

Московский Авиационный Институт  
(Национальный Исследовательский Университет)

Факультет: «Компьютерные науки и прикладная математика»  
Кафедра: 806 «Вычислительная математика и программирование»  
Дисциплина: «Базы данных»

**Курсовой проект**  
**Тема: «Сервис по курированию задач в общежитии»**

Студенты:	Ирисов Т. А.
Группа:	М8О-309Б-22
Преподаватель:	Малахов А. В.
Дата:	
Оценка:	

Москва 2024

# Оглавление

Таблицы и схемы данных .....	3
Таблица users .....	3
Таблица tasks .....	3
Таблица user_task .....	3
Таблица dorms .....	4
Таблица blocks .....	4
Таблица user_block .....	4
Таблица dorm_block .....	4
Таблица history .....	5
Таблица requests .....	5
Схема базы данных .....	6
Архитектура проекта .....	7
Сервисы и модели .....	8
User .....	8
Tasks .....	10
History .....	13
Dorm .....	14
Request .....	16
Руководство пользователя .....	20
Регистрация .....	20
Заявка на вступление .....	23
Главное окно .....	26
Участники .....	27
Задания .....	28
История .....	32
Выход из блока .....	33
Функционал администратора .....	34
Список используемых источников. ....	35
<a href="#">Репозиторий проекта</a> .....	35

## Таблицы и схемы данных

В данной работе БД состоит из девяти таблиц: users, tasks, user\_task, blocks, user\_block, dorms, dorm\_block, history, requestst.

### Таблица users

Описание: Хранит информацию о пользователях системы.

Поля:

- user\_id: уникальный идентификатор пользователя (первичный ключ).
- user\_login: логин пользователя.
- user\_password: пароль пользователя.
- user\_name: имя пользователя.
- user\_surname: фамилия пользователя.
- user\_middle\_name: отчество пользователя (опционально).
- user\_role: роль пользователя, по умолчанию — 'user'.

### Таблица tasks

Описание: Хранит информацию о задачах.

Поля:

- task\_id: уникальный идентификатор задачи (первичный ключ).
- task\_name: название задачи.
- task\_type: тип задачи, по умолчанию — 'single' (одноразовая задача).

### Таблица user\_task

Описание: Связывает пользователей и задачи, определяя, какие задачи назначены пользователям.

Поля:

- user\_id: идентификатор пользователя (внешний ключ на users.user\_id).
- task\_id: идентификатор задачи (внешний ключ на tasks.task\_id).

#### Таблица dorms

Описание: Хранит информацию об общежитиях.

Поля:

- dorm\_id: уникальный идентификатор общежития (первичный ключ).
- dorm\_name: название общежития.
- dorm\_adress: адрес общежития.

#### Таблица blocks

Описание: Хранит информацию о блоках в общежитиях.

Поля:

- block\_id: уникальный идентификатор блока (первичный ключ).
- block\_number: номер блока.

#### Таблица user\_block

Описание: Связывает пользователей и блоки, определяя, в каком блоке проживает пользователь.

Поля:

- user\_id: идентификатор пользователя (внешний ключ на users.user\_id и первичный ключ).
- block\_id: идентификатор блока (внешний ключ на blocks.block\_id).

#### Таблица dorm\_block

Описание: Связывает блоки и общежития, определяя, в каком общежитии находится блок.

Поля:

- block\_id: идентификатор блока (внешний ключ на blocks.block\_id).
- dorm\_id: идентификатор общежития (внешний ключ на dorms.dorm\_id).

### Таблица history

Описание: Хранит историю действий пользователей в системе.

#### Поля:

- id: уникальный идентификатор записи (первичный ключ).
- dorm\_id: идентификатор общежития, связанного с действием (внешний ключ на dorms.dorm\_id).
- block\_id: идентификатор блока, связанного с действием (внешний ключ на blocks.block\_id).
- user\_id: идентификатор пользователя, совершившего действие (внешний ключ на user\_block.user\_id).
- action: описание действия.
- date\_time: время и дата действия, по умолчанию — текущее время.

### Таблица requests

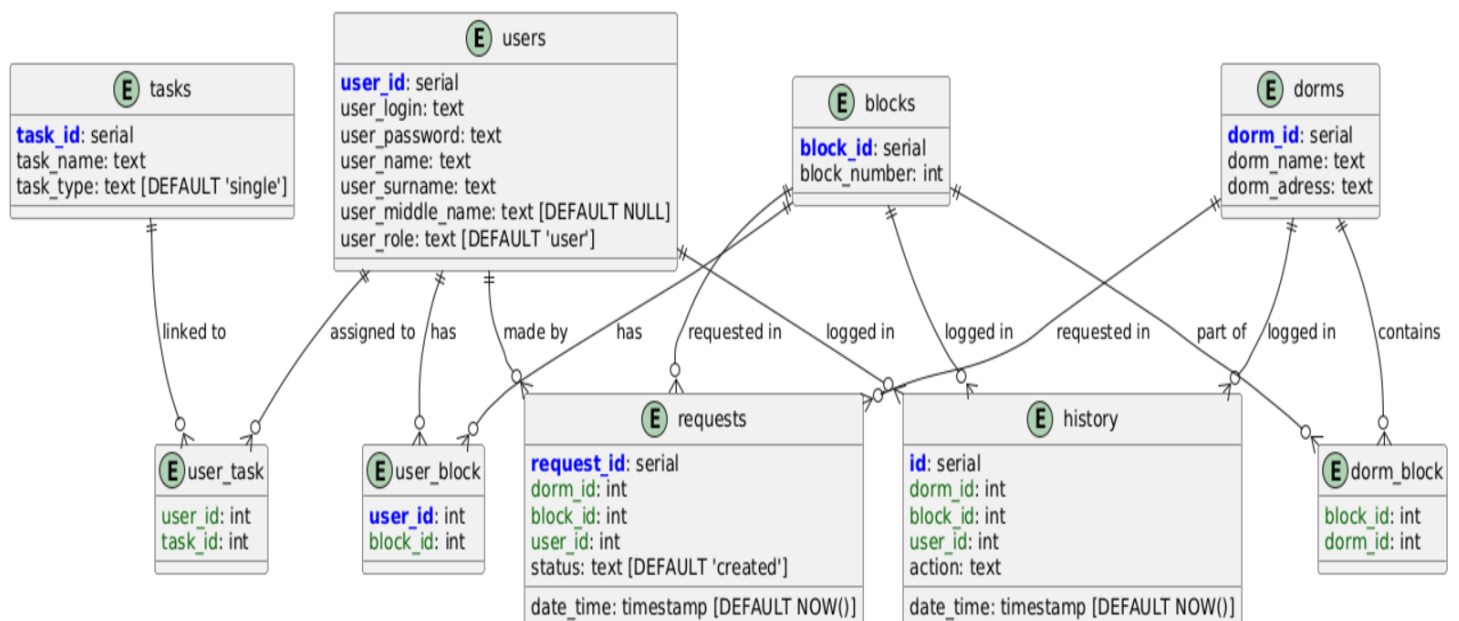
Описание: Хранит информацию о заявках, созданных пользователями.

#### Поля:

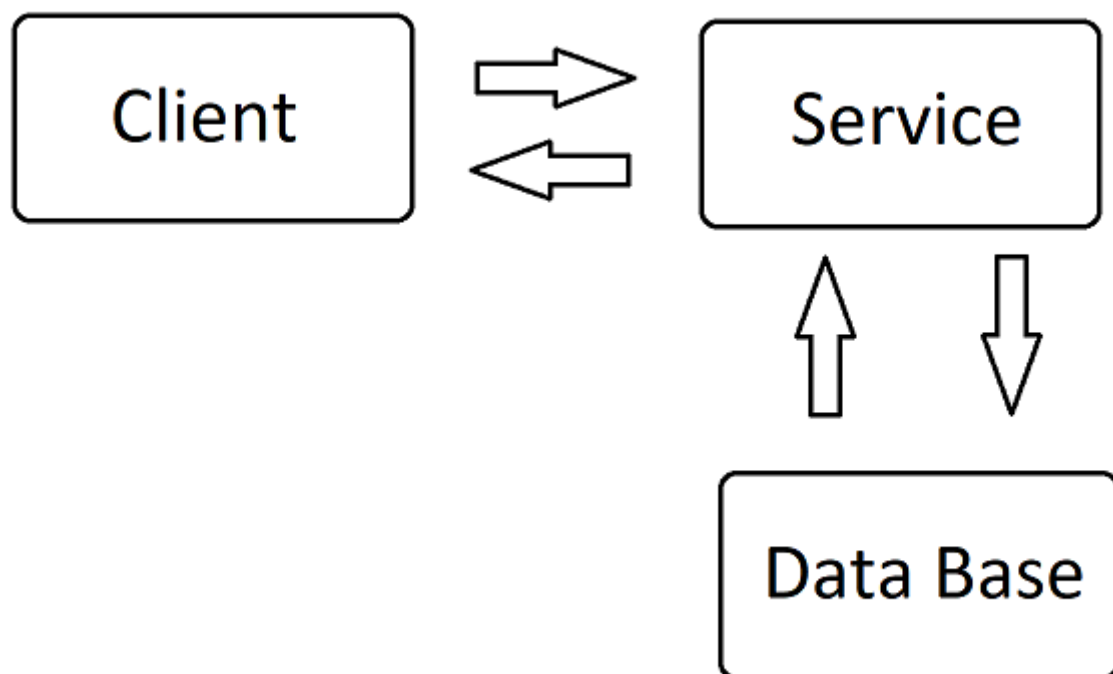
- request\_id: уникальный идентификатор заявки (первичный ключ).
- dorm\_id: идентификатор общежития, связанного с заявкой (внешний ключ на dorms.dorm\_id).
- block\_id: идентификатор блока, связанного с заявкой (внешний ключ на blocks.block\_id).
- user\_id: идентификатор пользователя, создавшего заявку (внешний ключ на users.user\_id).
- status: статус заявки, по умолчанию — 'created'.
- date\_time: время и дата создания заявки, по умолчанию — текущее время.

## Схема базы данных

Таблицы используют внешние ключи для обеспечения связности данных между пользователями, задачами, общежитиями, блоками и историями действий. Основная структура построена так, чтобы каждая запись была четко привязана к конкретному пользователю, задаче или объекту (общежитию/блоку).



## Архитектура проекта



Данное приложение является многопользовательским и имеет сервисную структуру, то есть обращение клиента посылается не напрямую в базу данных, а на определенный сервис который находится на сервере с backend и уже этот сервис в зависимости от типа запрос посылает разные запросы в базу данных, и возвращает ответы клиенту. Такой подход является эффективным, легко масштабируемым, и безопасным, так как клиент не хранит в себе какую либо секретную информацию. Также хочу упомянуть что пароли в базе данных хранятся в хэшированном виде, что тоже является признаком безопасности.

## Сервисы и модели.

### User

Данный сервис работает с моделью User и имеет методы:

```
service User{  
    rpc GetUser (GetUserRequest) returns (GetUserResponse);  
    rpc GetUsers (GetUsersRequest) returns (GetUsersResponse);  
    rpc CheckLogin (CheckLoginRequest) returns (CheckLoginResponse);  
    rpc Register (RegistrationRequest) returns (RegistrationResponse);  
    rpc ChangeRole (ChangeRoleRequest) returns (ChangeRoleResponse);  
}
```

1) GetUser принимает на вход GetUserRequest:

```
message GetUserRequest {  
    string user_login = 1;  
    string user_password = 2;  
}
```

и возвращает GetUserResponse:



```

message GetUserResponse {
    int32 user_id = 1;
    string user_role = 2;
    string user_name = 3;
    int32 block_id = 4;
    int32 dorm_id = 5;
}

```

Данный сервис используется для авторизации в приложении.

- 2) GetUsers принимает запрос GetUsersRequest и возвращает GetUsersResponse. Используется в приложении для просмотра информации и всех пользователей которые находятся с текущем блоке общежития

```

message GetUsersRequest{
    int32 block_id = 1;
}
message GetUsersResponse{
    repeated GetUserResponse users_info = 1;
}

```

- 3) CheckLogin используется при регистрации для проверки занятости логина.

```

message CheckLoginRequest{
    string user_login = 1;
}
message CheckLoginResponse{
    bool is_busy = 1;
}

```

- 4) Register регистрирует нового пользователя после проверки логина и пароля на соответствие требованиям сложности и доступности.

```
message RegistrationRequest{
    string user_name = 1;
    string user_surname = 2;
    string user_middle_name = 3;
    string user_login = 4;
    string user_password = 5;
}
message RegistrationResponse{
    int32 user_id = 1;
}
```

- 4) ChangeRole используется для назначения новых прав пользователю администратором, как назначение администратором, так и отбором прав администратора.

```
message ChangeRoleRequest{
    int32 user_id = 1;
    string role = 2;
}
message ChangeRoleResponse{
    bool is_changed = 1;
}
```

## Tasks

```
service Tasks{
    rpc GetUserTasks (GetTasksRequest) returns (GetTasksResponse);
    rpc TaskDone (TaskDoneRequest) returns (TaskDoneResponse);
    rpc MakeTask (MakeTaskRequest) returns (MakeTaskResponse);
    rpc GetBlockTasks (GetBlockTasksRequest) returns (GetBlockTasksResponse);
    rpc DeleteTask (DeleteTaskRequest) returns (DeleteTaskResponse);
}
```

- 1) GetUserTasks используется после успешной авторизации, для получения всех актуальных задач для текущего пользователя. Все задачи представлены в виде списка в главном окне приложения для пользователя.

```
message GetTasksRequest{
    int32 user_id = 1;
}
message task{
    int32 task_id = 1;
    string task_name = 2;
}
message GetTasksResponse{
    repeated task tasks = 1;
}
```

- 2) TaskDone. Данный метод вызывается при нажатии кнопки «Выполнено» на определенном задании, после чего данное задание будет перенесено из актуальных заданий в историю в базе данных.

```
message TaskDoneRequest{
    int32 task_id = 1;
    int32 user_id = 2;
}
message TaskDoneResponse{
    int32 is_done = 1;
}
```

- 3) MakeTask. Вызывается при создании задач для участников блока в общедоступности. Задачи могут быть как и однократными, так и поочередными, то есть если один участник выполнил задание, то оно переходит следующему после вызова TaskDone.

```
message MakeTaskRequest{
    string task_type = 1;
    string task_name = 2;
    GetUserResponse sender_info = 3;
}
message MakeTaskResponse{
    bool is_maded = 1;
}
```

- 4) GetBlockTasks. Метод используется чтобы получить список актуальных заданий для всех участников блока. Данный метод доступен для всех типов участников.

```
message GetBlockTasksRequest{
    int32 block_id = 1;
}
message GetBlockTasksResponse{
    repeated UserTasks users_task = 1;
}
```

- 5) DeleteTask. Метод доступный только администратору, который может удалить задание для определенного пользователя.

```
message DeleteTaskRequest{
    int32 user_id = 1;
    int32 task_id = 2;
}
message DeleteTaskResponse{
    bool is_deleted = 1;
}
```

## History

```
service History{
    rpc AddHistory (AddHistoryRequest) returns (AddHistoryResponse);
    rpc GetBlockHistory (GetBlockHistoryRequest) returns (GetBlockHistoryResponse);
}
```

- 1) AddHistory. Добавляет в историю какое либо значимое действие участника. Например если участник: выполнил задание, покинул блок, вступил в блок и тд.

```
message AddHistoryRequest{
    int32 user_id = 1;
    int32 block_id = 2;
    int32 dorm_id = 3;
    string action = 4;
}
message AddHistoryResponse{
    bool is_added = 1;
}
```

- 2) GetBlockHistory. Возвращает список всех значимых действий все участников блока, который может посмотреть любой участник блока.

```
message HistoryLine{
    int32 dorm_id = 1;
    int32 block_id = 2;
    int32 user_id = 3;
    string user_name = 4;
    string user_surname = 5;
    string action = 6;
}
message GetBlockHistoryRequest{
    int32 block_id = 1;
}
message GetBlockHistoryResponse{
    repeated HistoryLine history = 1;
}
```

## Dorm

```
service Dorm{
    rpc GetDormsBlocks (GetDormsBlocksRequest) returns (GetDormsBlocksResponse);
    rpc AddUser (AddUserRequest) returns (AddUserResponse);
    rpc KickUser (KickUserRequest) returns (KickUserResponse);
}
```

- 1) GetDormsBlocks. После регистрации пользователь может отправить заявку на вступления уже в сформированный блок или же создать свой из свободных. Метод возвращает список общежитий вместе с блоками.

```

message block{
    int32 block_number = 1;
    int32 block_id = 2;
}
message DormBlocks{
    int32 dorm_id = 1;
    string dorm_name = 2;
    repeated block blocks = 3;
}
message GetDormsBlocksRequest{
    int32 free = 1;
}
message GetDormsBlocksResponse{
    repeated DormBlocks DormsBlocks = 1;
}

```

- 2) AddUser. Если пользователь отправил заявку на вступление в блок, администратор может одобрить вступления или отклонить. Если администратор одобряет вступление нового участника, то вызывается данный метод и добавляет участника в блок.

```

message AddUserRequest{
    int32 user_id = 1;
    string user_role = 2;
    int32 block_id = 4;
}
message AddUserResponse{
    bool is_added = 1;
}

```

3) KickUser. Вызывается для удаления пользователя из блока. Данный метод доступен только администратору.

```
message KickUserRequest{
    int32 user_id = 1;
    int32 block_id = 2;
}
message KickUserResponse{
    bool is_kicked = 1;
}
```

## Request

```
service Request{
    rpc AddRequest (AddRequestRequest) returns (AddRequestResponse);
    rpc CancelRequest (CancelRequestRequest) returns (CancelRequestResponse);
    rpc GetRequest (GetRequestRequest) returns (GetRequestResponse);
    rpc GetRequests (GetRequestsRequest) returns (GetRequestsResponse);B
    rpc SetRequestStatus (SetRequestStatusRequest) returns (SetRequestStatusResponse);
    rpc DeleteRequests (DeleteRequestsRequest) returns (DeleteRequestsResponse);
}
```

1) AddRequest. Вызывается после отправки заявки на вступление в блок. Администраторы получают данную заявку в окне с заданиями и могут ее принять или отклонить.



```

message AddRequestResponse{
    int32 is_added = 1;
}
message AddRequestRequest{
    int32 dorm_id = 1;
    int32 block_id = 2;
    int32 user_id = 3;
}

```

- 2) CancelRequest. Если администратор отклоняет заявку на вступление в блок, вызывается данный метод.

```

message CancelRequestRequest{
    int32 request_id = 1;
}
message CancelRequestResponse{
    bool is_cancelled = 1;
}
message AddRequestResponse{
    int32 is_added = 1;
}

```

- 3) GetRequest. Используется для проверки наличия у пользователя каких либо запросов при входе в приложение, если же запросы у пользователя есть, то приложение сообщает пользователю об этом.

```

message GetRequestRequest{
    int32 user_id = 1;
}
message GetRequestResponse{
    int32 request_id = 1;
    int32 dorm_id = 2;
    string dorm_name = 3;
    int32 block_id = 4;
    int32 block_number = 5;
    string status = 6;
}

```

- 4) GetRequests. Используется для администраторов при входе в главное окно и проверяет на наличие каких либо запросов в текущий блок.

```

message request{
    int32 request_id = 1;
    int32 user_id = 2;
    string user_name = 3;
    string user_surname = 4;
    string user_middle_name = 5;
    string status = 6;
}
message GetRequestsRequest{
    int32 block_id = 1;
}
message GetRequestsResponse{
    repeated request requests = 1;
}

```

- 5) SetRequestStatus. Меняет текущий статус заявки в зависимости от ответа администратора. Заявка может быть принята или отклонена.

```
message SetRequestStatusRequest{
    int32 request_id = 1;
    string status = 2;
}
message SetRequestStatusResponse{
    bool is_changed = 1;
}
```

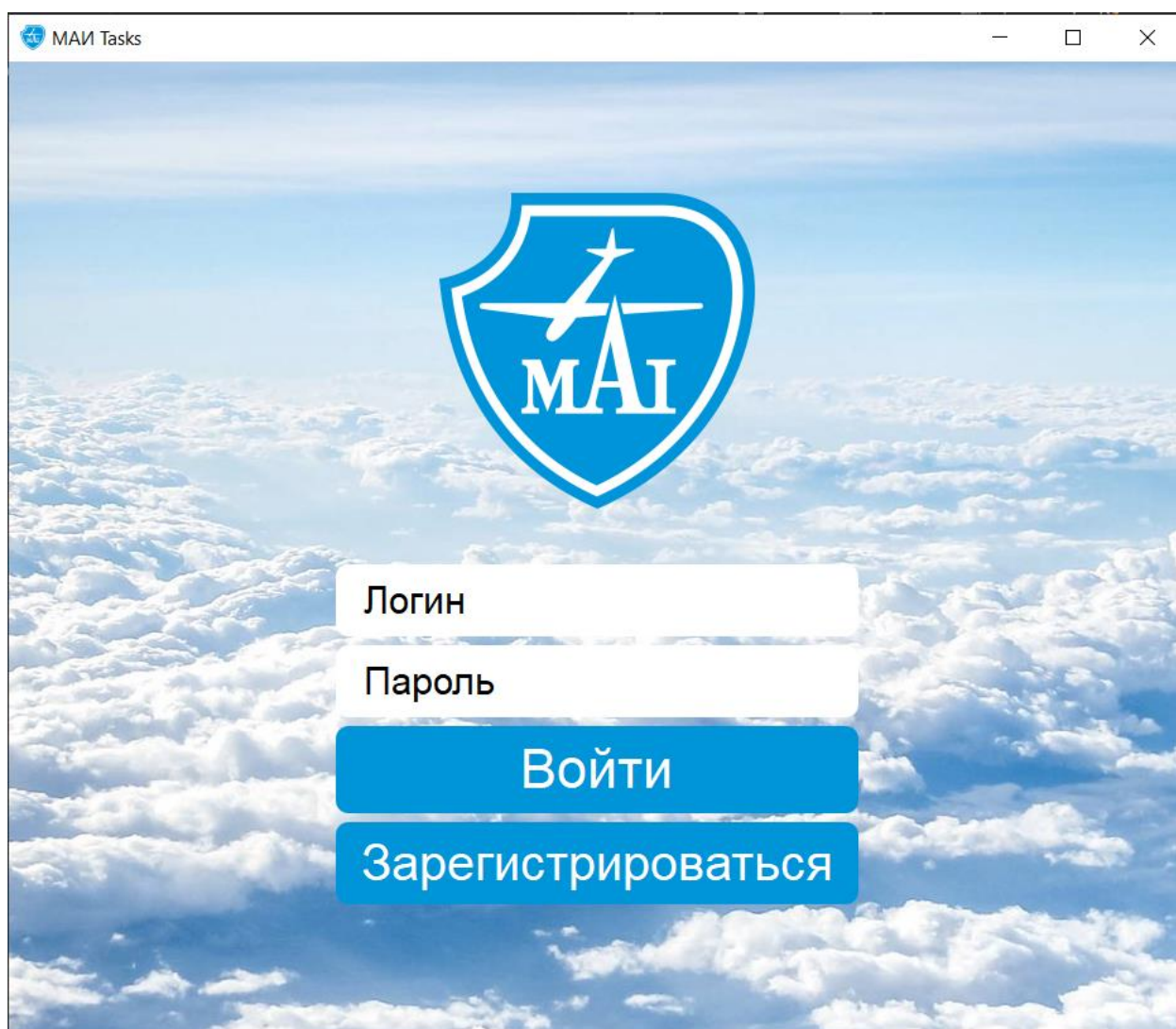
- 6) DeleteRequests. Метод удаляет все запросы блока, вызывается если последний участник блока покидает его. Участник ,который подал заявку в этот блок, может создать его сам и стать его администратором.

```
message DeleteRequestsRequest{
    int32 block_id = 1;
}
message DeleteRequestsResponse{
    bool are_deleted = 1;
}
```

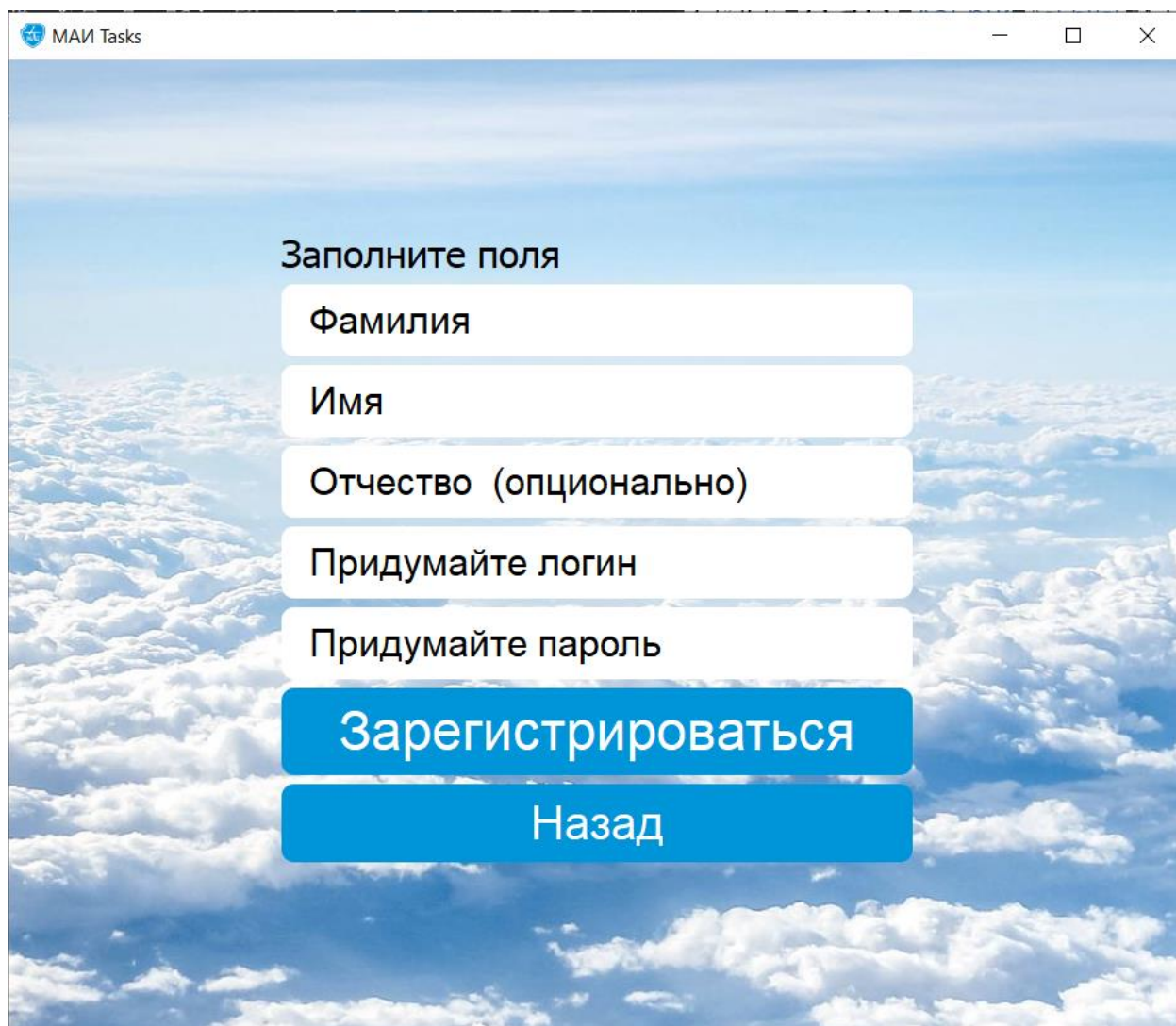
# Руководство пользователя

## Регистрация

Для начала регистрации нажмите на кнопку «Зарегистрироваться» в главном окне.



Далее заполните все обязательные поля для регистрации.



The image shows a web browser window titled "MAI Tasks". The background of the page is a blue sky with white clouds. The registration form is centered and consists of the following elements:

- Title: Заполните поля
- Input field: Фамилия
- Input field: Имя
- Input field: Отчество (опционально)
- Input field: Придумайте логин
- Input field: Придумайте пароль
- Blue button: Зарегистрироваться
- Blue button: Назад

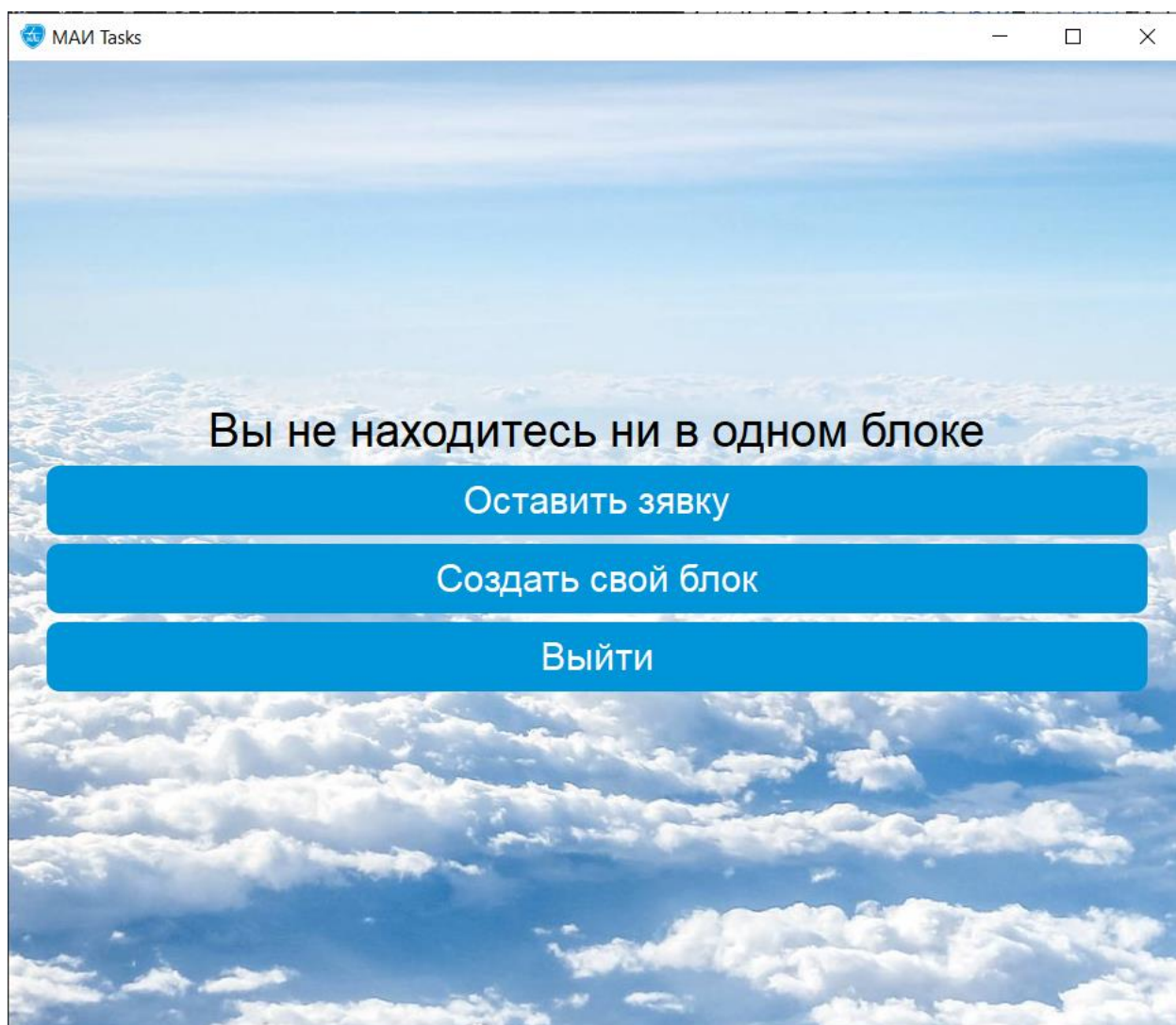
Логин должен составлять больше 4 символов. Пароль должен составлять больше 8 символов, обязательно должны быть хотя бы одна:

заглавная буква, специальный символ, цифра.

Раздел для заявок.

После регистрации вы можете или создать свой блок из свободных, или подать заявку на вступление в уже занятый кем либо блок.

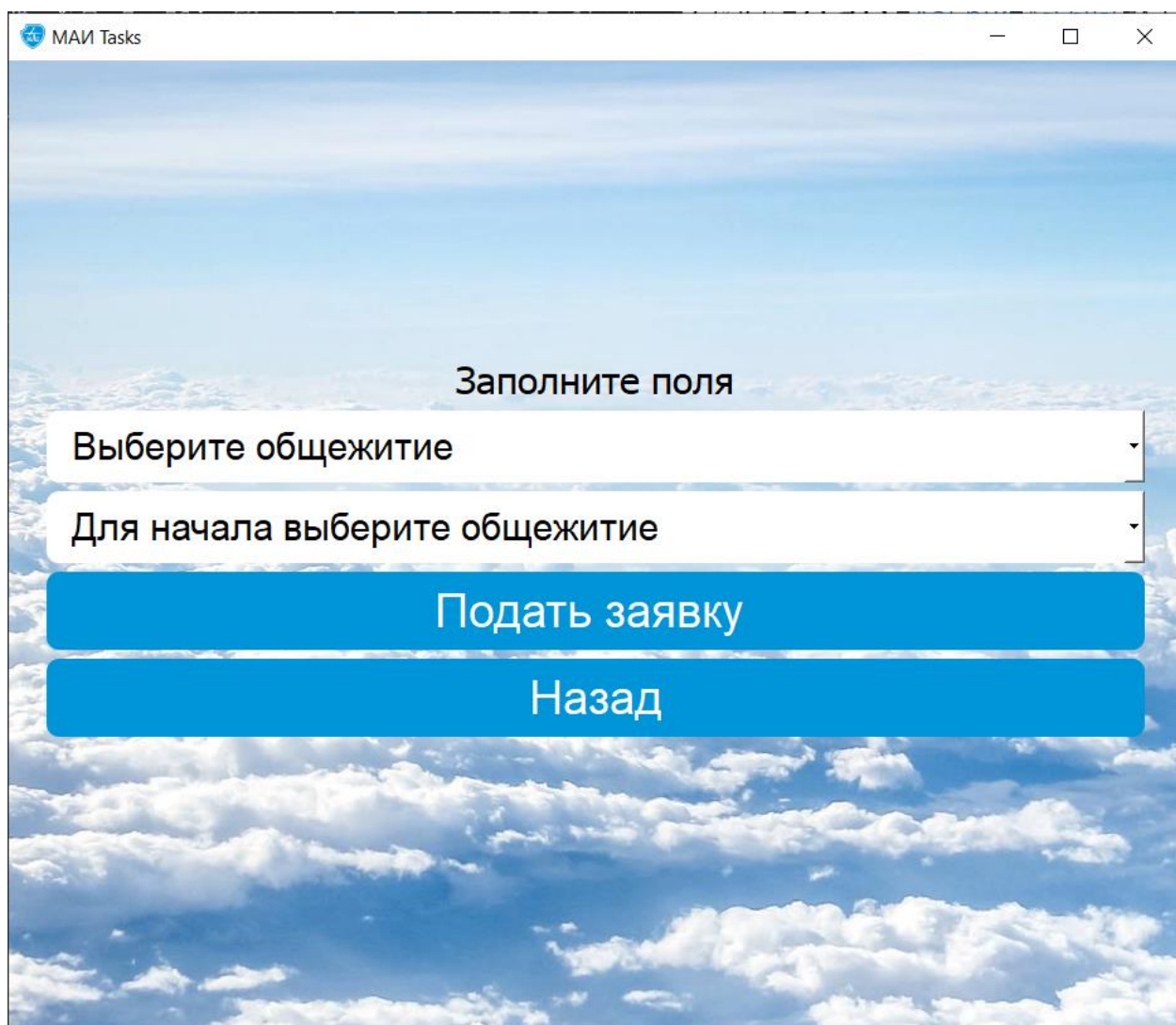
В нашем случае, подадим заявку на вступление.





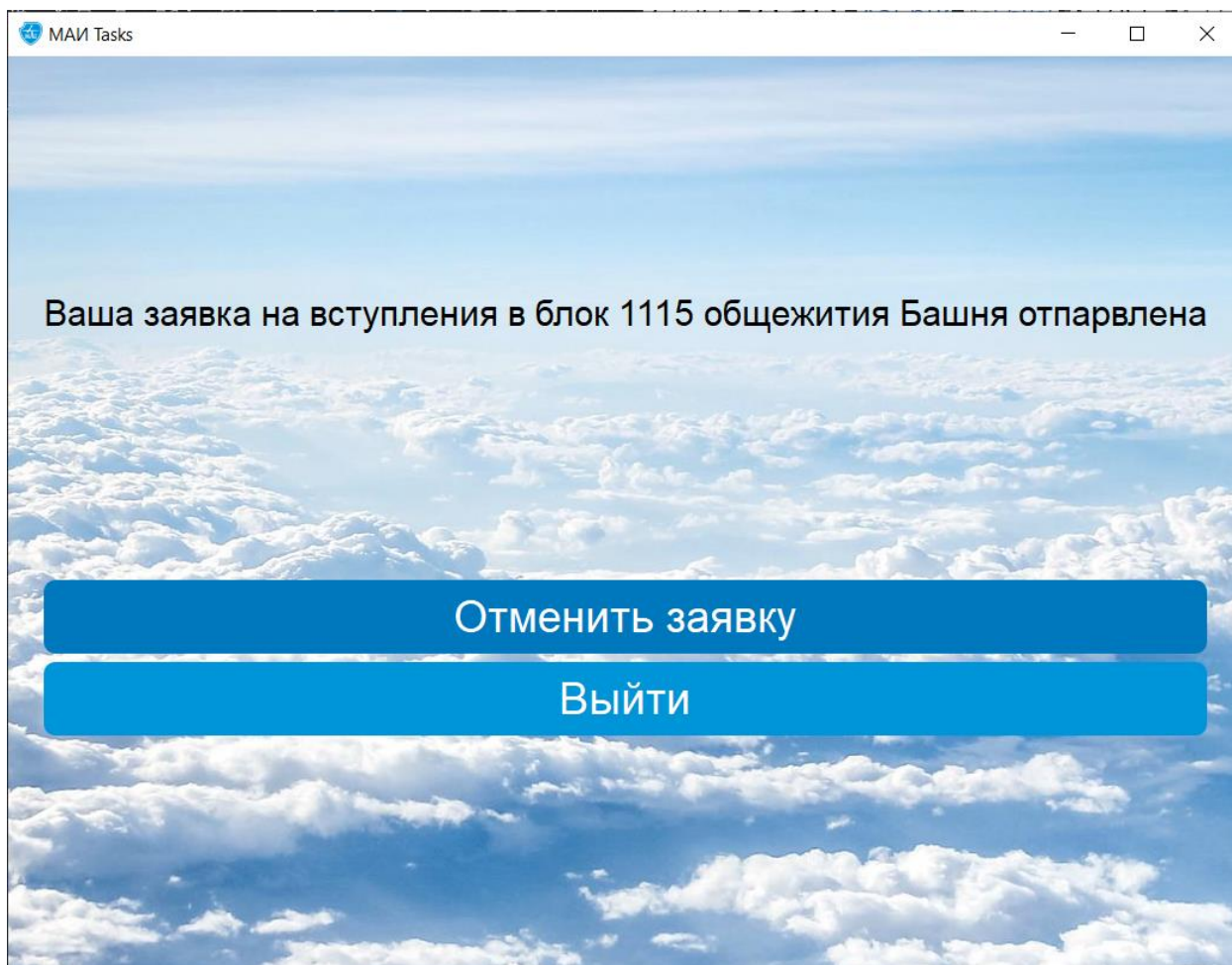
## Заявка на вступление

Выберите нужное вам общежитие и блок и нажмите кнопку «Подать заявку»



The screenshot shows a web application window titled "MAI Tasks". The background of the form is a blue sky with white clouds. The form contains the following elements:

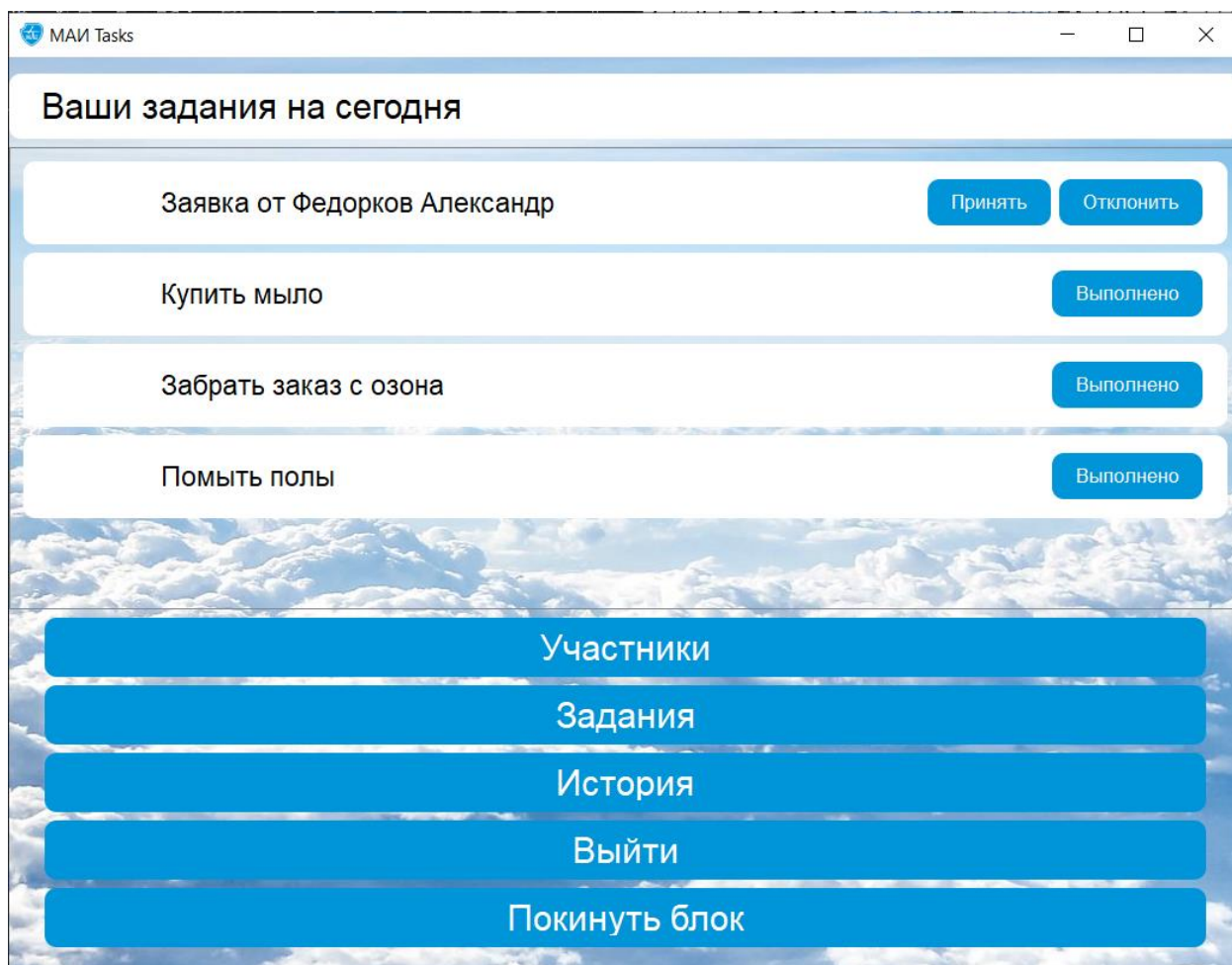
- The text "Заполните поля" (Fill in the fields) is centered above the input fields.
- A dropdown menu with the text "Выберите общежитие" (Select dormitory).
- A dropdown menu with the text "Для начала выберите общежитие" (At first, select a dormitory).
- A large blue button with the text "Подать заявку" (Submit application).
- A large blue button with the text "Назад" (Back).



После того, как вы отправили заявку на вступление, ожидайте ответа администратора блока, или же вы можете отменить заявку и попасть обратно к окну с разделом для заявок.

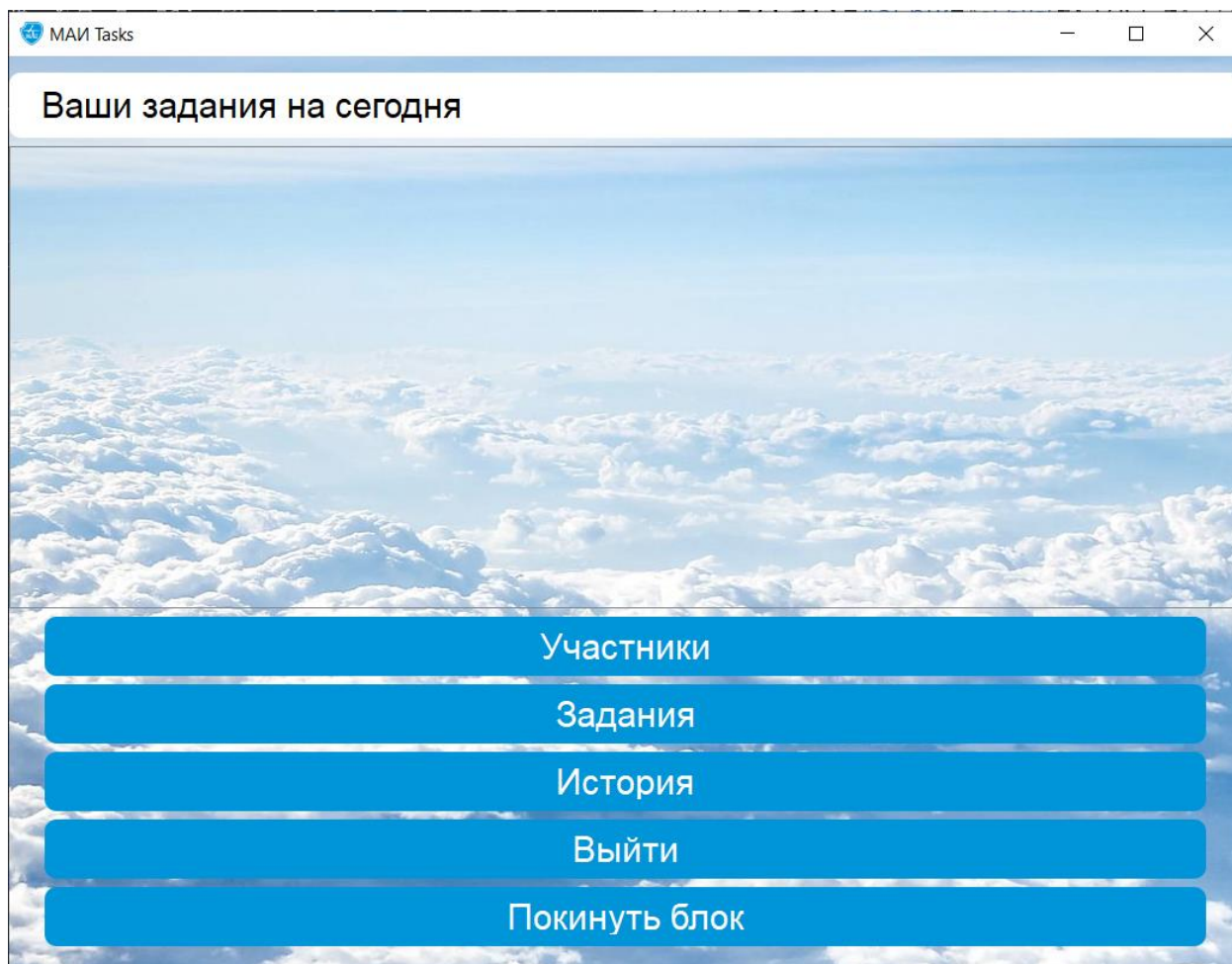


После того как вы отправили заявку, в главном окне, в разделе с заданиями, у администратора появится ваша заявка. И он сможет принять ее или отклонить.



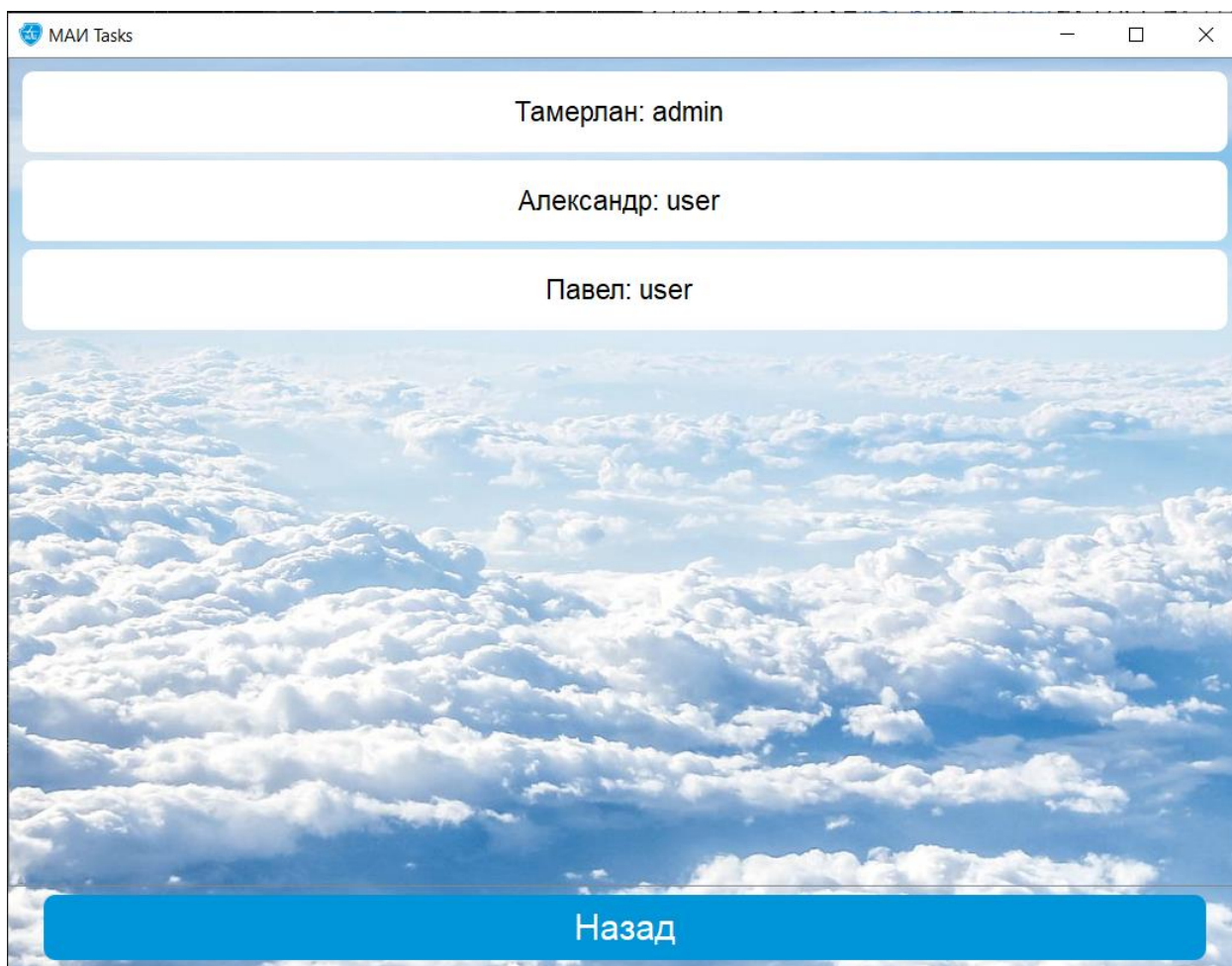
## Главное окно

При следующем входе после одобрения вашей заявки, вы окажитесь в главном окне вашего блока.



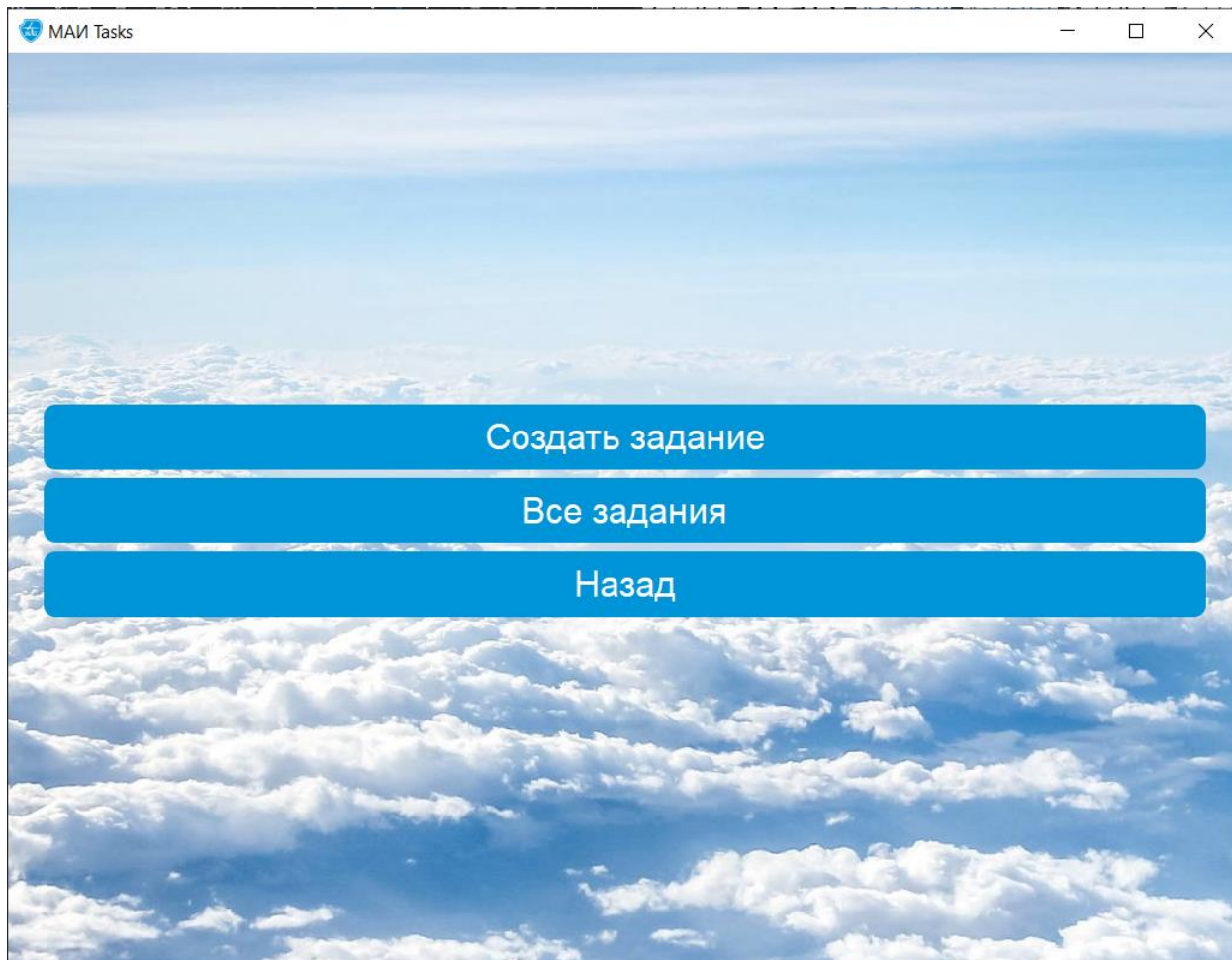
## Участники

Нажав на кнопку «Участники» вы можете посмотреть всех участников данного блока и их роли в блоке.



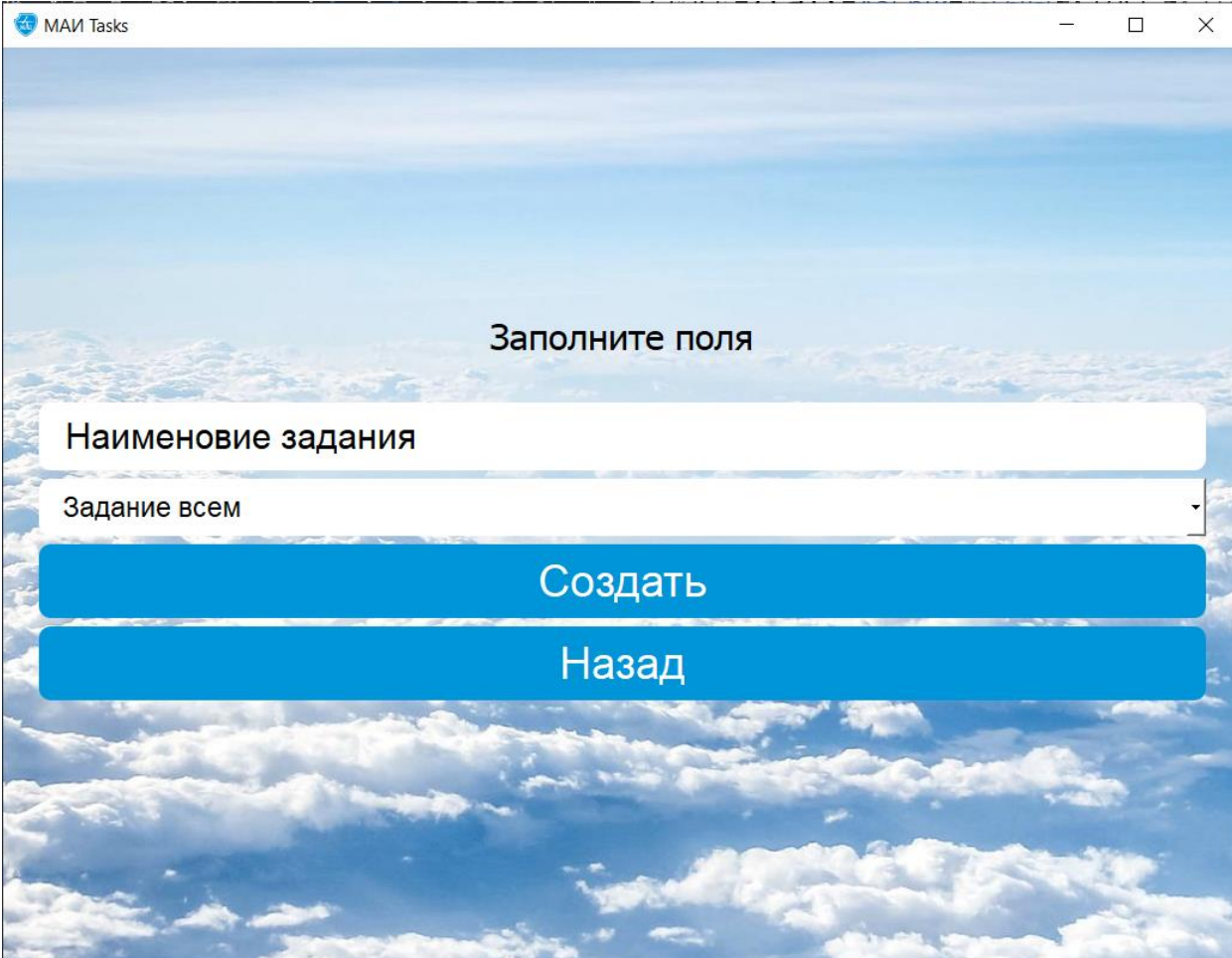
## Задания

В разделе задания вы можете просмотреть все задания или создать его.

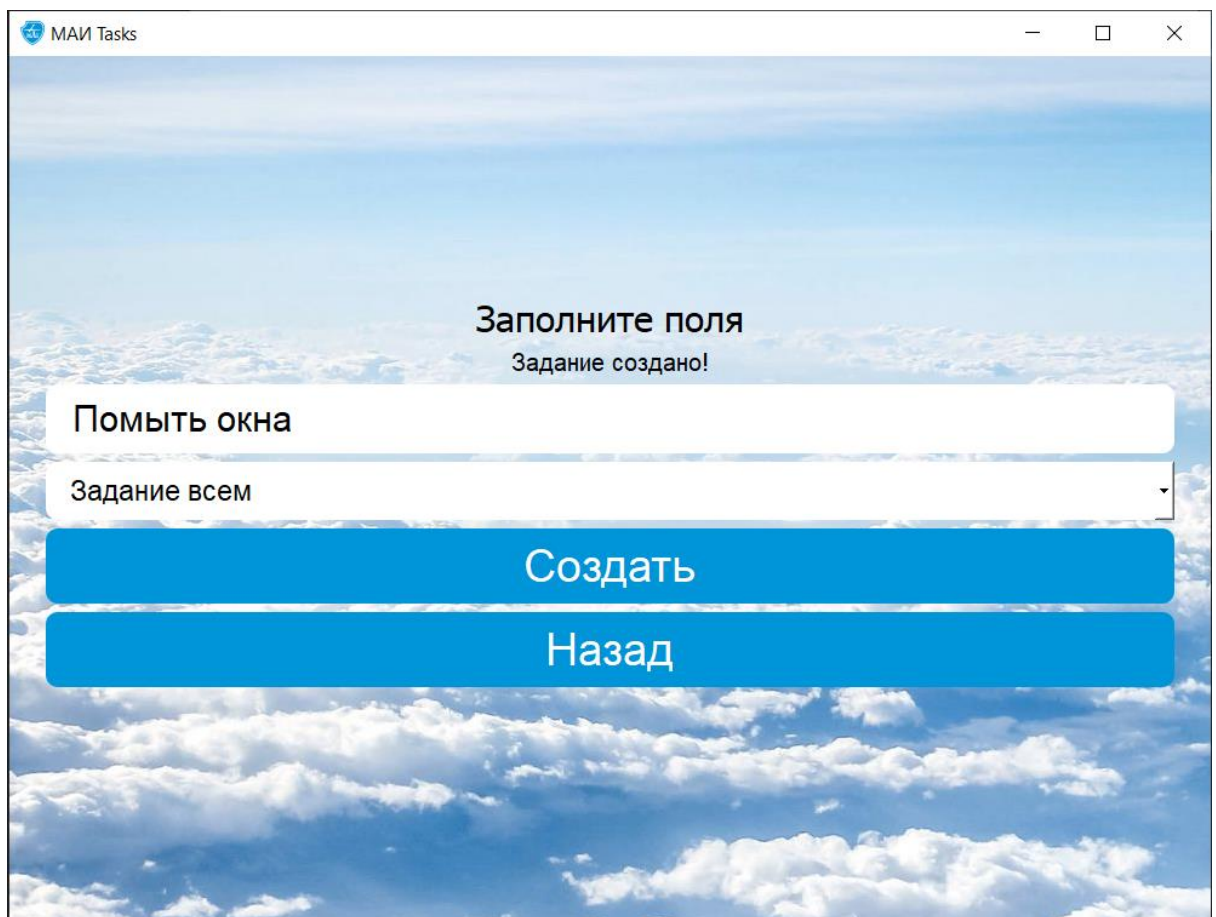
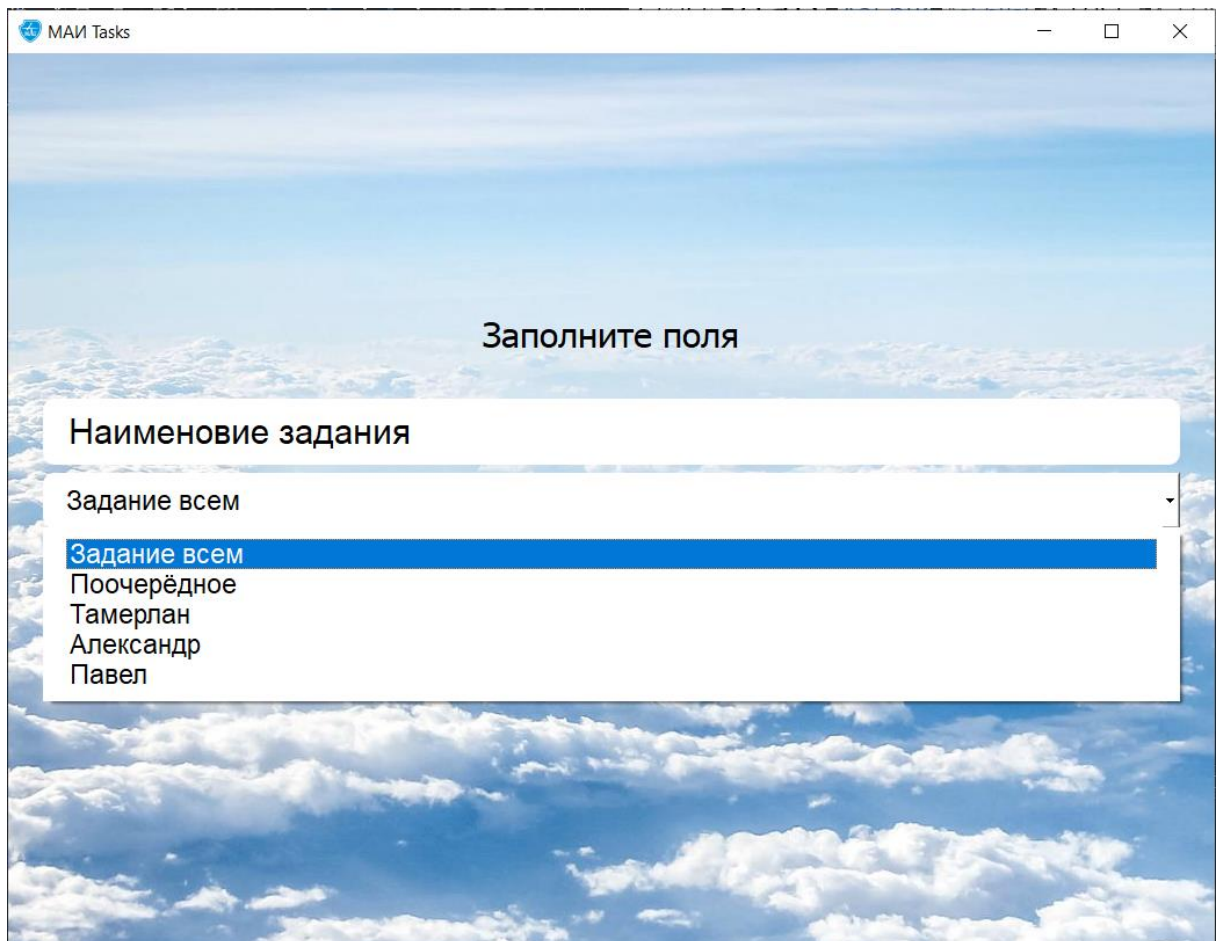




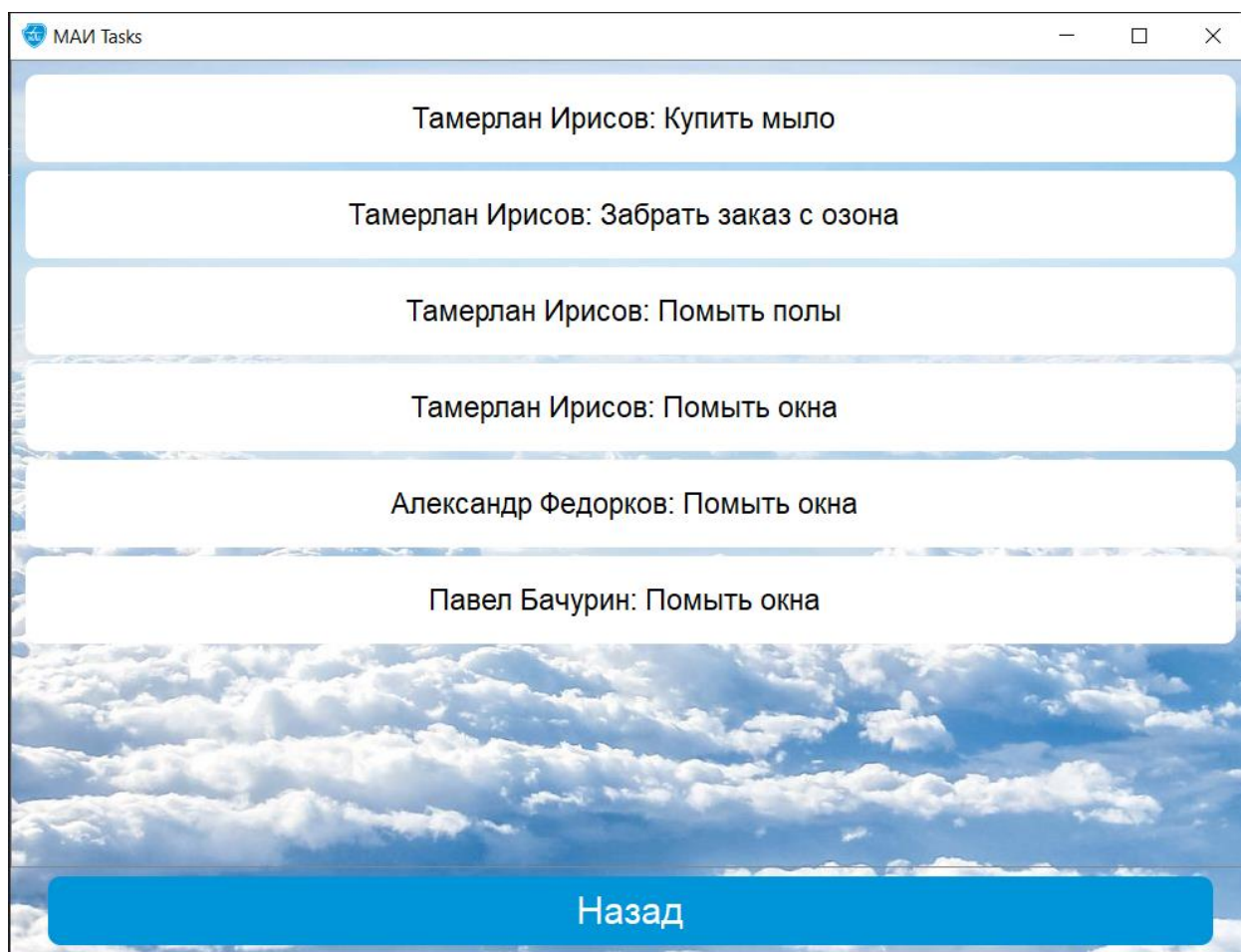
Нажав на кнопку «создать задание» в раздел, где сможете создать задание для любого участника блока, для всех сразу или же поочередное задание, которое будет переходить от участника к участнику после его выполнения.



The screenshot shows a web application window titled "MAI Tasks". The background of the page is a blue sky with white clouds. In the center, the text "Заполните поля" (Fill in the fields) is displayed. Below this text, there are two input fields: the first is labeled "Наименование задания" (Task name) and the second is labeled "Задание всем" (Task for all) with a dropdown arrow. At the bottom of the form, there are two large blue buttons: "Создать" (Create) and "Назад" (Back).

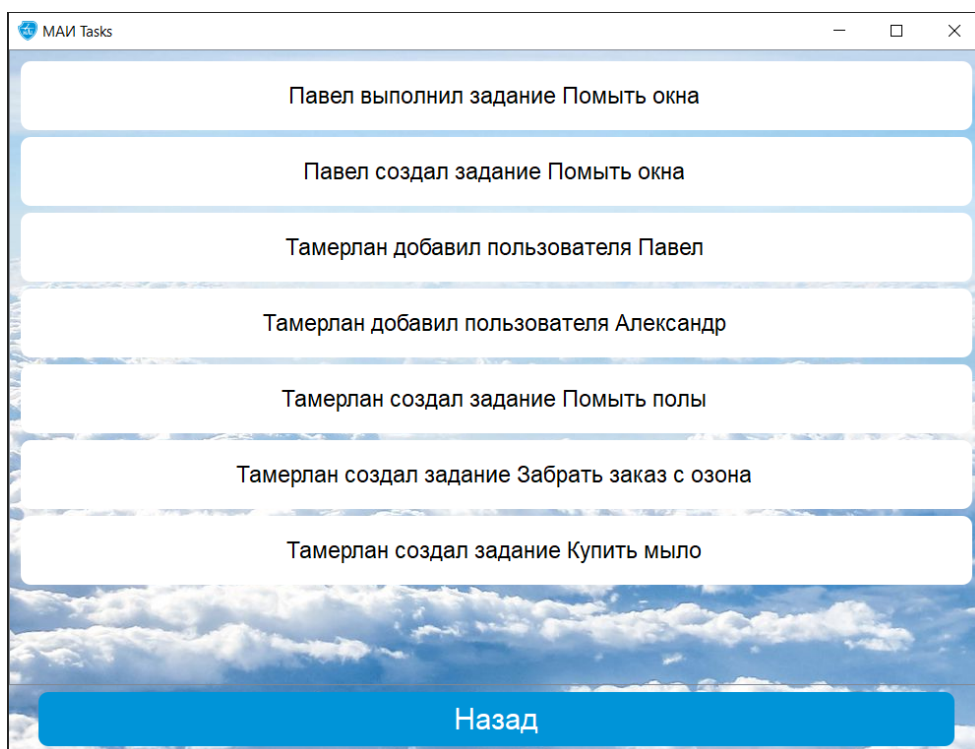
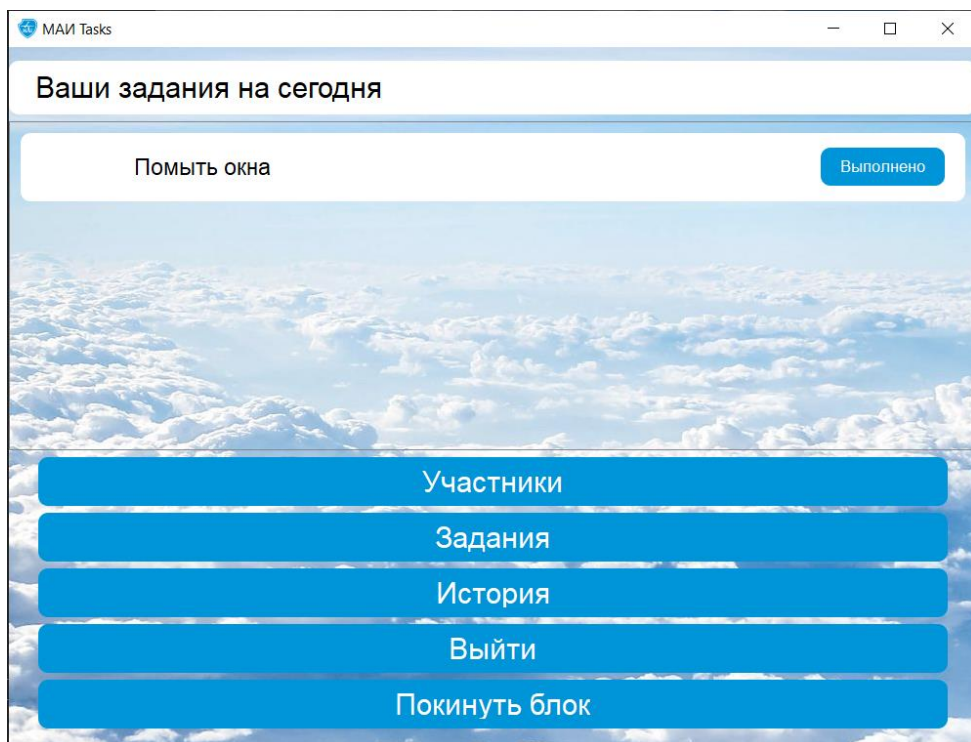


Теперь вернемся обратно и посмотри на наше задание в разделе  
«Все задания»



## История

В при нажатии на кнопку выполнено в разделе заданий, ваше выполненное задание записывается в историю блока и все участники могут ее просмотреть.



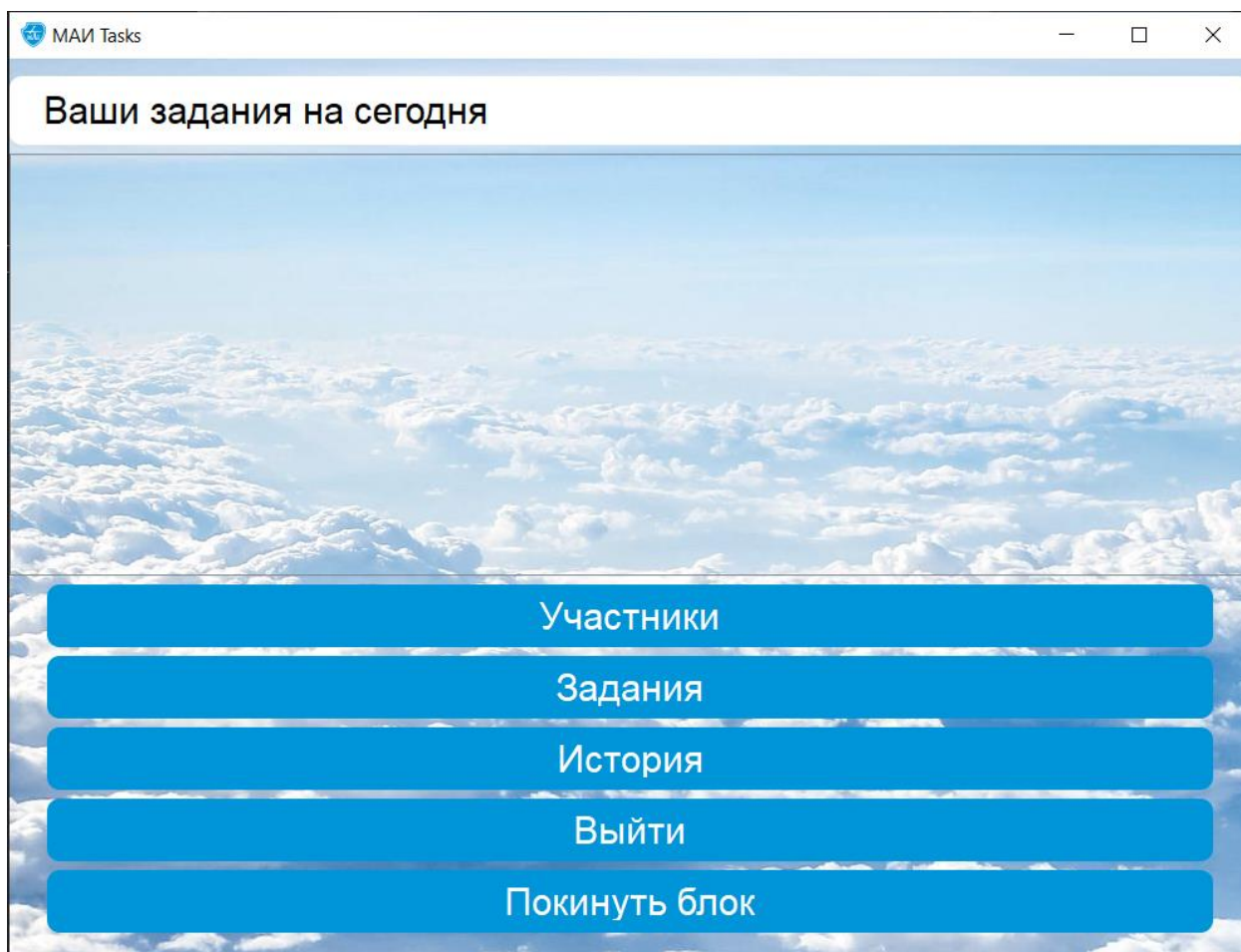


## Выход из блока

В главном меню мы можете «Выйти» и попасть в окно авторизации или же покинуть блок и вернуться в раздел с выбором блока.

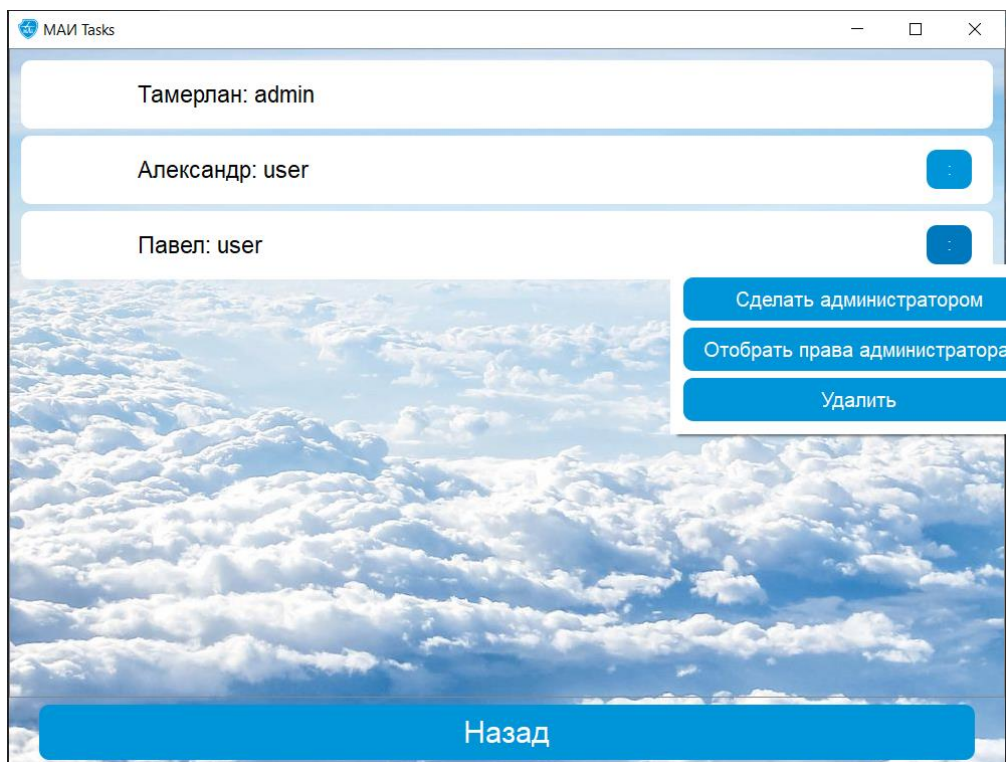
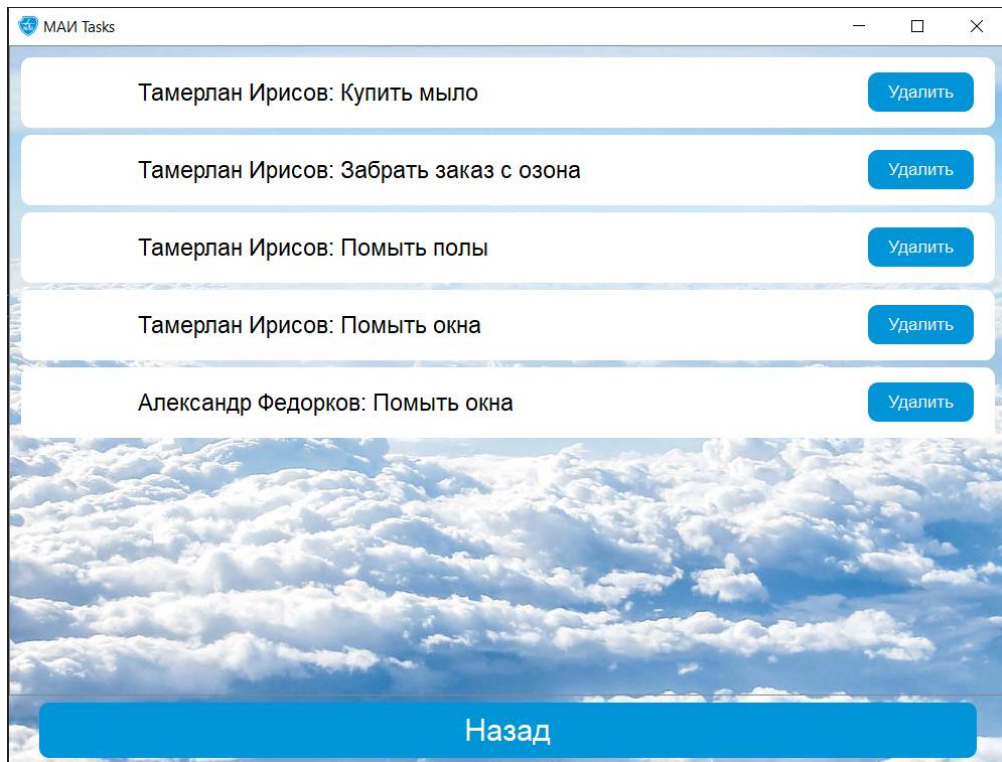
Если единственный администратор подкидает блок, то права администратора передаются следующему участнику по порядку.

Если участник блока является последний и покидает его, то блок становится доступным для создания другими пользователями.



## Функционал администратора

Администратор может: принимать и отклонять заявки, удалять задания у любых участников, удалять участников из блока, менять роли у всех участников в блоке и делать все что может обычный участник группы.



## Список используемых источников.

- 1) [«Запускаем PostgreSQL в Docker»](#)
- 2) [«Методы хэширования паролей»](#)
- 3) [«PsyCorg2 Docs»](#)
- 4) [«GRPC python»](#)
- 5) [«Qt Designer Manual»](#)
- 6) [«PyQt5 Guide»](#)

## [Репозиторий проекта](#)

