**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

СОГЛАСОВАНО

Приглашенный преподаватель
департамента программной инженерии
факультета компьютерных наук

_____ А. Н. Степанов

«*3*» *мая* 2022 г.

УТВЕРЖДАЮ

Академический руководитель
образовательной программы
«Программная инженерия», канд. техн. наук

_____ В. В. Шилов

«12 » _____мая_____ 2022 г.

| | |
|---|---|
| Подп. и дата | |
| Инв. № дубл. | |
| Взам. Инв. № | |
| Подп. и дата | |
| Инв. № подл. | |

# СЕРВЕРНАЯ ЧАСТЬ МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ПОМОЩИ В ЗАПОМИНАНИИ ТЕОРИТИЧЕСКОЙ ЧАСТИ ДИСЦИПЛИН

**Руководство программиста**

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU.17701729.05.10-01 РП 01-1-ЛУ**

Исполнитель:

_____ / И. Н. Дедов /

«*2*» *мая* 2022 г.

**Москва 2022**

УТВЕРЖДЕН

RU.17701729.05.10-01 РП 01-1-ЛУ

# СЕРВЕРНАЯ ЧАСТЬ МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ПОМОЩИ В ЗАПОМИНАНИИ ТЕОРИТИЧЕСКОЙ ЧАСТИ ДИСЦИПЛИН

**Руководство программиста**

**RU.17701729.05.10-01 РП 01-1**

**Листов 62**

| Подп. и дата | |
|---|---|
| Инв. № дубл. | |
| Взам. Инв. № | |
| Подп. и дата | |
| Инв. № подл. | |

**Москва 2022**

**СОДЕРЖАНИЕ**

# ГЛОССАРИЙ

1) *Пользователь* – человек, который использует приложение «Stoady». В контексте приложения «Stoady» существует два типа пользователей: «студенты» и «администраторы».
2) *Студент* (также, *участник, участник команды*) – пользователь, цель которого – лучше разобраться в теоретическом материале дисциплин.
3) *Администратор* – пользователь, имеющий доступ к редактированию материалов, содержащихся в приложении «Stoady», а именно:
   - добавление, удаление и редактирование прав участников команд;
   - добавление, удаление и редактирование предметов, тем и вопросов, содержащихся в команде.
4) *Команда* (также, *группа*) – группа людей, объединенных общим набором предметов.
5) *Предмет* (также, *дисциплина*) – верхний уровень иерархии теоретических материалов в приложении, который может содержать внутри себя одну или несколько тем.
6) *Тема* – раздел предмета, содержащий необходимую для изучения теорию и соответствующие ей вопросы.
7) *Вопрос* – элемент нижнего уровня иерархии теории в приложении. Вопрос содержит в себе сам текст вопроса, а также ответ на него.
8) *СУБД* – система управления базами данных.
9) *API* (Application Programming Interface) – описание способов (набор классов, методов и т. п.), которыми одна компьютерная программа (в данном случае, клиентская часть приложения) может взаимодействовать с другой (в данном случае, с сервером).
10) *JSON* (JavaScript Object Notation) – текстовый формат обмена данными, основанный на языке программирования JavaScript. Но при этом формат независим от JavaScript и может использоваться в любом языке программирования.
11) *Контроллер* – объект в API, предназначенный для обработки запросов. На вход он получает запрос, затем обрабатывает его одним из методов (хэндлеров) и выдает ответ пользователю.
12) *Хэндлер* – метод, обрабатывающий определенный запрос.

# 1. НАЗНАЧЕНИЕ ПРОГРАММЫ

## 1.1. Функциональное назначение

Мобильное приложение «Stoady» представляет собой сервис для помощи учащимся вузов и других учебных заведений в подготовке к различным элементам контроля. Сервис реализует в себе функциональность, позволяющую пользователям выбирать нужный предмет, а также тему для изучения внутри этого предмета.

Каждый пользователь может создавать команды и приглашать в них других пользователей, чтобы предоставить им доступ к материалам, содержащимся в их команде. Это позволит учащимся адаптировать приложение под образовательную программу их учебного заведения, что позволит лучше ее усвоить: у учащихся не будет необходимости в поиске и фильтрации информации из сети Интернет – вся теория, нужная им, будет сосредоточена в одной команде.

Серверная часть мобильного приложения «Stoady» предназначена для реализации внутренней логики приложения, управления безопасностью приложения, в том числе сохранности данных пользователей приложения, обеспечения взаимодействия с базой данных, и, наконец, предоставления своего API для использования клиентской частью мобильного приложения.

## 1.2. Эксплуатационная назначение

Мобильное приложение «Stoady» будет применяться в сфере образования и обучения. Инициативные студенты (а также преподаватели учебных заведений) могут в доступном формате давать необходимую теорию своим сокурсникам (или же подопечным студентам), которая будет агрегирована в одном приложении, что избавит студентов от необходимости хранить все ресурсы и ссылки на них в каких-либо хранилищах (например, Google Drive или Яндекс.Диск). Это улучшит пользовательский опыт студентов при подготовке к элементам контроля, а наличие функциональности для отслеживания статистики при изучении теории будет способствовать более эффективному обучению: каждый студент будет понимать, какие темы ему стоит повторить, на каких вопросах следует заострить внимание.

Целевая аудитория приложения – студенты, которые хотят лучше изучить теоретическую часть дисциплин или помочь своим сокурсникам сделать это благодаря добавлению необходимой теории в свою команду.

Серверная часть приложения «Stoady» предоставляет API, который будет использоваться клиентской частью приложения. Клиент будет получать доступ к необходимым ему данным из базы данных посредством вызова конечных точек API.

# 2. УСЛОВИЯ ВЫПОЛНЕНИЯ ПРОГРАММЫ

## 2.1. Минимальный состав аппаратных средств

Для успешного локального запуска программы необходимы:

1) Персональный компьютер с операционной системой Windows, Linux (любой дистрибутив) или MacOS;
2) Не менее 1 Гб свободной памяти на устройстве;
3) Не менее 2 Гб оперативной памяти.

Для взаимодействия с работающем на удаленном сервере приложением помимо перечисленных выше средств, необходимы также:

4) Доступ к сети Интернет.

## 2.2. Минимальный состав технических средств

Для наиболее удобной работы с приложением требуется следующий минимальный состав технических средств:

1) IDE JetBrains Rider (или другая IDE);
2) Установленный .NET;
3) Установленный PostgreSQL версии 13 или выше, а также DataGrip или pgAdmin для работы с ним;
4) Установленный Docker.

## 2.3. Требования к персоналу (пользователю, программисту)

Программист, использующий приложение, должен знать язык программирования C# (версии 9.0 или выше), а также иметь опыт работы с фреймворком ASP .NET Core [1], иметь знания в области реляционных баз данных, а также опыт работы с БД PostgreSQL (версии 13 или выше).

Программист должен иметь представление о клиент–серверном взаимодействии, а также обладать базовыми знаниями о Docker [2] и контейнеризации приложений.

# 3. ВЫПОЛНЕНИЕ ПРОГРАММЫ

## 3.1. Установка приложения

Для того, чтобы локально запустить серверную часть приложения, требуется склонировать репозиторий, находящийся по ссылке https://github.com/Ivan-Dedov/StoadyBackend на свое устройство с помощью Git (или приложений типа GitHub Desktop, GitKraken и т. п.).

Для получения доступа к API, находящемуся на удаленном сервере, следует перейти по ссылке https://stoady.herokuapp.com/swagger/index.html.

Корень API находится по адресу https://stoady.herokuapp.com.

## 3.2. Запуск приложения

Для подключения к базе данных, необходимо ввести в консоль команду `psql -h ec2-52-211-158-144.eu-west-1.compute.amazonaws.com -p 5432 -U rxxaapxbpdyrlk -d d9elrdlh8nmq04` или подключиться к ней через DataGrip или pgAdmin, используя данные хоста, порта и т. п. из приведенной выше строки.

Для локального запуска API необходимо ввести команду `dotnet run Stoady`, находясь в корне репозитория или запустить проект в IDE.

## 3.3. Работа с API

API, предоставляемое серверной частью, дает доступ к конечным точкам (можно найти по ссылке: https://stoady.herokuapp.com/swagger/index.html), вызывая которые, можно получать обработанные данные из базы данных:

### 3.3.1. Авторизация

- **`/auth/authorize`**

*Тип конечной точки*: POST

*Что делает конечная точка*: по введенному адресу электронной почты и паролю возвращает данные о пользователе.

*Параметры* (AuthenticationRequest):

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| email | string | Тело запроса | Адрес электронной почты пользователя, которого необходимо авторизовать | Обязательный |
| password | string | Тело запроса | Пароль пользователя, которого необходимо авторизовать | Обязательный |

*Возвращаемые данные* (AuthenticationResponse):

| Имя | Тип | Описание данных |
|---|---|---|
| id | long | Идентификатор пользователя из таблицы Users |
| name | string | Имя пользователя |
| avatarId | int | Идентификатор аватара пользователя (число от 0 до 14 включительно) |

*Описание алгоритма работы*:

В контроллере AuthenticationController вызывается хэндлер AuthorizationCommandHandler, который получает из таблицы user информацию о пользователе по адресу его электронной почты, передает полученный в запросе пароль в сервис PasswordValidatorService, который хэширует полученный пароль с помощью алгоритма SHA-256 [3] и сравнивает с хэшом, лежащим в базе данных.

Если хэши совпали, то контроллер возвращает данные о пользователе, иначе возвращает ошибку 401 Unauthorized.

### 3.3.2. Вопросы

- **`/questions/{topicId}`**

*Тип конечной точки*: GET

*Что делает конечная точка*: по введенному идентификатору темы получает вопросы этой темы.

*Параметры*:

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| topicId | long | Путь | Идентификатор темы из таблицы Topics | Обязательный, больше нуля |

*Возвращаемые данные* (GetQuestionsResponse) – массив объектов со следующим содержанием:

| Имя | Тип | Описание данных |
|---|---|---|
| id | long | Идентификатор вопроса из таблицы Questions |
| questionText | string | Текст вопроса |
| answerText | string | Текст ответа на вопрос |

*Описание алгоритма работы*:

В контроллере QuestionController вызывается хэндлер GetQuestionsCommandHandler, который получает из таблицы Questions вопросы, у которых значение topicId равно значению из запроса.

- **`/questions/saved/{userId}`**

*Тип конечной точки*: GET

*Что делает конечная точка*: по введенному идентификатору пользователя получает его сохраненные вопросы.

*Параметры*:

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| userId | long | Путь | Идентификатор пользователя из таблицы Users | Обязательный, больше нуля |

*Возвращаемые данные* (GetSavedQuestionsResponse) – массив объектов со следующим содержанием:

| Имя | Тип | Описание данных |
|---|---|---|
| id | long | Идентификатор вопроса из таблицы Questions |
| questionText | string | Текст вопроса |

| | | |
|---|---|---|
| answerText | string | Текст ответа на вопрос |

*Описание алгоритма работы*:

В контроллере QuestionController вызывается хэндлер GetSavedQuestionsCommandHandler, который получает из таблицы UserQuestions сохраненные вопросы, у которых значение userId равно значению userId из запроса.

- **`/questions/add`**

*Тип конечной точки*: POST

*Что делает конечная точка*: создает новый вопрос.

*Параметры* (AddQuestionRequest):

| **Имя** | **Тип** | **Откуда берется** | **Смысл параметра** | **Ограничения** |
|---|---|---|---|---|
| topicId | long | Тело запроса | Идентификатор темы из таблицы Topics, в которую требуется добавить новый вопрос | Обязательный, больше нуля |
| questionText | string | Тело запроса | Текст нового вопроса | Обязательный, длиной от 1 до 100 символов |
| answerText | string | Тело запроса | Текст ответа на новый вопрос | Обязательный, длиной от 1 до 100 символов |

*Возвращаемые данные*: нет.

*Описание алгоритма работы*:

В контроллере QuestionController вызывается хэндлер AddQuestionCommandHandler, который добавляет в базу данных, а именно в таблицу Questions, новый вопрос, с внешним ключом topicId.

Если темы с идентификатором topicId не обнаружено, возвращается ошибка с кодом 400 Bad Request.

- **`/questions/edit/{questionId}`**

*Тип конечной точки*: POST

*Что делает конечная точка*: редактирует существующий вопрос.

*Параметры* (EditQuestionRequest):

| **Имя** | **Тип** | **Откуда берется** | **Смысл параметра** | **Ограничения** |
|---|---|---|---|---|
| questionId | long | Путь | Идентификатор вопроса из таблицы Questions | Обязательный, больше нуля |
| questionText | string | Тело запроса | Новый текст вопроса | Обязательный, длиной от 1 до 100 символов |
| answerText | string | Тело запроса | Новый ответ на вопрос | Обязательный, длиной от 1 до 100 символов |

*Возвращаемые данные*: нет.

*Описание алгоритма работы*:

В контроллере QuestionController вызывается хэндлер EditQuestionCommandHandler, который редактирует в базе данных (в таблице Questions) вопрос с первичным ключом questionId. Если же вопроса с идентификатором questionId не обнаружено, возвращается ошибка с кодом 400 Bad Request.

- **`/questions/save/{questionId}`**

*Тип конечной точки*: PUT

*Что делает конечная точка*: по идентификатору пользователя и вопроса добавляет вопрос в сохраненные.

*Параметры*:

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| questionId | long | Путь | Идентификатор вопроса из таблицы Questions | Обязательный, больше нуля |
| userId | long | Хэдер запроса | Идентификатор вопроса из таблицы Users | Обязательный |

*Возвращаемые данные*: нет.

*Описание алгоритма работы*:

В контроллере QuestionController вызывается хэндлер SaveQuestionCommandHandler, который добавляет в таблицу UserQuestions новую запись с userId и questionId, переданными из запроса. Если же вопроса или пользователя с указанными идентификаторами нет, то возвращается ошибка с кодом 400 Bad Request.

- **`/questions/remove/{questionId}`**

*Тип конечной точки*: DELETE

*Что делает конечная точка*: удаляет вопрос с указанным идентификатором.

*Параметры*:

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| questionId | long | Путь | Идентификатор вопроса из таблицы Questions, который надо удалить. | Обязательный |

*Возвращаемые данные*: нет.

*Описание алгоритма работы*:

В контроллере QuestionController вызывается хэндлер RemoveQuestionCommandHandler, который удаляет из таблицы Questions вопрос с указанным идентификатором. Вследствие политики удаления ON DELETE CASCADE в

таблице UserQuestions, вопрос удалится и из сохраненных. Если вопроса с таким идентификатором нет, то вернется ошибка 400 Bad Request.

- **`/questions/unsave/{questionId}`**

*Тип конечной точки*: DELETE

*Что делает конечная точка*: убирает вопрос из сохраненных у выбранного пользователя.

*Параметры*:

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| questionId | long | Путь | Идентификатор вопроса из таблицы Questions, который необходимо удалить из сохраненных | Обязательный, больше нуля |
| userId | long | Хэдер запроса | Идентификатор пользователя из таблицы Users, у которого необходимо удалить вопрос | Обязательный |

*Возвращаемые данные*: нет.

*Описание алгоритма работы*:

В контроллере QuestionController вызывается хэндлер RemoveSavedQuestionCommandHandler, который удаляет из таблицы UserQuestions запись с указанными идентификаторами userId и questionId. Если вопроса или пользователя с такими идентификаторами нет, то вернется ошибка 400 Bad Request.

### 3.3.3. Статистика

- **`/stats/{userId}`**

*Тип конечной точки*: GET

*Что делает конечная точка*: возвращает статистику по пройденным темам у указанного пользователя.

*Параметры*:

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| userId | long | Путь | Идентификатор пользователя из таблицы Users, статистику которого нужно найти | Обязательный, больше нуля |

*Возвращаемые данные* (GetUserStatisticsResponse) – массив объектов со следующим содержанием:

| Имя | Тип | Описание данных |
|---|---|---|
| topicId | long | Идентификатор темы из таблицы Topics |
| topicName | string | Имя темы |
| result | int | Процент выполнения темы, округленный до целого числа |

*Описание алгоритма работы*:

В контроллере StatisticsController вызывается хэндлер GetUserStatisticsCommandHandler, который получает из таблицы Statistics информацию о пройденных пользователем тестах.

### 3.3.4. Дисциплины

- **`/subjects/{subjectId}`**

*Тип конечной точки*: GET

*Что делает конечная точка*: по введенному идентификатору предмета получает информацию о нем.

*Параметры*:

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| subjectId | long | Путь | Идентификатор предмета из таблицы Subjects, информацию о котором нужно найти | Обязательный, больше нуля |

*Возвращаемые данные* (GetSubjectInfoResponse):

<table>
<tr><th>Имя</th><th>Тип</th><th>Описание данных</th></tr>
<tr><td>name</td><td>string</td><td>Название дисциплины</td></tr>
<tr><td>description</td><td>string</td><td>Описание дисциплины</td></tr>
<tr><td>topics</td><td>Topic[]</td><td>Массив тем, о каждой из которых известно:
<table>
<tr><th>Имя</th><th>Тип</th><th>Описание данных</th></tr>
<tr><td>id</td><td>long</td><td>Идентификатор темы из таблицы Topics</td></tr>
<tr><td>name</td><td>string</td><td>Название темы</td></tr>
</table></td></tr>
</table>

*Описание алгоритма работы*:

В контроллере SubjectController вызывается хэндлер GetSubjectInfoCommandHandler, который получает из таблицы Subjects данные о предмете и соединяет их с помощью INNER JOIN с данными таблицы Topics, у которых идентификатор предмета совпадает с введенным.

- **`/subjects/add`**

*Тип конечной точки*: POST

*Что делает конечная точка*: создает новую дисциплину.

*Параметры* (AddSubjectRequest):

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| teamId | long | Тело запроса | Идентификатор команды из таблицы Teams, в которую нужно добавить дисциплину | Обязательный, больше нуля |
| subjectName | string | Тело запроса | Название новой дисциплины | Обязательный, длиной от 1 до 250 символов |

| | | | | |
|---|---|---|---|---|
| subjectDescription | string | Тело запроса | Описание новой дисциплины | Длиной от 0 до 250 символов |

*Возвращаемые данные*: нет.

*Описание алгоритма работы*:

В контроллере SubjectController вызывается хэндлер AddSubjectCommandHandler, который добавляет в таблицу Subjects новую запись с указанными данными.

- **`/subjects/edit/{subjectId}`**

*Тип конечной точки*: POST

*Что делает конечная точка*: редактирует информацию о дисциплине.

*Параметры* (EditSubjectRequest):

| **Имя** | **Тип** | **Откуда берется** | **Смысл параметра** | **Ограничения** |
|---|---|---|---|---|
| subjectId | long | Путь | Идентификатор дисциплины из таблицы Subjects, которую нужно отредактировать | Обязательный, больше нуля |
| subjectName | string | Тело запроса | Новое название дисциплины | Обязательный, длиной от 1 до 250 символов |
| subjectDescription | string | Тело запроса | Новое описание дисциплины | Длиной от 0 до 250 символов |

*Возвращаемые данные*: нет.

*Описание алгоритма работы*:

В контроллере SubjectController вызывается хэндлер EditSubjectCommandHandler, который редактирует информацию о предмете с введенным идентификатором. Если же такого предмета нет, то возвращает ошибку 400 Bad Request.

- **`/subjects/remove/{subjectId}`**

*Тип конечной точки*: DELETE

*Что делает конечная точка*: удаляет дисциплину.

*Параметры* (RemoveSubjectRequest):

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| subjectId | long | Путь | Идентификатор дисциплины из таблицы Subjects, которую нужно удалить | Обязательный, больше нуля |

*Возвращаемые данные*: нет.

*Описание алгоритма работы*:

В контроллере SubjectController вызывается хэндлер RemoveSubjectCommandHandler, который удаляет запись из таблицы Subjects с указанным идентификатором. Вследствие политики удаления ON DELETE CASCADE, из таблиц Topics, Questions, UserQuestions и Statistics также удалятся данные, зависящие от дисциплины (темы внутри нее, вопросы внутри тем, сохраненные вопросы и статистика по темам).

Если предмета с указанным идентификатором нет, то конечная точка возвращает ошибку 400 Bad Request.

### 3.3.5. Команды

- **`/teams/{teamId}/select`**

*Тип конечной точки*: POST

*Что делает конечная точка*: по введенному идентификатору команды и пользователя возвращает роль этого пользователя в команде.

*Параметры*:

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| teamId | long | Путь | Идентификатор команды из таблицы Teams, участника которой нужно найти | Обязательный, больше нуля |
| userId | long | Параметры запроса | Идентификатор участника из таблицы Users, роль которого нужно найти | Обязательный |

*Возвращаемые данные* (SelectTeamResponse):

| Имя | Тип | Описание данных |
|---|---|---|
| role | enum | Роль пользователя в команде. Может принимать одно из значений: «Creator», «Admin», «Member». |

*Описание алгоритма работы*:

В контроллере TeamController вызывается хэндлер SelectTeamCommandHandler, который получает из таблицы TeamUsers информацию об участнике команды, а затем преобразовывает полученный идентификатор роли roleId к значению перечисления Role.

Если пользователь с указанным идентификатором не состоит в команде с идентификатором teamId, то возвращается ошибка 400 Bad Request.

- **`/teams/{teamId}`**

*Тип конечной точки*: GET

*Что делает конечная точка*: по введенному идентификатору команды получает информацию о ней.

*Параметры*:

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| teamId | long | Путь | Идентификатор команды из таблицы Teams, информацию о которой нужно получить | Обязательный, больше нуля |

*Возвращаемые данные* (GetTeamInfoResponse):

<table>
<tr><th>Имя</th><th>Тип</th><th>Описание данных</th></tr>
<tr><td>name</td><td>string</td><td>Название команды</td></tr>
<tr><td>picture</td><td>string</td><td>Ссылка на картинку-аватар команды</td></tr>
<tr><td>subjects</td><td>Subject[]</td><td>Массив дисциплин в команде, о каждой из которых известно:
<table>
<tr><th>Имя</th><th>Тип</th><th>Описание данных</th></tr>
<tr><td>id</td><td>long</td><td>Идентификатор дисциплины из таблицы Subjects</td></tr>
<tr><td>name</td><td>string</td><td>Название дисциплины</td></tr>
<tr><td>description</td><td>string</td><td>Описание дисциплины</td></tr>
</table></td></tr>
</table>

*Описание алгоритма работы*:

В контроллере TeamController вызывается хэндлер GetTeamInfoCommandHandler, который получает из таблицы Teams информацию о теме с указанным идентификатором, а также получает из таблицы Subjects информацию о предметах, у которых внешний ключ teamId совпадает с введенным в параметрах значением. Затем результаты поиска по обеим таблицам объединяются и возвращаются пользователю.

- **`/teams/{teamId}/members`**

*Тип конечной точки*: GET

*Что делает конечная точка*: по введенному идентификатору команды получает ее участников.

*Параметры*:

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| teamId | long | Путь | Идентификатор команды из таблицы Teams | Обязательный, больше нуля |

*Возвращаемые данные* (GetTeamMembersResponse) – массив объектов со следующей информацией о пользователях:

| Имя | Тип | Описание данных |
|---|---|---|
| id | long | Идентификатор пользователя из таблицы Users |
| username | string | Имя пользователя |
| email | string | Адрес электронной почты пользователя |
| role | enum | Роль пользователя в команде. Одно из значений перечисления: «Creator», «Admin», «Member». |

*Описание алгоритма работы*:

В контроллере TeamController вызывается хэндлер GetTeamMembersCommandHandler, который из таблицы TeamUsers получает идентификаторы всех пользователей-участников команды с указанным идентификатором teamId, и посредством INNER JOIN объединяет данные с таблицей Users. После этого происходит преобразование идентификатора роли участника в команде к значению перечисления Role.

- **`/teams/{teamId}/members`**

*Тип конечной точки*: PUT

*Что делает конечная точка*: изменяет роль выбранного участника в команде.

*Параметры*:

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| executorId | Long | Хэдер запроса | Идентификатор пользователя, выполняющего изменение роли, из таблицы Users | Обязательный, больше нуля |
| teamId | long | Путь | Идентификатор команды из таблицы Teams | Обязательный, больше нуля |

| | | | | |
|---|---|---|---|---|
| userId | long | Параметры запроса | Идентификатор пользователя, роль которого изменяют, из таблицы Users | Обязательный, больше нуля, не равен executorId |
| userRole | enum | Параметры запроса | Новая роль пользователя. Одно из значений перечисления: «Creator», «Admin», «Member». | Обязательный, не равен «Creator» |

*Возвращаемые данные*: нет.

*Описание алгоритма работы*:

В контроллере TeamController вызывается хэндлер ChangeMemberStatusCommandHandler, который сначала проверяет целесообразность действий пользователя: есть ли в команде такой участник, не меняется ли роль создателя команды.

Затем, если эти проверки пройдены, с помощью сервиса RightsValidatorService, проверяем доступность действий по изменению роли у пользователя с идентификатором executorId. Если у него есть права на это, в таблице TeamUsers меняем roleId на нужное значение.

При отсутствии прав на выполнение действия или отсутствии участника с идентификатором userId в команде с идентификатором teamId, возвращается ошибка с кодом 400 Bad Request.

- **`/teams/create`**

*Тип конечной точки*: POST

*Что делает конечная точка*: создает новую команду.

*Параметры*:

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| userId | long | Параметры запроса | Идентификатор пользователя, создателя команды, из таблицы Users | Обязательный, больше нуля |
| teamName | string | Параметры запроса | Название новой команды | Обязательный, есть не пробельные символы |

*Возвращаемые данные*: нет.

*Описание алгоритма работы*:

В контроллере TeamController вызывается хэндлер CreateTeamCommandHandler, который создает новую команду в таблице Teams, добавляет в нее участника с идентификатором userId в качестве создателя, предварительно найдя созданную команду по имени и времени создания (берем самую новую).

- **`/teams/{teamId}/avatar`**

*Тип конечной точки*: POST

*Что делает конечная точка*: редактирует название и аватар команды.

*Параметры* (EditTeamRequest):

| **Имя** | **Тип** | **Откуда берется** | **Смысл параметра** | **Ограничения** |
|---|---|---|---|---|
| executorId | long | Хэдер запроса | Идентификатор пользователя, выполняющего редактирование команды, из таблицы Users | Обязательный, больше нуля |
| teamId | long | Путь | Идентификатор редактируемой команды из таблицы Teams | Обязательный, больше нуля |

| | | | | |
|---|---|---|---|---|
| teamName | string | Тело запроса | Новое название команды | Обязательный, состоит не только из пробельных символов |
| teamAvatar | string | Тело запроса | Новая ссылка на картинку-аватар команды | Обязательный, состоит не только из пробельных символов |

*Возвращаемые данные*: нет.

*Описание алгоритма работы*:

В контроллере TeamController вызывается хэндлер EditTeamCommandHandler, который с помощью сервиса RightsValidatorService проверяет наличие у пользователя с идентификатором executorId прав на редактирование команды, а затем изменяет данные таблицы Teams на те, которые были получены из запроса.

Если у пользователя нет прав на выполнение действия, возвращается ошибка 400 Bad Request.

- **`/teams/{teamId}/members/add`**

*Тип конечной точки*: POST

*Что делает конечная точка*: добавляет пользователя в команду.

*Параметры*:

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| executorId | long | Хэдер запроса | Идентификатор пользователя, добавляющего нового участника в | Обязательный, больше нуля |

| | | | | |
|---|---|---|---|---|
| | | | команду, из таблицы Users | |
| teamId | long | Путь | Идентификатор команды из таблицы Teams | Обязательный, больше нуля |
| email | string | Тело запроса | Адрес электронной почты нового участника команды | Обязательный, состоит не только из пробельных символов, является валидным адресом электронной почты |

*Возвращаемые данные*: нет.

*Описание алгоритма работы*:

В контроллере TeamController вызывается хэндлер AddMemberCommandHandler, который сначала с помощью сервиса RightsValidatorService проверяет у пользователя с идентификатором executorId наличие прав на добавление нового участника, и при наличии таковых прав ищет участника в таблице Users по его адресу электронной почты. Затем найденный участник добавляется в команду с ролью «Member».

Если в базе данных не нашлось команды с нужным идентификатором, участника с требуемым адресом электронной почты или пользователя с идентификатором executorId, возвращается ошибка 400 Bad Request.

- **`/teams/{teamId}/members/remove`**

*Тип конечной точки*: DELETE

*Что делает конечная точка*: удаляет участника из команды.

*Параметры*:

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| executorId | long | Хэдер запроса | Идентификатор пользователя, удаляющего | Обязательный, больше нуля |

| | | | | |
|---|---|---|---|---|
| | | | участника из команды, из таблицы Users | |
| teamId | long | Путь | Идентификатор команды из таблицы Teams | Обязательный, больше нуля |
| userId | long | Параметры запроса | Идентификатор участника, которого нужно удалить из команды, из таблицы Users | Обязательный, больше нуля |

*Возвращаемые данные*: нет.

*Описание алгоритма работы*:

В контроллере TeamController вызывается хэндлер RemoveMemberCommandHandler, который сперва валидирует действия пользователя: существует ли команда с идентификатором teamId, имеет ли смысл удаление участника (так как создателя удалять нельзя). Если эти проверки были пройдены, с помощью сервиса RightsValidatorService проверяется наличие у пользователя с идентификатором executorId прав на удаление участника.

Если у пользователя есть такие права, то из таблицы TeamUsers удаляется запись, содержащая информацию о том, что участник с идентификатором userId был членом команды teamId.

Если в запросе будут указаны неверные параметры, или идентификаторы не будут соответствовать никакой команде или никакому пользователю, возвращается ошибка 400 Bad Request.

### 3.3.6. Тестирование

- **`/tests`**

*Тип конечной точки*: POST

*Что делает конечная точка*: сохраняет результаты тестирования по теме.

*Параметры* (SaveTestResultsRequest):

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| userId | string | Тело запроса | Идентификатор пользователя из таблицы Users, результаты тестирования которого нужно сохранить | Обязательный, больше нуля |
| topicId | string | Тело запроса | Идентификатор темы из таблицы Topics, по которой проводилось тестирование | Обязательный, больше нуля |
| result | int | Тело запроса | Процент правильных ответов на вопросы темы, округленный до целого числа | Обязательный, в пределах от 0 до 100 включительно |

*Возвращаемые данные*: нет.

*Описание алгоритма работы*:

В контроллере TestingController вызывается хэндлер SaveTestResultsCommandHandler, который берет из таблицы Statistics всю статистику по пользователю с идентификатором userId и определяет, есть ли в ней запись с введенным идентификатором темы. Если есть, то редактирует ее, иначе создает новую запись.

Если любой из идентификаторов topicId или userId не найден или результат тестирования не лежит в заданном отрезке, то возвращается ошибка с кодом 400 Bad Request.

### 3.3.7. Темы

- **`/topics/{topicId}`**

*Тип конечной точки*: GET

*Что делает конечная точка*: по введенному идентификатору темы получает информацию о ней.

*Параметры*:

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| topicId | long | Путь | Идентификатор темы из таблицы Topics, информацию о которой нужно найти | Обязательный, больше нуля |

*Возвращаемые данные* (GetTopicInfoResponse):

| Имя | Тип | Описание данных |
|---|---|---|
| name | string | Название темы |
| description | string | Описание темы |
| questions | Question[] | Массив вопросов, о каждом из которых известно: |

| Имя | Тип | Описание данных |
|---|---|---|
| id | long | Идентификатор вопроса из таблицы Questions |

*Описание алгоритма работы*:

В контроллере TopicController вызывается хэндлер GetTopicInfoCommandHandler, который получает из таблицы Topics данные о теме и соединяет их с помощью INNER JOIN с данными таблицы Questions, у которых идентификатор темы совпадает с введенным, и возвращает необходимую информацию.

- **`/topics/add`**

*Тип конечной точки*: POST

*Что делает конечная точка*: создает новую тему.

*Параметры* (AddTopicRequest):

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| subjectId | string | Тело запроса | Идентификатор предмета из таблицы Subjects, в | Обязательный, больше нуля |

| | | | | |
|---|---|---|---|---|
| | | | который нужно добавить тему | |
| topicName | string | Тело запроса | Название новой темы | Обязательный, длиной от 1 до 250 символов |
| topicDescription | string | Тело запроса | Описание новой темы | Длиной от 0 до 250 символов |

*Возвращаемые данные*: нет.

*Описание алгоритма работы*:

В контроллере TopicController вызывается хэндлер AddTopicCommandHandler, который добавляет в таблицу Topics новую запись с указанными данными.

- **`/topics/edit/{topicId}`**

*Тип конечной точки*: POST

*Что делает конечная точка*: редактирует информацию о теме.

*Параметры* (EditTopicRequest):

| **Имя** | **Тип** | **Откуда берется** | **Смысл параметра** | **Ограничения** |
|---|---|---|---|---|
| topicId | long | Путь | Идентификатор темы из таблицы Topics, которую нужно отредактировать | Обязательный, больше нуля |
| topicName | string | Тело запроса | Новое название темы | Обязательный, длиной от 1 до 250 символов |
| topicDescription | string | Тело запроса | Новое описание темы | Длиной от 0 до 250 символов |

*Возвращаемые данные*: нет.

*Описание алгоритма работы*:

В контроллере TopicController вызывается хэндлер EditTopicCommandHandler, который редактирует информацию о теме с введенным идентификатором. Если же такой темы нет, то возвращает ошибку 400 Bad Request.

- **`/topics/remove/{topicId}`**

*Тип конечной точки*: DELETE

*Что делает конечная точка*: удаляет тему.

*Параметры* (RemoveTopicRequest):

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| topicId | long | Путь | Идентификатор темы из таблицы Topics, которую нужно удалить | Обязательный, больше нуля |

*Возвращаемые данные*: нет.

*Описание алгоритма работы*:

В контроллере TopicController вызывается хэндлер RemoveTopicCommandHandler, который удаляет запись из таблицы Topics с указанным идентификатором. Вследствие политики удаления ON DELETE CASCADE, из таблиц Questions, UserQuestions и Statistics также удалятся данные, зависящие от дисциплины (вопросы внутри темы, сохраненные вопросы в этой теме, и статистика по ней).

Если предмета с указанным идентификатором нет, то конечная точка возвращает ошибку 400 Bad Request.

### 3.3.8. Пользователи

- **`/users/teams`**

*Тип конечной точки*: GET

*Что делает конечная точка*: по переданному идентификатору пользователя получает список команд, в которых он состоит.

*Параметры*:

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| userId | long | Параметры запроса | Идентификатор пользователя из таблицы Users, команды которого нужно получить | Обязательный, больше нуля |

*Возвращаемые данные* (GetUserTeamsResponse) – массив объектов со следующим содержимым:

| Имя | Тип | Описание данных |
|---|---|---|
| teamId | long | Идентификатор команды из таблицы Teams |
| role | enum | Роль пользователя в команде с данным идентификатором. Принимает одно из значений: «Creator», «Admin», «Member». |
| teamName | string | Название команды |
| teamAvatar | string | Ссылка на картинку-аватар команды |

*Описание алгоритма работы*:

В контроллере UserController вызывается хэндлер GetUserTeamsCommandHandler, который из таблицы TeamUsers получает команды, в которых состоит пользователь вместе с ролью пользователя в этой команде, затем соединяет с помощью INNER JOIN с таблицей команд Teams, и наконец, для каждой из полученных записей преобразовывает идентификатор роли к ее названию, используя таблицу Roles.

- **`/users/register`**

*Тип конечной точки*: POST

*Что делает конечная точка*: регистрирует нового пользователя приложения.

*Параметры* (RegisterUserRequest):

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| username | string | Тело запроса | Имя пользователя, которого необходимо зарегистрировать | Обязательный, состоит не только из пробельных символов |
| email | string | Тело запроса | Адрес электронной почты пользователя, которого необходимо зарегистрировать | Обязательный, является валидным адресом электронной почты (проверка через регулярное выражение) |
| password | string | Тело запроса | Пароль пользователя, которого необходимо зарегистрировать | Обязательный, длиной от 5 до 30 символов включительно |
| avatarId | int | Тело запроса | Идентификатор аватара, который будет отображаться у пользователя в профиле | Обязательный, от 0 до 14 включительно |

*Возвращаемые данные* (RegisterUserResponse):

| **Имя** | **Тип** | **Описание данных** |
|---|---|---|
| userId | long | Идентификатор созданного пользователя в таблице Users |

*Описание алгоритма работы*:

В контроллере UserController вызывается хэндлер RegisterUserCommandHandler, который проверяет, что пользователя с введенным адресом электронной почты еще не существует, затем использует PasswordValidatorService для хэширования пароля, и создает новую запись в таблице Users.

После создания пользователя идет в таблицу Users и получает идентификатор пользователя с введенным адресом электронной почты, и возвращает его.

- **`/users/{userId}/avatar/set`**

*Тип конечной точки*: PUT

*Что делает конечная точка*: обновляет аватар пользователя.

*Параметры*:

| Имя | Тип | Откуда берется | Смысл параметра | Ограничения |
|---|---|---|---|---|
| userId | long | Путь | Идентификатор пользователя из таблицы Users, аватар которого нужно поменять | Обязательный, больше нуля |
| avatarId | int | Параметры запроса | Идентификатор аватара, который нужно поставить | Обязательный, от 0 до 14 включительно |

*Возвращаемые данные*: нет.

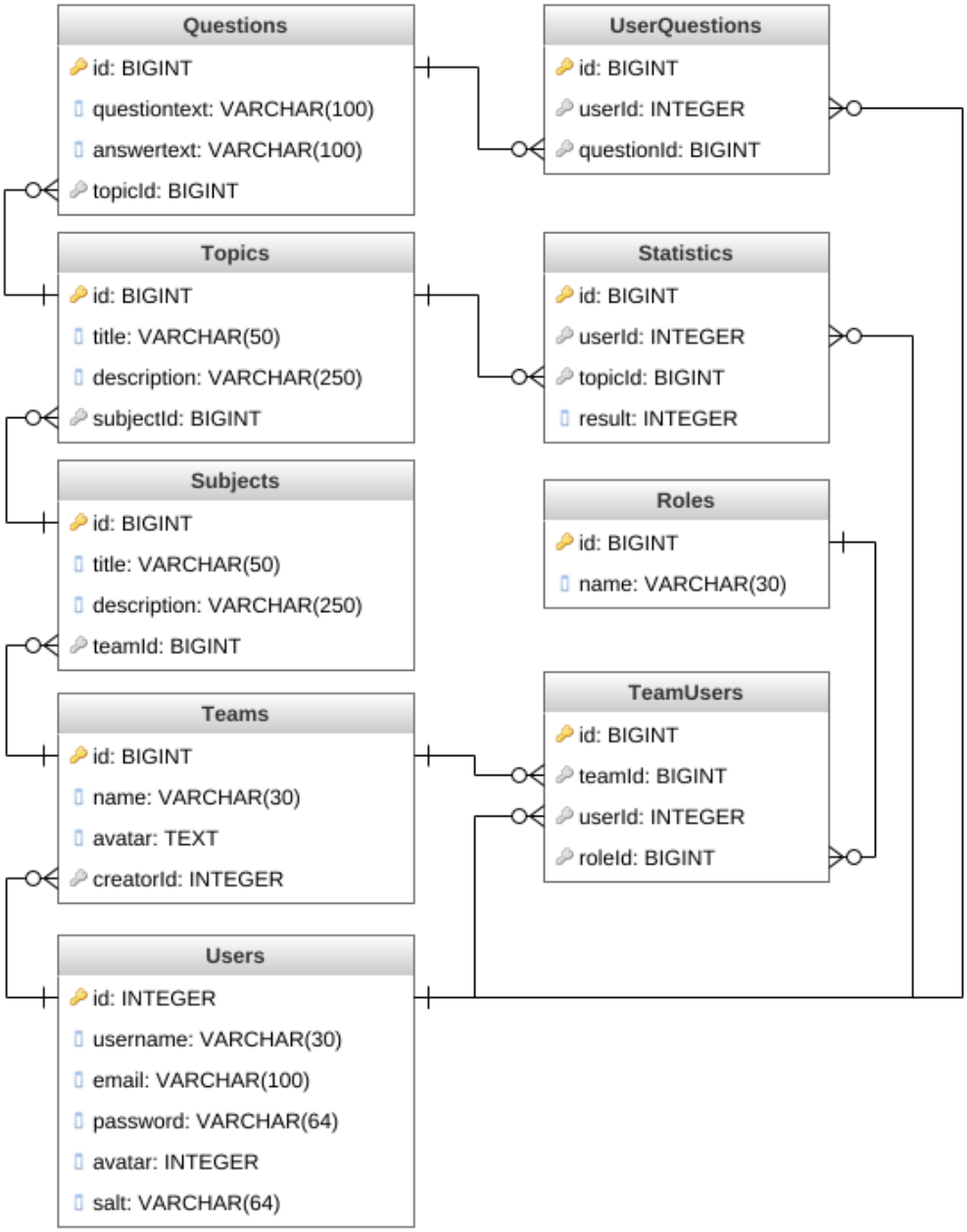
*Описание алгоритма работы*:

В контроллере UserController вызывается хэндлер UpdateUserAvatarCommandHandler, который изменяет аватар пользователя с указанным идентификатором в таблице Users на нужный.

### 3.4. Работа с базой данных

Для хранения данных используется реляционная база данных PostgreSQL. Взаимосвязь классов можно увидеть на UML-диаграмме:

*Диаграмма 1. Диаграмма взаимосвязей сущностей в базе данных*




Политика удаления сущностей в таблицах с внешними ключами – ON DELETE CASCADE.

Иерархическая структура приложения (см. левый столбец сущностей):

1) Questions – содержит в себе вопросы. Вопрос обязан принадлежать теме.
2) Topics – содержит в себе темы. В одной теме может быть много вопросов. Тема обязана принадлежать предмету.
3) Subjects – содержит в себе предметы (дисциплины). В одном предмете может быть много тем. Предмет обязан принадлежать команде.
4) Teams – описывает команду (группу) пользователей. В одной команде может быть много предметов. Команда обязательно имеет одного создателя.
5) Users – хранит в себе пользователей приложения. Один пользователь может быть создателем нескольких команд.

Принадлежность пользователей командам:

1) Один пользователь может состоять в нескольких командах.
2) Команда содержит в себе много пользователей.
3) У каждого пользователя в команде есть своя роль.
4) Для описания членов команды создана таблица TeamUsers – в одной записи содержится информация о паре «команда – пользователь», а также о роли, которую этот пользователь имеет в этой команде.

Статистика прохождения тем:

1) Один пользователь может проверять свои знания по нескольким темам.
2) Одна и та же тема может быть пройдена несколькими пользователями.
3) У каждого пользователя при прохождении темы есть свой результат.
4) Для описания успешности прохождения одним пользователем одной темы создана таблица Statistics, каждая запись которой содержит в себе информацию о паре «пользователь – тема», а также о результате прохождения тестирования этим пользователем этой темы.

Сохранение вопросов:

1) Один вопрос может быть сохраненным у нескольких пользователей.

2) Один пользователь может сохранить несколько вопросов.
3) Для описания many-to-many связи составлена таблица UserQuestions, хранящая в себе пары «пользователь – вопрос».

### 3.5. Внутренние сервисы

Для разделения ответственности при реализации API были введены следующие сервисы:

#### 3.5.1. PasswordValidatorService

- `GetHashedPassword`

*Что делает метод*: хэширует введенную строку.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| password | string | Пароль, который необходимо захэшировать |

*Возвращаемые данные*:

| Имя | Тип | Описание данных |
|---|---|---|
| HashedPassword | string | Захэшированный алгоритмом SHA-256 пароль |
| Salt | string | «Соль» [3] для дополнительной защищенности пароля |

*Алгоритм работы метода*: хэширует password алгоритмом SHA-256, добавляя к нему константу LocalSalt и случайно сгенерированный массив из 32 байт.

- `ValidatePassword`

*Что делает метод*: проверяет соответствие введенного пароля тому, который лежит в базе данных.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| password | string | Введенный пользователем пароль |
| truePassword | string | Хэш пароля, который лежал в базе данных |

| | | |
|---|---|---|
| salt | string | «Соль» [3] из базы данных |

*Возвращаемые данные*:

| Тип | Описание данных |
|---|---|
| bool | Возвращает значение true, если введенный пароль в захэшированном виде равен паролю из базы данных; и false, в противном случае |

*Алгоритм работы метода*: хэширует строку password и строку salt с помощью алгоритма SHA-256, проверяет на равенство строке truePassword.

### 3.5.2. RightsValidatorService

- `ValidateRights`

*Что делает метод*: проверяет, что у пользователя есть права администратора в данной команде.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| teamId | long | Идентификатор команды из таблицы Teams |
| userId | long | Идентификатор пользователя из таблицы Users |
| ct | CancellationToken | Токен отмены |

*Возвращаемые данные*:

| Тип | Описание данных |
|---|---|
| Task<bool> | Task, который содержит в себе значение true, если у пользователя есть права администратора; и false, в противном случае |

*Алгоритм работы метода*: из базы данных получает информацию о роли пользователя (из таблицы Roles) и проверяет на равенство значению «Admin» или «Creator», так как у создателя также есть права администратора.

## 3.6. Репозитории

Для взаимодействия с удаленной базой данных используются классы репозиториев, в которых есть различные методы для отправки SQL-запросов и получения ответов из базы данных (внутри используется ORM, которая сопоставляет классы из БД классам в API).

### 3.6.1. QuestionRepository

- `GetQuestionsByTopicId`

*Что делает метод*: получает все вопросы по идентификатору темы.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| topicId | long | Идентификатор темы из таблицы Topics |

*Возвращаемые данные*:

| Тип | Описание данных |
|---|---|
| QuestionDao[] | Массив вопросов, содержащихся в теме. О каждом вопросе известен его идентификатор, текст вопроса, ответа и идентификатор темы из таблицы Topics, которой он принадлежит |

*SQL-запрос*:

```
SELECT
q.id           as Id,
q.answerText   as AnswerText,
q.questionText as QuestionText,
q.topicId      as TopicId
FROM questions q
WHERE topicId = @topicId
ORDER BY q.id
```

- `AddQuestion`

*Что делает метод*: добавляет новый запрос в базу данных.

*Параметры* (AddQuestionParameters):

| Имя | Тип | Смысл параметра |
|---|---|---|
| questionText | string | Текст нового вопроса |
| answerText | string | Текст ответа на новый вопрос |
| topicId | long | Идентификатор темы из таблицы Topics, в которую нужно добавить вопрос |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
INSERT INTO questions
(questionText, answerText, topicId) VALUES
(@questionText, @answerText, @topicId)
```

- `EditQuestion`

*Что делает метод*: редактирует вопрос по идентификатору.

*Параметры* (EditQuestionParameters):

| Имя | Тип | Смысл параметра |
|---|---|---|
| questionId | long | Идентификатор редактируемого вопроса из таблицы Questions |
| questionText | string | Новый текст вопроса |
| answerText | string | Новый текст ответа |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
UPDATE questions
SET questionText = @questionText,
    answerText = @answerText
WHERE id = @id
```

- `RemoveQuestion`

*Что делает метод*: удаляет вопрос из базы данных по идентификатору.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| id | long | Идентификатор удаляемого вопроса из таблицы Questions |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
DELETE FROM questions
WHERE id = @id
```

- `GetSavedQuestions`

*Что делает метод*: возвращает сохраненные вопросы пользователя по идентификатору.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| userId | long | Идентификатор пользователя из таблицы Users, сохраненные вопросы которого нужно найти |

*Возвращаемые данные*:

| Тип | Описание данных |
|---|---|
| QuestionDao[] | Массив вопросов, содержащихся в теме. О каждом вопросе известен его идентификатор, текст вопроса, ответа и идентификатор темы из таблицы Topics, которой он принадлежит |

*SQL-запрос*:

```
SELECT
q.id           as Id,
q.answerText   as AnswerText,
q.questionText as QuestionText,
q.topicId      as TopicId
FROM userQuestions uq
INNER JOIN questions q ON uq.questionId = q.id
WHERE userId = @userId
ORDER BY q.id
```

- `SaveQuestion`

*Что делает метод*: добавляет выбранный вопрос в сохраненные.

*Параметры* (SaveQuestionParameters):

| Имя | Тип | Смысл параметра |
|---|---|---|
| userId | long | Идентификатор пользователя из таблицы Users, которому нужно добавить вопрос |
| questionId | long | Идентификатор вопроса из таблицы Questions, который нужно сохранить пользователю |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
INSERT INTO userQuestions
(userId, questionId) VALUES
(@userId, @questionId)
```

- `RemoveQuestionFromSaved`

*Что делает метод*: удаляет вопрос из сохраненных вопросов пользователя.

*Параметры* (RemoveQuestionFromSavedParameters):

| Имя | Тип | Смысл параметра |
|---|---|---|
| userId | long | Идентификатор пользователя из таблицы Users, которому нужно удалить вопрос |
| questionId | long | Идентификатор вопроса из таблицы Questions, который нужно удалить из сохраненных вопросов пользователя |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
DELETE FROM userQuestions uq
WHERE userId = @userId
AND questionId = @questionId
```

### 3.6.2. RoleRepository

- `GetRoleByName`

*Что делает метод*: получает информацию о роли по ее названию.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| name | string | Название роли в таблице Roles |

*Возвращаемые данные*:

| Тип | Описание данных |
|---|---|
| RoleDao | Роль с идентификатором и именем |

*SQL-запрос*:

```
SELECT
r.id   as Id,
r.name as Name
FROM roles r
WHERE name = @name
```

- `GetUserRoleByTeamId`

*Что делает метод*: получает роль пользователя в команде по ее идентификатору.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| userId | long | Идентификатор пользователя из таблицы Users, роль которого нужно получить |
| teamId | long | Идентификатор команды из таблицы Teams, роль пользователя в которой нужно получить |

*Возвращаемые данные*:

| Тип | Описание данных |
|---|---|
| RoleDao | Роль участника в команде |

*SQL-запрос*:

```
SELECT
r.id   as Id,
r.name as Name
FROM roles r
INNER JOIN teamUsers tu ON tu.roleId = r.id
WHERE teamId = @teamId
AND   userId = @userId
```

### 3.6.3. StatisticsRepository

- `GetStatisticsByUserId`

*Что делает метод*: получает статистику пользователя по его идентификатору.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| userId | long | Идентификатор пользователя из таблицы Users, статистику прохождения тем которого нужно получить |

*Возвращаемые данные*:

| Тип | Описание данных |
|---|---|
| StatisticsDao[] | Информация о результатах прохождения темы: идентификатор пользователя из таблицы Users, идентификатор и название темы из таблицы Topics и результат прохождения темы |

*SQL-запрос*:

```
SELECT
s.id      as Id,
s.topicId as TopicId,
t.title   as TopicName,
s.userId  as UserId,
s.result  as Result
FROM statistics s
INNER JOIN topics t ON t.Id = s.topicId
```

```
WHERE userId = @userId
ORDER BY s.id
```

- `AddStatistics`

*Что делает метод*: добавляет новую статистику по пройденной теме.

*Параметры* (AddStatisticsParameters):

| Имя | Тип | Смысл параметра |
|---|---|---|
| topicId | long | Идентификатор темы из таблицы Topics, по которой пройдено тестирование |
| userId | long | Идентификатор пользователя из таблицы Users, который проходил тестирование |
| result | int | Результат прохождения темы: процент правильных ответов, округленный до целого числа |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
INSERT INTO statistics
(userId, topicId, result) VALUES
(@userId, @topicId, @result)
```

- `EditStatistics`

*Что делает метод*: редактирует существующую статистику пользователя по теме.

*Параметры* (EditStatisticsParameters):

| Имя | Тип | Смысл параметра |
|---|---|---|
| topicId | long | Идентификатор темы из таблицы Topics, по которой пройдено тестирование |
| userId | long | Идентификатор пользователя из таблицы Users, который проходил тестирование |
| result | int | Результат прохождения темы: процент правильных ответов, округленный до целого числа |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
UPDATE statistics
SET result = @result
WHERE userId = @userId
AND topicId = @topicId
```

### 3.6.4. SubjectRepository

- `GetSubjectById`

*Что делает метод*: получает информацию о дисциплине по ее идентификатору.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| id | long | Идентификатор дисциплины из таблицы Subjects, информацию о которой нужно получить |

*Возвращаемые данные*:

| Тип | Описание данных |
|---|---|
| SubjectDao | Объект с информацией о дисциплине: идентификатор дисциплины, ее название, описание и идентификатор команды, которой принадлежит дисциплина |

*SQL-запрос*:

```
SELECT
s.id          as Id,
s.title       as Title,
s.description as Description,
s.teamId      as TeamId
FROM subjects s
WHERE id = @id
```

- `GetSubjectsByTeamId`

*Что делает метод*: получает информацию о предметах в команде по идентификатору команды.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| teamId | long | Идентификатор команды из таблицы Teams, информацию о которой нужно получить |

*Возвращаемые данные*:

| Тип | Описание данных |
|---|---|
| SubjectDao[] | Массив объектов с информацией о дисциплине: идентификатор дисциплины, ее название, описание и идентификатор команды, которой принадлежит дисциплина |

*SQL-запрос*:

```
SELECT
s.id          as Id,
s.title       as Title,
s.description as Description,
s.teamId      as TeamId
FROM subjects s
WHERE teamId = @teamId
ORDER BY s.Id
```

- `AddSubject`

*Что делает метод*: добавляет новую дисциплину в команду.

*Параметры* (AddSubjectParameters):

| Имя | Тип | Смысл параметра |
|---|---|---|
| teamId | long | Идентификатор команды из таблицы Teams, в которую нужно добавить предмет |
| subjectName | string | Имя новой дисциплины |
| subjectDescription | string | Описание новой дисциплины |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
INSERT INTO subjects
(title, description, teamId) VALUES
(@title, @description, @teamId)
```

- `EditSubject`

*Что делает метод*: редактирует дисциплину.

*Параметры* (EditSubjectParameters):

| Имя | Тип | Смысл параметра |
|---|---|---|
| subjectId | long | Идентификатор предмета из таблицы Subjects, который нужно отредактировать |
| subjectName | string | Новое имя дисциплины |
| subjectDescription | string | Новое описание дисциплины |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
UPDATE subjects
SET title = @title,
    description = @description
WHERE id = @id
```

- `RemoveSubject`

*Что делает метод*: удаляет дисциплину по ее идентификатору.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| id | long | Идентификатор предмета из таблицы Subjects, который нужно удалить |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
DELETE FROM subjects
WHERE id = @id
```

### 3.6.5. TeamRepository

- `GetTeamById`

*Что делает метод*: получает информацию о команде по ее идентификатору.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| id | long | Идентификатор команды из таблицы Teams, информацию о которой нужно найти |

*Возвращаемые данные*:

| Тип | Описание данных |
|---|---|
| TeamDao | Объект с информацией о команде: идентификатор команды в таблице Teams, название этой команды, ее картинка-аватар и идентификатор создателя из таблицы Users |

*SQL-запрос*:

```
SELECT
t.id        as Id,
t.name      as Name,
t.avatar    as Avatar,
t.creatorId as CreatorId
FROM teams t
WHERE t.id = @id
```

- `GetTeamByNameAndCreator`

*Что делает метод*: получает информацию о команде по ее названию и идентификатору создателя.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| name | string | Название команды |
| creatorId | long | Идентификатор создателя команды из таблицы Users |

*Возвращаемые данные*:

| Тип | Описание данных |
|---|---|
| TeamDao | Объект с информацией о команде: идентификатор команды в таблице Teams, название этой команды, ее картинка-аватар и идентификатор создателя из таблицы Users |

*SQL-запрос*:

```
SELECT
t.id        as Id,
t.name      as Name,
t.avatar    as Avatar,
t.creatorId as CreatorId
FROM teams t
WHERE t.name = @name
AND t.creatorId = @creatorId
ORDER BY t.id DESC
```

- `GetTeamMembersByTeamId`

*Что делает метод*: получает информацию об участниках команды по идентификатору команды.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| teamId | long | Идентификатор команды из таблицы Teams, информацию об участниках которой нужно получить |

*Возвращаемые данные*:

| Тип | Описание данных |
|---|---|
| UserDao[] | Массив объектов с информацией об участниках команды: идентификатор пользователя из таблицы Users, имя пользователя, его адрес электронной почты и идентификатор аватара пользователя |

*SQL-запрос*:

```
SELECT
u.id       as Id,
u.username as Username,
u.email    as Email,
u.password as Password,
u.avatarId as AvatarId
FROM teamUsers tu
INNER JOIN users u ON u.id = tu.userId
WHERE tu.teamId = @teamId
ORDER BY tu.id
```

- `CreateTeam`

*Что делает метод*: создает новую команду.

*Параметры* (CreateTeamParameters):

| Имя | Тип | Смысл параметра |
|---|---|---|
| teamName | string | Название новой команды |
| avatar | int | Аватар новой команды |
| creatorId | long | Идентификатор создателя новой команды из таблицы Users |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
INSERT INTO teams
(name, avatar, creatorId) VALUES
(@name, @avatar, @creatorId)
```

- `AddMember`

*Что делает метод*: добавляет нового участника в команду.

*Параметры* (AddMemberParameters):

| Имя | Тип | Смысл параметра |
|---|---|---|
| roleId | long | Идентификатор роли нового участника в команде из таблицы Roles |

| | | |
|---|---|---|
| userId | long | Идентификатор пользователя из таблицы Users, которого нужно добавить в команду |
| teamId | long | Идентификатор команды из таблицы Teams, в которую нужно добавить участника |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
INSERT INTO teamUsers
(teamId, userId, roleId) VALUES
(@teamId, @userId, @roleId)
```

- `ChangeMemberStatus`

*Что делает метод*: изменяет роль участника в команде.

*Параметры* (ChangeMemberStatusParameters):

| **Имя** | **Тип** | **Смысл параметра** |
|---|---|---|
| roleId | long | Идентификатор роли в команде из таблицы Roles, которую нужно присвоить участнику |
| userId | long | Идентификатор пользователя из таблицы Users, роль которого нужно изменить |
| teamId | long | Идентификатор команды из таблицы Teams, роль участника в которой нужно изменить |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
UPDATE teamUsers
SET roleId = @roleId
WHERE userId = @userId
AND   teamId = @teamId
```

- `EditTeam`

*Что делает метод*: редактирует информацию о команде.

*Параметры* (EditTeamParameters):

| Имя | Тип | Смысл параметра |
|---|---|---|
| teamId | long | Идентификатор команды из таблицы Teams, которую нужно отредактировать |
| teamName | string | Новое название команды |
| teamAvatar | string | Новый аватар команды |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
UPDATE teams
SET avatar = @avatar,
    name = @name
WHERE id = @id
```

- `RemoveMember`

*Что делает метод*: удаляет участника из команды.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| userId | long | Идентификатор пользователя из таблицы Users, которого нужно удалить из команды |
| teamId | long | Идентификатор команды из таблицы Teams, из которой нужно удалить пользователя |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
DELETE FROM teamUsers tu
WHERE userId = @userId
AND   teamId = @teamId
```

### 3.6.6. TopicRepository

- `GetTopicById`

*Что делает метод*: получает информацию о теме по ее идентификатору.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| id | long | Идентификатор темы из таблицы Topics, информацию о которой нужно получить |

*Возвращаемые данные*:

| Тип | Описание данных |
|---|---|
| TopicDao | Объект с информацией о теме: идентификатор темы, ее название, описание и идентификатор дисциплины, которой принадлежит тема |

*SQL-запрос*:

```
SELECT
t.id          as Id,
t.title       as Title,
t.description as Description,
t.subjectId   as SubjectId
FROM topics t
WHERE t.id = @id
```

- `GetTopicsBySubjectId`

*Что делает метод*: получает информацию о темах в команде по идентификатору дисциплины.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| subjectId | long | Идентификатор дисциплины из таблицы Subjects, информацию о которой нужно получить |

*Возвращаемые данные*:

| Тип | Описание данных |
|---|---|

| | |
|---|---|
| TopicDao[] | Массив объектов с информацией о темах, для каждой из которых известно: идентификатор темы, ее название, описание и идентификатор дисциплины, которой принадлежит тема |

*SQL-запрос*:

```
SELECT
t.id          as Id,
t.title       as Title,
t.description as Description,
t.subjectId   as SubjectId
FROM topics t
WHERE subjectId = @subjectId
ORDER BY t.id
```

- `AddTopic`

*Что делает метод*: добавляет новую тему в дисциплину.

*Параметры* (AddTopicParameters):

| Имя | Тип | Смысл параметра |
|---|---|---|
| subjectId | long | Идентификатор дисциплины из таблицы Subjects, в которую нужно добавить тему |
| topicName | string | Имя новой темы |
| topicDescription | string | Описание новой темы |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
INSERT INTO topics
(title, description, subjectId) VALUES
(@title, @description, @subjectId)
```

- `EditTopic`

*Что делает метод*: редактирует тему.

*Параметры* (EditTopicParameters):

| Имя | Тип | Смысл параметра |
|---|---|---|
| topicId | long | Идентификатор темы из таблицы Topics, которую нужно отредактировать |
| topicName | string | Имя новой темы |
| topicDescription | string | Описание новой темы |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
UPDATE topics
SET title = @title,
    description = @description
WHERE id = @id
```

- `RemoveTopic`

*Что делает метод*: удаляет тему по ее идентификатору.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| id | long | Идентификатор темы из таблицы Topics, которую нужно удалить |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
DELETE FROM topics
WHERE id = @id
```

### 3.6.7. UserRepository

- `GetUserById`

*Что делает метод*: получает информацию о пользователе по его идентификатору.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| userId | long | Идентификатор пользователя из таблицы Users |

*Возвращаемые данные*:

| Тип | Описание данных |
|---|---|
| UserDao | Информация о пользователе: идентификатор пользователя из таблицы Users, имя пользователя, его адрес электронной почты и идентификатор аватара пользователя |

*SQL-запрос*:

```
SELECT
u.id       as Id,
u.username as Username,
u.email    as Email,
u.avatarId as AvatarId
FROM users u
WHERE u.id = @id
```

- `GetUserWithPasswordByEmail`

*Что делает метод*: получает информацию о пользователе, включая его захэшированный пароль, по адресу электронной почты.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| email | string | Адрес электронной почты пользователя, по которому производить поиск |

*Возвращаемые данные*:

| Тип | Описание данных |
|---|---|
| UserWithPasswordDao | Информация о пользователе с заданным адресом электронной почты: идентификатор пользователя из таблицы Users, имя пользователя, его адрес электронной почты, идентификатор |

| | |
|---|---|
| | аватара пользователя, захэшированный пароль и «соль» к паролю |

*SQL-запрос*:

```
SELECT
u.id       as Id,
u.username as Username,
u.email    as Email,
u.avatarId as AvatarId
FROM users u
WHERE email = @email
```

- `GetUserByEmail`

*Что делает метод*: получает основную информацию о пользователе по адресу электронной почты.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| email | string | Адрес электронной почты пользователя, по которому производить поиск |

*Возвращаемые данные*:

| Тип | Описание данных |
|---|---|
| UserDao | Информация о пользователе с заданным адресом электронной почты: идентификатор пользователя из таблицы Users, имя пользователя, его адрес электронной почты и идентификатор аватара пользователя |

*SQL-запрос*:

```
SELECT
u.id       as Id,
u.username as Username,
u.email    as Email,
u.avatarId as AvatarId,
u.password as Password,
u.salt     as Salt
```

```
FROM users u
WHERE email = @email
```

- `GetTeamsByUserId`

*Что делает метод*: получает команды пользователя по его идентификатору.

*Параметры*:

| Имя | Тип | Смысл параметра |
|---|---|---|
| userId | long | Идентификатор пользователя из таблицы Users, команды которого нужно получить |

*Возвращаемые данные*:

| Тип | Описание данных |
|---|---|
| TeamDao[] | Массив объектов с информацией о командах пользователя: идентификатор команды в таблице Teams, название этой команды, ее картинка-аватар и идентификатор создателя из таблицы Users |

*SQL-запрос*:

```
INSERT INTO users
(username, email, password, avatarId, salt) VALUES
(@username, @email, @password, @avatarId, @salt)
```

- `AddUser`

*Что делает метод*: создает нового пользователя.

*Параметры* (AddUserParameters):

| Имя | Тип | Смысл параметра |
|---|---|---|
| username | string | Имя нового пользователя |
| email | string | Адрес электронной почты нового пользователя |
| password | string | Захэшированный пароль нового пользователя |
| salt | string | Персональная «соль» к паролю нового пользователя |
| avatarId | int | Идентификатор аватара нового пользователя |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
UPDATE users
SET avatarId = @avatarId
WHERE id = @id
```

- `ChangeUserAvatarById`

*Что делает метод*: изменяет аватар пользователя.

*Параметры* (ChangeUserAvatarParameters):

| Имя | Тип | Смысл параметра |
|---|---|---|
| userId | long | Идентификатор пользователя из таблицы Users, аватар которого нужно поменять |
| avatarId | long | Идентификатор нового аватара |

*Возвращаемые данные*: нет.

*SQL-запрос*:

```
SELECT
t.id     as Id,
t.name   as Name,
t.avatar as Avatar
FROM teamUsers tu
INNER JOIN teams t ON tu.teamId = t.id
WHERE tu.userId = @userId
```

# 4. СООБЩЕНИЯ ОПЕРАТОРУ

API может возвращать пользователю сообщения и ошибки со следующими кодами:

| Код | Текст | Причина |
|---|---|---|
| 200 | Success | Запрос выполнен успешно, при необходимости возвращены корректные данные |
| 400 | Bad Request | В запросе содержится информация, которой нет в базе данных |
| 401 | Unautorized | У пользователя нет прав на выполнение данного запроса, или пользователь не прошел авторизацию |
| 404 | Not Found | По отправленному запросу не нашлось требуемых данных или запрашиваемая конечная точка API не существует |
| 405 | Method Not Allowed | В запросе указан неверный тип конечной точки (например, GET вместо POST) |
| 500 | Internal Server Error | При запросе произошла ошибка на стороне базы данных |

# СПИСОК ИСТОЧНИКОВ

1) Документация ASP .NET Core [Электронный ресурс] / Режим доступа: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-6.0, свободный.
2) Docker [Электронный ресурс] / Режим доступа: https://www.docker.com, свободный.
3) Как работает алгоритм SHA-256 [Электронный ресурс] / Режим доступа: https://tproger.ru/translations/sha-2-step-by-step, свободный.

## ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

| Лист регистрации изменений | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Номера листов (страниц) | | | | | Всего листов (страниц в докум.) | № документа | Входящий № сопроводительного докум. и дата | Подп. | Дата |
| Изм. | Измененных | Замененных | Новых | Аннулированных | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |