Université de Liège
3rd year bachelor in engineering
Academic year 2023-2024

Report project 1

Introduction to computer networking

INFO0010

- DONATO Sophia (s212861)
- INNAURATO Arnaud (s211234)

Liège

December 2023

1 Software architecture

The implementation of our project is divided into several independent Java classes. Here is a list of all these classes and their main utilities.

- **WordleServer**: This class initializes the Wordle server and handles all the connections (threads) using a `FixedThreadPool`.
- **WordleConnection extends Thread**: This class is instantiated when a new connection is accepted on the server Socket. It responds to user queries, handles exceptions, and manages HTTP error codes.
- **PageHandler**: This class contains the HTML, CSS, and JS code. It provides the entire interface for the client to play and send queries to the server. Can deal with gzip compression aswell
- **WordleGameState**: This class encapsulates all the information necessary to fully describe a game state of Wordle, including the cookie, the hidden word, the number of trials, and a list of previous trials along with their respective replies and the game status. It is responsible for accurately responding to CHEAT and TRY queries.
- **CookiesStorage**: This class manages the association between each game state of a specific game and its corresponding cookie.
- **CookieState** : This class represents the token utilized within the *CookiesStorage* class. It stores the game state, its associated cookie, and the timestamp indicating the creation time of the cookie.
- **CookiesNotInListException extends Exception**: The exception is triggered when the browser provides an unknown or unrecognized cookie.

In our implementation, there's an enumeration called **GameStatus** with values: RUNNING, TOCLOSE, or LASTQUERY. These values determine the appropriate timing for cookie deletion.

2 Multi-thread coordination

Each time an exchange happens between the client and the server, a new thread is created in *WordleServer* and the request is managed by the *WordleConnection* class that extends *Thread*.However, if too many threads are active at the same time, this could cause the server to be unable to treat them all and so, to fail.

To avoid this problem, we create a thread pool with the help of the `newFixedThreadPool(int maxThreads)` method of the *java.util.concurrent.Executors* library. This thread pool allows the server to deal with only several tasks at a time, with a number of thread fixed by the user, while the other tasks are waiting for a thread to be free.

Since the list of cookies is a common object between all the threads, we have to `synchronized` several methods to ensure the proper functioning of our thread pool. This is particularly the case for all methods (and constructors) of the *CookieStorage* class that use the cookie list.

We can also remark the use of `synchronized` in two other cases :
→ in *WordleServer*, when creating a new connection, we synchronize the `threadIdentifier` argument so that the number of threads increment correctly.
→ in *WordleGameState*, we synchronize the list of guesses (and their colors) to avoid confusion between them.

3 Limits

The first problem lies in the fact that exchanges between server and client aren't encrypted, so data can be read by anyone. A good solution would be to implement an HTTPs server.

Concerning protection of user's data, we can also note that a malicious hacker could, if they really wanted to, find an active cookie and access a other people game. A password and user's name would be interesting in this case, so that only you can access your game.

We can also remark that the number of threads for the thread pool is fixed when compiling the program. So if a user decides to set this number really high, we could still have crashes and failures and the thread pool effect would be cancelled. A better idea would be to fix a limit on the user's input so that the number of threads simultaneously active stays low.

4 Possible improvements

We already have discussed about improvements concerning robustness of the server in the precedent point. Let's focus on the user side.
Here is a non-exhaustive list of the possible improvements that we can make for the game to be more user friendly :

- Coloring the letters on the screen keyboard to show if they are in the secret word or not.
- Adding a quit button that allows the user to start a new game without having to finishing the current one or to actively change cookie.
- Adding a parameter button to eventually change the theme as it can be done in other applications : darker mode for playing at night, other colors for color blind people, ...
- Adding some kind of button that shows the rules of the game.
- when JavaScript is disabled, adding a 'Cheating' button.
- ...

We have think that we my also ameliorate the following point to have an even more efficient program :

- Sending the HTML page 'line by line' to the server in order to reduce processing time (instead of sending a huge string containing all of the page content).