

# Проект «Сетевой чат»

Так как основная часть кода уже была написана преподавателем и мной при выполнении домашних работ, то в данной презентации указываю только пункты задания, реализованные в рамках курсового проекта.

## **Серверная часть:**

- Возможность регистрации и авторизации клиентов, с хранением данных о клиентах в базе данных.
- Сервер к сообщениям должен добавлять время рассылки
- Возможность изменения ника клиента по его запросу
- Рассылка списка активных клиентов
- Автоматическое отключение клиентов, которые не активны более 20 минут
- Возможность банить клиентов, баны могут быть по времени или перманентные (так может делать админ админ)
- Возможность остановить сервер командой /shutdown
- SQL команды для создания таблиц с данными пользователей
- Что хотелось бы сделать, но не хватило времени

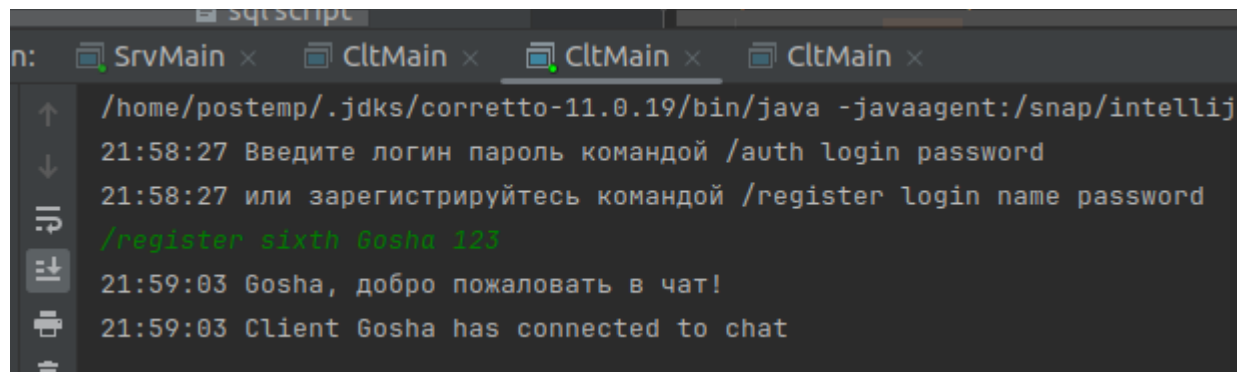
Клиентскую часть выбрал терминальную, значимых изменений не вносил.

## Регистрация и авторизации клиентов, с хранением данных о клиентах в базе данных

В качестве базы данных (далее - БД) использовал контейнер Docker с PostgreSQL

Методы для работы с БД вносил в класс `InMemoryAuthProvider` т.к. он работает с регистрацией и авторизацией

Для регистрации клиент использует команду `/register login name password`, вызывается метод `server.getAuthProvider().register`, который регистрирует пользователя в БД и добавляет новый объект класса `USER` в List массив `users`.



```

n:  SrvMain x  CltMain x  CltMain x  CltMain x
↑ /home/postemp/.jdk/corretto-11.0.19/bin/java -javaagent:/snap/intellij
↓ 21:58:27 Введите логин пароль командой /auth login password
: 21:58:27 или зарегистрируйтесь командой /register login name password
⇅ /register sixth Gosha 123
⇅ 21:59:03 Gosha, добро пожаловать в чат!
⇅ 21:59:03 Client Gosha has connected to chat
⇅
```

Роль `USER` присваивается новому пользователю автоматически, изменение на роль `ADMIN` выполняется с помощью SQL скрипта через программу DBeaver:

```
update user_to_roles ur set role_id = (select id from roles r where r.roles_name = 'ADMIN')
where user_id = (select id from users u where u.login = 'second')
```

Хотел добавить команду /makeadmin login для выполнения из под пользователя с административными правами, но времени не хватило (

В момент старта приложения сервера данные о пользователях загружаются из БД в массив List<User> users в конструкторе InMemoryAuthenticationProvider()

Старт конструктора производится при создании объекта server из метода main класса SrvMain  
Server server = new Server(port, new InMemoryAuthenticationProvider());

Зарегистрированный клиент входит в чат используя команду /auth login password

```
/home/postemp/.jdkcs/corretto-11.0.19/bin/java -javaagent:/snap/intellij-id  
22:49:28 Введите логин пароль командой /auth login password  
22:49:28 или зарегистрируйтесь командой /register login name password  
/auth third 123  
22:49:33 Pasha, добро пожаловать в чат!  
22:49:33 Пользователь Pasha подключился к чату  
|
```

Все пользователям получают сообщение о входе нового пользователя в чат.

Вход в этом случае происходит без взаимодействия с БД, аутентификация и авторизация используют массив List<User> users

**Удаление пользователя производится из под DBeaver**

delete from public.users where id = (select id from users u where u.login = 'second')

запись из таблицы public.user\_to\_roles удаляется каскадно

## Сервер к сообщениям добавляет время рассылки

Реализовал с помощью класса Date в ClientHandler в методе sendMessage, который используется при любой посылке сообщения пользователям.

```
public void sendMessage(String message) {  
    try {  
//        System.out.println("We try to send message:" + message);  
        Date currentDate = new Date();  
        SimpleDateFormat timeFormatter = new SimpleDateFormat("HH:mm:ss");  
        out.writeUTF(timeFormatter.format(currentDate) + " " + message);  
    }  
}
```

посылаем сообщение другому пользователю:

```
22:50:54 Пользователь Gosha подключился к чату  
/w Gosha 123123123  
|
```

другой пользователь видит его с пометкой времени:

```
22:50:54 Пользователь Gosha подключился к чату  
22:57:48 123123123|
```

## Изменение ника клиента по его запросу

Производится командой /changenick newusername

```
23:16:10 nick server: принятая команда: /register login name password
/ auth fifth 123
23:16:11 Dasha, добро пожаловать в чат!
23:16:11 Пользователь Dasha подключился к чату

23:16:14 Broadcast message:
/ changenick Dasha123
23:17:04 Ваш ник сменен на Dasha123
```

При вызове команды производится проверка на наличие аргумента, потом ник меняется в БД и в памяти через `user.setUsername(newNick)`;

```
case "/changenick": { // смена своего ника
    String newNick;
    try {
        newNick = args[1];
    } catch (ArrayIndexOutOfBoundsException e) {
        sendMessage("Не указан новый ник");
        continue;
    }
    boolean isWrittenToDB = server.getAuthenticationProvider().changeNickDB(this.username, newNick);
    if (isWrittenToDB) {
        this.username = newNick;
        sendMessage("Ваш ник сменен на " + this.username);
    } else {
        sendMessage("Не удалось сменить ваш ник, при записи в БД возникли проблемы");
    }
}
```

```
}  
    continue;  
}  
default: {  
    System.out.println("default");  
    sendMessage("Неопознанная команда");  
}
```

## Рассылка списка активных клиентов

Осуществляется командой **/activelist**, выводит список активных пользователей

```
00:37:20 Пользователь Masha подключился к чату  
00:37:35 Пользователь Gosha подключился к чату  
/activelist  
00:37:42 Pasha, Masha, Gosha  
|
```

```
List<String> userList = new ArrayList<>();  
for (ClientHandler clientHandler : clients) {  
    System.out.println("clientHandler.getUsername()" + clientHandler.getUsername());  
    userList.add(clientHandler.getUsername());  
}  
return userList;
```

Дополнительно добавлена команда **/whoami**, выводит на экран ник пользователя и его роль

```
00:33:43 Пользователь Pasha подключился к чату  
/whoami  
00:33:49 Вы Pasha ваша роль:USER
```

```
public String getRoleByUsername(String username) {  
    for (User user : users) {  
        if (Objects.equals(user.getUsername(), username)) {  
            return user.getRole();  
        }  
    }  
    return null;  
}
```

Добавлена команда **/allclients**, выводит список всех пользователей в БД, команда срабатывает только для пользователя с правами ADMIN

```
12:41:00 или зарегистрируйтесь командой /register - login name - пароль  
/auth first 123  
12:42:07 Sasha, добро пожаловать в чат!  
12:42:07 Пользователь Sasha подключился к чату  
/allclients  
12:42:11 Список всех клиентов из БД: Sasha; Pasha; Misha; Dasha123; Masha; Gosha;
```

Список всех клиентов получаем через метод `server.getAuthenticationProvider().getAllClientsList()`



## Автоматическое отключение клиентов, которые не активны более 20 минут

Для пользователя это выглядит как отключение через 20 минут работы в чате:

```
23:10:19 Ну нельзя так долго сидеть в чате, идите работать! :)
Disconnect

Process finished with exit code 0
```

В консоли сервера

```
Пользователь Gosha находился в чате в течение 19 min
Пользователя отключили, поэтому exception
ClientHandler Thread disconnect
Пользователь Gosha находился в чате в течение 20 min
Отключаем пользователя Gosha
Пользователя отключили, поэтому exception
ClientHandler Thread disconnect
```

При входе пользователя в чат через сеттер записывается дата и время его подключения в классе `ClientHandler` в поле `Date loginDate`;

При запуске метода `start()` класса `Server` запускается отдельный `Thread`, в котором в цикле раз в минуту проверяется время нахождения каждого пользователя в чате

Высчитывается разница между временем логирования и текущим в минутах и отключается клиент, если разница больше 20 мин.

```
new Thread() -> {
    try {
        while (!toExit) {
            Date currentDate = new Date();
            long diffInMillies = 0;
            Iterator<ClientHandler> clientHandlerIterator = clients.iterator();
            while (clientHandlerIterator.hasNext()) {
                ClientHandler clientHandler = clientHandlerIterator.next();
                diffInMillies = Math.abs(currentDate.getTime() - clientHandler.getLoginDate().getTime()) / 60000;
                if (diffInMillies >= 20) {
                    System.out.println("Пользователь " + clientHandler.getUsername() + " находился в чате в течение " +
Long.toString(diffInMillies) + " min");
                    clientHandler.sendMessage("Ну нельзя так долго сидеть в чате, идите работать! :)");
                    System.out.println("Отключаем пользователя " + clientHandler.getUsername());
                    Socket socket = clientHandler.getSocket();
                    DataInputStream in = clientHandler.getIn();
                    DataOutputStream out = clientHandler.getOut();
                    if (in != null) {
                        try {
                            in.close();
                        } catch (IOException e) {
```

Наткнулся на проблему, если отключаем пользователя в момент, когда программа ждет от пользователя ввод строки, `message = in.readUTF()`; то закрытие сокета приводит к exception, пришлось использовать `try catch` и скрыть этот exception. На лучшее решение нет времени (

## Возможность банить клиентов, баны могут быть по времени или перманентные (так может делать админ админ)

Осуществляется с помощью команды `/ban username min`

где `min` - на сколько минут отключаем

работает только для пользователя с правами администратора

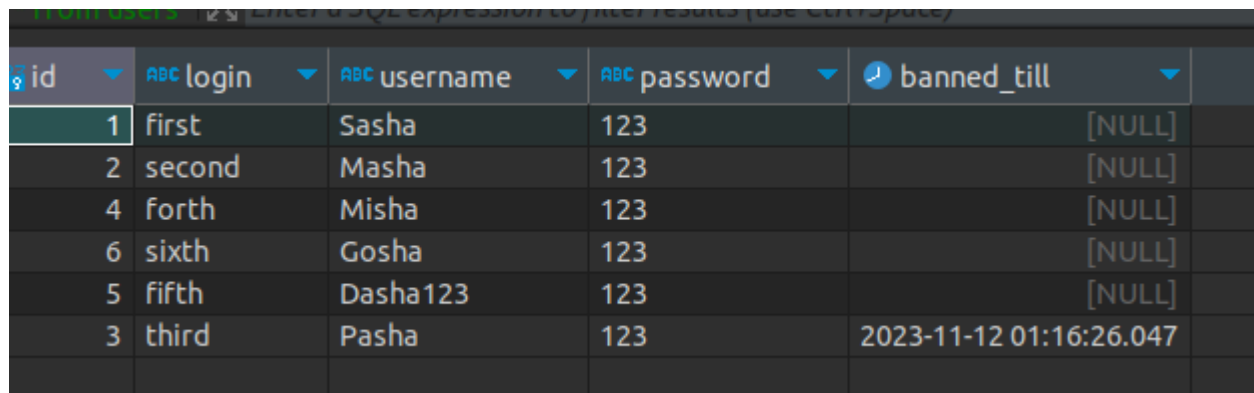
```
00:48:04 пользователь sasha подключился к чату  
/ban Pasha  
00:48:08 Ошибка ввода периода блокировки, введите количество минут, 0 - до окончания веков, отрицательное число разблокирует пользователя  
/ban Pasha 1  
00:48:33 Пользователь Pasha заблокирован в БД, результат: true  
00:48:33 Пользователь Pasha отключился  
00:48:33 Отключили пользователя:Pasha
```

для пользователя, которого отключают, это выглядит так:

```
00:48:33 Вас отключают  
Disconnect  
  
Process finished with exit code 0  
|
```

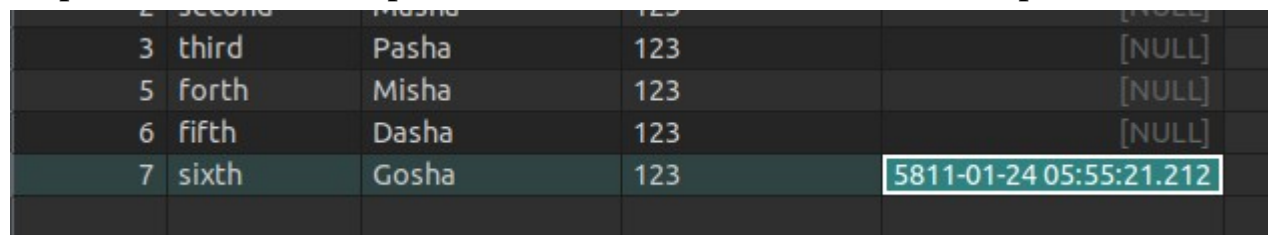
Блокировка осуществляется через метод `server.getAuthenticationProvider().banUser(bannedUser, bannedPeriod)`

- вычисляется дата и время разблокировки
- дата записывается в поле `Date bannedTill` класса `User` и БД в поле `users.banned_till` в виде даты.



id	login	username	password	banned_till
1	first	Sasha	123	[NULL]
2	second	Masha	123	[NULL]
4	forth	Misha	123	[NULL]
6	sixth	Gosha	123	[NULL]
5	fifth	Dasha123	123	[NULL]
3	third	Pasha	123	2023-11-12 01:16:26.047

Если при вводе минут ввели 0, то устанавливаем очень позднюю дату, считаем что это перманентная блокировка, ни пользователь ни админ просто не доживут до разблокировки )



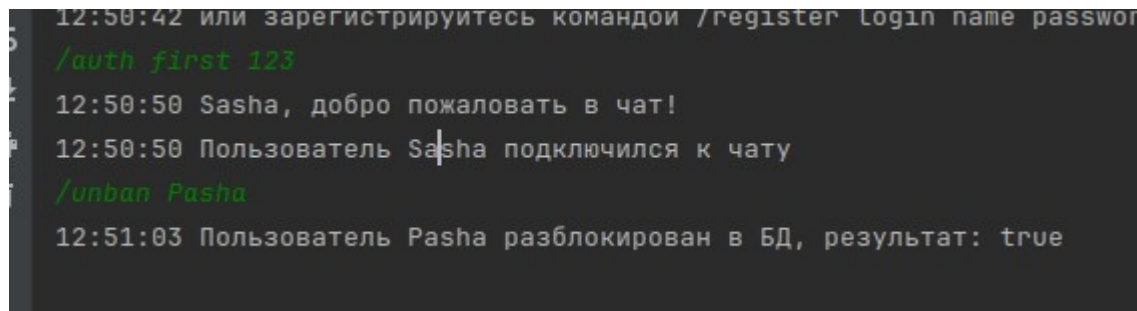
3	third	Pasha	123	[NULL]
5	forth	Misha	123	[NULL]
6	fifth	Dasha	123	[NULL]
7	sixth	Gosha	123	5811-01-24 05:55:21.212

- пользователь удаляется из чата с помощью метода `client.disconnect();`

- при попытке подключиться в методе `authenticateUser` установлена проверка заблокирован ли пользователь

```
Date currentDate = new Date();  
if (currentDate.before(user.getBannedTill())) {  
    sendMessage("Вы заблокированы до :" + user.getBannedTill());  
    break;  
}
```

Так же создана обратная команда `/unban username` позволяющая разблокировать пользователя



```
12:50:42 или зарегистрируйтесь командой /register login name password  
/auth first 123  
12:50:50 Sasha, добро пожаловать в чат!  
12:50:50 Пользователь Sasha подключился к чату  
/unban Pasha  
12:51:03 Пользователь Pasha разблокирован в БД, результат: true
```


для этого запускается тот же метод, но с отрицательным значением минут

```
server.getAuthenticationProvider().banUser(unBannedUser, -100)
```

в методе работает проверка, что если `bannedPeriod` меньше 0, то устанавливаем самую раннюю дату

```
} else if (bannedPeriod < 0) {
```

```
blockedUntilDate = new Date(1L); // 1970-01-01 03:00:00.001
```

t * from users  Enter a SQL expression to filter results (use Ctrl+Space)					
123 id	login	username	password	banned_till	
1	first	Sasha	123	[NULL]	
2	second	Masha	123	[NULL]	
4	forth	Misha	123	[NULL]	
6	sixth	Gosha	123	[NULL]	
5	fifth	Dasha123	123	[NULL]	
3	third	Pasha	123	1970-01-01 03:00:00.001	

## Остановка сервера командой /shutdown

При запуске команды /shutdown:

- Проверяется, является ли пользователь администратором

```
if (!AmIAdmin()) {  
    sendMessage("Вы не админ, нет у вас таких прав");  
    continue;  
}
```

- сервер отключается через метод server.serverShutdown();
  - устанавливается переменная this.toExit = true;
  - вызывается новый поток, в нем через итератор перебираются и отключаются все пользователи

```
Iterator<ClientHandler> clientHandlerIterator = clients.iterator();  
while (clientHandlerIterator.hasNext()) {  
    ClientHandler clientHandler = clientHandlerIterator.next();  
    clientHandler.sendMessage("Сервер отключается, все на выход!");  
    System.out.println("Отключаем пользователя " + clientHandler.getUsername());  
    Socket socket = clientHandler.getSocket();  
    DataInputStream in = clientHandler.getIn();  
    DataOutputStream out = clientHandler.getOut();  
    if (in != null) {  
        try {  
            in.close();  
        } catch (IOException e) {  
            System.out.println("in exception ");  
            throw new RuntimeException(e);  
        }  
    }  
    if (out != null) {  
        try {  
            out.close();  
        }  
    }  
}
```

```

    } catch (IOException e) {
        System.out.println("out exception ");
        throw new RuntimeException(e);
    }
}

if (socket != null) {
    try {
        socket.close();
    } catch (IOException e) {
        System.out.println("socket exception ");
        throw new RuntimeException(e);
    }
}

}
clientHandlerIterator.remove();
}

```

пришлось в этом же методе закрывать потоки и сокет, т.к. иначе при вызове метода `disconnect()` возникал `exception`

- закрываем серверный сокет, слушающий подключение новых пользователей методом -  
`new Socket(serverSocket.getInetAddress(), serverSocket.getLocalPort()).close();`

- после отключения всех пользователей и серверного сокета в памяти остается поток отключающий пользователей по времени в 20 мин, (запускаемый в методе `server.start()`)

он работает в цикле `while (!toExit)` и использует засыпание на минуту.

Для ускорения выхода из цикла была использована синхронизация с методами `wait()` и `notifyAll()`

```

public void serverShutdown() {
    this.toExit = true;
}

```



```
new Thread() -> {  
    try {  
        synchronized (mon) {  
.....здесь отключение пользователей  
        }  
        mon.notifyAll(); // будим поток в методе start(), что бы быстрее вышел из цикла  
    }  
}.....
```

На этом работа сервера завершается.

Вывод лога в консоли сервера:

```
command = /shutdown  
a new serverSocket.accept()  
Отключаем пользователя Sasha  
Отключаем сервер по команде пользователя  
Сервер отключается, выход  
Пользователя отключили, поэтому exception  
ClientHandler Thread disconnect  
  
Process finished with exit code 0
```

## SQL команды для создания таблиц с данными пользователей

```
--создание таблиц
create table public.users (
    id serial4 not null,
    login varchar(255),
    username varchar(255),
    password varchar(255),
    banned_till timestamp default null,
    constraint users_pk primary key (id)
)

create table public.roles (
    id serial4 not null,
    roles_name varchar(255) ,
    constraint roles_pk primary key (id)
)

create table public.user_to_roles (
    user_id integer,
    role_id integer,
    primary key(user_id,role_id),
    constraint user_id_fk foreign key (user_id) references public.users(id) MATCH SIMPLE ON UPDATE NO ACTION ON DELETE
CASCADE,
    constraint role_id_fk foreign key (role_id) references public.roles(id)
)

--наполнение таблиц данными
insert into public.users ( login, username, password) values ('first', 'Sasha', 123);
insert into public.users ( login, username, password) values ('second', 'Masha', 123);
insert into public.users ( login, username, password) values ('third', 'Pasha', 123);
insert into public.users ( login, username, password) values ('forth', 'Misha', 123);

insert into public.roles ( roles_name) values ('ADMIN');
insert into public.roles ( roles_name) values ('USER');

insert into public.user_to_roles (user_id, role_id) values (1,1);
insert into public.user_to_roles (user_id, role_id) values (2,2);
insert into public.user_to_roles (user_id, role_id) values (3,2);
```

```
insert into public.user_to_roles (user_id, role_id) values (4,2);
```

```
--проверка данных
```

```
select * from users
```

```
select * from roles r
```

```
select * from user_to_roles ur
```

```
select u.username, r.roles_name from users u, roles r, user_to_roles ur where ur.user_id = u.id and ur.role_id = r.id and  
r.roles_name = 'ADMIN'
```

```
select u.id as id, u.login as login, u.username as username, r.roles_name, u.password from users u, roles r,  
user_to_roles ur where ur.user_id = u.id and ur.role_id = r.id
```

```
--изменение данных
```

```
update public.users set username = 'Pasha' where id = 3
```

```
--удаление пользователя
```

```
delete from public.users where id = (select id from users u where u.login = 'second')
```

```
--делаем пользователя админом
```

```
update user_to_roles ur
```

```
set role_id = (select id from roles r where r.roles_name = 'ADMIN')
```

```
where user_id = (select id from users u where u.login = 'second')
```

```
-- при пересоздании таблиц используем следующие скрипты:
```

```
drop table public.user_to_roles
```

```
drop table public.roles
```

```
drop table public.users
```

## **Что хотелось бы сделать, но не хватило времени:**

- Добавить логирование, пока логирование производится простым выводом в консоль сервера.
- Сделать так, что бы приложение брало настройки из конфигурационного файла, например порт подключения, время работы пользователя в чате (20 мин) и т.д.
- Продумать и запретить одновременный вход под одним и тем же пользователем в систему с нескольких терминалов.
- Запустить еще один отдельный поток, слушать консольный ввод на сервере, добавить возможность входа в консоль под пользователем.