

Дисциплина «УПРАВЛЕНИЕ ПРОГРАММНЫМИ ПРОЕКТАМИ»

ОТЧЕТ О ПРАКТИЧЕСКИХ РАБОТАХ

Выполнил:

Кузнецов Дмитрий Александрович

Соколов Егор Александрович

студент группы 143

электронная почта

cuznetsov.dmitri2003@yandex.ru

faositb@yandex.ru

Проверил:

Пруцков Александр Викторович

д-р техн. наук, профессор кафедры ВПМ

Рязань 2023

1. Описание сетевой информационной системы

1.1. Цель работы

Цель работы – сетевая информационная система, позволяющая автоматизировать совместную работу экспертов и команд.

1.2. Концептуальная и логическая модели базы данных

На основании анализа предметной области были выделены следующие сущности, представленные в таблице 1.

Таблица 1

Название сущности	Назначение	Таблица в БД
Пользователь	Перечень всех лиц, зарегистрированных в СИС	USER
Команда	Перечень всех созданных команд	TEAM
Сообщение	Сообщения между пользователями	MESSAGE
Роль	Роль пользователя, позволяющая выполнять определённые действия в СИС	ROLE
Группа ролей	Обобщение ролей	ROLE_GROUP
Санкция	Санкции, наложенные на пользователя	SANCTION
Тип санкции	Определяет тип накладываемой санкции	SANCTION_TYPE
Настройка	Настройки СИС	SETTING
Взаимодействие командой	с Действие пользователя по отношению к команде	TEAM_INTERACT
Тип взаимодействия командой	с Тип действия пользователя по отношению к команде	TEAM_INTERACT_TYPE
Прикрепление сообщения	Прикрепление конкретного сообщения к конкретной команде	MESSAGE_ATTACHING
Удаленное сообщение	Перечень удаленных сообщений пользователей, которые могут быть восстановлены	DELETED_MESSAGE
Назначение на роль	История назначений на роль пользователей СИС	ROLE_ASSIGNMENT

Концептуальная схема базы данных представлена на рисунке 1.

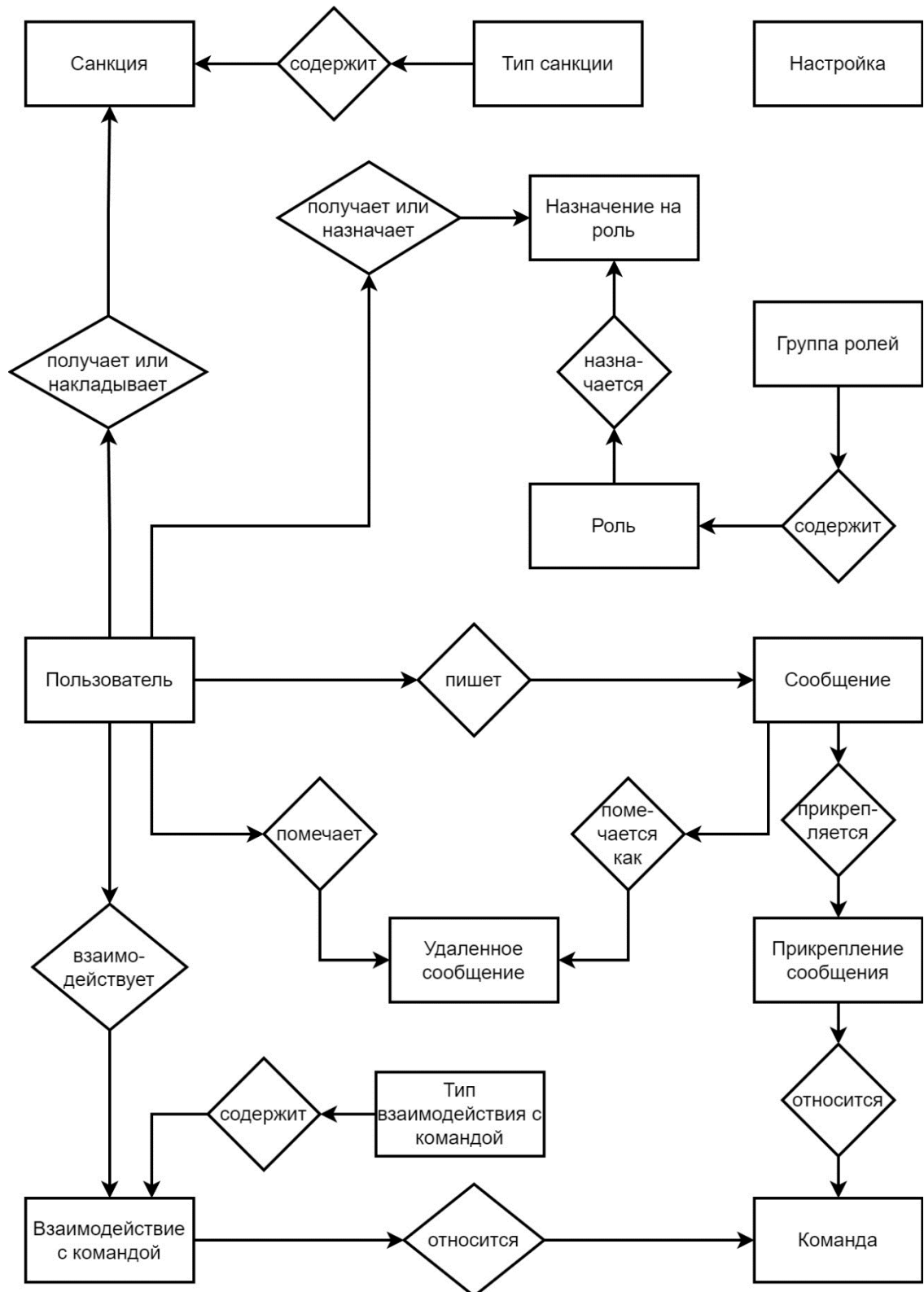


Рисунок 1 – Концептуальная модель базы данных

Удаление пользователя	Администраторы	Удаление строки в таблице USER и в связанных с ней таблицах
Назначение пользователя на роль	Администраторы	Создание строки в таблице ROLE_ASSIGNMENT
Редактирование данных пользователя	Администраторы	Изменение строки в таблице USER
Редактирование настроек	Администраторы	Изменение строк в таблице SETTING
Наложение санкций на пользователя	Модераторы	Создание строки в таблице SANCTION, обозначающей наложение санкций
Снятие санкций с пользователя	Модераторы	Создание строки в таблице SANCTION, обозначающей снятие санкций
Удаление сообщения пользователя	Модераторы	Создание строки в таблице DELETED_MESSAGE, обозначающей пометку о удалении сообщения пользователя
Создание команды	Пользователи (обычные пользователи)	Добавление строки в таблицу TEAM, добавление строки в таблицу TEAM_INTERACT с информацией о вступлении капитана в команду
Присоединение к команде	Пользователи	Добавление строки в TEAM_INTERACT
Выход из команды	Пользователи	Создание строки в TEAM_INTERACT
Отправка сообщения	Пользователи	Добавление строки в MESSAGE, прикрепление сообщения к команде путем добавления строки в MESSAGE_ATTACHING
Удаление своих сообщений	Пользователи	Создание строки в DELETED_MESSAGE
Отказ от консультирования	Пользователи (Эксперт, капитан команды)	Создание строки в TEAM_INTERACT
Исключение из команды	Пользователи (Капитан команды)	Создание строки в TEAM_INTERACT

1.4. Отчеты, предоставляемые администратору, модератору и другим группам пользователей

Группам пользователей выдаются различные отчеты (таблица 3).

Таблица 3 – Отчеты для групп пользователей

Группа пользователей	Наименование отчета
Администраторы	Вывод информации о всех учетных записях
Администраторы	Вывод информации о истории назначения на роль по конкретной учетной записи
Модераторы	Вывод всех учётных записей, являющихся пользователями
Модераторы	Статистика блокировок и разблокировок пользователей данным модератором
Модераторы	Вывод пользователей, заблокированных N или более раз
Эксперт	Вывод команд, консультирующихся у данного эксперта
Эксперт	Вывод N команд, которым данный эксперт написал больше всего сообщений
Эксперт	Вывод команд, отказавшихся от консультирования у данного эксперта
Пользователь	Вывод сообщений, которые удалил не сам пользователь
Пользователь	История санкций, наложенных на данного пользователя
Пользователь	Вывод количества оставленных и удалённых сообщений у данного пользователя

2. Требуемое программное обеспечение

Для функционирования разработанной сетевой информационной системы требуется программное обеспечение (Таблица 4).

Таблица 4 – Требуемое программное обеспечение

Название	Версия	Назначение
JDK	11.0.18	Среда выполнения СИС
Oracle XE Express Edition	18c	СУБД, хранящая базу данных СИС
Apache Tomcat	9.0.80	ПО для запуска веб-сервера СИС
Библиотека JDBC	18.21.0.0	ПО для доступа к базе данных из СИС
Библиотека JSTL	1.2	ПО для встраивания кода на Java в веб-приложение СИС.

3. Список используемых запросов на языке SQL

3.1. Запросы создания таблиц базы данных

-- Создание таблицы с пользователями

```
CREATE TABLE "USER" (
  "user_id" int generated always as identity(start with 1 increment by 1) PRIMARY KEY not NULL,
  "login" nvarchar2(30) UNIQUE not NULL,
  "password" nvarchar2(50) not NULL,
  "user_name" NVARCHAR2(50),
  "email" NVARCHAR2(50),
  "is_authorized" CHAR(1) DEFAULT '0' CHECK ("is_authorized" IN ('0','1')) not NULL
);
```

-- Создание таблицы с информацией о группах ролей

```
CREATE TABLE "ROLE_GROUP" (
  "role_group_id" int generated always as identity(start with 1 increment by 1) PRIMARY KEY not NULL,
  "role_group_name" nvarchar2(30) UNIQUE not NULL
);
```

-- Создание таблицы с перечнем всех команд

```
CREATE TABLE "TEAM" (
  "team_id" int generated always as identity(start with 1 increment by 1) PRIMARY KEY not NULL,
  "team_name" nvarchar2(50) UNIQUE not NULL
);
```

-- Создание таблицы с информацией о ролях пользователей

```
CREATE TABLE "ROLE" (
  "role_id" int generated always as identity(start with 1 increment by 1) PRIMARY KEY not NULL,
  "role_name" nvarchar2(30) UNIQUE not NULL,
  "group" int not NULL
);
```

-- Создание таблицы с историей назначения пользователей на роли

```
CREATE TABLE "ROLE_ASSIGNMENT" (
  "role_assignment_id" int generated always as identity(start with 1 increment by 1) PRIMARY KEY not NULL,
  "role" int not NULL,
  "sender" int NULL,
  "receiver" int not NULL,
  "time" date
);
```

-- Создание таблицы, хранящей настройки СИС

```
CREATE TABLE "SETTING" (
  "id" int generated always as identity(start with 1 increment by 1) PRIMARY KEY not NULL,
  "name" nvarchar2(30) not NULL UNIQUE,
  "value" int
);
```

-- Создание таблицы с историей действий пользователя по отношению к команде

```
CREATE TABLE "TEAM_INTERACT" (
  "team_interact_id" int generated always as identity(start with 1 increment by 1) PRIMARY KEY not NULL,
  "user" int not NULL,
  "type" int not NULL,
  "team" int not NULL,
  "time" date not NULL
);
```

-- Создание таблицы с сообщениями пользователей

```
CREATE TABLE "MESSAGE" (
  "message_id" int generated always as identity(start with 1 increment by 1) PRIMARY KEY not NULL,
  "data" nvarchar2(250) not NULL,
  "author" int not NULL,
  "message_time" date not NULL
);
```

-- Создание таблицы, хранящей информацию о удаленных сообщениях

```
CREATE TABLE "DELETED_MESSAGE" (
  "del_message_id" int generated always as identity(start with 1 increment by 1) PRIMARY KEY not NULL,
  "sender" int NULL,
  "message" int not NULL,
  "del_message_time" date not NULL
);
```

-- Создание таблицы с информацией о санкциях, наложенных на пользователя

```
CREATE TABLE "SANCTION" (
  "sanction_id" int generated always as identity(start with 1 increment by 1) PRIMARY KEY not NULL,
  "type" int not NULL,
  "sender" int NULL,
  "receiver" int not NULL,
  "reason" nvarchar2(100),
  "time" date not NULL
);
```

-- Создание таблицы с информацией о типах санкций


```
CREATE TABLE "SANCTION_TYPE" (
  "sanction_t_id" int generated always as identity(start with 1 increment by 1) PRIMARY KEY not NULL,
  "sanction_t_name" nvarchar2(30) UNIQUE not NULL
);
```

-- Создание таблицы с информацией о прикреплении сообщений к конкретным командам

```
CREATE TABLE "MESSAGE_ATTACHING" (
  "message_attach_id" int generated always as identity(start with 1 increment by 1) PRIMARY KEY not NULL,
  "team" int not NULL,
  "message" int not NULL
);
```

-- Создание таблицы с информацией о типах действий пользователя по отношению к команде

```
CREATE TABLE "TEAM_INTERACT_TYPE" (
  "type_id" int generated always as identity(start with 1 increment by 1) PRIMARY KEY not NULL,
  "type_name" nvarchar2(30) UNIQUE not NULL
);
```

```
ALTER TABLE "TEAM_INTERACT" ADD FOREIGN KEY ("user") REFERENCES "USER" ("user_id") ON
DELETE CASCADE;
```

```
ALTER TABLE "MESSAGE" ADD FOREIGN KEY ("author") REFERENCES "USER" ("user_id") ON
DELETE CASCADE;
```

```
ALTER TABLE "TEAM_INTERACT" ADD FOREIGN KEY ("team") REFERENCES "TEAM" ("team_id") ON
DELETE CASCADE;
```

```
ALTER TABLE "ROLE_ASSIGNMENT" ADD FOREIGN KEY ("role") REFERENCES "ROLE" ("role_id");
```

```
ALTER TABLE "ROLE_ASSIGNMENT" ADD FOREIGN KEY ("sender") REFERENCES "USER" ("user_id")
ON DELETE SET NULL;
```

```
ALTER TABLE "ROLE_ASSIGNMENT" ADD FOREIGN KEY ("receiver") REFERENCES "USER" ("user_id")
ON DELETE CASCADE;
```

```
ALTER TABLE "SANCTION" ADD FOREIGN KEY ("sender") REFERENCES "USER" ("user_id") ON
DELETE SET NULL;
```

```
ALTER TABLE "SANCTION" ADD FOREIGN KEY ("receiver") REFERENCES "USER" ("user_id") ON
DELETE CASCADE;
```

```
ALTER TABLE "ROLE" ADD FOREIGN KEY ("group") REFERENCES "ROLE_GROUP" ("role_group_id");
```

```
ALTER TABLE "DELETED_MESSAGE" ADD FOREIGN KEY ("message") REFERENCES "MESSAGE"
("message_id") ON DELETE CASCADE;
```

```
ALTER TABLE "DELETED_MESSAGE" ADD FOREIGN KEY ("sender") REFERENCES "USER" ("user_id")
ON DELETE SET NULL;
```

```
ALTER TABLE "SANCTION" ADD FOREIGN KEY ("type") REFERENCES "SANCTION_TYPE"
("sanction_t_id");
```

```
ALTER TABLE "MESSAGE_ATTACHING" ADD FOREIGN KEY ("message") REFERENCES "MESSAGE"
("message_id") ON DELETE CASCADE;
```

```
ALTER TABLE "MESSAGE_ATTACHING" ADD FOREIGN KEY ("team") REFERENCES "TEAM"
("team_id") ON DELETE CASCADE;
```

```
ALTER TABLE "TEAM_INTERACT" ADD FOREIGN KEY ("type") REFERENCES
"TEAM_INTERACT_TYPE" ("type_id");
```

3.2. Запросы выборки данных из таблиц базы данных

-- Вывод информации о всех учетных записях

```
select * from "USER";
```

-- Вывод истории о назначении пользователя на роли (prepared statement)

```
select * from "USER" join "ROLE_ASSIGNMENT" on "USER"."user_id" = "ROLE_ASSIGNMENT"."receiver"
where "USER"."login" = ?;
```

-- Вывод всех учетных записей, являющихся пользователями

```
select "USER".* from (("USER" join "ROLE_ASSIGNMENT" on "USER"."user_id" =
"ROLE_ASSIGNMENT"."receiver")
join "ROLE" on "ROLE_ASSIGNMENT"."role" = "ROLE"."role_id")
join "ROLE_GROUP" on "ROLE"."group" = "ROLE_GROUP"."role_group_id"
where "ROLE_GROUP"."role_group_name" = 'User';
```

-- Статистика блокировок/разблокировок

```
select * from "USER" join "SANCTION" on "USER"."user_id" = "SANCTION"."sender";
```

-- Вывод пользователей, заблокированных N или более раз (prepared statement)

```
select "USER"."user_id", count("login")
from ("USER" join "SANCTION" on "USER"."user_id" = "SANCTION"."receiver")
join "SANCTION_TYPE" ON "SANCTION_TYPE"."sanction_t_id" = "SANCTION"."type"
where "SANCTION_TYPE"."sanction_t_name" = 'Block'
group by "USER"."user_id"
having count("login") >= ?;
```

--Вывод команд, консультирующихся у данного эксперта (prepared statement)

```
select "TEAM".*
  from "TEAM" join "TEAM_INTERACT" on
    "TEAM"."team_id" = "TEAM_INTERACT"."team" join "TEAM_INTERACT_TYPE" on
    "TEAM_INTERACT"."type" = "TEAM_INTERACT_TYPE"."type_id"
  where "TEAM_INTERACT"."user" in (select "ROLE_ASSIGNMENT"."receiver"
    from "ROLE_ASSIGNMENT" join "ROLE" on "ROLE_ASSIGNMENT"."role" =
"ROLE"."role_id" join "USER" on "USER"."user_id" = "ROLE_ASSIGNMENT"."receiver"
    where "ROLE"."role_name" = 'Expert' and "USER"."login" = ?);
```

--Вывод N команд, которым данный эксперт написал больше всего сообщений (prepared statement)

```
select "TEAM"."team_name", count("MESSAGE"."author")
  from "TEAM" join "MESSAGE_ATTACHING" on
    "TEAM"."team_id" = "MESSAGE_ATTACHING"."team" join "MESSAGE" on
    "MESSAGE_ATTACHING"."message" = "MESSAGE"."message_id"
  where "MESSAGE"."author" in (select "ROLE_ASSIGNMENT"."receiver"
    from "ROLE_ASSIGNMENT" join "ROLE" on "ROLE_ASSIGNMENT"."role" =
"ROLE"."role_id" join "USER" on "USER"."user_id" = "ROLE_ASSIGNMENT"."receiver"
    where "ROLE"."role_name" = 'Expert' and "USER"."login" = ?) and rownum <= ?
 group by "TEAM"."team_name"
 order by count("MESSAGE"."author") desc;
```

-- Вывод команд, отказавшихся от консультирования у данного эксперта

```
select distinct "TEAM"."team_name"
  from "TEAM" join "TEAM_INTERACT" on "TEAM"."team_id" = "TEAM_INTERACT"."team"
    join "TEAM_INTERACT_TYPE" on "TEAM_INTERACT_TYPE"."type_id" =
"TEAM_INTERACT"."type"
  where "TEAM_INTERACT_TYPE"."type_name" = 'Expert_ejected' and "TEAM_INTERACT"."user" in
(select "USER"."user_id"
    from "USER" join "ROLE_ASSIGNMENT" on
"USER"."user_id" = "ROLE_ASSIGNMENT"."receiver"
    join "ROLE" on "ROLE"."role_id" =
"ROLE_ASSIGNMENT"."role" where "ROLE"."role_name" = 'Expert' and "USER"."user_id" = ?);
```

--Вывод сообщений, которые удалил не сам пользователь

```
select "MESSAGE".*
  from "USER" join "MESSAGE" on "USER"."user_id" = "MESSAGE"."author"
  join "DELETED_MESSAGE" on "MESSAGE"."message_id" = "DELETED_MESSAGE"."message"
  where "DELETED_MESSAGE"."sender" != "USER"."user_id" and "USER"."login" = ?;
```

-- История санкций, наложенных на данного пользователя

```
select "SANCTION".*
  from "USER" join "SANCTION" on "USER"."user_id" = "SANCTION"."receiver"
```

```
where "USER"."login" = ?;
```

```
-- Вывод количества оставленных и удалённых сообщений у данного пользователя
```

```
select count("MESSAGE"."message_id") as "count_messages"
  from "USER" join "MESSAGE" on "USER"."user_id" = "MESSAGE"."author"
 where "USER"."login" = ?;
```

```
select count("MESSAGE"."message_id") as "count_del_messages"
  from "USER" join "MESSAGE" on "USER"."user_id" = "MESSAGE"."author"
 where "MESSAGE"."message_id" in (select "message" from "DELETED_MESSAGE") and
 "USER"."login" = ?;
```

3.3. Запросы изменения данных в таблицах базы данных

```
-- Создание пользователя
```

```
INSERT INTO "USER" ("login", "password", "user_name", "email") VALUES (?, ?, ?, ?);
```

```
-- Удаление пользователя
```

```
delete from "USER"
where "user_id" = ?;
```

```
-- Назначение на роль
```

```
insert into "ROLE_ASSIGNMENT" ("role_assignment_id", "sender", "receiver", "time") values ((select "role_id"
from "ROLE" where "role_name" = ?), ?, ?, (select sysdate from dual));
```

```
-- Редактирование данных пользователя
```

```
update "USER"
set "login" = ?,
  "password" = ?,
  "is_authorized" = ?,
  "user_name" = ?,
  "email" = ?
where "USER"."user_id" = ?;
```

```
-- Редактирование настроек
```

```
update "SETTING"
set "name" = ?,
  "value" = ?
where "SETTING"."id" = ?;
```

```
-- Наложение санкций на пользователя и их снятие
```

```
INSERT INTO "SANCTION" ("type", "sender", "receiver", "reason", "time") VALUES ((select "id" from
"SANCTION_TYPE" where "sanction_t_name" = ?), ?, ?, ?, (select sysdate from dual));
```

-- Удаление сообщения пользователя

```
INSERT INTO "DELETED_MESSAGE" ("sender", "message", "del_message_time") VALUES (?, ?, (select sysdate from dual));
```

-- Создание команды

```
INSERT INTO "TEAM" ("team_name") VALUES (?);
```

```
INSERT INTO "TEAM_INTERACT" ("user", "type", "team", "time") VALUES (?, (select "type_id" from "TEAM_INTERACT_TYPE" where "type_name" = 'Join'), (select "team_id" from "TEAM" where "team_name" = ?), (select sysdate from dual));
```

-- Присоединение к команде

```
INSERT INTO "TEAM_INTERACT" ("user", "type", "team", "time") VALUES (?, (select "type_id" from "TEAM_INTERACT_TYPE" where "type_name" = 'Join'), ?, (select sysdate from dual));
```

-- Выход из команды

```
INSERT INTO "TEAM_INTERACT" ("user", "type", "team", "time") VALUES (?, (select "type_id" from "TEAM_INTERACT_TYPE" where "type_name" = 'Exit'), ?, (select sysdate from dual));
```

-- Отправка сообщения

```
INSERT INTO "MESSAGE" ("data", "author", "message_time") VALUES (?, ?, (select sysdate from dual));
```

-- Удаление своих сообщений

```
INSERT INTO "DELETED_MESSAGE" ("sender", "message", "del_message_time") VALUES (?, ?, (select sysdate from dual));
```

-- Отказ от консультирования экспертом

```
INSERT INTO "TEAM_INTERACT" ("user", "type", "team", "time") VALUES (?, (select "type_id" from "TEAM_INTERACT_TYPE" where "type_name" = 'Exit'), ?, (select sysdate from dual));
```

-- Отказ от консультирования капитаном

```
INSERT INTO "TEAM_INTERACT" ("user", "type", "team", "time") VALUES (?, (select "type_id" from "TEAM_INTERACT_TYPE" where "type_name" = 'Expert_ejected'), ?, (select sysdate from dual));
```

-- Исключение из команды

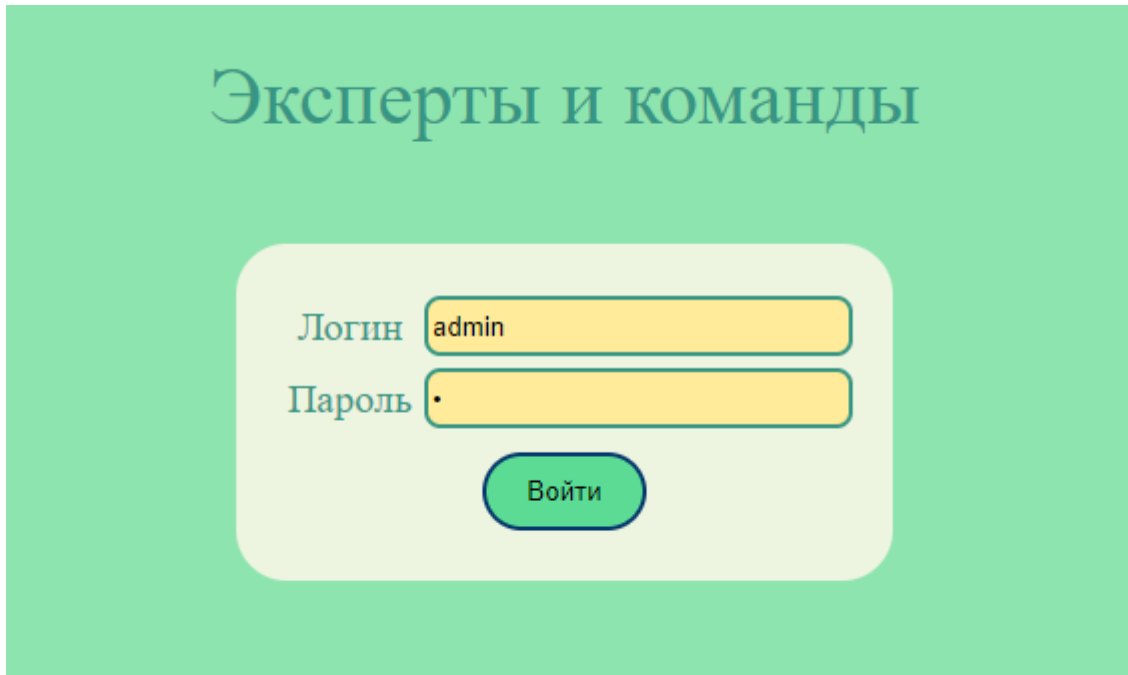
```
INSERT INTO "TEAM_INTERACT" ("user", "type", "team", "time") VALUES (?, (select "type_id" from "TEAM_INTERACT_TYPE" where "type_name" = 'Exit'), ?, (select sysdate from dual));
```

4. Инструкции по работе с сетевой информационной системой

4.1. Инструкция администратора

Перед началом работы с сетевой информационной системой с правами администратора пользователь должен войти в

систему на странице авторизации, используя соответствующий логин и пароль и нажав на кнопку «Войти» (представлено на рисунке 3).



The image shows a web interface for authorization. At the top, the title 'Эксперты и команды' is displayed in a teal font. Below the title is a light yellow rounded rectangle containing the login form. The form has two input fields: the first is labeled 'Логин' and contains the text 'admin'; the second is labeled 'Пароль' and contains a single black dot, indicating a masked password. Below these fields is a teal button with the text 'Войти'.

Рисунок 3 – Страница авторизации

В случае ввода неправильных логина и/или пароля пользователь будет перенаправлен на страницу авторизации, на которой также будет отображена надпись «Login or password incorrect». В случае успешной авторизации пользователь попадет на главную страницу администратора, позволяющую редактировать данные пользователей (представлено на рисунке 4).

The screenshot shows a web application interface for managing users. The main content area is titled 'Изменение данных пользователей' (Change user data). It contains the following form fields:

- Логин (Login): user1
- Пароль (Password): 1234567890
- Имя (Name): Userov User Userovich
- Электронная почта (Email): 123@yandex.ru
- Роль (Role): Administrator (selected from a dropdown)

Below the form fields are five buttons: 'Сохранить' (Save), 'Создать пользователя' (Create user), 'Удалить пользователя' (Delete user), 'Поиск пользователя по логину' (Search user by login), and 'Очистить форму' (Clear form).

The right sidebar, titled 'Список всех пользователей СИС' (List of all users in the system), includes a search filter 'Фильтр' and a list of users: root, admin, moderator_1, moderator_2, expert_1, expert_2, dmitry, egor, olya, tiger, mitter, cat, and admin_hater. The 'admin' user is currently selected, as indicated by the 'admin' label and 'Logout' button in the top right corner.

Рисунок 4 – Страница изменения данных пользователей

В центральной области данной страницы располагаются текстовые поля, позволяющие вводить соответственно логин пользователя, его пароль, имя, электронную почту. Также есть выпадающий список для выбора роли редактируемого пользователя (Administrator, Moderator, Expert, Common User – соответственно администратор, модератор, эксперт и обычный пользователь). Значение по умолчанию для данного списка – Administrator. Дополнительные ограничения на длины вводимых строк: для логина – не более 30 символов, для остальных полей – не более 50 (система не позволит вводить в поля новые символы по достижении ограничения).

В правой части страницы находятся список с логинами всех пользователей СИС (если пользователь был заблокирован модератором, то его логин отображается как перечёркнутый) и поле для фильтрации по ним: в списке с пользователями отображаются только те логины, в которых есть вхождение строки, введенной в это поле. Если администратор кликнет по какому-либо логину из списка, то текстовые поля заполнятся соответствующей информацией для пользователя с выбранным логином.

В верхней правой части данной страницы (и остальных страниц, рассматриваемых в инструкции) отображается логин текущего администратора и находится кнопка «Logout»,

позволяющая выйти из системы и перейти к странице авторизации.

В центральной части страницы ниже элементов для ввода информации располагаются кнопки, позволяющие непосредственно редактировать данные пользователя.

Описание функциональности кнопок:

- «Сохранить» – при нажатии на данную кнопку система сначала определяет, существует ли пользователь с указанным логином. Если существует, то вся информация, введенная администратором в редактируемые элементы, сохраняется в базе данных для пользователя с данным логином. В противном случае данные в системе не меняются. При любом исходе после нажатия данной кнопки информация, введенная в текстовые поля, исчезает, а значение в выпадающем списке с ролью устанавливается на значение по умолчанию. Дополнительное требование: пароль пользователя не должен быть пустым;
- «Создать пользователя» – при нажатии на данную кнопку система сначала определяет, существует ли пользователь с указанным логином. Если не существует, то создается новый пользователь с учетными данными, указанными в редактируемых элементах. Если же существует, то новый пользователь создан не будет. В любом случае после нажатия данной кнопки информация, введенная в текстовые поля, исчезает, а значение в выпадающем списке с ролью устанавливается на значение по умолчанию. Дополнительные требования для создания пользователя – его логин и пароль не должны быть пустыми;
- «Удалить пользователя» – при нажатии на данную кнопку пользователь с указанным логином удаляется из системы, если он был в ней зарегистрирован. В любом случае после нажатия данной кнопки информация, введенная в текстовые поля, исчезает, а значение в выпадающем списке с ролью устанавливается на значение по умолчанию. При попытке администратора удалить свою учетную запись из системы она не удаляется;
- «Поиск пользователя по логину» – при нажатии на данную кнопку редактируемые элементы заполняются информацией о найденном пользователе, если пользователь с заданным логином существует. В противном случае информация, находящаяся в текстовых полях,

исчезает, а значение в выпадающем списке с ролью устанавливается на значение по умолчанию;

- «Очистить форму» – при нажатии на данную кнопку информация, находящаяся в текстовых полях, исчезает, а значение в выпадающем списке с ролью устанавливается на значение по умолчанию.

Для перехода администратора на другие предназначенные для него страницы существуют ссылки на них в левой боковой панели (левая часть интерфейса, представленная на рисунке 4), при нажатии на которые администратор попадает на выбранную страницу.

Для редактирования настроек СИС существует страница «Настройки СИС», представленная на рисунке 5.




Рисунок 5 - Страница для редактирования настроек СИС

На данной странице располагаются два редактируемых числовых поля которые отображают и позволяют устанавливать значения соответственно для максимального количества человек в одной команде и максимального количества консультируемых у одного эксперта команд. После внесения необходимых изменений администратору нужно нажать на кнопку «Сохранить» для сохранения изменений в базе данных. Ограничения на вводимые значения: они должны быть положительными и не превышать 100 (при попытке сохранения неверных параметров будет выведено предупреждающее сообщение, а сами изменения сохранены не будут).

Для просмотра отчетов существует страница «Отчеты», состоящая из двух отчетов. Сама страница и отчет по умолчанию представлены на рисунке 6.

Логин	Пароль	ФИО	email	Роль	Авторизован
root	1			Administrator	false
admin	1			Administrator	true
moderator_1	1			Moderator	false
moderator_2	1			Moderator	false
expert_1	1			Expert	false
expert_2	1			Expert	false
dmitry	1	Кузнецов Дмитрий	cuznetsov.dmitri2003@yandex.ru	Common user	false
egor	1	Соколов Егор	faositb@yandex.ru	Common user	false
olya	1			Common user	false
tiger	1			Common user	false
mitter	1			Common user	false
cat	1			Common user	false
admin_hater	1			Common user	false
killer	1			Common user	false

Рисунок 6 – Страница «Отчеты» с отчетом по умолчанию

Первый отчет, предлагаемый по умолчанию - вывод информации о всех учетных записях. В данном подразделе выводится таблица с информацией о логине, пароле, ФИО, адресе электронной почты, роли и статусе авторизации (true – авторизован, иначе false) всех пользователей СИС.

Для перехода ко другому отчету или для возврата на первый на правой части страницы с отчетами существует панель «Доступные отчеты», на которой располагаются ссылки на все доступные для данного пользователя отчеты.

В оставшемся отчете содержится информация о истории назначения на роль по конкретной учетной записи. Для ее получения необходимо в текстовом поле рядом с меткой «login» ввести логин искомого пользователя и нажать на кнопку «Найти». Тогда в таблице, располагающейся ниже, появятся строки с информацией о логине пользователя, факте установки роли, времени ее установки и логине администратора, который назначил роль. После нажатия на кнопку «Найти» текстовое поле с искомым логином очищается. Результат поиска истории

назначения на роль для пользователя с логином «olya» представлен на рисунке 7.

The screenshot shows a web application interface with a green header and sidebar. The main content area is titled "Вывод информации о истории назначения на роль по конкретной учетной записи". It features a search form with a label "login =", a text input field containing "Введите логин пользователя", and a green "Найти" button. Below the form is a table with the following data:

Пользователь	Роль	Кто назначил	Время назначения
olya	Common user	admin	2023-09-23 00:00:00.0

The sidebar on the left contains links: "Главное меню", "Настройки СИС", and "Отчеты". The top right corner shows the user "admin" and a "Logout" button. The right sidebar contains links: "Доступные отчеты", "Вывод информации о всех учетных записях", and a link to the current page: "Вывод информации о истории назначения на роль по конкретной учетной записи".

Рисунок 7 – Поиск истории назначения на роль для пользователя с логином «olya» во втором отчете, доступном администратору

4.2. Инструкция модератора

Перед началом работы с сетевой информационной системой с правами модератора пользователь должен войти в систему на странице авторизации, используя соответствующий логин и пароль и нажав на кнопку «Войти» (представлено на рисунке 8).

The screenshot shows a login page titled "Эксперты и команды". It features a login form with a "Логин" label and a text input field containing "moderator". Below it is a "Пароль" label and a password input field with a yellow background and a single dot. A green "Войти" button is positioned below the password field.

Рисунок 8 – Страница авторизации

В случае ввода неправильных логина и/или пароля пользователь будет перенаправлен на страницу авторизации, на которой также будет отображена надпись «Login or password»

incorrect». В случае успешной авторизации пользователь попадет на главную страницу модератора, представленную на рисунке 9.

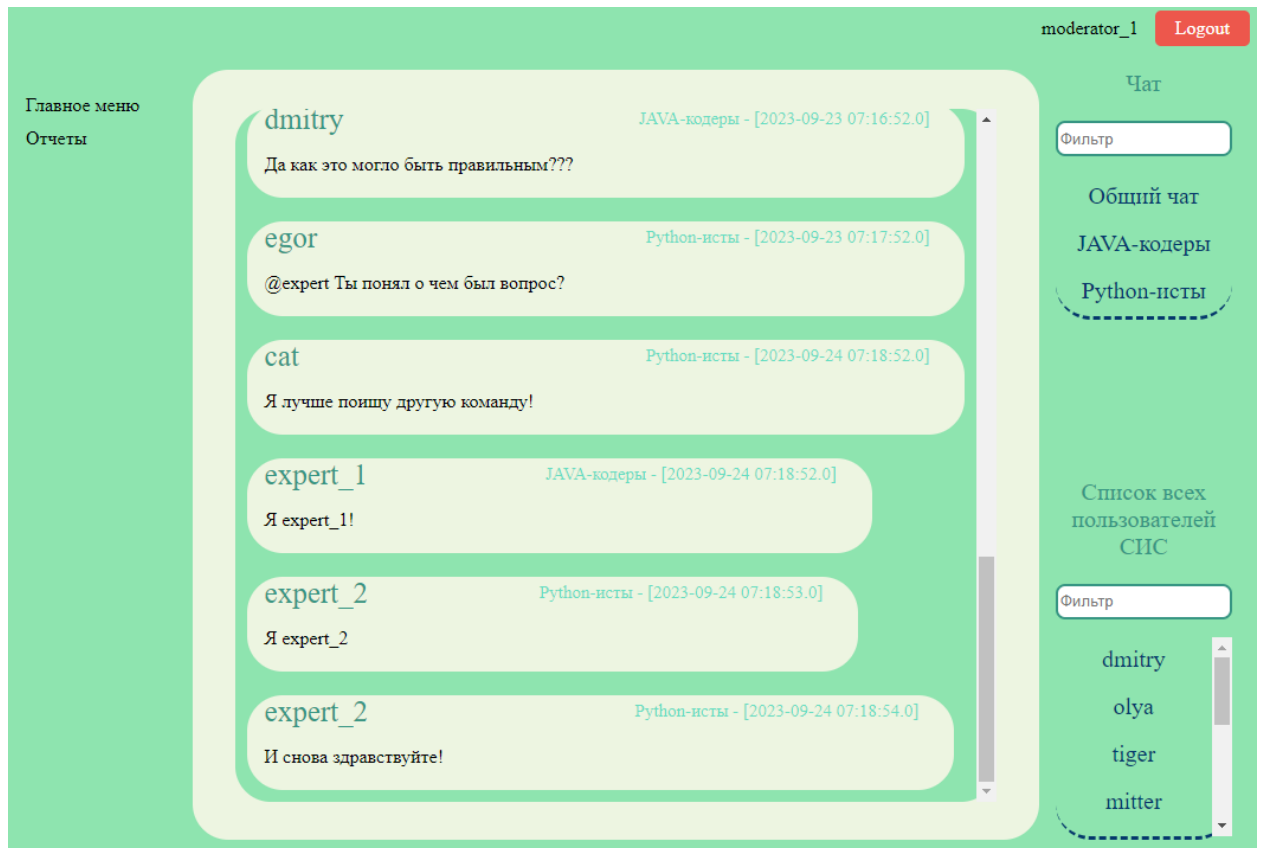


Рисунок 9 – Главная страница модератора

В центральной области данной страницы располагается чат пользователей, который может просматривать модератор. По умолчанию модератору виден общий чат (все сообщения пользователей из всех команд). Для выбора конкретного чата в правой верхней части страницы присутствует список с наименованиями команд и поле для фильтрации по ним: в списке отображаются только те команды, в которых есть вхождение строки, введенной в это поле. При нажатии на наименование команды в этом списке открывается соответствующий чат.

В правой части страницы находятся список с логинами всех непривилегированных пользователей СИС, то есть всех пользователей за исключением администраторов и модераторов (если пользователь был заблокирован модератором, то его логин отображается как перечёркнутый) и поле для фильтрации по ним: в списке с пользователями отображаются только те логины, в которых есть вхождение строки, введенной в это поле.

В верхней правой части данной страницы (и остальных страниц, рассматриваемых в инструкции) отображается логин текущего модератора и находится кнопка «Logout», позволяющая выйти из системы и перейти к странице авторизации.

Для удаления сообщения модератор должен навести на него указатель мыши. Тогда на сообщении появится красный крестик, при нажатии на который сообщение будет удалено для остальных пользователей СИС, его смогут просматривать только модераторы и сам пользователь, оставивший данное сообщение, причем внешний вид сообщения станет тусклым. Внешний вид сообщения непосредственно перед его удалением представлен на рисунке 10.

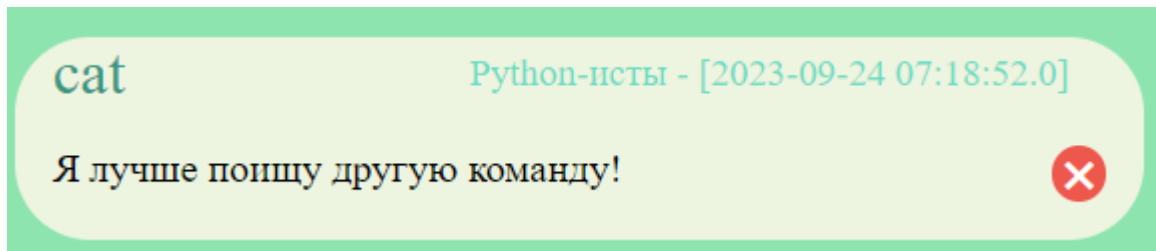


Рисунок 10 – Вид сообщения непосредственно перед его удалением модератором

Для восстановления удаленного сообщения модератор должен навести на него указатель мыши. Тогда на сообщении появится значок закругленной стрелочки, при нажатии на которую сообщение будет восстановлено для остальных пользователей СИС, а его внешний вид перестанет быть тусклым. Внешний вид сообщения непосредственно перед его восстановлением представлен на рисунке 11.

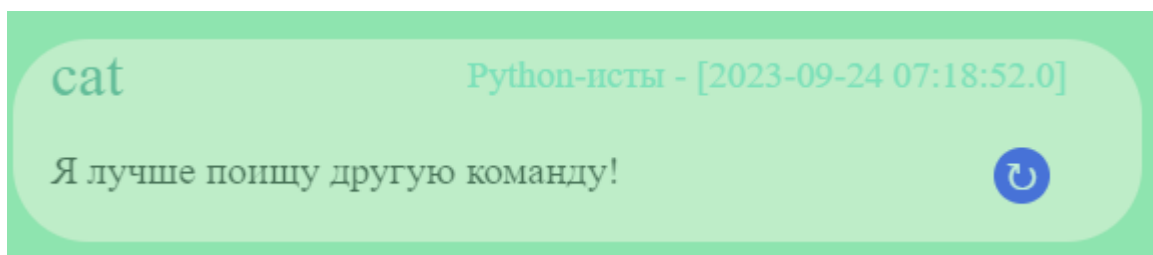


Рисунок 11 – Вид сообщения непосредственно перед его восстановлением модератором

Для блокирования или разблокирования пользователя нужно навести указатель мыши на его логин либо в чате, либо в списке пользователей, тогда появится всплывающее окно, представленное на рисунке 12.

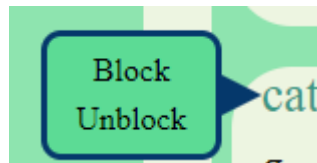


Рисунок 12 – Окно блокирования/разблокирования пользователя

Для блокирования или разблокирования пользователя нужно выбрать соответственно пункты «Block» и «Unblock». Появится окно, позволяющее указать причину блокировки или разблокировки и подтвердить действие. Данное окно представлено на рисунке 13.

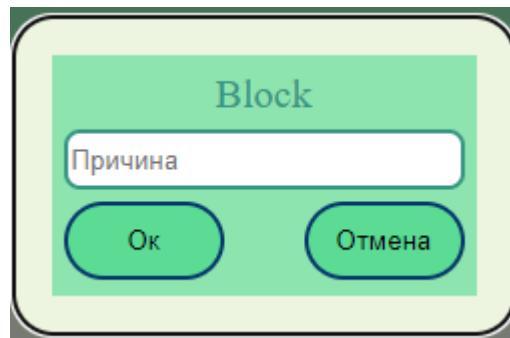


Рисунок 13 – Окно с указанием причины блокировки или разблокировки и подтверждением действия

В случае нажатия на кнопку «Ок» будет установлена причина блокировки или разблокировки (она может быть и пустой), действие блокировки/разблокировки будет выполнено, а в случае нажатия на кнопку «Отмена» действие не будет выполнено.

Для подгрузки новых сообщений в чат модератору следует обновить страницу.

Для просмотра отчетов существует страница «Отчеты», состоящая из трех отчетов. Сама страница и отчет по умолчанию представлены на рисунке 14.

moderator_1 Logout

Главное меню
Отчеты

Статистика блокировок и разблокировок пользователей данным модератором

Тип санкции	Пользователь	Причина	Время
Block	mitter	Спам	2023-09-23 10:30:01.0
Unblock	mitter		2023-09-23 12:30:52.0
Block	cat	Неадекватное поведение, не разблокировать.	2023-12-07 18:58:29.0
Unblock	cat	Добрый человек	2023-12-07 18:58:39.0
Block	admin_hater	Неадекватное поведение	2023-09-23 07:09:52.0

Доступные отчеты

Вывод учетных записей, являющихся пользователями

[Статистика блокировок и разблокировок пользователей данным модератором](#)

Вывод пользователей, заблокированных N или более раз

Рисунок 14 – Страница «Отчеты» с отчетом по умолчанию

Первый отчет, предлагаемый по умолчанию - статистика блокировок и разблокировок пользователей данным модератором. В данном подразделе выводится таблица с информацией о типе санкции (Block – блокировка, Unblock – разблокировка), пользователе (его логин), причине блокировки или разблокировки, а также времени совершения этого действия.

Для перехода к другим отчетам на правой части страницы с отчетами существует панель «Доступные отчеты», на которой располагаются ссылки на все доступные для данного модератора отчеты.

В отчете «Вывод учетных записей, являющихся пользователями» модератору показывается таблица со следующей информацией о непривилегированных пользователях СИС (т.е. обо всех пользователях, кроме администраторов и модераторов): логин, ФИО, email, статус авторизации (true – авторизован, иначе false). Скриншот данного отчета представлен на рисунке 15.

moderator_1
Logout

Главное меню

Отчеты

Вывод учетных записей, являющихся пользователями

Логин	ФИО	email	Авторизован
dmitry	Кузнецов Дмитрий	cuznetsov.dmitri2003@yandex.ru	false
olya			false
tiger			false
mitter			false
cat			false
admin_hater			false
killer			false
expert_2			false
expert_1			false
egor	Соколов Егор	faositb@yandex.ru	false

Доступные отчеты

[Вывод учетных записей, являющихся пользователями](#)

Статистика блокировок и разблокировок пользователей данным модератором

[Вывод пользователей, заблокированных N или более раз](#)

Рисунок 15 – Отчет для модератора с информацией о учетных записях, являющихся пользователями

В отчете «Вывод пользователей, заблокированных N или более раз» отображается числовое поле, в которое нужно ввести число N. После этого модератор должен нажать на кнопку «Найти». В результате в таблице, располагающейся чуть ниже, будут отображены логины пользователей, удовлетворяющие условию. Дополнительное ограничение: число N должно быть положительным и не превышать 10000 (при попытке поиска данных с неверным параметром будет выведено предупреждающее сообщение). Скриншот данного отчета представлен на рисунке 16.

The screenshot shows a web interface for a moderator. At the top right, the user is logged in as 'moderator_1' with a 'Logout' button. On the left, there is a sidebar with links for 'Главное меню' (Main menu) and 'Отчеты' (Reports). The main content area is titled 'Вывод пользователей, заблокированных N или более раз' (Output of users blocked N or more times). Below the title, there is a search form with 'N =' followed by a text input containing the number '2' and a green 'Найти' (Find) button. Below the search form, a green box displays the results: 'Пользователь' (User) and 'cat'. On the right side, there is a sidebar titled 'Доступные отчеты' (Available reports) with links for 'Вывод учетных записей, являющихся пользователями' (Output of accounts that are users), 'Статистика блокировок и разблокировок пользователей данным модератором' (Statistics of blocking and unblocking of users by this moderator), and a link for 'Вывод пользователей, заблокированных N или более раз' (Output of users blocked N or more times), which is the current page.

Рисунок 16 – Отчет модератора «Вывод пользователей, заблокированных N или более раз»

4.3. Инструкция эксперта

Эксперт является непривилегированным пользователем СИС. Перед началом работы с сетевой информационной системой эксперт должен войти в систему на странице авторизации, используя соответствующий логин и пароль и нажав на кнопку «Войти» (представлено на рисунке 17).

The screenshot shows a login page titled 'Эксперты и команды' (Experts and teams). It features a login form with two input fields: 'Логин' (Login) with the text 'expert' and 'Пароль' (Password) with a single dot. Below the password field is a green 'Войти' (Login) button.

Рисунок 17 – Страница авторизации

В случае ввода неправильных логина и/или пароля пользователь будет перенаправлен на страницу авторизации, на которой также будет отображена надпись «Login or password incorrect». В случае успешной авторизации пользователь попадет на главную страницу эксперта, представленную на рисунке 18).

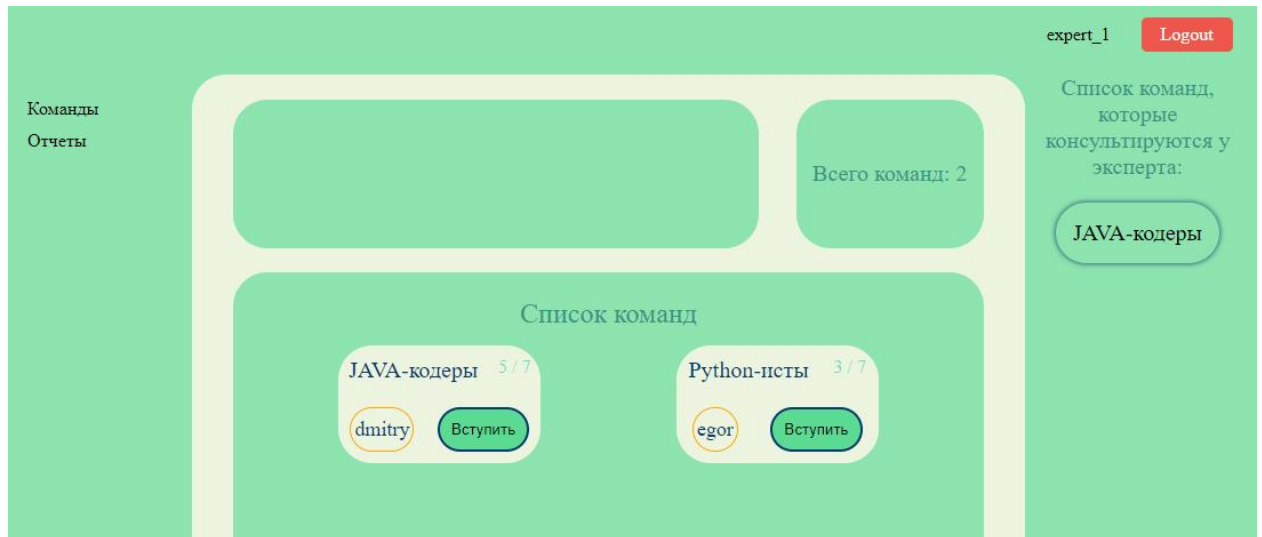


Рисунок 18 – Главная страница эксперта

В центральной области данной страницы располагается список существующих команд (отображается название команды, текущее количество участников, максимальное количество человек в команде, логин капитана команды). Для прикрепления эксперта к команде он должен нажать на кнопку «Вступить», которая располагается рядом с названием выбранной команды. В случае успеха (у выбранной команды нет прикрепленного эксперта, у текущего эксперта количество консультируемых команд меньше ограничения, установленного администратором) эксперт прикрепляется к команде.

В правой части страницы находятся список команд, консультирующихся у данного эксперта.

В верхней правой части данной страницы (и остальных страниц, рассматриваемых в инструкции) отображается логин текущего эксперта и находится кнопка «Logout», позволяющая выйти из системы и перейти к странице авторизации.

В левой части страницы находятся ссылки на другие страницы, доступные эксперту.

Для консультирования выбранной команды эксперт должен кликнуть на ее название в правой части страницы, тогда он

попадет на страницу с чатом с данной командой (представлено на рисунке 19).

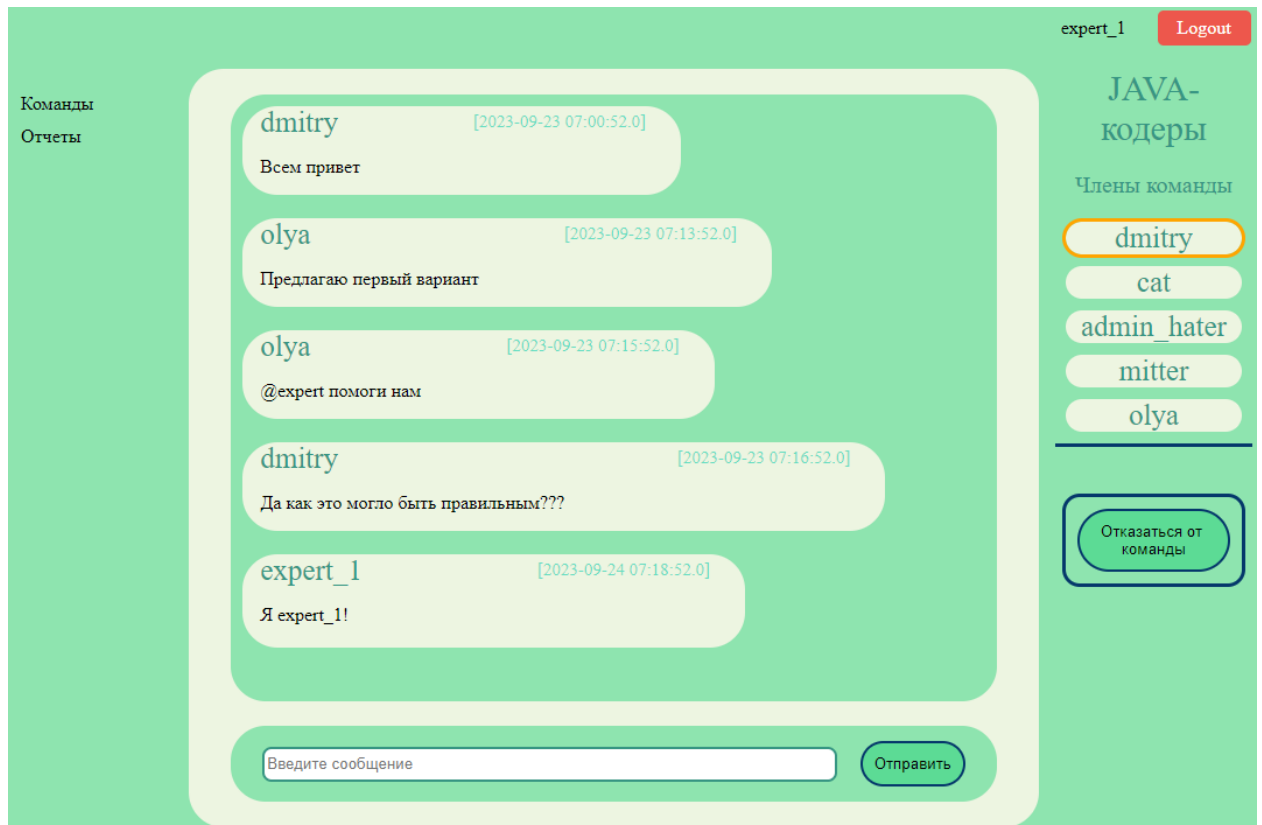


Рисунок 19 – Страница эксперта с чатом с выбранной командой

В центральной части страницы находятся сообщения в чате (отображается само сообщение, его автор и время отправки), для написания сообщения эксперту следует набрать его в текстовом поле, расположенном ниже чата и нажать на кнопку «Отправить», располагающуюся рядом. Ограничение на длину сообщения: она должна быть ненулевой и не превышать 250 символов, иначе будет выведено предупреждающее сообщение, а введенный текст не будет отправлен. После отправки сообщения чат обновится и автоматически пролистнется в его низ, к последним оставленным сообщениям. Для подгрузки новых сообщений в чате нужно обновить страницу.

Эксперт также может удалять и восстанавливать сообщения, оставленные им. Для удаления сообщения в чате эксперт должен навести на него указатель мыши. Тогда на сообщении появится красный крестик, при нажатии на который сообщение будет удалено для остальных пользователей СИС, его смогут просматривать только модераторы и сам эксперт,

оставивший данное сообщение, причем внешний вид сообщения станет тусклым. Внешний вид сообщения непосредственно перед его удалением представлен на рисунке 20.

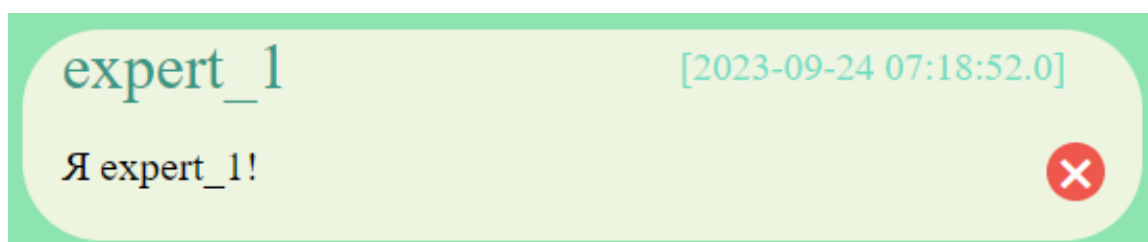


Рисунок 20 – Вид сообщения непосредственно перед его удалением

Для восстановления удаленного сообщения эксперт должен навести на него указатель мыши. Тогда на сообщении появится значок закругленной стрелочки, при нажатии на которую сообщение будет восстановлено для остальных пользователей СИС, а его внешний вид перестанет быть тусклым. Внешний вид сообщения непосредственно перед его восстановлением представлен на рисунке 21.

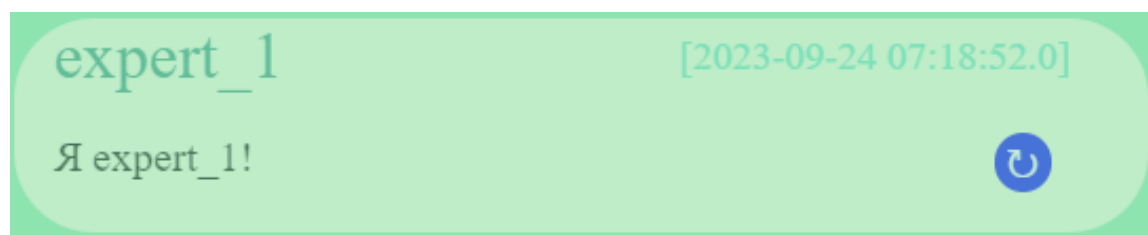


Рисунок 21 – Вид сообщения непосредственно перед его восстановлением

В правой части страницы с чатом с командой располагаются список с логинами участников команды (капитан находится наверху списка) и кнопка «Отказаться от команды», при нажатии на которую эксперт отказывается от консультирования команды и система перенаправляет его на главную страницу, которая была представлена на рисунке 18.

Эксперту также доступны отчеты. Для перехода к ним нужно нажать на ссылку «Отчеты» на левой панели. Отчет, который отображается по умолчанию, позволяет выводить наименования команд, отказавшихся от консультирования у данного эксперта (представлено на рисунке 22).

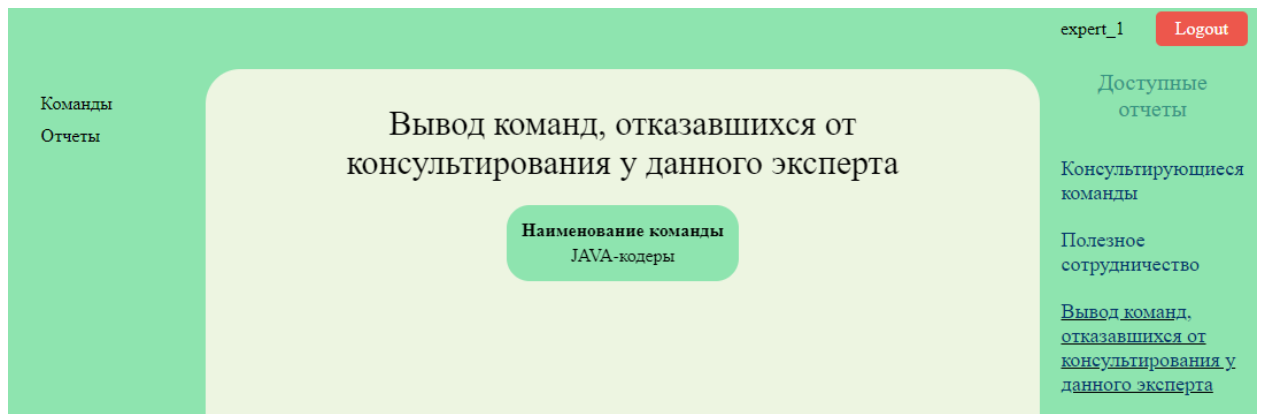


Рисунок 22 – Отчет по умолчанию для эксперта

Другой отчет называется «Полезное сотрудничество» и позволяет вывести N команд, которым данный эксперт писал больше всего сообщений (представлено на рисунке 23).

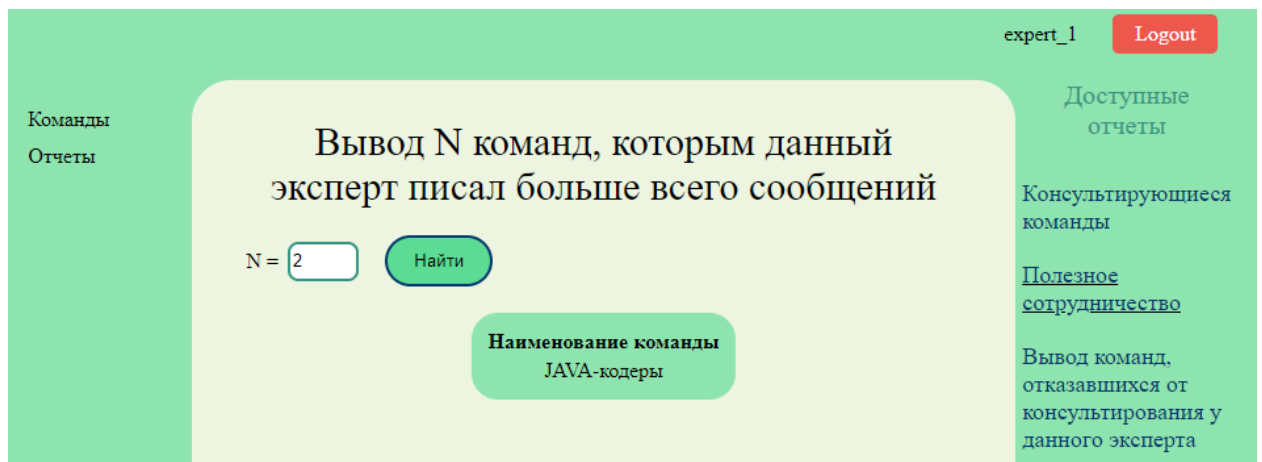


Рисунок 23 – отчет «Полезное сотрудничество»

Для вывода наименований команд эксперт должен ввести число N в текстовое поле и нажать на кнопку «Найти». Дополнительное ограничение: число N должно быть положительным и не превышать 10000 (при попытке поиска данных с неверным параметром будет выведено предупреждающее сообщение).

Третий доступный эксперту отчет позволяет вывести наименования команд, отказавшихся от консультирования у данного эксперта (представлено на рисунке 24).

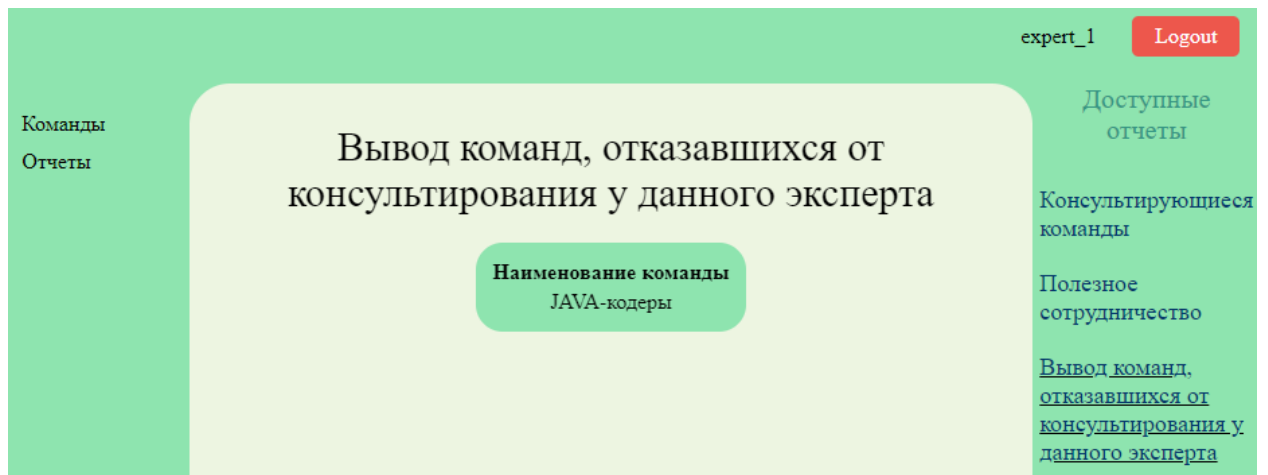


Рисунок 24 – Третий отчет, доступный эксперту

4.4. Инструкция обычного пользователя

Роль обычного пользователя (далее в инструкции – пользователь) является непривилегированной. Перед началом работы с сетевой информационной системой пользователь должен войти в систему на странице авторизации, используя соответствующий логин и пароль и нажав на кнопку «Войти» (представлено на рисунке 25).

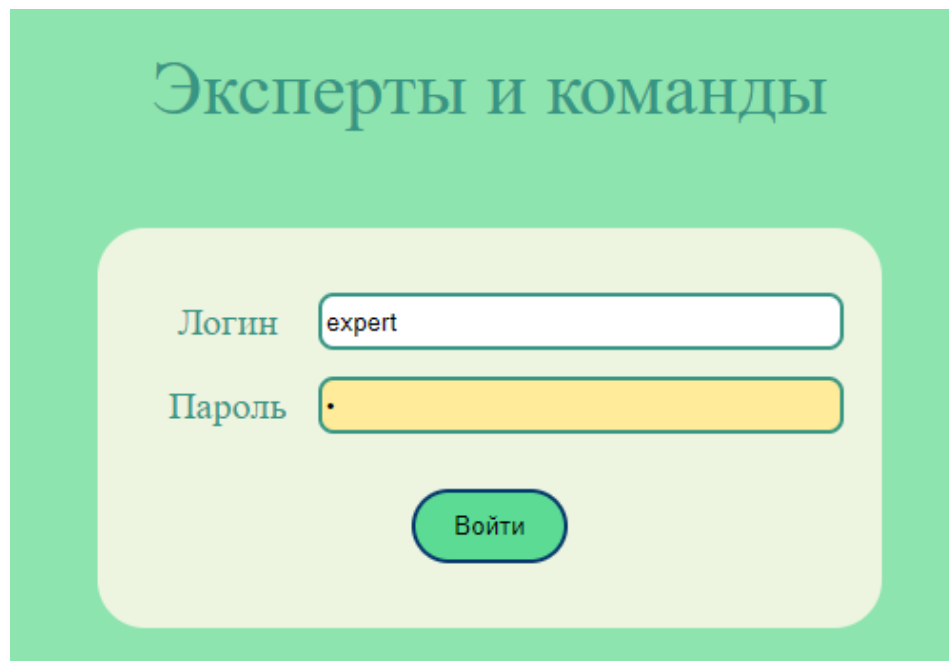


Рисунок 25 – Страница авторизации

В случае ввода неправильных логина и/или пароля пользователь будет перенаправлен на страницу авторизации, на которой также будет отображена надпись «Login or password incorrect». В случае успешной авторизации пользователь попадет на главную страницу, представленную на рисунке 26.



Рисунок 26 – Главная страница пользователя

В центральной области данной страницы располагается список существующих команд (отображается название команды, текущее количество участников, максимальное количество человек в команде, логин капитана команды). Для вступления пользователя в команду он должен нажать на кнопку «Вступить», которая располагается рядом с названием выбранной команды. В случае успеха (текущее число человек в команде меньше ограничения, установленного администратором) пользователь вступает в команду. Дополнительное ограничение – если пользователь является капитаном какой-либо команды, то он не может вступить в другую команду. Если пользователь состоял в какой-либо команде, то он выходит из нее.

Выше расположен подраздел для создания команды. Пользователь может ввести название команды в текстовое поле и нажать на кнопку «Создать». Если пользователь на данный момент не был капитаном команды и команды с создаваемым наименованием не существует, то будет создана новая команда, пользователь станет ее капитаном и его перенаправит на страницу чата с командой, причем если пользователь уже состоял в какой-либо команде, то он выходит из нее. Описание страницы чата с командой будет дано позднее.

В правой части главной страницы находится ссылка на текущую команду пользователя. При нажатии на нее пользователя перенаправляет на страницу чата с командой.

В верхней правой части страницы (и остальных страниц, рассматриваемых в инструкции) отображается логин текущего

пользователя и находится кнопка «Logout», позволяющая выйти из системы и перейти к странице авторизации.

В левой части страницы находятся ссылки на другие страницы, доступные пользователю: «Моя команда» для перехода в чат с командой пользователя (ссылка отсутствует, если пользователь не вступил ни в одну команду), «Команды» для перехода на страницу выбора и создания команд, «Покинуть команду» для выхода из команды и перехода на главную страницу (ссылка отсутствует, если пользователь не вступил ни в одну команду, либо если пользователь является капитаном), «Отчеты» для перехода к отчетам.

Вид страницы чата с командой для капитана команды представлен на рисунке 27.

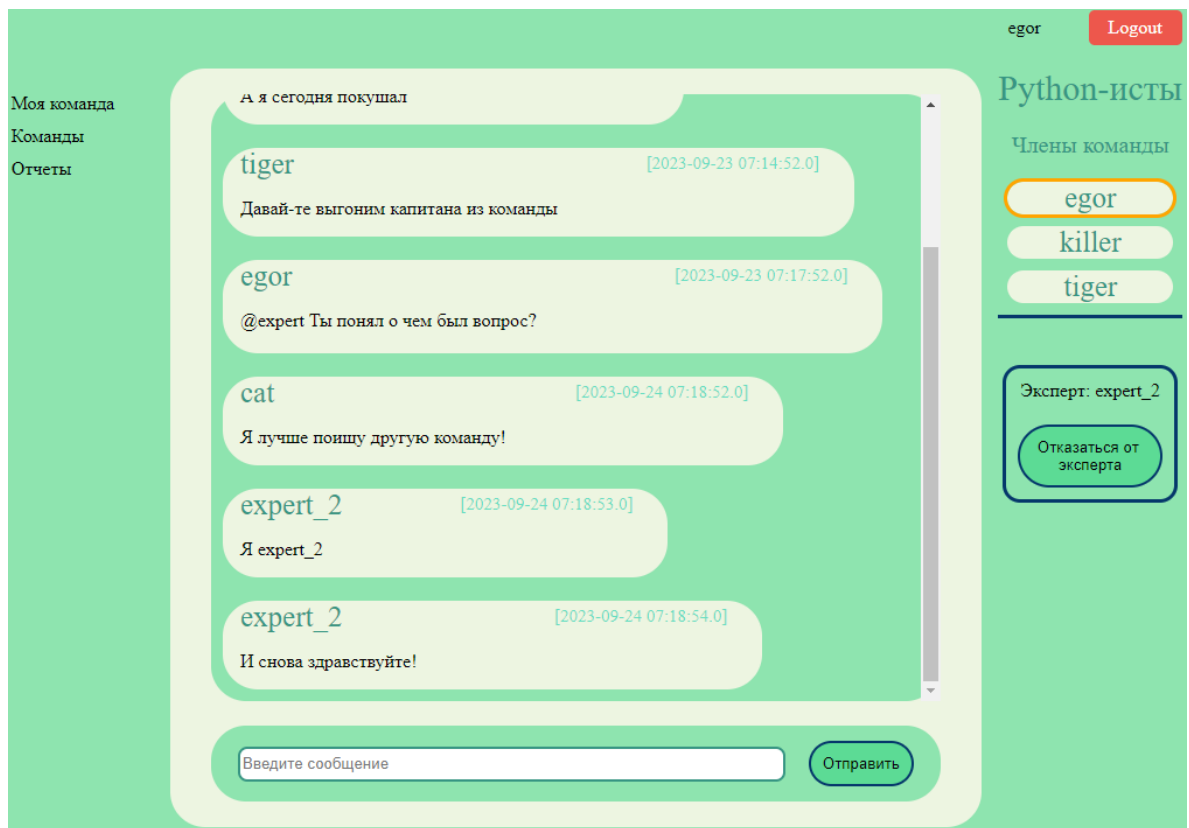


Рисунок 27 – Страница с чатом с командой для капитана команды

В центральной части страницы находятся сообщения в чате (отображается само сообщение, его автор и время отправки), для написания сообщения пользователю следует набрать его в текстовом поле, расположенном ниже чата и нажать на кнопку «Отправить», располагающуюся рядом. Ограничение на длину сообщения: она должна быть ненулевой и не превышать 250 символов, иначе будет выведено предупреждающее

сообщение, а введенный текст не будет отправлен. После отправки сообщения чат обновится и автоматически пролистнется в его низ, к последним оставленным сообщениям. Для подгрузки новых сообщений в чате нужно обновить страницу.

Пользователь также может удалять и восстанавливать сообщения, оставленные им. Для удаления сообщения в чате пользователь должен навести на него указатель мыши. Тогда на сообщении появится красный крестик, при нажатии на который сообщение будет удалено для остальных пользователей СИС, его смогут просматривать только модераторы и сам пользователь, оставивший данное сообщение, причем внешний вид сообщения станет тусклым. Внешний вид сообщения непосредственно перед его удалением представлен на рисунке 28.

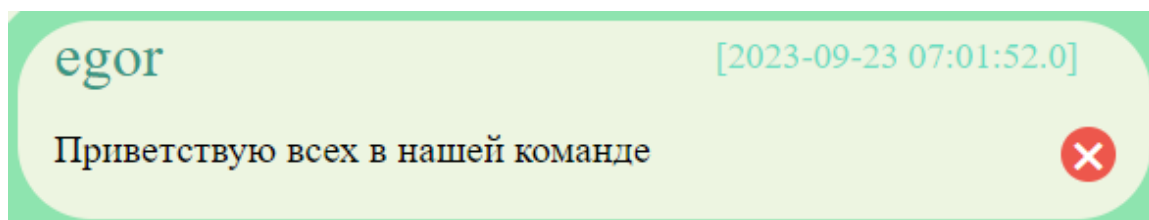


Рисунок 28 – Вид сообщения непосредственно перед его удалением пользователем

Для восстановления удаленного сообщения пользователь должен навести на него указатель мыши. Тогда на сообщении появится значок закругленной стрелочки, при нажатии на которую сообщение будет восстановлено для остальных пользователей СИС, а его внешний вид перестанет быть тусклым. Внешний вид сообщения непосредственно перед его восстановлением представлен на рисунке 29.

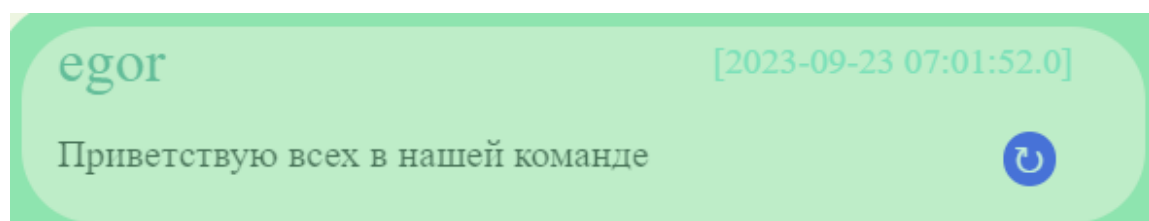


Рисунок 29 – Вид сообщения непосредственно перед его восстановлением пользователем

В правой части страницы с чатом с командой располагаются список с логинами участников команды (капитан находится наверху списка), метка с логином эксперта, консультирующего данную команду, и кнопка «Отказаться от эксперта» (доступна

только капитану), при нажатии на которую капитан отказывается от консультирования команды прикрепленным экспертом. Вид страницы с чатом с командой для пользователя, не являющегося капитаном команды, представлен на рисунке 30.

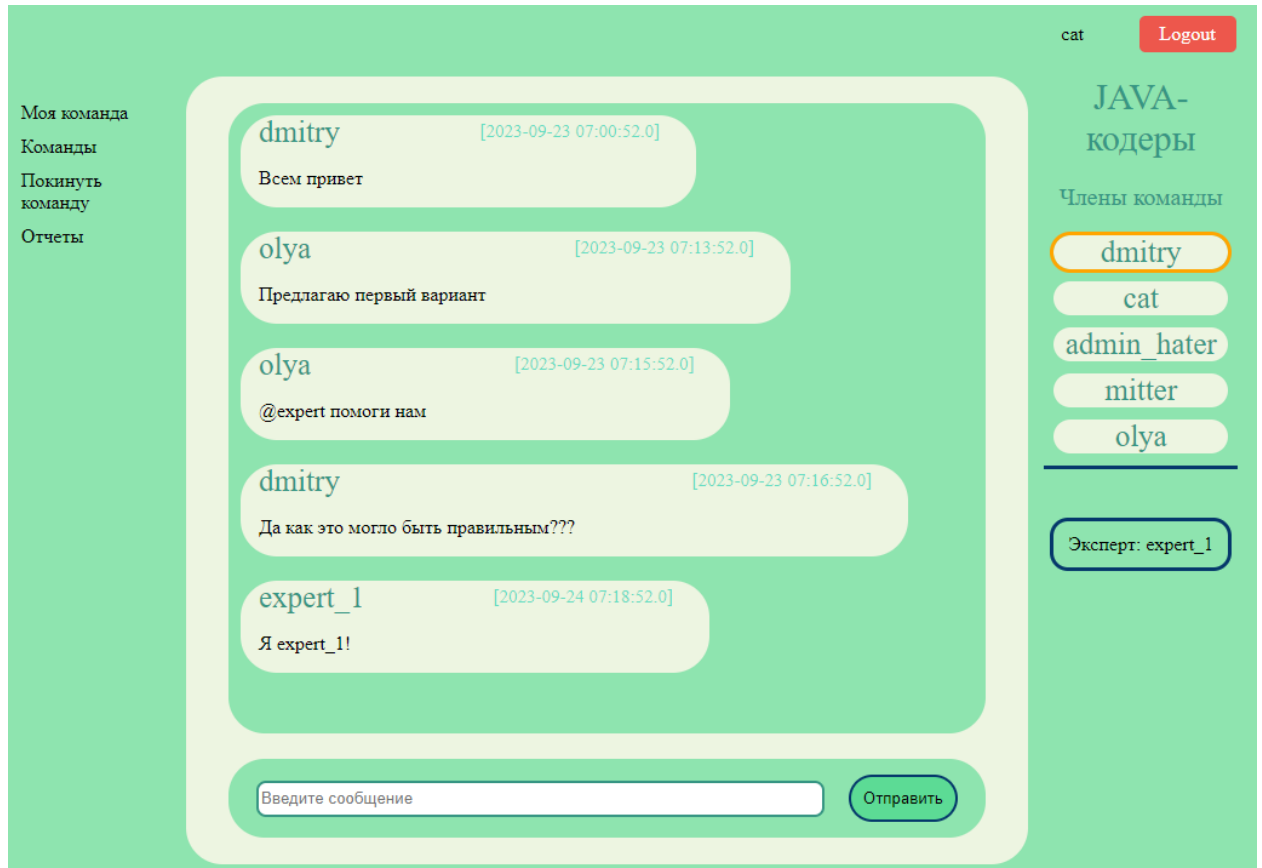


Рисунок 30 – Страница с чатом с командой для пользователя

Пользователю также доступны отчеты. Для перехода к ним нужно нажать на ссылку «Отчеты» на левой панели. Отчет, который отображается по умолчанию, позволяет выводить сообщения, которые удалил не сам пользователь: указывается текст самого сообщения и время его отправки (представлено на рисунке 31).

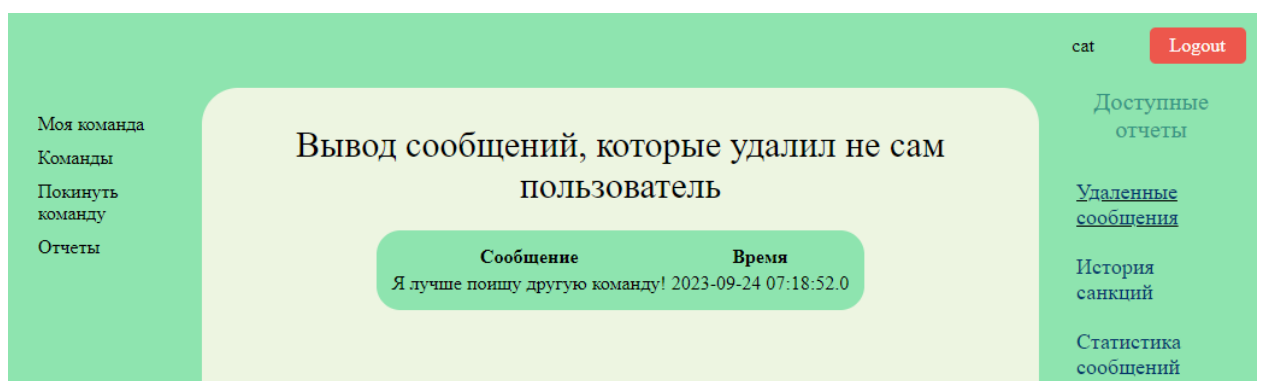


Рисунок 31 – Отчет по умолчанию для пользователя

Второй отчет называется «История санкций» и позволяет вывести историю санкций, наложенных на данного пользователя модераторами: в таблице указывается тип санкции («Block» – блокировка, «Unblock» – разблокировка), отправитель (логин модератора, назначившего санкцию), причина и время блокировки (представлено на рисунке 32).

Тип санкции	Отправитель	Причина	Время
Block	moderator_1	Неадекватное поведение, не разблокировать.	2023-12-07 18:58:29.0
Unblock	moderator_1	Добрый человек	2023-12-07 18:58:39.0
Block	moderator_2	Флуд	2023-09-23 07:10:58.0
Unblock	moderator_2	Пользователь извинился	2023-09-23 11:32:42.0

Рисунок 32 – Отчет «Полезное сотрудничество»

Третий отчет, доступный пользователю, позволяет вывести количество оставленных и удаленных сообщений у данного пользователя (представлено на рисунке 33).

Статистика
Всего сообщений: 2
Удалено сообщений: 1

Рисунок 33 – Третий отчет, доступный пользователю

5. Листинг основных классов программы с комментариями Javadoc

5.1. Класс FrontController

/**

- * The {@code FrontController} class serves as the controller for handling requests and responses.
- * It accepts requests from associated JSP pages, invokes the corresponding business logic for processing,
- * and, depending on the result, determines which JSP page to associate with the result.
- *

```

* <p>This servlet class implements the doGet and doPost methods to handle HTTP GET and POST
requests,
* respectively. The doGet method forwards the request to the appropriate JSP page using a
RequestDispatcher,
* while the doPost method redirects to the JSP page based on the result of request processing.
*

```

```

* <p>The class includes methods for processing requests, defining commands, and redirecting to the index
page.
*/

```

```

public class FrontController extends HttpServlet {

```

```

    private static final long serialVersionUID = 1L;

```

```

    /**

```

```

     * Handles GET requests, processes it, gets the page to be redirected, forwards
     * the request to the appropriate JSP page.

```

```

     * {@inheritDoc}

```

```

     *

```

```

     * @param request HttpServletRequest object.

```

```

     * @param response HttpServletResponse object.

```

```

     * @throws ServletException If errors with servlet occurs.

```

```

     * @throws IOException If an I/O error occurs.

```

```

     */

```

```

    @Override

```

```

    protected void doGet(HttpServletRequest request, HttpServletResponse response)

```

```

        throws ServletException, IOException {

```

```

        String page = processRequest(request, response);

```

```

        if (page != null) {

```

```

            RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(page);

```

```

            dispatcher.forward(request, response);

```

```

        } else {

```

```

            redirectToIndexPage(request, response);

```

```

        }

```

```

    }

```

```

    /**

```

```

     * Handles POST requests, processes it, gets the page to be redirected.

```

```

     * {@inheritDoc}

```

```

     *

```

```

     * @param request HttpServletRequest object.

```

```

     * @param response HttpServletResponse object.

```

```

     * @throws ServletException If errors with servlet occurs.

```

```

     * @throws IOException If an I/O error occurs.

```

```

*/
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String page = processRequest(request, response);
    if (page != null) {
        response.sendRedirect(request.getContextPath() + page);
    } else {
        redirectToIndexPage(request, response);
    }
}

/**
 * Processes Command and returns the page to be redirected
 * @param request HttpServletRequest object
 * @param response HttpServletResponse object
 * @return the page to be redirected
 * @throws ServletException
 * @throws IOException if the request for the could not be handled
 */
private String processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    request.setCharacterEncoding("UTF-8");
    String page = null;
    ActionFactory client = new ActionFactory();
    ActionCommand command = client.defineCommand(request);
    page = command.execute(request);
    return page;
}

/**
 * Processes redirect to index page
 * @param request HttpServletRequest object
 * @param response HttpServletResponse object
 * @throws IOException if an input or output error is detected when the servlet handles the request
 */
private void redirectToIndexPage(HttpServletRequest request, HttpServletResponse response) throws
IOException {
    String page = ConfigurationManager.getProperty("path.page.index");
    request.getSession().setAttribute("nullPage",
MessageManager.getProperty("message.nullpage"));
    response.sendRedirect(request.getContextPath() + page);
}

```

```
}
```

5.2. Интерфейс DeletedMessageDAO

```
/**
 * This interface provides methods for interacting with a database table
 * storing information about deleted messages.
 *
 * <p>Implementations of this interface should handle the storage and retrieval of deleted messages.
 */
public interface DeletedMessageDAO {

    /**
     * Adds a deleted message to the table storing deleted messages in the database.
     *
     * @param deletedMessage The deleted message to be added.
     * @throws SQLException If a database exception error occurs.
     */
    void addDeletedMessage(DeletedMessage deletedMessage) throws SQLException;

    /**
     * Restores a deleted message and removes it from the deleted messages table.
     *
     * @param deletedMessage The deleted message to be restored.
     * @throws SQLException If a database error occurs.
     */
    void removeFromDeletedMessage(DeletedMessage deletedMessage) throws SQLException;

    /**
     * Gets a deleted message.
     *
     * @param message The original deleted message to be gets.
     * @return The deleted message corresponding to the original message, or {@code null} if not found.
     * @throws SQLException If a database error occurs.
     */
    DeletedMessage getDeletedMessage(Message message) throws SQLException;

}
```

5.3. Интерфейс MessageAttachingDAO

```
/**
 * This interface provides methods for attaching messages to specific commands.
 *
 * <p>This interface defines methods for retrieving different types of message attachments (undeleted,
```

* deleted, and both) and for attaching messages to commands.

*/

public interface MessageAttachingDAO {

/**

* Retrieves all attachments of undeleted messages.

*

* @return A list of all attachments of undeleted messages.

* @throws SQLException If a database error occurs.

*/

List<MessageAttaching> getAllUndeletedMessageAttachs() throws SQLException;

/**

* Retrieves all attachments of deleted messages.

*

* @return A set of all attachments of deleted messages.

* @throws SQLException If a database error occurs.

*/

Set<MessageAttaching> getAllDeletedMessageAttachs() throws SQLException;

/**

* Retrieves all attachments of both deleted and undeleted messages.

*

* @return A list of all attachments of messages, both deleted and undeleted.

* @throws SQLException If a database error occurs.

*/

List<MessageAttaching> getAllMessageAttachs() throws SQLException;

/**

* Attaches a message to a command.

*

* @param messageAttach The message attachment to be added.

* @throws SQLException If a database error occurs.

*/

void addMessage(MessageAttaching messageAttach) throws SQLException;

}

5.4. Интерфейс MessageDAO

/**

* The interface provides methods for interacting with a database table

* storing information about messages.

*

```

* <p>This interface defines methods for retrieving different types of messages, including
* undeleted, deleted, and all messages for a team. It also includes methods to retrieve
* messages sent or deleted by a specific user.
*/

```

```

public interface MessageDAO {

```

```

    /**

```

```

        * Retrieves undeleted messages for a specific team.

```

```

        *

```

```

        * @param team The team for which to retrieve undeleted messages.

```

```

        * @return A list of undeleted messages for the specified team.

```

```

        * @throws SQLException If a database error occurs.

```

```

    */

```

```

    List<Message> getUndeletedMessagesForTeam(Team team) throws SQLException;

```

```

    /**

```

```

        * Retrieves deleted messages for a specific team along with their deletion counts.

```

```

        *

```

```

        * @param team The team for which to retrieve deleted messages.

```

```

        * @return A map where keys are deleted messages and values are their deletion counts.

```

```

        * @throws SQLException If a database error occurs.

```

```

    */

```

```

    Map<Message, Integer> getDeletedMessagesForTeam(Team team) throws SQLException;

```

```

    /**

```

```

        * Retrieves all messages for a specific team, both deleted and undeleted.

```

```

        *

```

```

        * @param team The team for which to retrieve all messages.

```

```

        * @return A list of all messages for the specified team.

```

```

        * @throws SQLException If a database error occurs.

```

```

    */

```

```

    List<Message> getAllMessagesForTeam(Team team) throws SQLException;

```

```

    /**

```

```

        * Retrieves all messages, both deleted and undeleted.

```

```

        *

```

```

        * @return A list of all messages.

```

```

        * @throws SQLException If a database error occurs.

```

```

    */

```

```

    List<Message> getAllMessages() throws SQLException;

```

```

    /**

```

```

        * Retrieves messages deleted by a specific user, excluding self-deleted messages.

```



```

*
* @param user The user for whom to retrieve deleted messages.
* @return A list of messages deleted by the specified user.
* @throws SQLException If a database error occurs.
*/
List<Message> getMessagesDeletedByNoSelfUser(User user) throws SQLException;

/**
* Retrieves the count of messages sent by a specific user.
*
* @param user The user for whom to retrieve the message count.
* @return The count of messages sent by the specified user.
* @throws SQLException If a database error occurs.
*/
int getCountMessagesSendedByUser(User user) throws SQLException;

/**
* Retrieves the count of deleted messages sent by a specific user.
*
* @param user The user for whom to retrieve the deleted message count.
* @return The count of deleted messages sent by the specified user.
* @throws SQLException If a database error occurs.
*/
int getCountDeletedMessagesSendedByUser(User user) throws SQLException;
}

```

5.5. Интерфейс RoleAssignmentDAO

```

/**
* The interface provides methods for assigning roles to users
* and retrieving information about role assignments.
*
* <p>This interface includes methods for getting role assignments for a specific user,
* retrieving all role assignments, and adding new role assignments.
*/
public interface RoleAssignmentDAO {

    /**
    * Retrieves role assignments for a specific user.
    *
    * @param user The user for whom to retrieve role assignments.
    * @return A list of role assignments for the specified user.
    * @throws SQLException If a database error occurs.
    */

```

```

List<RoleAssignment> getRoleAssignmentsForUser(User user) throws SQLException;

/**
 * Retrieves all role assignments.
 *
 * @return A list of all role assignments.
 * @throws SQLException If a database error occurs.
 */
List<RoleAssignment> getAllRoleAssignments() throws SQLException;

/**
 * Adds a new role assignment.
 *
 * @param roleAssignment The role assignment to be added.
 * @throws SQLException If a database error occurs.
 */
void addRoleAssignment(RoleAssignment roleAssignment) throws SQLException;
}

```

5.6. Интерфейс RoleDAO

```

/**
 * The interface provides methods for working with user roles.
 *
 * <p>This interface includes methods for retrieving the role of a specific user,
 * getting a role by its name, and retrieving a list of all roles.
 */
public interface RoleDAO {

    /**
     * Retrieves the role of a specific user.
     *
     * @param user The user for whom to retrieve the role.
     * @return The role of the specified user.
     * @throws SQLException If a database error occurs.
     */
    Role getUserRole(User user) throws SQLException;

    /**
     * Retrieves a role by its name.
     *
     * @param name The name of the role to retrieve.
     * @return The role with the specified name.
     * @throws SQLException If a database error occurs.
     */
}

```

```

*/
Role getRoleByName(String name) throws SQLException;

/**
 * Retrieves a list of all roles.
 *
 * @return A list of all roles.
 * @throws SQLException If a database error occurs.
 */
List<Role> getAllRoles() throws SQLException;
}

```

5.7. Интерфейс SanctionDAO

```

/**
 * The interface provides methods for working with sanctions
 * (blocking and unblocking) imposed on users.
 *
 * <p>This interface includes methods for retrieving sanctions received or sent by a specific user,
 * getting the last sanction imposed on a user, retrieving a sanction type by its name, and adding new
 * sanctions.
 */
public interface SanctionDAO {

    /**
     * Retrieves sanctions received by a specific user.
     *
     * @param receiver The user for whom to retrieve sanctions.
     * @return A list of sanctions received by the specified user.
     * @throws SQLException If a database error occurs.
     */
    List<Sanction> getUserSanctions(User receiver) throws SQLException;

    /**
     * Retrieves sanctions sent by a specific user.
     *
     * @param sender The user for whom to retrieve sanctions.
     * @return A list of sanctions sent by the specified user.
     * @throws SQLException If a database error occurs.
     */
    List<Sanction> getSanctionsByUser(User sender) throws SQLException;

    /**
     * Retrieves the last sanction imposed on a specific user.

```

```

*
* @param user The user for whom to retrieve the last sanction.
* @return The last sanction imposed on the specified user.
* @throws SQLException If a database error occurs.
*/
Sanction getLastUserSanction(User user) throws SQLException;

/**
 * Retrieves a sanction type by its name.
 *
 * @param name The name of the sanction type to retrieve.
 * @return The sanction type with the specified name.
 * @throws SQLException If a database error occurs.
 */
SanctionType getSanctionTypeByName(String name) throws SQLException;

/**
 * Adds a new sanction.
 *
 * @param sanction The sanction to be added.
 * @throws SQLException If a database error occurs.
 */
void addSanction(Sanction sanction) throws SQLException;
}

```

5.8. Интерфейс SettingDAO

```

/**
 * The interface works with the settings of a network information system.
 *
 * <p>This interface includes methods for retrieving a list of settings and updating settings.
 */
public interface SettingDAO {

    /**
     * Retrieves a list of settings for the network information system.
     *
     * @return A list of Setting objects representing the settings.
     * @throws SQLException If a database error occurs.
     */
    List<Setting> getSetting() throws SQLException;

    /**

```

```

    * Updates a setting in the network information system.
    *
    * @param setting The Setting object representing the setting to be updated.
    * @throws SQLException If a database error occurs.
    */
    void setSetting(Setting setting) throws SQLException;
}

```

5.9. Интерфейс TeamDAO

```

/**
 * The interface provides methods for working with teams.
 *
 * <p>This interface includes methods for retrieving information about teams,
 * such as getting all teams, getting a team by name or ID, retrieving teams consulted
 * by an expert, getting the N best-cooperated teams with an expert, retrieving teams
 * ejected by an expert, getting teams associated with a specific user, retrieving
 * the count of team members, adding a new team, and deleting a team.
 */
public interface TeamDAO {

    /**
     * Retrieves information about all teams.
     *
     * @return A map where keys are teams, and values are additional information
     *         (the name of the command and count of members).
     * @throws SQLException If a database error occurs.
     */
    Map<Team, Map<String, Integer>> getAllTeam() throws SQLException;

    /**
     * Retrieves a team by its name.
     *
     * @param name The name of the team to retrieve.
     * @return The team with the specified name.
     * @throws SQLException If a database error occurs.
     */
    Team getTeamByName(String name) throws SQLException;

    /**
     * Retrieves a team by its id.
     *
     * @param id The ID of the team to retrieve.
     * @return The team with the specified id.
     */
}

```

```
* @throws SQLException If a database error occurs.
*/
```

```
Team getTeamById(int id) throws SQLException;
```

```
/**
 * Retrieves teams consulted by a specific expert.
 *
 * @param expert The expert for whom to retrieve teams.
 * @return A list of teams consulted by the specified expert.
 * @throws SQLException If a database error occurs.
 */
```

```
List<Team> getTeamsConsultedByExpert(User expert) throws SQLException;
```

```
/**
 * Retrieves the N best-cooperated teams with a specific expert.
 *
 * @param expert The expert for whom to retrieve teams.
 * @param N The number of teams to retrieve.
 * @return A map where keys are teams, and values are cooperation scores or counts.
 * @throws SQLException If a database error occurs.
 */
```

```
Map<Team, Integer> getNTeamsBestCooperatedExpert(User expert, int N) throws SQLException;
```

```
/**
 * Retrieves teams ejected by a specific expert.
 *
 * @param expert The expert for whom to retrieve teams.
 * @return A list of teams ejected by the specified expert.
 * @throws SQLException If a database error occurs.
 */
```

```
List<Team> getTeamsEjectedExpert(User expert) throws SQLException;
```

```
/**
 * Retrieves teams associated with a specific user.
 *
 * @param user The user for whom to retrieve teams.
 * @return A list of teams associated with the specified user.
 * @throws SQLException If a database error occurs.
 */
```

```
List<Team> getTeamsForUser(User user) throws SQLException;
```

```
/**
 * Retrieves the count of members in a specific team.
```

```

*
* @param team The team for which to retrieve the member count.
* @return The count of members in the specified team.
* @throws SQLException If a database error occurs.
*/
int getCountTeamMembers(Team team) throws SQLException;

/**
* Adds a new team.
*
* @param team The team to be added.
* @throws SQLException If a database error occurs.
*/
void addTeam(Team team) throws SQLException;

/**
* Deletes a team.
*
* @param team The team to be deleted.
* @throws SQLException If a database error occurs.
*/
void deleteTeam(Team team) throws SQLException;
}

```

5.10.Интерфейс TeamInteractDAO

```

/**
* The interface manages the interaction between a user and a team.
*
* <p>This interface includes methods for retrieving team interactions for a specific user,
* getting all team interactions, retrieving a team interact type by its name, adding new team interactions,
* and deleting team interactions for a user.
*/
public interface TeamInteractDAO {

    /**
    * Retrieves team interactions for a specific user.
    *
    * @param user The user for whom to retrieve team interactions.
    * @return A list of team interactions for the specified user.
    * @throws SQLException If a database error occurs.
    */
    List<TeamInteract> getTeamInteractsForUser(User user) throws SQLException;
}

```

```

/**
 * Retrieves all team interactions.
 *
 * @return A list of all team interactions.
 * @throws SQLException If a database error occurs.
 */
List<TeamInteract> getAllTeamInteracts() throws SQLException;

/**
 * Retrieves a team interact type by its name.
 *
 * @param name The name of the team interact type to retrieve.
 * @return The team interact type with the specified name.
 * @throws SQLException If a database error occurs.
 */
TeamInteractType getTeamInteractTypeByName(String name) throws SQLException;

/**
 * Adds a new team interact.
 *
 * @param teamInteract The team interact to be added.
 * @throws SQLException If a database error occurs.
 */
void addTeamInteract(TeamInteract teamInteract) throws SQLException;

/**
 * Deletes team interactions for a specific user.
 *
 * @param user The user for whom to delete team interactions.
 * @throws SQLException If a database error occurs.
 */
void deleteTeamInteractsForUser(User user) throws SQLException;
}

```

5.11. Интерфейс UserDAO

```

/**
 * The {@code UserDAO} interface provides methods for working with tables storing user data.
 *
 * <p>This interface includes methods for retrieving users by ID or login, getting the expert for a team,
 * retrieving a list of all users, getting a list of unprivileged users, retrieving a list of blocked users
 * more than N times, getting the user list for a specific team, retrieving a map of users with their roles,
 * getting the team captain for a team, adding a new user, updating user information, deleting a user, and
 * getting a set of blocked users.

```



```

*/
public interface UserDao {

    /**
     * Retrieves a user by their ID.
     *
     * @param id The ID of the user to retrieve.
     * @return The user with the specified ID.
     * @throws SQLException If a database error occurs.
     */
    User getUserById(int id) throws SQLException;

    /**
     * Retrieves a user by their login.
     *
     * @param login The login of the user to retrieve.
     * @return The user with the specified login.
     * @throws SQLException If a database error occurs.
     */
    User getUserByLogin(String login) throws SQLException;

    /**
     * Retrieves the expert for a specific team.
     *
     * @param team The team for which to retrieve the expert.
     * @return The expert for the specified team.
     * @throws SQLException If a database error occurs.
     */
    User getExpertForTeam(Team team) throws SQLException;

    /**
     * Retrieves a list of all users.
     *
     * @return A list of all users.
     * @throws SQLException If a database error occurs.
     */
    List<User> getUsers() throws SQLException;

    /**
     * Retrieves a list of unprivileged users.
     *
     * @return A list of unprivileged users.
     * @throws SQLException If a database error occurs.
     */

```

```

*/
List<User> getUnprivilegedUsers() throws SQLException;

/**
 * Retrieves a list of users who have been blocked more than N times.
 *
 * @param N The number of times a user must be blocked to be included in the list.
 * @return A list of users blocked more than N times.
 * @throws SQLException If a database error occurs.
 */
List<User> getBlockedUsersMoreNTimes(int N) throws SQLException;

/**
 * Retrieves the user list for a specific team.
 *
 * @param team The team for which to retrieve the user list.
 * @return A list of users associated with the specified team.
 * @throws SQLException If a database error occurs.
 */
List<User> getTeamUserList(Team team) throws SQLException;

/**
 * Retrieves a map of users with their roles.
 *
 * @return A map where keys are users, and values are their roles.
 * @throws SQLException If a database error occurs.
 */
Map<User, Role> getUsersWithRole() throws SQLException;

/**
 * Retrieves the team captain for a specific team.
 *
 * @param team The team for which to retrieve the team captain.
 * @return The team captain for the specified team.
 * @throws SQLException If a database error occurs.
 */
User getTeamCapitan(Team team) throws SQLException;

/**
 * Adds a new user.
 *
 * @param user The user to be added.
 * @throws SQLException If a database error occurs.

```

```

*/
void addUser(User user) throws SQLException;

/**
 * Updates user information.
 *
 * @param user The user with updated information.
 * @throws SQLException If a database error occurs.
 */
void updateUser(User user) throws SQLException;

/**
 * Deletes a user.
 *
 * @param user The user to be deleted.
 * @throws SQLException If a database error occurs.
 */
void deleteUser(User user) throws SQLException;

/**
 * Retrieves a set of blocked users.
 *
 * @return A set of blocked users.
 * @throws SQLException If a database error occurs.
 */
public Set<User> getBlockedUsers() throws SQLException;
}

```

5.12. Класс AbstractEntity

```

/**
 * The {@code AbstractEntity} class serves as a superclass for all entity classes related to database tables.
 * It contains a single attribute, id, which uniquely identifies the entity in the database.
 *
 * <p>This abstract class provides methods for getting and setting the id attribute, as well as implementing
 * equals and hashCode methods based on the id for proper comparison and hashing in collections.
 */
public abstract class AbstractEntity {

    private int id;

    /**
     * Constructs an AbstractEntity without initialized id.
     */

```

```

protected AbstractEntity() {

}

/**
 * Constructs an AbstractEntity with the specified id.
 *
 * @param id The id to set for the entity.
 */
protected AbstractEntity(int id) {
    this.setId(id);
}

/**
 * Gets the id of the entity.
 *
 * @return The id of the entity.
 */
public int getId() {
    return id;
}

/**
 * Sets the id of the entity.
 *
 * @param id The id to set for the entity.
 */
public void setId(int id) {
    this.id = id;
}

/**
 * Implements equals method to compare objects
 * {@inheritDoc}
 *
 * @param obj The object to be compared
 */
@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;

```

```

        if (getClass() != obj.getClass())
            return false;
        AbstractEntity other = (AbstractEntity) obj;
        return this.getId() == other.getId();
    }

    /**
     * Implements hashCode method to get the hash of the object
     * {@inheritDoc}
     */
    @Override
    public int hashCode() {
        return this.getId();
    }
}

```

5.13. Класс DeletedMessage

```

/**
 * The {@code DeletedMessage} class represents a deleted message by the sender.
 * It extends the {@link AbstractEntity} class and includes information about the sender,
 * the deleted message, and the timestamp of deletion.
 *
 * <p>This class provides constructors for creating instances with different sets of attributes.
 * It includes getter and setter methods for accessing and modifying the sender, message, and time attributes.
 */
public class DeletedMessage extends AbstractEntity {

    private User sender;
    private Message message;
    private Timestamp time;

    /**
     * Constructs a DeletedMessage with uninitialized values.
     */
    public DeletedMessage() {
    }

    /**
     * Constructs a DeletedMessage with the specified id and uninitialized attributes.
     *
     * @param id The id to set for the DeletedMessage.
     */

```

```

public DeletedMessage(int id) {
    super(id);
}

/**
 * Constructs a DeletedMessage with the specified id, sender, message, and time.
 *
 * @param id    The id to set for the DeletedMessage.
 * @param sender The User representing the sender of the deleted message.
 * @param message The Message representing the deleted message.
 * @param time   The Timestamp representing the time of deletion.
 */
public DeletedMessage(int id, User sender, Message message, Timestamp time) {
    this(id);
    this.setSender(sender);
    this.setMessage(message);
    this.setTime(time);
}

/**
 * Gets the sender of the deleted message.
 *
 * @return The User representing the sender of the deleted message.
 */
public User getSender() {
    return sender;
}

/**
 * Sets the sender of the deleted message.
 *
 * @param sender The User representing the sender of the deleted message.
 */
public void setSender(User sender) {
    this.sender = sender;
}

/**
 * Gets the deleted message.
 *
 * @return The Message representing the deleted message.
 */
public Message getMessage() {

```

```

        return message;
    }

    /**
     * Sets the deleted message.
     *
     * @param message The Message representing the deleted message.
     */
    public void setMessage(Message message) {
        this.message = message;
    }

    /**
     * Gets the timestamp of deletion.
     *
     * @return The Timestamp representing the time of deletion.
     */
    public Timestamp getTime() {
        return time;
    }

    /**
     * Sets the timestamp of deletion.
     *
     * @param time The Timestamp representing the time of deletion.
     */
    public void setTime(Timestamp time) {
        this.time = time;
    }
}

```

5.14.Класс Message

```

/**
 * The {@code Message} class represents a message written by a User.
 * It extends the {@link AbstractEntity} class and includes information about the message data,
 * the author (User), and the timestamp of creation.
 *
 * <p>This class provides constructors for creating instances with different sets of attributes.
 * It includes getter and setter methods for accessing and modifying the data, author, and time attributes.
 */
public class Message extends AbstractEntity {

```

```

private String data;
private User author;
private Timestamp time;

/**
 * Constructs a Message with uninitialized values.
 */
public Message() {
}

/**
 * Constructs a Message with the specified id and default (uninitialized) attributes.
 *
 * @param id The id to set for the Message.
 */
public Message(int id) {
    super(id);
}

/**
 * Constructs a Message with the specified id, data, author, and time.
 *
 * @param id The id to set for the Message.
 * @param data The String representing the message data.
 * @param author The User representing the author of the message.
 * @param time The Timestamp representing the time of creation.
 */
public Message(int id, String data, User author, Timestamp time) {
    this(id);
    this.setData(data);
    this.setAuthor(author);
    this.setTime(time);
}

/**
 * Gets the data of the message.
 *
 * @return The String representing the message data.
 */
public String getData() {
    return data;
}

```



```

/**
 * Sets the data of the message.
 *
 * @param data The String representing the message data.
 */
public void setData(String data) {
    this.data = data;
}

/**
 * Gets the author of the message.
 *
 * @return The User representing the author of the message.
 */
public User getAuthor() {
    return author;
}

/**
 * Sets the author of the message.
 *
 * @param author The User representing the author of the message.
 */
public void setAuthor(User author) {
    this.author = author;
}

/**
 * Gets the timestamp of creation.
 *
 * @return The Timestamp representing the time of creation.
 */
public Timestamp getTime() {
    return time;
}

/**
 * Sets the timestamp of creation.
 *
 * @param time The Timestamp representing the time of creation.
 */
public void setTime(Timestamp time) {
    this.time = time;
}

```

```
}
```

```
}
```

5.15. Класс MessageAttaching

```
/**
```

```
* The {@code MessageAttaching} class represents the attachment of a message to a team.
```

```
* It extends the {@link AbstractEntity} class and includes information about the associated team
```

```
* and the attached message.
```

```
*
```

```
* <p>This class provides constructors for creating instances with different sets of attributes.
```

```
* It includes getter and setter methods for accessing and modifying the team and message attributes.
```

```
*/
```

```
public class MessageAttaching extends AbstractEntity {
```

```
    private Team team;
```

```
    private Message message;
```

```
/**
```

```
* Constructs a MessageAttaching with uninitialized values.
```

```
*/
```

```
    public MessageAttaching() {
```

```
    }
```

```
/**
```

```
* Constructs a MessageAttaching with the specified id and uninitialized attributes.
```

```
*
```

```
* @param id The id to set for the MessageAttaching.
```

```
*/
```

```
    public MessageAttaching(int id) {
```

```
        super(id);
```

```
    }
```

```
/**
```

```
* Constructs a MessageAttaching with the specified id, team, and message.
```

```
*
```

```
* @param id    The id to set for the MessageAttaching.
```

```
* @param team  The Team representing the associated team.
```

```
* @param message The Message representing the attached message.
```

```
*/
```

```
    public MessageAttaching(int id, Team team, Message message) {
```

```
        this(id);
```

```
        this.setTeam(team);
```

```

        this.setMessage(message);
    }

    /**
     * Gets the associated team for the message attachment.
     *
     * @return The Team representing the associated team.
     */
    public Team getTeam() {
        return team;
    }

    /**
     * Sets the associated team for the message attachment.
     *
     * @param team The Team representing the associated team.
     */
    public void setTeam(Team team) {
        this.team = team;
    }

    /**
     * Gets the attached message.
     *
     * @return The Message representing the attached message.
     */
    public Message getMessage() {
        return message;
    }

    /**
     * Sets the attached message.
     *
     * @param message The Message representing the attached message.
     */
    public void setMessage(Message message) {
        this.message = message;
    }
}

```

5.16. Класс Role

```

/**

```

```

* The {@code Role} class represents a role for separating user powers.
* It extends the {@link AbstractEntity} class and includes information about the role name
* and the role group to which it belongs.
*
* <p>This class provides constructors for creating instances with different sets of attributes.
* It includes getter and setter methods for accessing and modifying the name and group attributes.
*/

```

```

public class Role extends AbstractEntity {

```

```

    private String name;
    private RoleGroup group;

```

```

    /**
     * Constructs a Role with default (uninitialized) values.
     */

```

```

    public Role() {
    }

```

```

    /**
     * Constructs a Role with the specified id and default (uninitialized) attributes.
     *
     * @param id The id to set for the Role.
     */

```

```

    public Role(int id) {
        super(id);
    }

```

```

    /**
     * Constructs a Role with the specified id, name, and role group.
     *
     * @param id The id to set for the Role.
     * @param name The String representing the role name.
     * @param group The RoleGroup representing the role group.
     */

```

```

    public Role(int id, String name, RoleGroup group) {
        this(id);
        this.setName(name);
        this.setGroup(group);
    }

```

```

    /**
     * Gets the name of the role.
     *

```

```

    * @return The String representing the role name.
    */
    public String getName() {
        return name;
    }

    /**
     * Sets the name of the role.
     *
     * @param name The String representing the role name.
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Gets the role group to which the role belongs.
     *
     * @return The RoleGroup representing the role group.
     */
    public RoleGroup getGroup() {
        return group;
    }

    /**
     * Sets the role group to which the role belongs.
     *
     * @param group The RoleGroup representing the role group.
     */
    public void setGroup(RoleGroup group) {
        this.group = group;
    }
}

```

5.17. Класс RoleAssignment

```

/**
 * The {@code RoleAssignment} class represents the assignment of a role to a user (receiver) by a sender.
 * It extends the {@link AbstractEntity} class and includes information about the assigned role, sender,
 * receiver,
 * and the timestamp of assignment.
 *
 * <p>This class provides constructors for creating instances with different sets of attributes.

```

* It includes getter and setter methods for accessing and modifying the assigned role, sender, receiver, and time attributes.

*/

```
public class RoleAssignment extends AbstractEntity {
```

```
    private Role role;
```

```
    private User sender;
```

```
    private User receiver;
```

```
    private Timestamp time;
```

```
    /**
```

```
     * Constructs a RoleAssignment with default (uninitialized) values.
```

```
    */
```

```
    public RoleAssignment() {
```

```
    }
```

```
    /**
```

```
     * Constructs a RoleAssignment with the specified id and default (uninitialized) attributes.
```

```
     *
```

```
     * @param id The id to set for the RoleAssignment.
```

```
    */
```

```
    public RoleAssignment(int id) {
```

```
        super(id);
```

```
    }
```

```
    /**
```

```
     * Constructs a RoleAssignment with the specified id, assigned role, sender, receiver, and time.
```

```
     *
```

```
     * @param id    The id to set for the RoleAssignment.
```

```
     * @param role  The Role representing the assigned role.
```

```
     * @param sender The User representing the sender of the assignment.
```

```
     * @param receiver The User representing the receiver of the assignment.
```

```
     * @param time   The Timestamp representing the time of assignment.
```

```
    */
```

```
    public RoleAssignment(int id, Role role, User sender, User receiver, Timestamp time) {
```

```
        this(id);
```

```
        this.setRole(role);;
```

```
        this.setSender(sender);;
```

```
        this.setReceiver(receiver);;
```

```
        this.setTime(time);;
```

```
    }
```

```
    /**
```

```

    * Gets the assigned role.
    *
    * @return The Role representing the assigned role.
    */
    public Role getRole() {
        return role;
    }

    /**
     * Sets the assigned role.
     *
     * @param role The Role representing the assigned role.
     */
    public void setRole(Role role) {
        this.role = role;
    }

    /**
     * Gets the sender of the assignment.
     *
     * @return The User representing the sender of the assignment.
     */
    public User getSender() {
        return sender;
    }

    /**
     * Sets the sender of the assignment.
     *
     * @param sender The User representing the sender of the assignment.
     */
    public void setSender(User sender) {
        this.sender = sender;
    }

    /**
     * Gets the receiver of the assignment.
     *
     * @return The User representing the receiver of the assignment.
     */
    public User getReceiver() {
        return receiver;
    }

```

```

/**
 * Sets the receiver of the assignment.
 *
 * @param receiver The User representing the receiver of the assignment.
 */
public void setReceiver(User receiver) {
    this.receiver = receiver;
}

```

```

/**
 * Gets the timestamp of assignment.
 *
 * @return The Timestamp representing the time of assignment.
 */
public Timestamp getTime() {
    return time;
}

```

```

/**
 * Sets the timestamp of assignment.
 *
 * @param time The Timestamp representing the time of assignment.
 */
public void setTime(Timestamp time) {
    this.time = time;
}

```

```

}

```

5.18. Класс RoleGroup

```

/**
 * The {@code RoleGroup} class represents the grouping of roles.
 * It extends the {@link AbstractEntity} class and includes information about the group name.
 *
 * <p>This class provides constructors for creating instances with different sets of attributes.
 * It includes getter and setter methods for accessing and modifying the group name.
 */
public class RoleGroup extends AbstractEntity {

    private String name;

```



```

/**
 * Constructs a RoleGroup with uninitialized values.
 */
public RoleGroup() {
}

/**
 * Constructs a RoleGroup with the specified id and uninitialized attributes.
 *
 * @param id The id to set for the RoleGroup.
 */
public RoleGroup(int id) {
    super(id);
}

/**
 * Constructs a RoleGroup with the specified id and group name.
 *
 * @param id The id to set for the RoleGroup.
 * @param name The string representing the group name.
 */
public RoleGroup(int id, String name) {
    this(id);
    this.name = name;
}

/**
 * Gets the name of the role group.
 *
 * @return The string representing the group name.
 */
public String getName() {
    return name;
}

/**
 * Sets the name of the role group.
 *
 * @param name The string representing the group name.
 */
public void setName(String name) {
    this.name = name;
}

```

```
}
```

5.19. Класс Sanction

```
/**
 * The {@code Sanction} class represents the delivery of a sanction to a user (receiver) by a sender.
 * It extends the {@link AbstractEntity} class and includes information about the sanction type, sender,
 receiver,
 * reason for the sanction, and the timestamp of delivery.
 *
 * <p>This class provides constructors for creating instances with different sets of attributes.
 * It includes getter and setter methods for accessing and modifying the sanction type, sender, receiver,
 reason, and time attributes.
 */
public class Sanction extends AbstractEntity {

    private SanctionType type;
    private User sender;
    private User receiver;
    private String reason;
    private Timestamp time;

    /**
     * Constructs a Sanction with uninitialized values.
     */
    public Sanction() {
    }

    /**
     * Constructs a Sanction with the specified id and uninitialized attributes.
     *
     * @param id The id to set for the Sanction.
     */
    public Sanction(int id) {
        super(id);
    }

    /**
     * Constructs a Sanction with the specified id, sanction type, sender, receiver, reason, and time.
     *
     * @param id The id to set for the Sanction.
     * @param type The SanctionType representing the type of sanction.
     * @param sender The User representing the sender of the sanction.

```

```

    * @param receiver The User representing the receiver of the sanction.
    * @param reason The String representing the reason for the sanction.
    * @param time The Timestamp representing the time of delivery.
    */
    public Sanction(int id, SanctionType type, User sender, User receiver, String reason, Timestamp time)
    {
        this(id);
        this.setType(type);
        this.setSender(sender);
        this.setReceiver(receiver);
        this.setReason(reason);
        this.setTime(time);
    }

    /**
     * Gets the type of the sanction.
     *
     * @return The SanctionType representing the type of sanction.
     */
    public SanctionType getType() {
        return type;
    }

    /**
     * Sets the type of the sanction.
     *
     * @param type The SanctionType representing the type of sanction.
     */
    public void setType(SanctionType type) {
        this.type = type;
    }

    /**
     * Gets the sender of the sanction.
     *
     * @return The User representing the sender of the sanction.
     */
    public User getSender() {
        return sender;
    }

    /**
     * Sets the sender of the sanction.

```

```

*
* @param sender The User representing the sender of the sanction.
*/
public void setSender(User sender) {
    this.sender = sender;
}

/**
 * Gets the receiver of the sanction.
 *
 * @return The User representing the receiver of the sanction.
 */
public User getReceiver() {
    return receiver;
}

/**
 * Sets the receiver of the sanction.
 *
 * @param receiver The User representing the receiver of the sanction.
 */
public void setReceiver(User receiver) {
    this.receiver = receiver;
}

/**
 * Gets the reason for the sanction.
 *
 * @return The String representing the reason for the sanction.
 */
public String getReason() {
    return reason;
}

/**
 * Sets the reason for the sanction.
 *
 * @param reason The String representing the reason for the sanction.
 */
public void setReason(String reason) {
    this.reason = reason;
}

```

```

/**
 * Gets the timestamp of delivery.
 *
 * @return The Timestamp representing the time of delivery.
 */
public Timestamp getTime() {
    return time;
}

/**
 * Sets the timestamp of delivery.
 *
 * @param time The Timestamp representing the time of delivery.
 */
public void setTime(Timestamp time) {
    this.time = time;
}
}

```

5.20. Класс SanctionType

```

/**
 * The {@code SanctionType} class represents the type of a sanction.
 * It extends the {@link AbstractEntity} class and includes information about the type name.
 *
 * <p>This class provides constructors for creating instances with different sets of attributes.
 * It includes getter and setter methods for accessing and modifying the type name.
 */
public class SanctionType extends AbstractEntity {

    private String name;

    /**
     * Constructs a SanctionType with uninitialized values.
     */
    public SanctionType() {
    }

    /**
     * Constructs a SanctionType with the specified id and uninitialized attributes.
     *
     * @param id The id to set for the SanctionType.
     */

```

```

public SanctionType(int id) {
    super(id);
}

/**
 * Constructs a SanctionType with the specified id and type name.
 *
 * @param id The id to set for the SanctionType.
 * @param name The String representing the type name.
 */
public SanctionType(int id, String name) {
    this(id);
    this.name = name;
}

/**
 * Gets the name of the sanction type.
 *
 * @return The String representing the type name.
 */
public String getName() {
    return name;
}

/**
 * Sets the name of the sanction type.
 *
 * @param name The String representing the type name.
 */
public void setName(String name) {
    this.name = name;
}
}

```

5.21. Класс Setting

```

/**
 * The {@code Setting} class represents the settings of an information system.
 * It extends the {@link AbstractEntity} class and includes information about the setting name and its value.
 *
 * <p>This class provides constructors for creating instances with different sets of attributes.
 * It includes getter and setter methods for accessing and modifying the setting name and value.
 */

```

```

public class Setting extends AbstractEntity {

    private String name;
    private int value;

    /**
     * Constructs a Setting with uninitialized attributes.
     */
    public Setting() {
    }

    /**
     * Constructs a Setting with the specified id and uninitialized attributes.
     *
     * @param id The id to set for the Setting.
     */
    public Setting(int id) {
        super(id);
    }

    /**
     * Constructs a Setting with the specified name and value, and uninitialized id.
     *
     * @param name The String representing the setting name.
     * @param value The int representing the setting value.
     */
    public Setting(String name, int value) {
        this.setName(name);
        this.setValue(value);
    }

    /**
     * Constructs a Setting with the specified id, name, and value.
     *
     * @param id The id to set for the Setting.
     * @param name The String representing the setting name.
     * @param value The int representing the setting value.
     */
    public Setting(int id, String name, int value) {
        this(id);
        this.setName(name);
        this.setValue(value);
    }
}

```

```

/**
 * Gets the name of the setting.
 *
 * @return The String representing the setting name.
 */
public String getName() {
    return name;
}

/**
 * Sets the name of the setting.
 *
 * @param name The String representing the setting name.
 */
public void setName(String name) {
    this.name = name;
}

/**
 * Gets the value of the setting.
 *
 * @return The int representing the setting value.
 */
public int getValue() {
    return value;
}

/**
 * Sets the value of the setting.
 *
 * @param value The int representing the setting value.
 */
public void setValue(int value) {
    this.value = value;
}
}

```

5.22. Класс Team

```

/**
 * The {@code Team} class represents a team that users can join.
 * It extends the {@link AbstractEntity} class and includes information about the team name.

```



```

*
* <p>This class provides constructors for creating instances with different sets of attributes.
* It includes getter and setter methods for accessing and modifying the team name.
*/

```

```

public class Team extends AbstractEntity {

    private String name;

    /**
     * Constructs a Team with uninitialized values.
     */
    public Team() {
    }

    /**
     * Constructs a Team with the specified id and uninitialized attributes.
     *
     * @param id The id to set for the Team.
     */
    public Team(int id) {
        super(id);
    }

    /**
     * Constructs a Team with the specified id and team name.
     *
     * @param id The id to set for the Team.
     * @param name The String representing the team name.
     */
    public Team(int id, String name) {
        this(id);
        this.setName(name);
    }

    /**
     * Gets the name of the team.
     *
     * @return The String representing the team name.
     */
    public String getName() {
        return name;
    }
}

```

```

/**
 * Sets the name of the team.
 *
 * @param name The String representing the team name.
 */
public void setName(String name) {
    this.name = name;
}
}

```

5.23. Класс TeamInteract

```

/**
 * The {@code TeamInteract} class represents the interaction of a user with a team.
 * It extends the {@link AbstractEntity} class and includes information about the user, interaction type, team,
 * and the timestamp of the interaction.
 *
 * <p>This class provides constructors for creating instances with different sets of attributes.
 * It includes getter and setter methods for accessing and modifying the user, interaction type, team, and time
 * attributes.
 */
public class TeamInteract extends AbstractEntity {

    private User user;
    private TeamInteractType type;
    private Team team;
    private Timestamp time;

    /**
     * Constructs a TeamInteract with uninitialized values.
     */
    public TeamInteract() {
    }

    /**
     * Constructs a TeamInteract with the specified id and uninitialized attributes.
     *
     * @param id The id to set for the TeamInteract.
     */
    public TeamInteract(int id) {
        super(id);
    }
}

```

```

/**
 * Constructs a TeamInteract with the specified id, user, interaction type, team, and time.
 *
 * @param id The id to set for the TeamInteract.
 * @param user The User representing the user involved in the interaction.
 * @param type The TeamInteractType representing the type of interaction.
 * @param team The Team representing the team involved in the interaction.
 * @param time The Timestamp representing the time of the interaction.
 */
public TeamInteract(int id, User user, TeamInteractType type, Team team, Timestamp time) {
    this(id);
    this.setUser(user);
    this.setType(type);
    this.setTeam(team);
    this.setTime(time);
}

/**
 * Gets the user involved in the interaction.
 *
 * @return The User representing the user involved in the interaction.
 */
public User getUser() {
    return user;
}

/**
 * Sets the user involved in the interaction.
 *
 * @param user The User representing the user involved in the interaction.
 */
public void setUser(User user) {
    this.user = user;
}

/**
 * Gets the type of interaction.
 *
 * @return The TeamInteractType representing the type of interaction.
 */
public TeamInteractType getType() {
    return type;
}

```

```

/**
 * Sets the type of interaction.
 *
 * @param type The TeamInteractType representing the type of interaction.
 */
public void setType(TeamInteractType type) {
    this.type = type;
}

/**
 * Gets the team involved in the interaction.
 *
 * @return The Team representing the team involved in the interaction.
 */
public Team getTeam() {
    return team;
}

/**
 * Sets the team involved in the interaction.
 *
 * @param team The Team representing the team involved in the interaction.
 */
public void setTeam(Team team) {
    this.team = team;
}

/**
 * Gets the timestamp of the interaction.
 *
 * @return The Timestamp representing the time of the interaction.
 */
public Timestamp getTime() {
    return time;
}

/**
 * Sets the timestamp of the interaction.
 *
 * @param time The Timestamp representing the time of the interaction.
 */
public void setTime(Timestamp time) {

```

```

        this.time = time;
    }

}

```

5.24. Класс TeamInteractType

```

/**
 * The {@code TeamInteractType} class represents the type of interaction with a team.
 * It extends the {@link AbstractEntity} class and includes information about the interaction type name.
 *
 * <p>This class provides constructors for creating instances with different sets of attributes.
 * It includes getter and setter methods for accessing and modifying the interaction type name.
 */
public class TeamInteractType extends AbstractEntity {

    private String name;

    /**
     * Constructs a TeamInteractType with uninitialized values.
     */
    public TeamInteractType() {

    }

    /**
     * Constructs a TeamInteractType with the specified id and uninitialized attributes.
     *
     * @param id The id to set for the TeamInteractType.
     */
    public TeamInteractType(int id) {
        super(id);
    }

    /**
     * Constructs a TeamInteractType with the specified id and interaction type name.
     *
     * @param id The id to set for the TeamInteractType.
     * @param name The String representing the interaction type name.
     */
    public TeamInteractType(int id, String name) {
        this(id);
        this.name = name;
    }
}

```

```

/**
 * Gets the name of the interaction type.
 *
 * @return The String representing the interaction type name.
 */
public String getName() {
    return name;
}

/**
 * Sets the name of the interaction type.
 *
 * @param name The String representing the interaction type name.
 */
public void setName(String name) {
    this.name = name;
}
}

```

5.25. Класс User

```

/**
 * The {@code User} class represents an entity that uses the information system.
 * It extends the {@link AbstractEntity} class and includes information about the user's login, password,
 * name, email, and authorization status.
 *
 * <p>This class provides constructors for creating instances with different sets of attributes.
 * It includes getter and setter methods for accessing and modifying the login, password, name, email,
 * and authorization status attributes. Additionally, it provides a method for checking the password.
 */
public class User extends AbstractEntity {

    private String login;
    private String password;
    private String name;
    private String email;
    private boolean is_authorized = false;

    /**
     * Constructs a User with uninitialized values.
     */
    public User() {
    }
}

```

```

/**
 * Constructs a User with the specified id and uninitialized attributes.
 *
 * @param id The id to set for the User.
 */
public User(int id) {
    super(id);
}

/**
 * Constructs a User with the specified login and password, and uninitialized id.
 *
 * @param login    The String representing the user's login.
 * @param password The String representing the user's password.
 */
public User(String login, String password) {
    this.setLogin(login);
    this.setPassword(password);
}

/**
 * Constructs a User with the specified login, password, name, and email.
 *
 * @param login    The String representing the user's login.
 * @param password The String representing the user's password.
 * @param name     The String representing the user's name.
 * @param email    The String representing the user's email.
 */
public User(String login, String password, String name, String email) {
    this(login, password);
    this.setName(name);
    this.setEmail(email);
}

/**
 * Constructs a User with the specified id, login, password, name, and email.
 *
 * @param id      The id to set for the User.
 * @param login    The String representing the user's login.
 * @param password The String representing the user's password.
 * @param name     The String representing the user's name.
 * @param email    The String representing the user's email.

```

```

*/
public User(int id, String login, String password, String name, String email) {
    this(login, password, name, email);
    this.setId(id);
}

/**
 * Constructs a User with the specified id, login, password, name, email, and authorization status.
 *
 * @param id          The id to set for the User.
 * @param login       The String representing the user's login.
 * @param password    The String representing the user's password.
 * @param name        The String representing the user's name.
 * @param email       The String representing the user's email.
 * @param isAuthorized The boolean representing the authorization status of the user.
 */
public User(int id, String login, String password, String name, String email, boolean is_authorized) {
    this(id, login, password, name, email);
    this.setIsAuthorized(is_authorized);
}

/**
 * Gets the user's login.
 *
 * @return The String representing the user's login.
 */
public String getLogin() {
    return login;
}

/**
 * Sets the user's login.
 *
 * @param login The String representing the user's login.
 */
public void setLogin(String login) {
    this.login = login;
}

/**
 * Gets the user's password.
 *
 * @return The String representing the user's password.

```



```
*/  
public String getPassword() {  
    return password;  
}  
  
/**  
 * Sets the user's password.  
 *  
 * @param password The String representing the user's password.  
 */  
public void setPassword(String password) {  
    this.password = password;  
}  
  
/**  
 * Gets the user's name.  
 *  
 * @return The String representing the user's name.  
 */  
public String getName() {  
    return name;  
}  
  
/**  
 * Sets the user's name.  
 *  
 * @param name The String representing the user's name.  
 */  
public void setName(String name) {  
    this.name = name;  
}  
  
/**  
 * Gets the user's email.  
 *  
 * @return The String representing the user's email.  
 */  
public String getEmail() {  
    return email;  
}  
  
/**  
 * Sets the user's email.
```

```

*
* @param email The String representing the user's email.
*/
public void setEmail(String email) {
    this.email = email;
}

/**
 * Checks if the user is authorized.
 *
 * @return The boolean representing the authorization status of the user.
 */
public boolean isIsAuthorized() {
    return is_authorized;
}

/**
 * Sets the authorization status of the user.
 *
 * @param isAuthorized The boolean representing the authorization status of the user.
 */
public void setIsAuthorized(boolean is_authorized) {
    this.is_authorized = is_authorized;
}

/**
 * Checks if the entered password matches the user's password.
 *
 * @param enterPass The String representing the entered password.
 * @return True if the entered password matches the user's password; otherwise, false.
 */
public boolean checkPassword(String enterPass) {
    return this.getPassword().equals(enterPass);
}
}

```

6. Библиографический список

1. Блинов И.Н., Романчик В.С. Java. Методы программирования: учеб.-метод. пособие. — Минск: Четыре четверти, 2013. — 896 с.

2. Пруцков А. В. Программирование на языке Java. Введение в курс с примерами и практическими заданиями: учебник. — М.: КУРС, 2018. — 208 с.
3. Пруцков А. В. Тонкости программирования в примерах: учебник. — М.: КУРС, 2022. — 228 с.
4. Пруцков А. В. Сборник документов для учебных занятий 2020 года / Рязан. гос. радиотехн. ун-т им. В. Ф. Уткина. – Рязань, 2020. – 36 с. – № 5500.
5. Пруцков А. В. Сборник документов для учебных занятий 2022 года / Рязан. гос. радиотехн. ун-т им. В. Ф. Уткина. – Рязань, 2022. – 24 с. – № 7077.