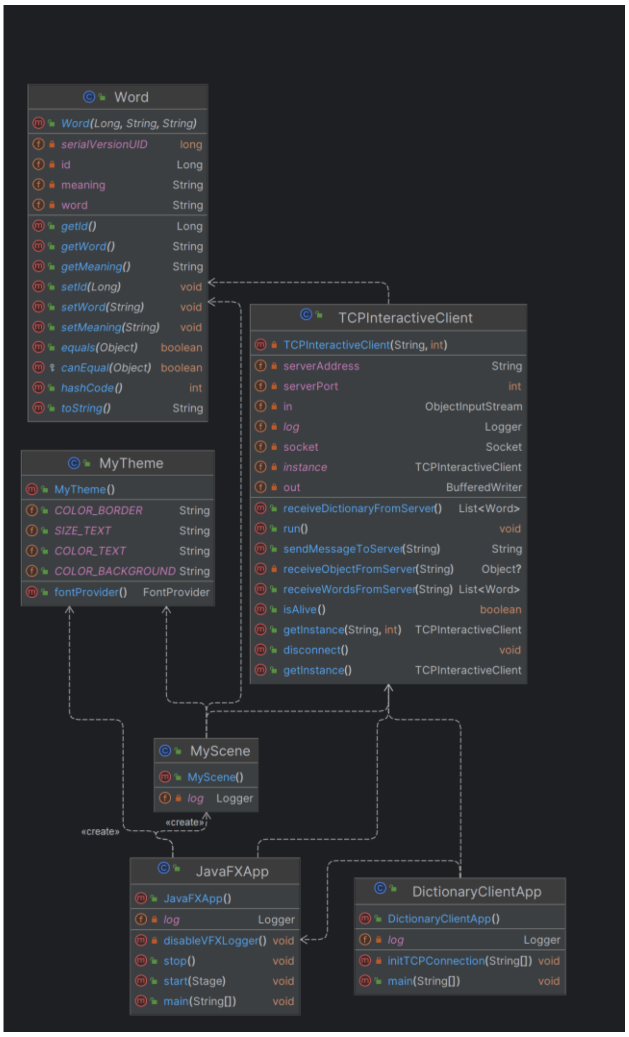


```
13:43:27.879 [Thread-0] INFO edu.unimelb.jarvis.client.TCPInteractiveClient - Connection established
13:43:28.968 [JavaFX Application Thread] INFO edu.unimelb.jarvis.client.TCPInteractiveClient - Dictionary received: 36 words
13:43:56.608 [JavaFX Application Thread] INFO edu.unimelb.jarvis.client.TCPInteractiveClient - Message sent: ADD Green a color
13:43:56.612 [JavaFX Application Thread] INFO edu.unimelb.jarvis.client.TCPInteractiveClient - Message received: SUCCESS - added word: Green
13:43:56.614 [JavaFX Application Thread] INFO edu.unimelb.jarvis.client.TCPInteractiveClient - Dictionary received: 37 words
13:44:02.206 [JavaFX Application Thread] INFO edu.unimelb.jarvis.client.TCPInteractiveClient - Message sent: DELETE 36
13:44:02.209 [JavaFX Application Thread] INFO edu.unimelb.jarvis.client.TCPInteractiveClient - Message received: SUCCESS - deleted word with id: 36
13:44:02.211 [JavaFX Application Thread] INFO edu.unimelb.jarvis.client.TCPInteractiveClient - Dictionary received: 36 words
```

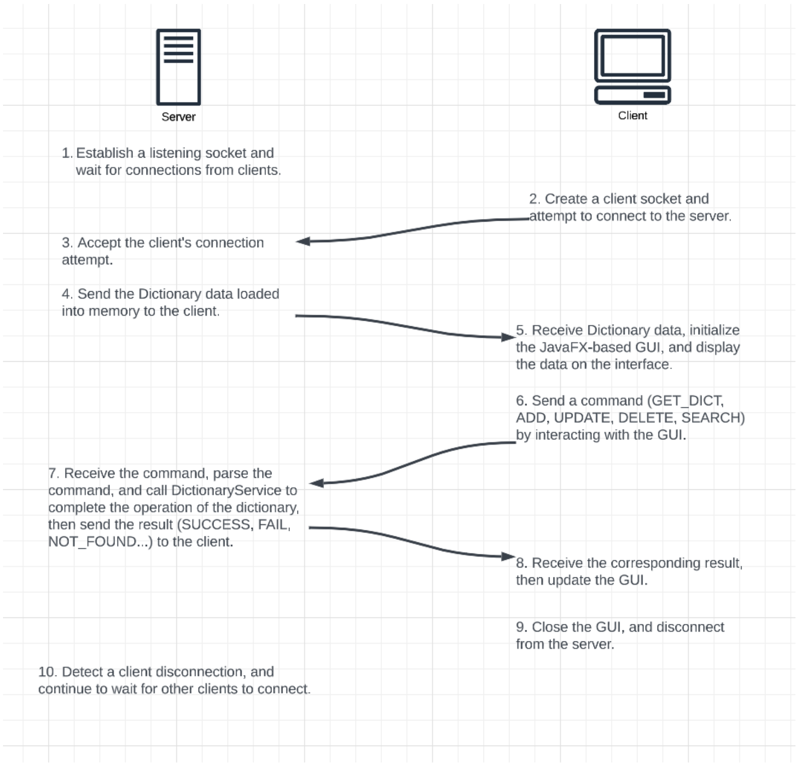
An overall class design and an interaction diagram.

Server & Client UML class diagram





Interaction diagram



1. Establish a listening socket and wait for connections from clients.

2. Create a client socket and attempt to connect to the server.

3. Accept the client's connection attempt.

4. Send the Dictionary data loaded into memory to the client.

5. Receive Dictionary data, initialize the JavaFX-based GUI, and display the data on the interface.

6. Send a command (GET_DICT, ADD, UPDATE, DELETE, SEARCH) by interacting with the GUI.

7. Receive the command, parse the command, and call DictionaryService to complete the operation of the dictionary, then send the result (SUCCESS, FAIL, NOT_FOUND...) to the client.

8. Receive the corresponding result, then update the GUI.

9. Close the GUI, and disconnect from the server.

10. Detect a client disconnection, and continue to wait for other clients to connect.

In this project, I have adopted a straightforward and intuitive class design without incorporating complex design patterns. Below are the key classes and their interactions:

On the server side,

DictionaryServerApp: This class serves as the entry point for the server application. It initializes the server by loading the dictionary data from a JSON file using the DictionaryService class, and then starts the TCP server on the specified port. Command line arguments can be used to configure the server's listening port and dictionary file path. If not provided, default values are used.

TCPInteractiveServer: This class is responsible for creating and managing the server socket and handling incoming client connections. It listens on the specified port for incoming client connections and assigns a unique client index to each new connection. The TCPInteractiveServer class utilizes a custom thread pool (MyThreadPool) to manage the threads that handle client connections, ensuring efficient use of resources and scalability.

ClientHandler: This class implements the Runnable interface and is responsible for handling the communication between the server and a specific client. When a new client connection is accepted, a new instance of ClientHandler is created and executed in a separate thread. Each ClientHandler instance is provided with a unique client index to keep track of connected clients.

The ClientHandler class manages the input and output streams for the client connection, reading incoming messages from the client and processing them accordingly. It interprets client messages as commands for CRUD operations (e.g., ADD, DELETE, UPDATE, SEARCH), and uses the DictionaryService class to perform the requested operations on the dictionary data. The results of these operations are then sent back to the client as serialized objects or UTF-encoded strings.

DictionaryService: The dictionary data is initially stored in a JSON file and is loaded into memory when the server starts. The json-simple library is utilized to parse and manage the JSON data. The DictionaryService class has several methods to handle CRUD operations while addressing multithreading and concurrency concerns:

Loading and managing the dictionary: The loadDictionary method is responsible for loading the dictionary data from the JSON file into memory. The getDictionary, setDictionary, getMaxId, and setMaxId synchronized methods are used to safely access and modify the dictionary data and maximum ID value, preventing potential issues related to concurrent access by multiple threads.

Adding a word: The addWord synchronized method adds a new word to the dictionary, ensuring that it does not already exist in the dictionary and that the new word has a unique ID. It also updates the maximum ID value accordingly.

Deleting a word: The deleteWord synchronized method removes a word from the dictionary based on its ID, updating the dictionary data accordingly.

Updating a word: The updateWord synchronized method modifies a word and its meaning in the dictionary based on the given ID, updating the dictionary data accordingly.

Searching for a word: The searchWord synchronized method finds words that contain the specified word as a substring, returning a list of matching words.

On the client side,

DictionaryClientApp: This class serves as the entry point for the client application. It initializes the TCP connection to the server using the initTCPConnection method, and then starts the JavaFX application. Command line arguments can be used to configure the server address and port. If not provided, default values are used.

TCPInteractiveClient: This class implements the Runnable interface and is responsible for establishing and maintaining the TCP connection to the server. The class follows the Singleton design pattern, ensuring that only one instance of the client is created. It manages the input and output streams for the connection, allowing the client to send messages to the server and receive responses. The TCPInteractiveClient class provides various methods for sending commands (e.g., ADD, DELETE, UPDATE, SEARCH) to the server and processing the received responses.

The run() method of the TCPInteractiveClient class is responsible for establishing the connection to the server and creating the input and output streams. Other utility methods, such as disconnect() and isAlive(), handle the disconnection and connection status checking processes.

JavaFXApp: This class is a JavaFX based application class, which is responsible for creating, configuring, and displaying the main window of the application. This class extends from JavaFX's Application class and overrides the start() method to set the window's title, size, and layout. It also uses a custom scene class called MyScene to build the user interface. In addition, when the application is closed, the stop() method of the JavaFXApp class calls the disconnect() method of TCPInteractiveClient to disconnect from the server.

MyScene: This class is responsible for creating the graphical user interface (GUI) of the client application using JavaFX. It interacts with the TCPInteractiveClient class to send user-generated commands to the server and receive responses. When a user performs an action, such as pressing a button, the MyScene class calls the appropriate method in the TCPInteractiveClient class to communicate with the server. After receiving the server's response, the MyScene class updates the GUI accordingly.

A critical analysis of the work done followed by the conclusions.

Throughout the development process, I successfully implemented the main goals and features of the project using the requested **Sockets** and **Threads**. However, after reflecting on the project, I have to admit that there are still many areas that are not considered and need subsequent improvement.

Strengths:

1. The dictionary application features a clean and easy-to-understand user interface with a JetBrains-style design, providing excellent user experience.
2. The application handles concurrency correctly, ensuring the stability and consistency of the dictionary data in multi-threaded scenarios.
3. The use of a more reliable TCP connection instead of UDP ensures secure and accurate data transmission between the client and server.
4. Proper error handling has been implemented, preventing crashes in both the client and server applications.

Areas for improvement:

1. Real-time rendering of the user interface has not been implemented. Ideally, when client A deletes a word, the server should broadcast an update message to all connected clients after removing the word from the shared dictionary. This way, all clients can receive notifications, and then update their GUIs accordingly.
2. The communication protocol between the client and server needs improvement. Currently, instructions are separated by spaces, e.g., "ADD newWord newMeaning", which can cause parsing issues if the new word or meaning contains spaces. A possible solution is to use commands in JSON format to make the communication between the client and server side more reliable and accurate.

In conclusion, the dictionary application has successfully met its primary objectives and delivers a functional and user-friendly experience. The critical analysis reveals that while the project has several strengths, improvements can be made in real-time user interface rendering and communication protocol between the client and server. By addressing these areas for improvement, the application can be further enhanced, providing an even more seamless and robust user experience.

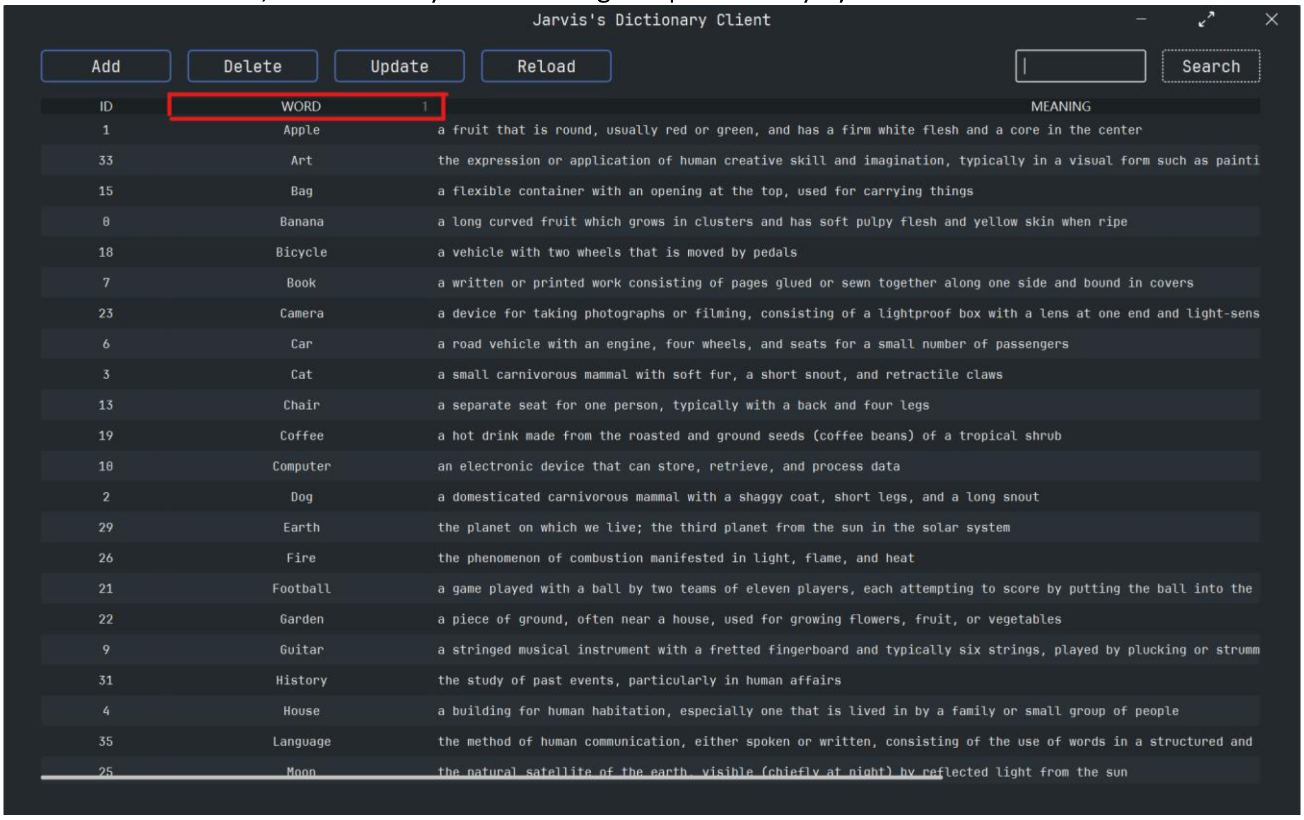
Creativity.

On the server side,

1. I have implemented my own thread pool. By controlling the number of threads and the task queue size, the custom thread pool helps prevent excessive resource consumption, leading to better overall performance. The use of a BlockingQueue for task scheduling ensures that tasks are executed in order and resources are utilized efficiently. Furthermore, the implementation provides appropriate handling of InterruptedExceptions, facilitating proper thread termination and resource release.
2. I maintain two self-incrementing variables in the TCPInteractiveServer class: clientCount and clientId. The clientCount variable is used to keep track of the current number of clients connected to the server, while clientId is assigned to a client upon a successful connection. By utilizing these two variables, I can better understand the operations of different clients when logging. Additionally, if further development of the server's GUI interface is required in the future, the server side can display the currently connected clients and their actions in real-time.
3. I maintain an auto-incrementing maxId variable in the TCPInteractiveServer class, which is used to assign unique and sequential IDs to each word. By using this ID, if there is a need to improve the efficiency of searching for words in the future, binary search can be employed.

On the client side,

1. I used the vfx library to develop the GUI. vfx is an open source component library developed using JavaFX, through which I was able to develop the GUI for this project more quickly.
2. The table supports sorting based on selected columns by clicking on the column names. As shown in the screenshot below, the dictionary data is arranged alphabetically by word.



ID	WORD 1	MEANING
1	Apple	a fruit that is round, usually red or green, and has a firm white flesh and a core in the center
33	Art	the expression or application of human creative skill and imagination, typically in a visual form such as painti
15	Bag	a flexible container with an opening at the top, used for carrying things
8	Banana	a long curved fruit which grows in clusters and has soft pulpy flesh and yellow skin when ripe
18	Bicycle	a vehicle with two wheels that is moved by pedals
7	Book	a written or printed work consisting of pages glued or sewn together along one side and bound in covers
23	Camera	a device for taking photographs or filming, consisting of a lightproof box with a lens at one end and light-sens
6	Car	a road vehicle with an engine, four wheels, and seats for a small number of passengers
3	Cat	a small carnivorous mammal with soft fur, a short snout, and retractile claws
13	Chair	a separate seat for one person, typically with a back and four legs
19	Coffee	a hot drink made from the roasted and ground seeds (coffee beans) of a tropical shrub
10	Computer	an electronic device that can store, retrieve, and process data
2	Dog	a domesticated carnivorous mammal with a shaggy coat, short legs, and a long snout
29	Earth	the planet on which we live; the third planet from the sun in the solar system
26	Fire	the phenomenon of combustion manifested in light, flame, and heat
21	Football	a game played with a ball by two teams of eleven players, each attempting to score by putting the ball into the
22	Garden	a piece of ground, often near a house, used for growing flowers, fruit, or vegetables
9	Guitar	a stringed musical instrument with a fretted fingerboard and typically six strings, played by plucking or strumm
31	History	the study of past events, particularly in human affairs
4	House	a building for human habitation, especially one that is lived in by a family or small group of people
35	Language	the method of human communication, either spoken or written, consisting of the use of words in a structured and
25	Moon	the natural satellite of the earth, visible (chiefly at night) by reflected light from the sun