Introduction to Computer Networking
Project 2

Université de Liège
10 December 2023
Gilles Ooms, s192136
Martin Dengis, s193348

1 Software Architecture

1. **WordleServer.java** : The WordleServer class represents the server component of the Wordle game. It handles incoming client connections, manages game sessions, and provides methods to access and manipulate session data.
2. **SessionData.java** : The SessionData class represents the data associated with a game session. It stores information such as the number of attempts, last activity time, game status, secret word, and game state.
3. **HttpHandler.java** : The HttpHandler class is responsible for handling HTTP requests from clients.It implements the Runnable interface to allow for concurrent handling of requests.
4. **HTML.java** : Generates dynamic HTML content for the Wordle game. It includes methods to generate the entire page, the Wordle board, keyboard layout, styles, and other necessary components. Some methods are tailored to handle the users' desire about enabling JavaScript or not.
5. **HttpMethod.java** : Represents the HTTP methods supported by the application and provides a method to check if a given HTTP method is allowed.
6. **ImageEncoder.java** : Provides functionality to encode images to Base64 format.

2 Multi-thread coordination

We structure our multi-thread coordination inside the ***WordleServer*** and ***HttpHandler*** classes.

Thread Pool Management in *WordleServer.java*

— Creation and Configuration of Thread Pool : The application employs an ExecutorService to create a fixed-size thread pool. The size of the thread pool is configurable and is determined by a command-line argument. This design choice is pivotal in managing the threads efficiently, allowing for the reuse of threads across multiple requests and mitigating the overhead associated with the continuous creation and destruction of threads.

— Client Request Handling : Upon a new client connection, the server accepts the connection (serverSocket.accept()) and allocates an instance of HttpHandler for processing the request. This instance is subsequently executed within the thread pool (executorService.execute(httpHandler)), ensuring each request is processed in an isolated thread. This method facilitates the server's capability to handle multiple requests simultaneously, promoting non-blocking operations and efficient request processing.

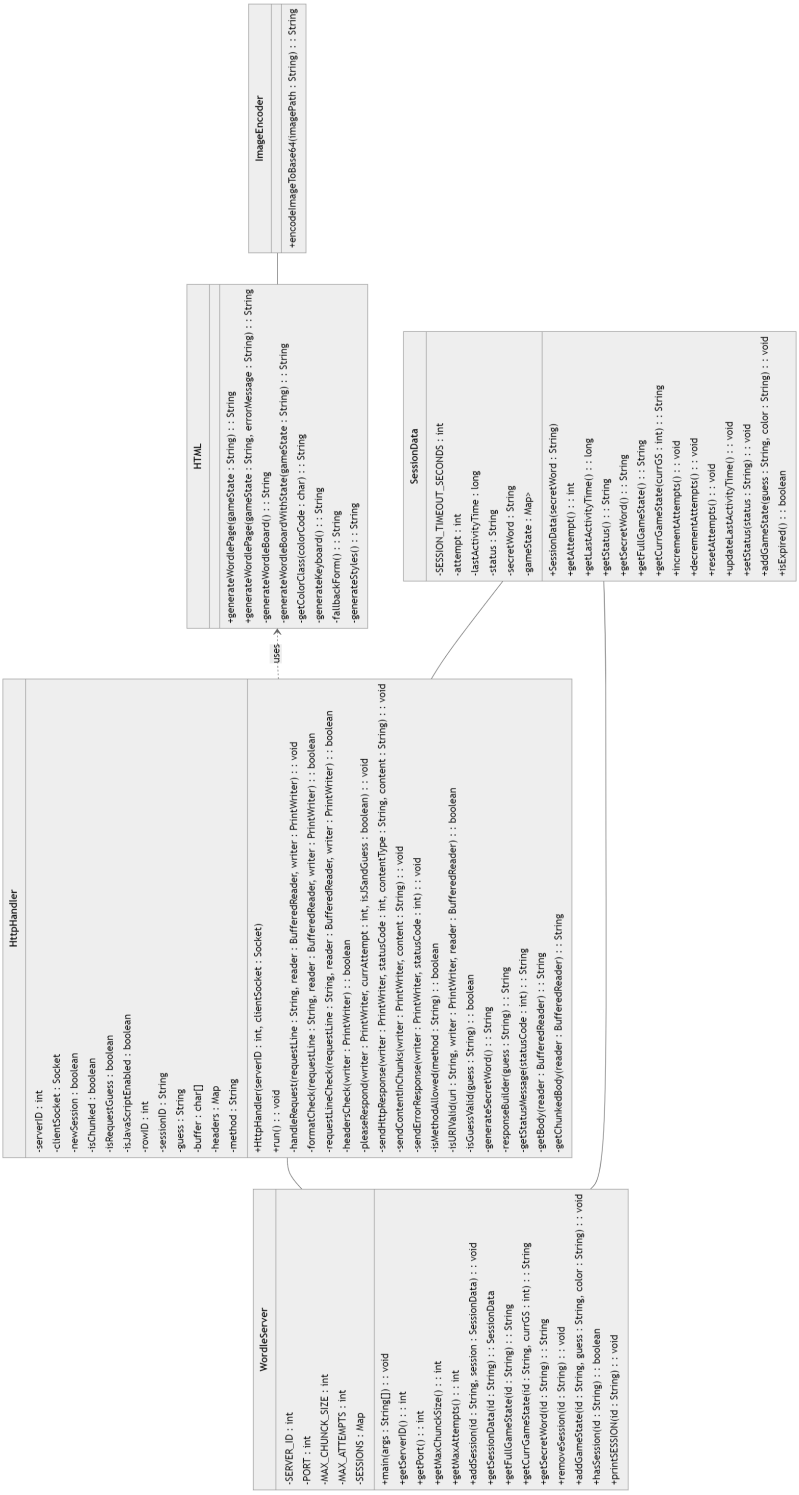

FIGURE 1 – Class diagram.

— Session Data Management : A ConcurrentHashMap is utilized for managing client sessions, offering thread-safe operations. This concurrent map is crucial in preserving the integrity and consistency of session data amidst concurrent access by multiple threads.

Request Processing in *HttpHandler.java*

— Runnable Implementation : HttpHandler, implementing the Runnable interface, is designed for thread execution. The run() method encapsulates the core logic for handling client requests : reading, processing, generating responses, and communication back to the client. Each instance of HttpHandler is responsible for one client request, promoting independence and isolation in request processing.

— Independent Execution : Each HttpHandler instance operates on a separate thread, handling all aspects of a client's request independently. This includes parsing the request, interacting with the WordleServer to access or modify session data, and crafting the appropriate response. Such an approach ensures that each client request is self-contained, minimizing the risk of interference with other requests and enhancing the server's reliability.

— Exception Handling : The run() method is fortified with try-catch blocks, crucial for robust error handling in a multi-threaded environment. This ensures that exceptions in individual threads do not propagate or impact the server's overall functionality, maintaining stability and reliability.

— Scalability and Performance : While the thread pool strategy significantly enhances resource utilization and scalability, the server's ability to handle requests is inherently bound to the thread pool's size and the system's resource limits. The thread pool acts as a control mechanism, preventing potential resource exhaustion under high loads but also imposes an upper limit on concurrent request processing capability.

3 Limits

1. Scalability : The use of a fixed-size thread pool in ***WordleServer.java*** can be a double-edged sword. While it prevents resource exhaustion, it also limits the server's capacity to handle surges in client requests. During peak loads, incoming requests might face delays or even timeouts.
2. Error Handling : Our current handling allows efficient debugging but might lack information for new people working on the game.
3. Dependency on Client-Side Cookies : The session management strategy seems to rely on client-side cookies. This approach might not be reliable if cookies are disabled or manipulated on the client side.

4 Possible Improvements

By addressing the aforementioned areas, the Wordle Server application could see significantly enhancements concerning its scalability, robustness, security, and overall user experience. These

improvements, while relatively small and incremental, have the potential to provide substantial benefits, making the server more resilient, secure, and enjoyable for its users. Implementing these recommendations will not only address the current limitations but also lay a stronger foundation for future enhancements and scalability.
Of course, some of them are out of our current reach based on the knowledge gathered throughout the semester. But if we wanted to launch the game with the same structures, the following improvements would be the first we would do :

— User Experience Improvements : We would enhance the support for non JavaScript users and make sure the UI we created works seamlessly and is positively received by the public.

— Scalability Enhancements : We could implement a dynamic thread pool scaling systems so that we could handle the demand more efficiently.

— Proactive Session Management : By implementing a routine to periodically check and clean up expired sessions, we could manage memory more efficiently and keep the session map lean.