Charles University Faculty of Mathematics and Physics

Programming 2 [NPRG031]

Simple chat with GUI

Mozharov Nazar

Prague 2023

Content

General information about the program. 3
Description of the Graphic User Interface (GUI) 4
Description of user interaction with the program 5
Describing the technical part 6
About the server part code 6
About the client part code 8
About security and client-server interaction 10
About code testing 12
Conclusion about the created program 13

General information about the program.

The purpose of the task is to develop a simple chat. The program must meet the following requirements:

- Create a program with Graphic User Interface (GUI)
- Develop Client side and server side.
- Implement ability to send and receive text messages in real-time using the TCP protocol.
- Implement simple message encryption using the RSA cryptosystem

Description of the Graphic User Interface (GUI)

I implemented the graphic user interface using the Windows Presentation Foundation (WPF), which is a graphical subsystem as part of the .NET Framework. To change the general appearance of the program, I used third-party open-source library called “ModernWPF UI Library”.

Game GUI structure

Description of user interaction with the program

When a user opens the program, they enter a waiting window. They remain in this window until they can connect to the server. After that, they will need to enter their username. If they enter a name that is already taken by another person, the program will ask them to choose a different username for the current session. Afterwards, the user is taken to the room selection window. Each room represents a closed chat accessible only to those who have the password for that room. All messages written within a room remain there. Now the user needs to choose whether to create a room or join an existing one.

- If the user decides to create a room, they will need to enter its name and a password for accessing it. After creation, the program automatically enters this room.
- If the user decides to join an existing room, they are taken to the room selection window where they can choose the desired room from the list or enter its name manually. If the user enters a non-existent room name, the program will ask them to enter a correct room name. Afterwards, the user will need to enter the password for the room they are trying to join. If an incorrect password is entered, the program notifies the user and prompts them to enter the correct password.

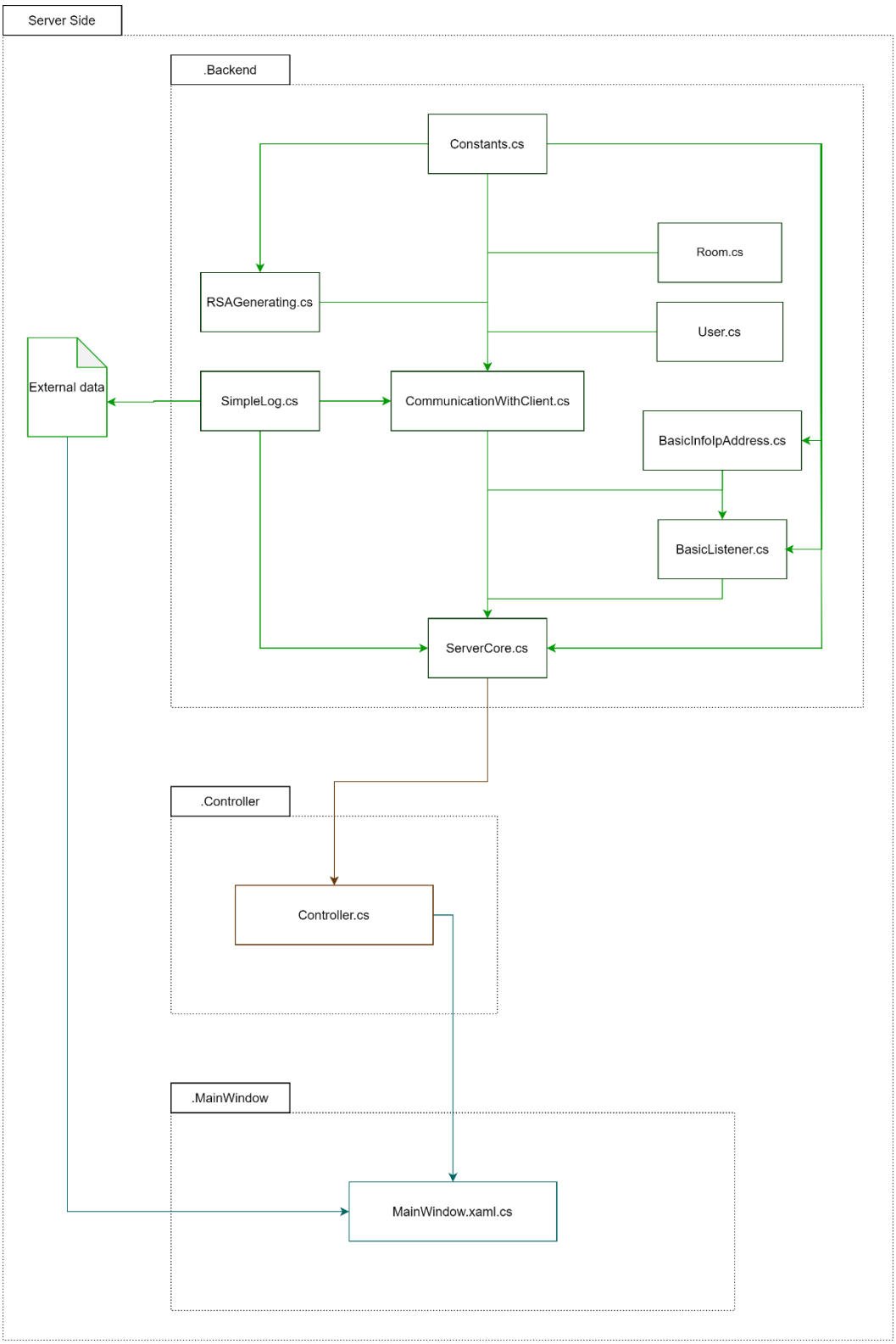
Once the user has chosen a room, they enter the chat of that room, where they can anonymously communicate with its participants. When all participants leave the room, it will be deleted for security purposes.

Describing the technical part

About the server part code

To implement the server part, I divided code into 3 parts: the backend, the controller, and the GUI:

- The backend is responsible for analyzing, securing and processing all the data that comes in through the TCP protocol in real time.
- The controller is responsible for the communication between the backend and the GUI.
- The GUI is responsible for displaying logs and entering basic information to configure the server before it starts.

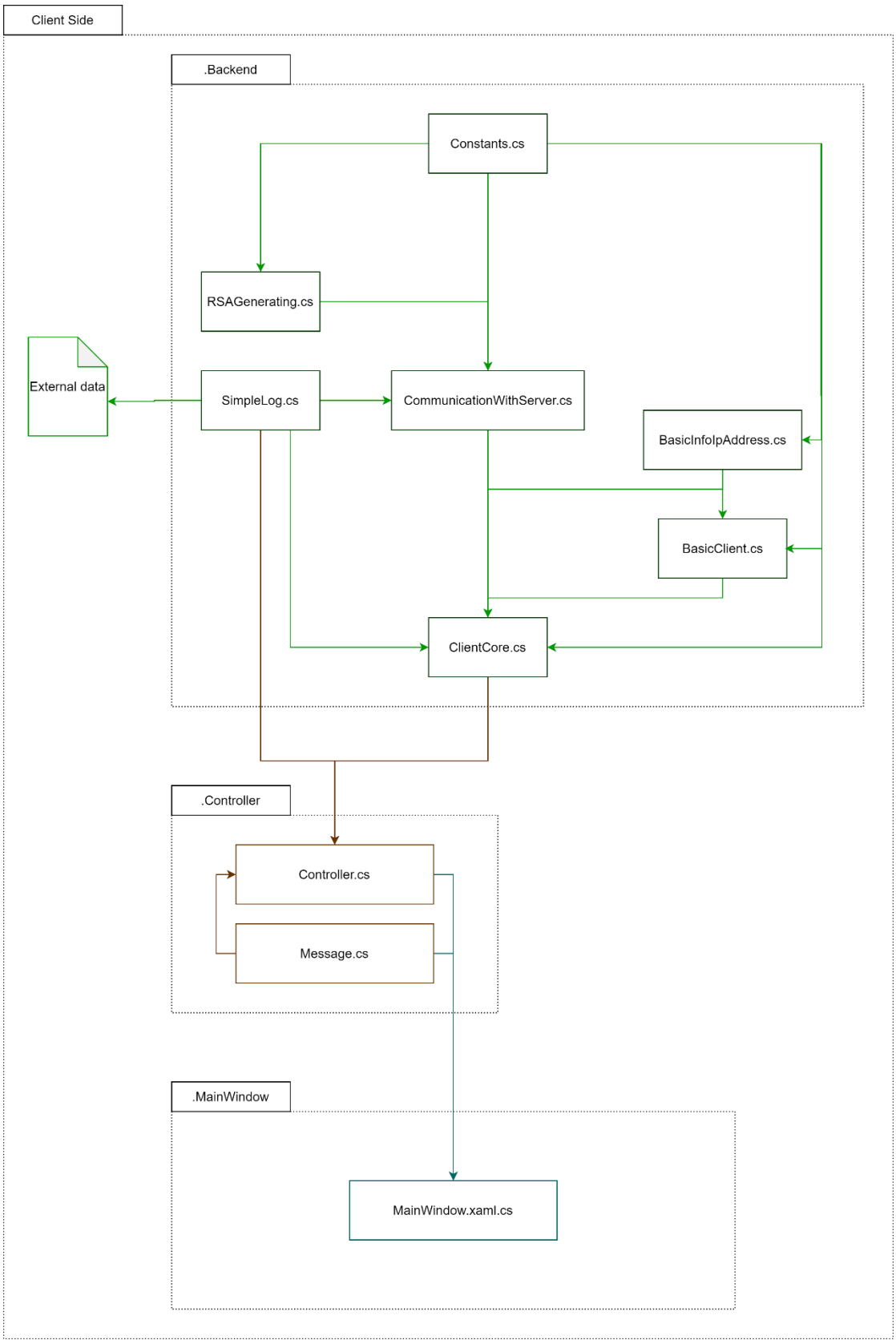


Picture of dependencies between backend, controller and GUI of the server side

About the client part code

To implement the client part, I also divided code into 3 parts: the backend, the controller, and the GUI:

- The backend is responsible for analyzing, securing and processing all data that is sent to and from the server.
- The controller is responsible for the communication between the backend and the GUI.
- The user interface is responsible for displaying logs and entering basic information to configure the server before it starts.

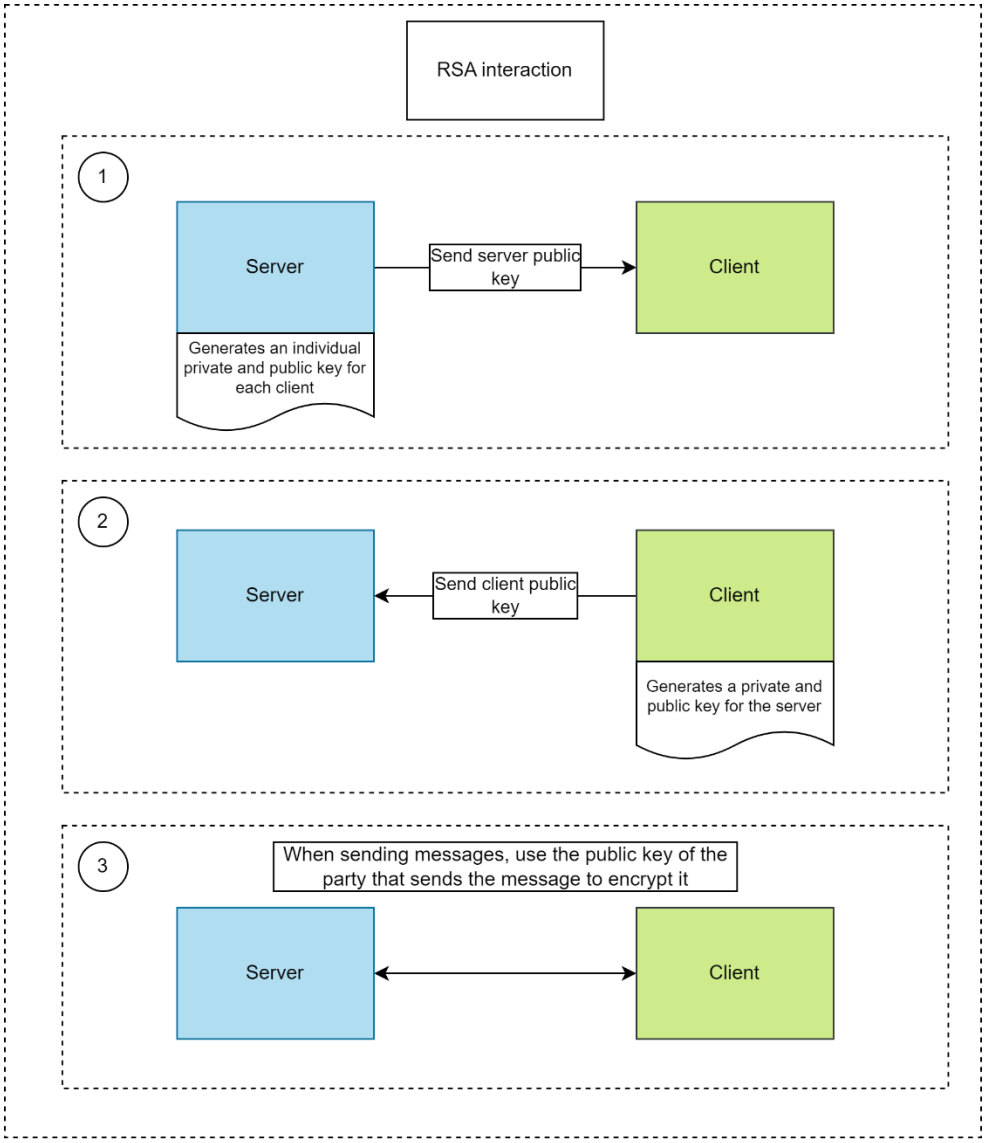


Picture of dependencies between backend, controller and GUI of the client side

About security and client-server interaction

To exchange messages between the client and the server over TCP I use RSA encryption, here are some reasons why I chose this RSA algorithm:

- Security: RSA is one of the most widespread and proven cryptographic algorithms that provides a high level of security.
- If the key is intercepted during transmission, and then the message encrypted with that key is intercepted, the attacker will not be able to decrypt it because the public key is only intended for message encryption.



Picture of client-server encryption key exchange

- By default, the program is set up to exchange data at 6KB per transfer. Of which 3 KB is available for useful information, while the rest of the message space is spent on encryption.

- The server and the client have predefined commands that are added to the beginning of a message before sending it. These commands are necessary for the client and server to understand what they want to obtain or accomplish with a particular message.
  One example of such a command set is the application of a user name. The client sends a message with a command at the beginning indicating that the client wants to set a user name. The server recognizes this and responds with either a positive command if the user's name is available or a negative command if the user's name is already taken by someone else.

- During operation, the server identifies users based on their unique ID generated by hashing their input data using the SHA algorithm. This provides additional security on the server side.

- The server does not store any user data. All conversations conducted between users are local and are immediately deleted upon the closure of all clients involved in the conversation. This ensures additional anonymity and security.

About code testing

To verify the correctness of the code, you have written a series of unit tests for the most important and necessary parts of the code. Some of these unit tests are integration tests, which means they require a running server or client. If a test is an integration test, the necessary steps for its proper activation are described in code comments.

Conclusion about the created program

During the development of a small chat application, I learned how RSA encryption works, how to write client and server components, and how to work with the WPF graphical interface. Additionally, I gained experience in writing unit tests, some of which were integration tests.

In the future, I would like to add new usage scenarios to the program. These include an improved user interface and enhanced user experience. I want to add new commands to the client and server parts to enable file transfer. I also want to give users the option to choose whether they want to save their conversations in a database for future reference. To achieve this, I will need to write secure and reliable interactions between the server and the database.