VRIJE
UNIVERSITEIT
BRUSSEL

DISTRIBUTION PROJECT

Distributed Wikipedia System

Rayane Kouidane - 0587073

02 June 2024

Sciences and Bio-engineering Sciences

1 Project Overview

The assignment was to design and implement a distributed system that allows users to search and make changes to articles stored on Wikipedia. Due to the large number of articles, the data is distributed over several peers. To develop such a system some assumptions have been made:

- When users register, they enter a valid username. There is no check for a specific username length or format. However, this can be easily adapted if necessary.
- When a problem occurs in the system, such as when all data peers are full and the user tries to add an article, the user should be notified that the action cannot be performed, and to try again later.
- In the assignment, Section 3.2.3 states that the data peer stores the articles together with all changes applied to an article. The history allows users to see how an article evolved over time. This means that a user should be able to retrieve the history of an article. Therefore, a new command **ShowHistory** was added.
- When using public keys, we can assume that these can be trusted and cannot be forged.
- The Wikipedia server handles the requests of the client and communicates with the database and data peers.

1.1 Application Capabilities and Limitations

1.1.1 Capabilities

- **Register**: Users can register with a unique username and password.
- **Login**: Users can log in with their username and password. If the credentials are correct, the user is notified and considered to be logged in and can start using the system. If the credentials are not correct, the system notifies the user that the password is incorrect.

- **Get**: A user can retrieve an article by giving the title of the article. If the article exist, the content of the article will be shown to the user. If it does not exist the user is notified.
- **Add**: Users can add new articles to the system. They need to give the title and the content of the new article. If another article has the same title, the user will be notified with an error that says that another article with this title already exists.
- **Start Edit**: Users can start an edit session for an article, by giving the title of the article. If the article exists, the user is notified, the article is locked for the user and no one else can start an edit session for this article. The user is also notified if the article does not exist.
- **Edit**: Users can edit an existing article. The user needs to enter the new content of the article. The user is notified if the content was successfully edited. If an error occurred, the user is also notified.
- **End Edit**: Once a user finishes editing an article, the session can be closed. The user is notified if the session closed successfully or not.
- **Show History**: Once a user is in an edit session, he can see the past version of an article by choosing this command. It will show the user a list of the past version of the article.
- **Encryption**: The communication between all the components is encrypted using hybrid encryption. When sending a message to another component, the arguments are encrypted before sending. Once the component receives the message it will decrypt the arguments and execute the correct function.

1.1.2 Limitations

- When a user wants to get an article, it will find it based on the exact title
- The users credentials and articles are not saved and loaded (in a file for example). They are saved in memory.

1.2 Communication Mechanism

All the communication between the distributed components was implemented using Java RMI. RMI offers a simple way to communicate with other components. When sending a message, it is as if the object was local (`Object.message(m)`). RMI also handles the parsing of objects, such as the Client. This way, I can easily send messages to the client. The Distributed Garbage Collector (DGC) also ensures that when a client exits, the object will no longer be in use. DGC helps clean up these objects.

2 Project Execution

To run this project, you just need to run the main method in `MainServer` class. This will set up the connection to the RMI registry and run the Wikipedia server, the user database and the data peers. Finally, each user can run the main method in the `MainClient` class which will run the client and set up the connection. The terminal will open and show the possible commands to the user. Figures 1 and 2 show how to run the servers and the client in IntelliJ. Figure 3 shows what the user sees on the terminal.

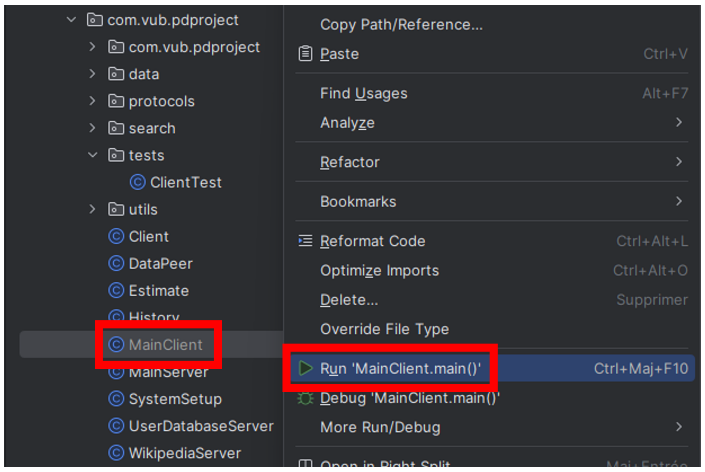


Figure 1: Start the servers in IntelliJ.

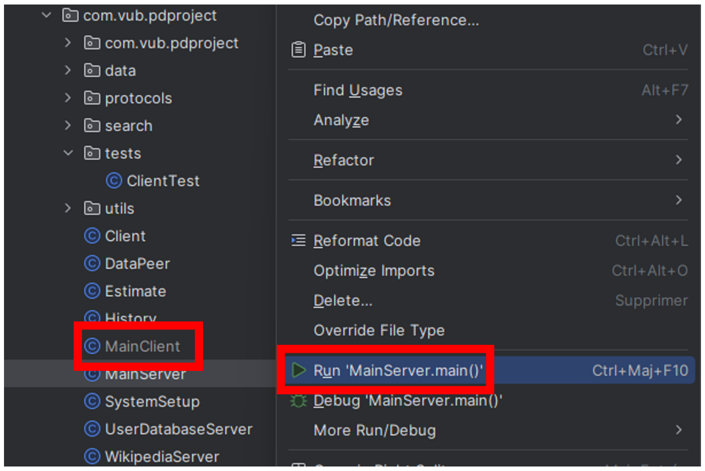
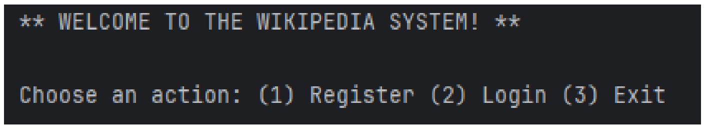


Figure 2: Start the clients in IntelliJ.



```
** WELCOME TO THE WIKIPEDIA SYSTEM! **

Choose an action: (1) Register (2) Login (3) Exit
```

Figure 3: Terminal for the user in IntelliJ.

3 Code Overview

3.1 Classes

- **Client**: The client is represented by the `Client` class. It is responsible for the user interaction, such as registration (`register`), login (`login`) and articles management (`get`, `add`, `startEdit`, `edit`, `endEdit`, `showHistory`)
- **Wikipedia Server**: The `WikipediaServer` class represents the Wikipedia server. It receives the requests from the client and ensures that the correct operations are performed by the other components, such as registration to the database (`register`), login (`login`) managing articles with data peers (`get`, `add`, `startEdit`, `edit`, `endEdit`, `showHistory`). It also holds a reference of the data peer in which an article is saved (`indexes`) and the locks for articles to ensure that clients cannot edit the same article at the same time (`editLocks`).
- **User Database Server**: The `UserDatabaseServer` class is responsible for the

registration of users in the database (`register`). It stores the username and the hashed password of a user in a hash map (`credentials`) and also retrieves the hashed password of a user (`retrieve`). I found the hashing method on [1].

- **Data Peer**: The articles are stored in data peers, implemented by the `DataPeer` class. They store the articles locally (`articles`), along with the history of modifications (`changes`). The data peer handles the articles (`get`, `add`, `startEdit`, `edit`, `endEdit`, `showHistory`).
- **History**: The class `History` implements the necessary methods to keep track of the modifications of an article (`getHistory`). When an article is edited the old version is saved (`addEntry`).
- **Hybrid Encryption**: `SecurityUtil` implements the necessary methods for the generation of private, public and session keys, and the encryption and decryption of data in the system. Public-key cryptography (RSA) is used to encrypt and decrypt the session key. Symmetric key cryptography (AES-256) is used to encrypt and decrypt the actual data. To implement this class I used [2].

3.2 External libraries

For this assignment, I used the Gson and Bcrypt libraries. The Gson library was used to be able to parse an article into a JSON string to be able to encrypt it and send it to the server. On the server, the message was decrypted into a JSON string and parsed back to an article. The Bcrypt library was used for password hashing and to verify if a plain-text password and a hashed password were the same.

3.3 Design Decisions

Many pieces of information, such as articles and user credentials, are stored using a concurrent hash map. This ensures thread safety, which means that when multiple clients register at the same time, it will not create any data inconsistencies.

On the Wikipedia server, the data peers are stored in a priority blocking queue. This ensures that accessing elements is thread-safe and that when the method `peek()`

is called, the smallest loaded data peer is returned. To do so, I implemented a comparator that compares the loads of two peers and ensures that full peers are never at the front of the queue.

References

[1] `https://www.baeldung.com/java-password-hashing`

[2] `https://stackoverflow.com/questions/25351717/how-to-implement-password-based-hybrid-encryption-in-java`

[3] `https://www.geeksforgeeks.org/runnable-interface-in-java/`