ФГБОУ ВО “Чувашский государственный университет им. И. Н. Ульянова” Факультет: ИВТ

Кафедра: Вычислительной техники

Предмет: Сети и телекоммуникации

Лабораторная работа 8

**Сетевое приложение**

Выполнил: студент группы ИВТ-41-20

Галкин Дмитрий

Проверила: Путевская И.В.

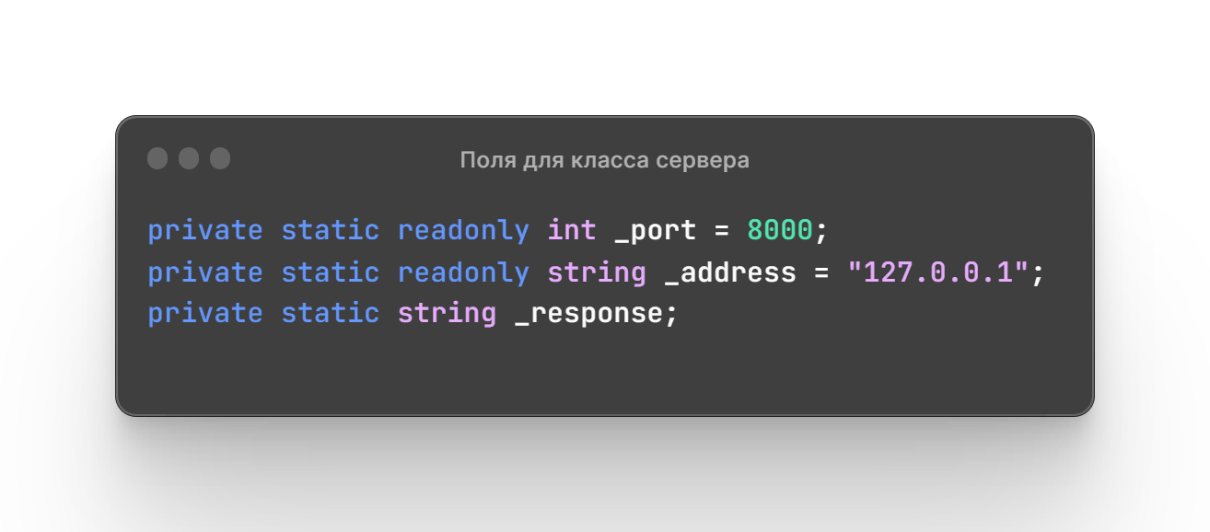
Чебоксары 2023 г.

**Цель:** научиться писать сетевые приложения

**Задача 1:** изучить примеры с сетевого сокета и примеры сетевых протоколов

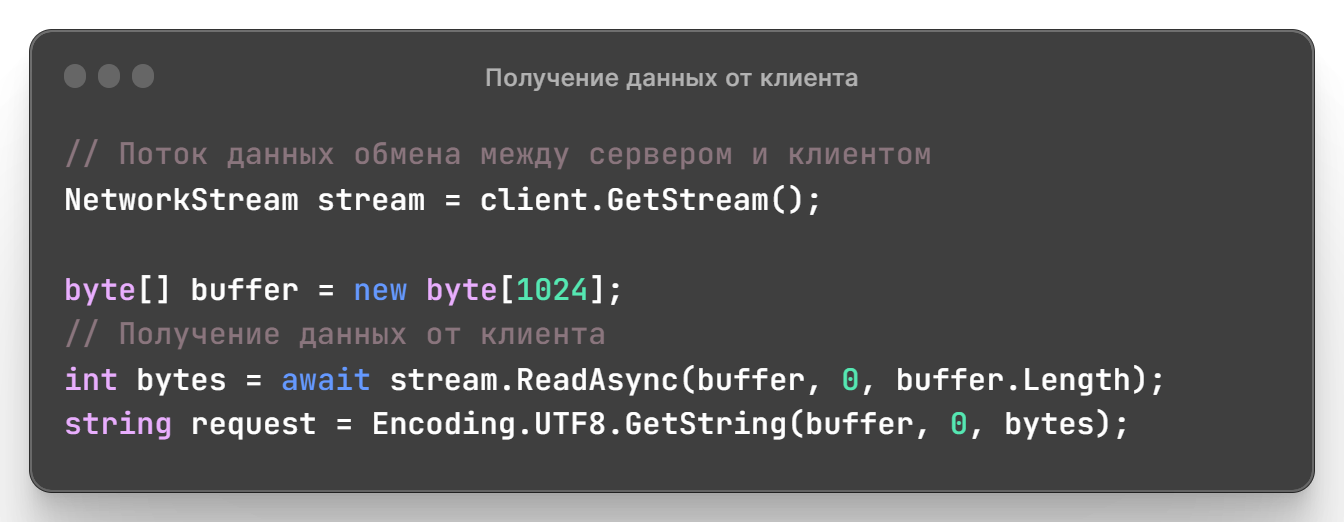
**Задача 2:** создать серверное приложение, реализующее одну из функций:

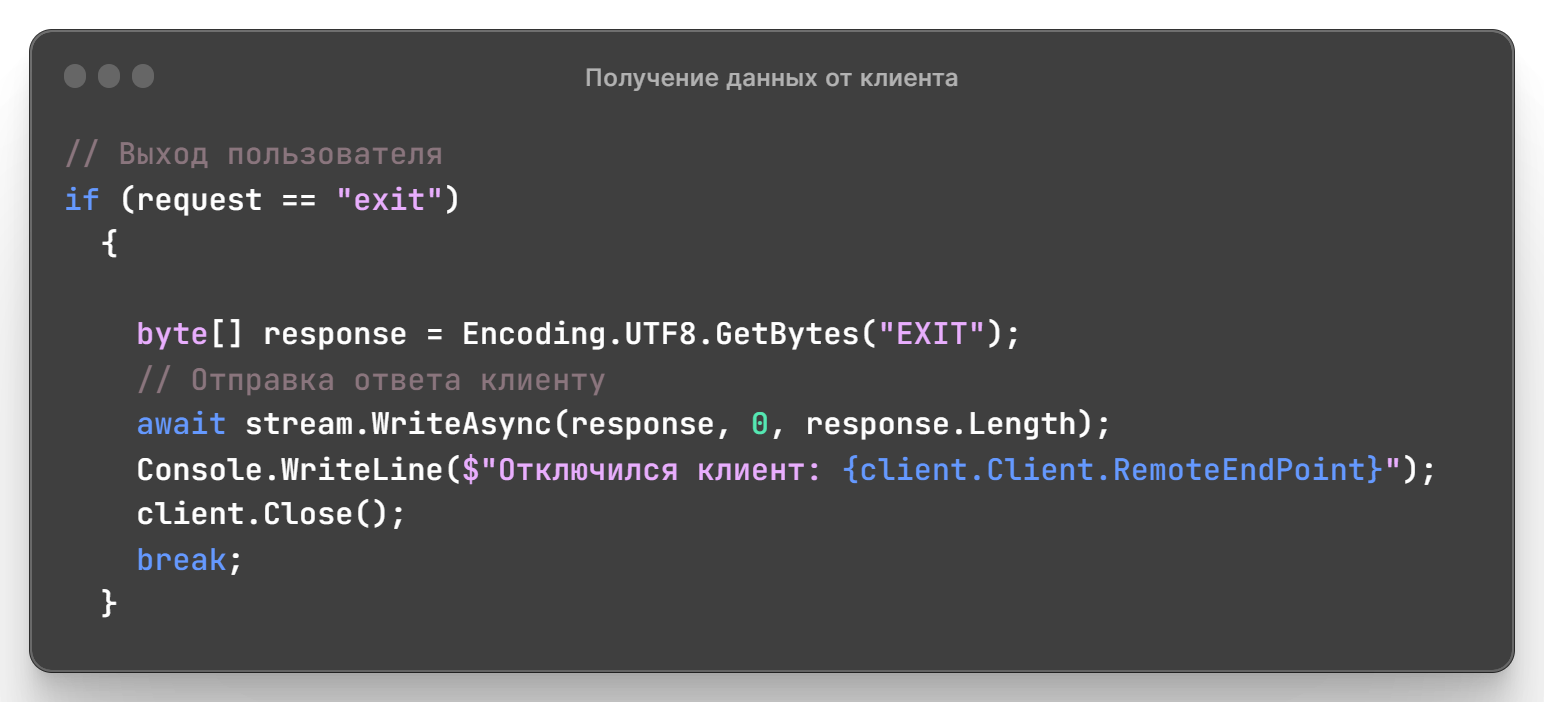
* Криптопреобразование сообщений, аутенфикация пользователя, скачивания файла/каталога, доставка файла/каталога, удаленное выполнение кода, получение информации о компьютере, чат, проверка сети.

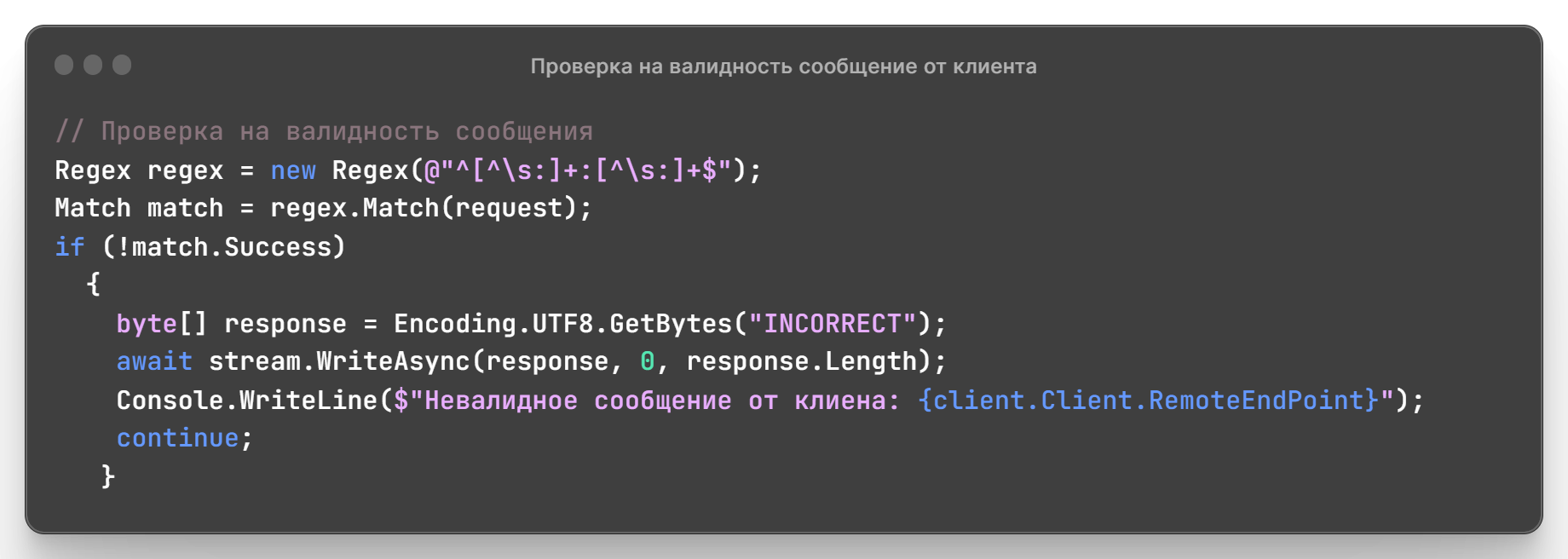


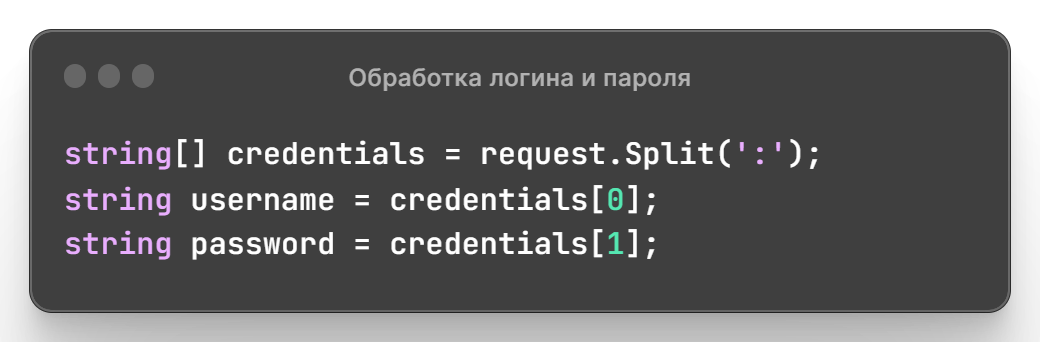


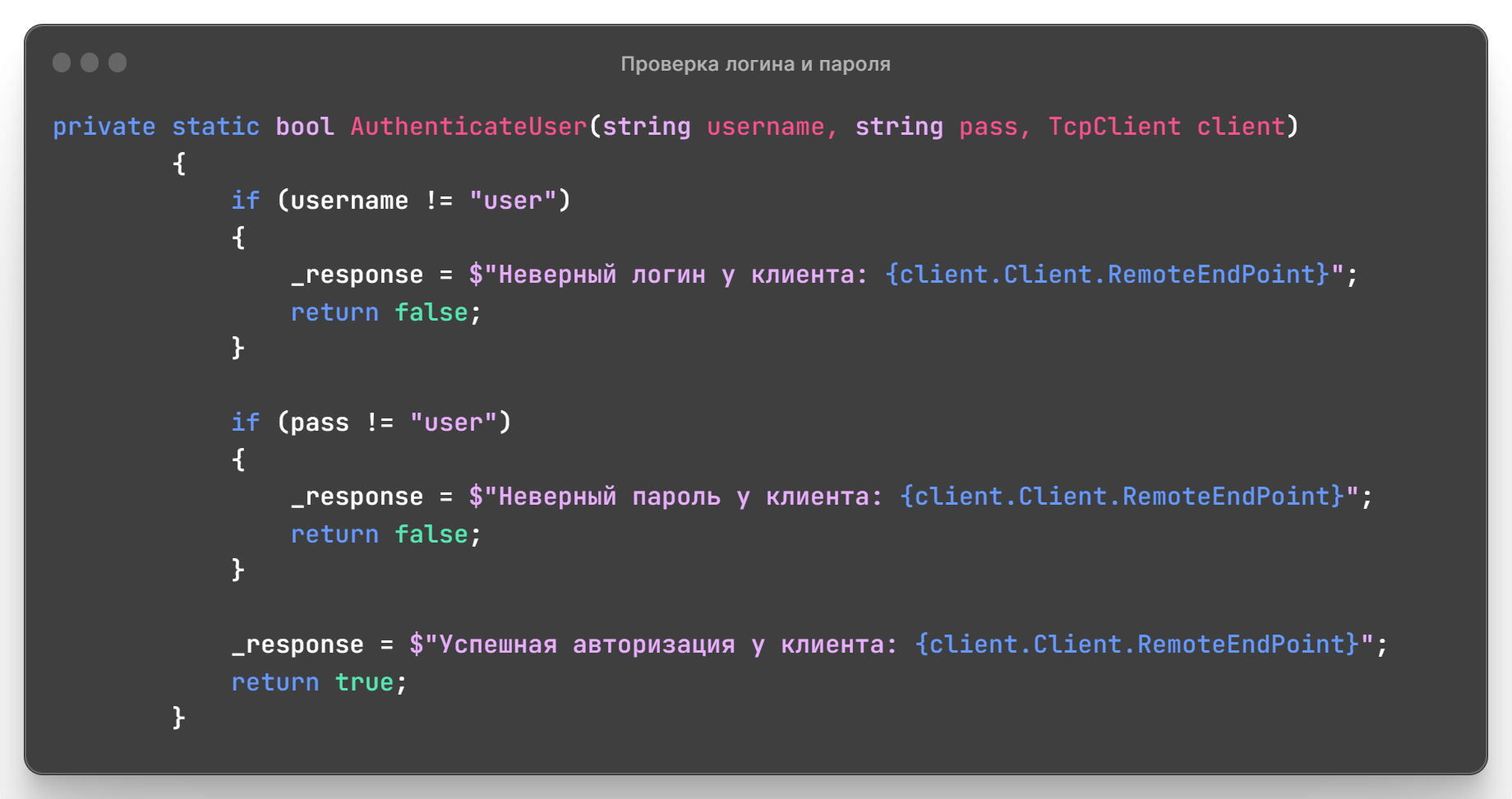
Метод обработки клиента (HandleClient)

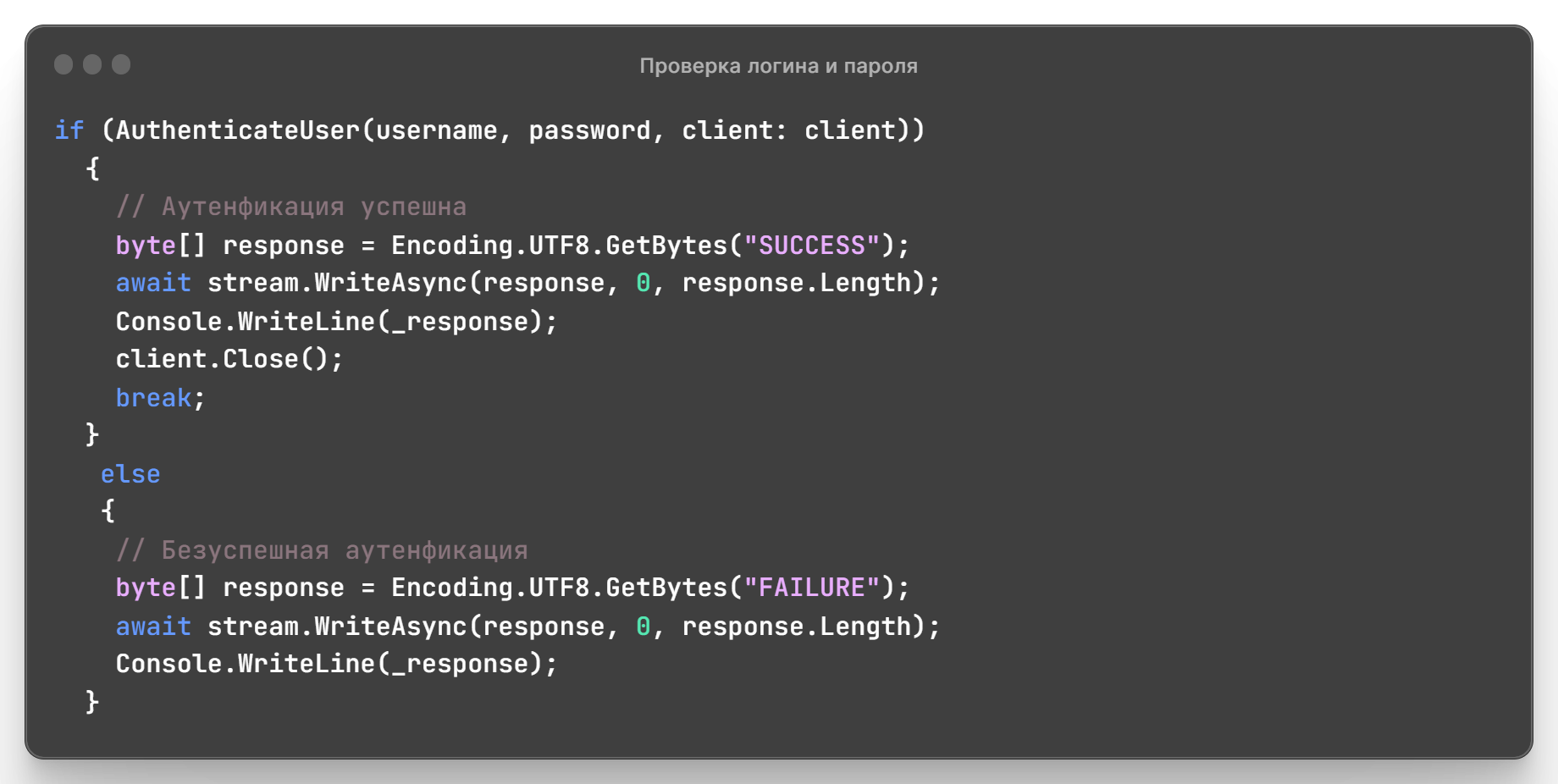




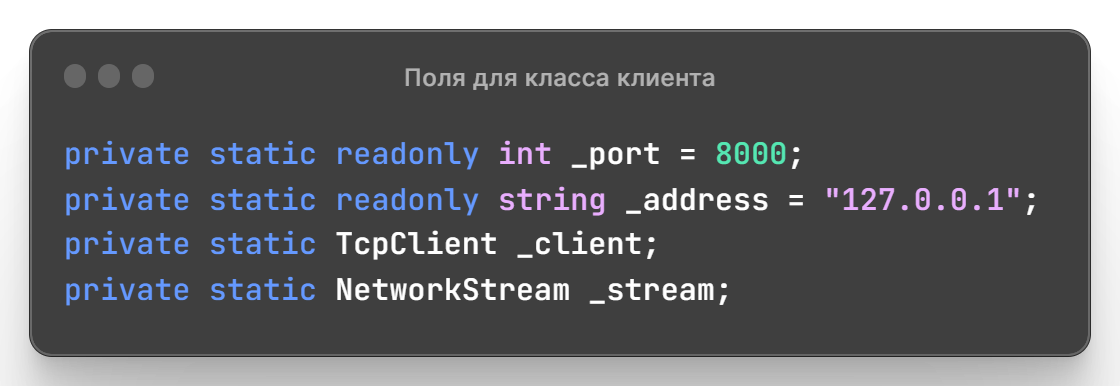








**Задача 3:** создать клиентское приложение







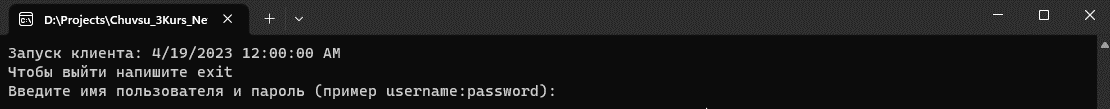


**Задача 4:** протестировать

* Запуск сервера

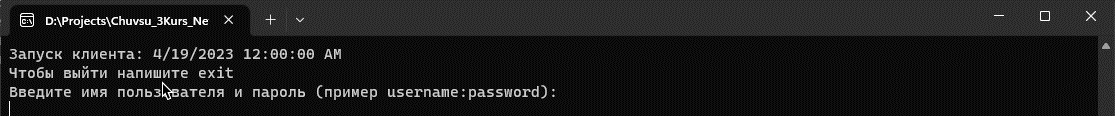


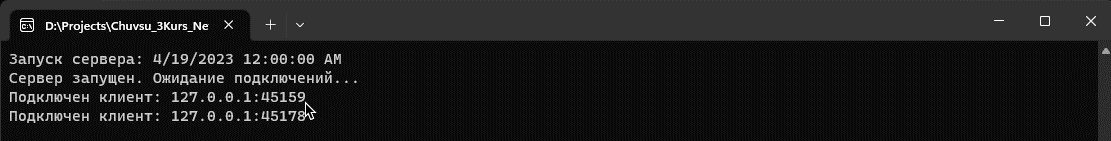
* Запуск клиента 1



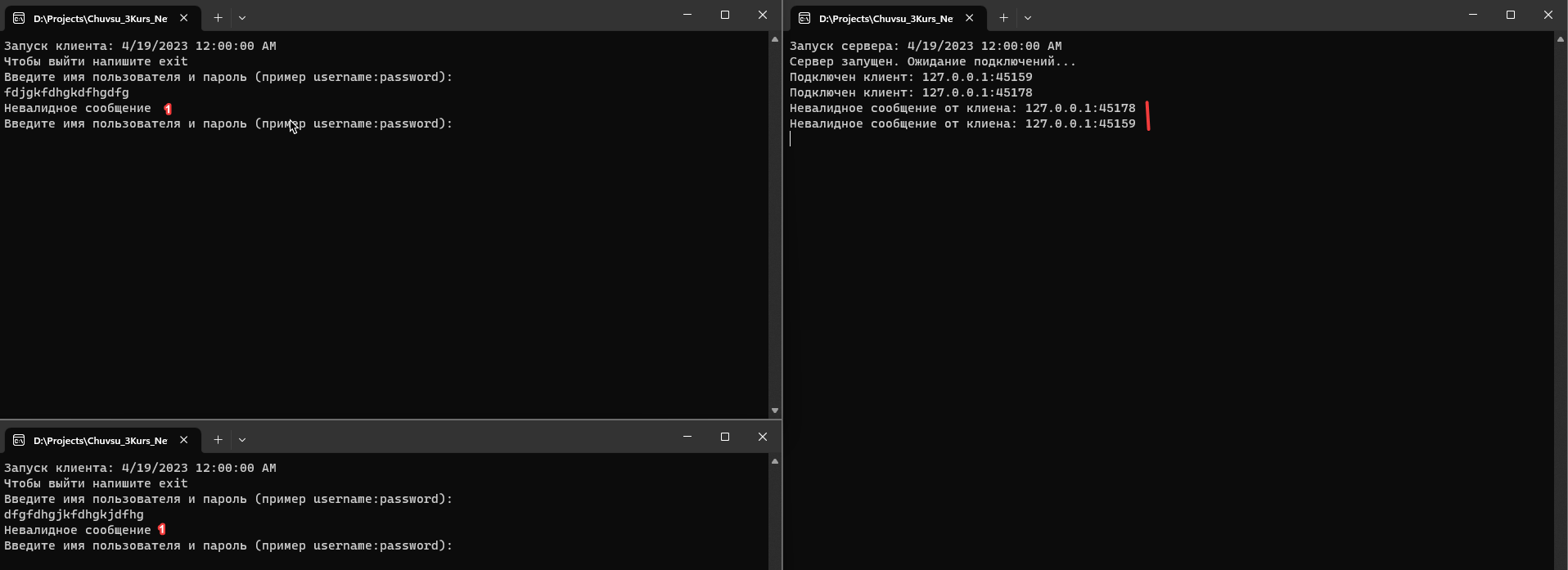


* Запуск клиента 2

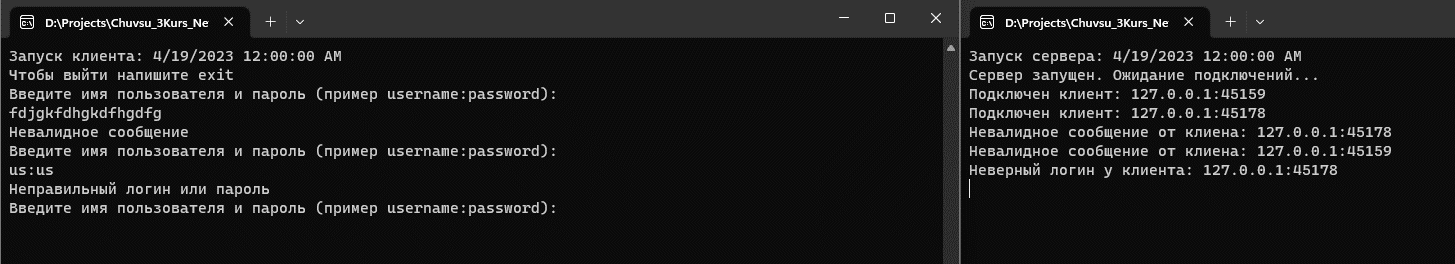


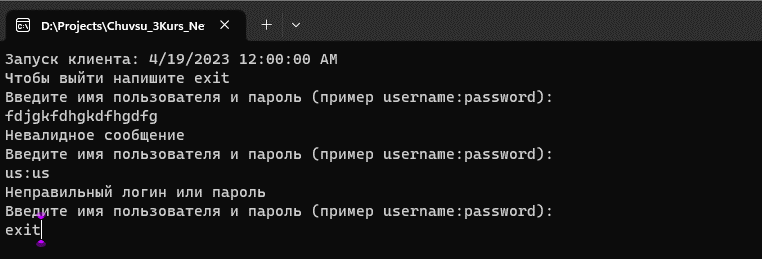


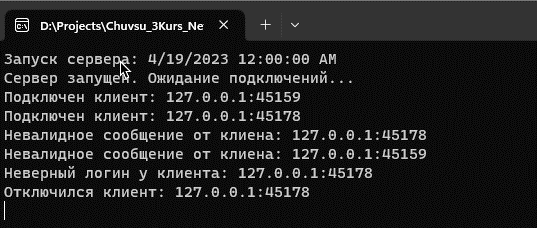
* Проверки на валидность сообщений



* Проверки на клиенте 1







* Успешная авторизация на клиенте 2





**Ответы на вопросы:**

1. Какая структура кода сетевого приложения?

* Определение протокола обмена данными (например: HTTP, SMPT и т.д)
* Создание сокетов на сторонах клиента и сервера
* Установить соединение между клиентами и сервером
* Реализация обмена данными
* Реализация обработки ошибок
* Завершение работы сокетов (закрытие)

1. Как реализовать разделение сообщений по протокол обмена?

* Через фиксированную длину сообщений
* Через специальный символ (разделитель) или последовательность символов
* Через заголовок сообщений: подход предусматривает использование заголовка сообщений, который содержит информацию о сообщении.
* Через использование стандартных протоколов (TCP, HTTP и т.д)

1. Как создать клиент-серверное сообщение?

* Реализовать клиентское приложение, которое будет отправлять сообщение на сервер
* Реализовать серверное приложение, которое будет получать сообщения о клиентах, обрабатывать и отправлять ответ

1. Как создать распределённое сообщение?

* Подход RPC (Remote Procedure Call) – позволяет вызывать функции на удаленных узлах и получать результат их выполнения
* Подход через Kafka – организация передачи сообщений между узлами и обеспечивают механизмы синхронизации работы и обработка ошибок

1. Как реализовать аутенфикацию и авторизацию?

* Определить какую информацию нужно использовать для аутенфикации
* Разработать систему аутенфикации (например: логин и пароль)
* Создать систему авторизации, которая определяет какие действия пользователь может выполнить после успешной аутенфикации
* Реализовать механизм безопасности, такие как шифрование и подпись данных
* Тестирование системы аутенфикации для нахождения ошибок

1. Как защитить передачу данных по сети?

* Протокол SSL/TLS: это протокол использует шифрование данных между клиентом и сервером
* Шифрование данных на стороне приложения
* Аутенфикация пользователей
* Использование сетевых устройств, такие как брандмауэры
* Ограничение доступа к данным

**Листинг программы сервера**

*using* System;  
*using* System.Net;  
*using* System.Net.Sockets;  
*using* System.Text;  
*using* System.Text.RegularExpressions;  
*using* System.Threading.Tasks;  
  
*namespace* Server  
{  
 *class* Program  
 {  
 *private static readonly int \_port* = 8000;  
 *private static readonly string \_address* = "127.0.0.1";  
 *private static string \_response*;  
   
 *static async* Task Main(*string*[] args)  
 {  
 Console.WriteLine($"Запуск сервера: {DateTime.Now.Date.ToString()}");  
   
 *try* {  
 TcpListener server = *new* TcpListener(IPAddress.Parse(\_address), \_port);  
 server.Start();  
   
 Console.WriteLine("Сервер запущен. Ожидание подключений...");  
   
 *while* (*true*)  
 {  
 TcpClient client = *await* server.AcceptTcpClientAsync();  
 Console.WriteLine($"Подключен клиент: {client.Client.RemoteEndPoint}");  
  
 Task.Run(() => HandleClient(client));  
 }  
 }  
 *catch*(Exception ex)  
 {  
 Console.WriteLine(ex.Message);  
 }  
 }  
  
 *private static async* Task HandleClient(TcpClient client)  
 {  
 *while* (*true*)  
 {  
 NetworkStream stream = client.GetStream();  
  
 *byte*[] buffer = *new byte*[1024];  
 *int* bytes = *await* stream.ReadAsync(buffer, 0, buffer.Length);  
 *string* request = Encoding.UTF8.GetString(buffer, 0, bytes);  
  
 *if* (request == "exit")  
 {  
 *// Выход пользователя  
 byte*[] response = Encoding.UTF8.GetBytes("EXIT");  
 *await* stream.WriteAsync(response, 0, response.Length);  
 Console.WriteLine($"Отключился клиент: {client.Client.RemoteEndPoint}");  
 client.Close();  
 *break*;  
 }  
  
 *// Проверка на валидность сообщения* Regex regex = *new* Regex(@"^[^\s:]+:[^\s:]+$");  
 Match match = regex.Match(request);  
 *if* (!match.Success)  
 {  
 *byte*[] response = Encoding.UTF8.GetBytes("INCORRECT");  
 *await* stream.WriteAsync(response, 0, response.Length);  
 Console.WriteLine($"Невалидное сообщение от клиена: {client.Client.RemoteEndPoint}");  
 }  
 *else* {  
 *string*[] credentials = request.Split(':');  
 *string* username = credentials[0];  
 *string* password = credentials[1];  
  
 *if* (AuthenticateUser(username, password, client: client))  
 {  
 *// Аутенфикация успешна  
 byte*[] response = Encoding.UTF8.GetBytes("SUCCESS");  
 *await* stream.WriteAsync(response, 0, response.Length);  
 Console.WriteLine(\_response);  
 client.Close();  
 *break*;  
 }  
 *else* {  
 *// Безуспешная аутенфикация  
 byte*[] response = Encoding.UTF8.GetBytes("FAILURE");  
 *await* stream.WriteAsync(response, 0, response.Length);  
 Console.WriteLine(\_response);  
 }  
 }  
 }  
   
 *return*;  
 }  
  
 *private static bool AuthenticateUser*(*string* username, *string* pass, TcpClient client)  
 {  
 *if* (username != "user")  
 {  
 *\_response* = $"Неверный логин у клиента: {client.Client.RemoteEndPoint}";  
 *return false*;  
 }  
  
 *if* (pass != "user")  
 {  
 *\_response* = $"Неверный пароль у клиента: {client.Client.RemoteEndPoint}";  
 *return false*;  
 }  
  
 *\_response* = $"Успешная авторизация у клиента: {client.Client.RemoteEndPoint}";  
 *return true*;  
 }  
 }  
}

**Листинг программы клиента**

*using* System;  
*using* System.Net;  
*using* System.Net.Sockets;  
*using* System.Text;  
  
*namespace* Client  
{  
 *class* Program  
 {  
 *private static readonly int \_port* = 8000;  
 *private static readonly string \_address* = "127.0.0.1";  
 *private static* TcpClient *\_client*;  
 *private static* NetworkStream *\_stream*;  
   
 *static void Main*(*string*[] args)  
 {  
 Console.*WriteLine*($"Запуск клиента: {DateTime.Now.Date.ToString()}");  
   
 *try* {  
 *ConnectClient*();   
 Console.*WriteLine*("Чтобы выйти напишите exit");  
  
 *while* (*true*)  
 {  
 Console.*WriteLine*("Введите имя пользователя и пароль (пример username:password):");  
 *string* input = Console.*ReadLine*();  
  
 *\_stream* = *\_client*.GetStream();  
 *byte*[] buffer = Encoding.UTF8.GetBytes(input);  
 *\_stream*.Write(buffer, 0, buffer.Length);  
  
 buffer = *new byte*[1024];  
 *int* bytes = *\_stream*.Read(buffer, 0, buffer.Length);  
 *string* response = Encoding.UTF8.GetString(buffer, 0, bytes);  
  
 *if* (!*CheckResponse*(response)) *break*;  
 }  
  
 *DisconnectClient*();  
 }  
 *catch*(Exception ex)  
 {  
 Console.*WriteLine*(ex.Message);  
 }  
 Console.*WriteLine*("Отключение системы...");  
 *return*;  
 }  
  
 *private static bool CheckResponse*(*string* response)  
 {  
 *switch* (response)  
 {  
 *case* "SUCCESS":  
 {  
 Console.*WriteLine*("Успешная аутенфикация!");  
 *return false*;  
 }  
 *case* "EXIT":  
 {  
 *return false*;  
 };  
 *case* "INCORRECT":  
 {  
 Console.*WriteLine*("Невалидное сообщение");  
 *return true*;  
 }  
 *case* "FAILURE":  
 {  
 Console.*WriteLine*("Неправильный логин или пароль");  
 *return true*;  
 }  
 *default*: *return true*;  
 }  
 }  
  
 *private static void ConnectClient*()  
 {  
 *\_client* = *new* TcpClient();  
 *\_client*.Connect(*\_address*, *\_port*);  
 }  
   
 *private static void DisconnectClient*()  
 {  
 *\_stream*.Close();  
 *\_client*.Close();  
 }  
 }  
}