**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ**

**по дисциплине «Введение в нереляционные СУБД»**

**Тема: Система автоматической проверки задач по математике**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9303 |  | Ефимов М.Ю. |
| Студент гр. 9303 |  | Махаличев Н.А. |
| Студент гр. 9303 |  | Эйсвальд М.И. |
| Преподаватель |  | Заславский М.М. |

Санкт-Петербург

2022

**ЗАДАНИЕ**

**НА ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ**

|  |
| --- |
| Студент Ефимов М.Ю.  Студент Махаличев Н.А.  Студент Эйсвальд М.И. |
| Группа 9303 |
| Тема работы: Система автоматической проверки задач по математике |
| Исходные данные:  Необходимо сделать простую систему проверки генерируемых случайно математических примеров. |
| Содержание пояснительной записки:  «Содержание»  «Введение»  «Качественные требования к решению»  «Сценарий использования»  «Модель данных»  «Разработанное приложение»  «Вывод»  «Список использованных источников»  «Приложения» |
| Предполагаемый объем пояснительной записки:  Не менее 10 страниц. |
| Дата выдачи задания: |
| Дата сдачи реферата: |
| Дата защиты реферата: |

|  |  |  |
| --- | --- | --- |
| Студент |  | Ефимов М.Ю. |
| Студент |  | Махаличев Н.А. |
| Студент |  | Эйсвальд М.И. |
| Преподаватель |  | Заславский М.М. |

**АННОТАЦИЯ**

В качестве индивидуального домашнего задания была выбрана тема «Система автоматической проверки задач по математике» предполагающая использование в процессе выполнения нереляционную СУБД MongoDB. Для создания приложения были использованы следующие технологии:

* Frontend: React с использованием Vite;
* Backend: JavaScript (библиотеки сервера – Express; библиотека для работы с MongoDB – Mongoose).

**СОДЕРЖАНИЕ**

|  |  |  |
| --- | --- | --- |
|  | Введение | 6 |
| 1. | Качественные требования к решению | 7 |
| 2. | Сценарий использования | 8 |
| 2.1. | Макет UI | 8 |
| 2.2. | Соглашения | 8 |
| 2.3. | Сценарии использования | 9 |
| 3. | Модель данных | 19 |
| 3.1. | Нереляционная модель данных | 19 |
| 3.2. | Аналог – реляционная модель данных | 31 |
| 3.3. | Сравнение моделей | 43 |
| 4. | Разработанное приложение | 45 |
| 4.1. | Краткое описание | 45 |
| 4.2. | Страницы экранов приложения | 45 |
|  | Вывод | 49 |
|  | Список использованных источников | 50 |
|  | Приложение А. ИНСТРУКЦИЯ ПО РАЗВЁРТЫВАНИЮ ПРИЛОЖЕНИЯ | 51 |

**ВВЕДЕНИЕ**

Цель работы – создать сервис для автоматический проверки математических заданий. Было решено разработать веб-приложение, генерирующее математические примеры с возможностью решения и моментальной проверки. Также присутствует функция логирования действий пользователей и просмотра статистики.

**1. КАЧЕСТВЕННЫЕ ТРЕБОВАНИЯ К РЕШЕНИЮ**

Необходимо создать систему автоматической проверки генерируемых случайно математических примеров. Система должна хранить: задачи, пользователей, попытки решения, сами решения, историю редактирования и историю действий, а также проводить поиск и аналитику по данным – у кого проблемы с какими действиями, кто отвлекается и т.д. Используемая БД – MongoDB.

**2. СЦЕНАРИЙ ИСПОЛЬЗОВАНИЯ**

**2.1. Макет UI**

Макет пользовательского интерфейса представлен на рис. 1. Изображение в высоком разрешении доступно по ссылке:

https://github.com/moevm/nosql2h22-math/blob/main/wiki/images/Use-Case.png



Рисунок 1 – Макет UI

**2.2. Соглашения**

Ученик ‒ пользователь, цель которого ‒ научиться решать примеры; Учитель ‒ пользователь, цель которого ‒ обучить Учеников решать примеры; Администратор ‒ пользователь, целью которого является поддержка работоспособности продукта.

Подпункты в сценариях являются ветвлениями. Если в пункте сценария использования с номером N есть несколько подпунктов, подпункты следует понимать как альтернативы. Ближайшая реакция системы на события в таком случае тоже разделена на подпункты; выполнен будет подпункт с тем же номером, что и альтернативное событие ранее. По умолчанию альтернативы являются взаимоисключающими.

* "Страница с примером" == "Главная страница ученика";
* "Адрес электронной почты" == "Логин".

**2.3. Сценарии использования**

**Решение примеров учеником.**

Действующее лицо: ученик.

Предусловие: ученик авторизован, открыта страница с примером.

Сценарий:

1. Приложение отображает текст примера и поле для ввода ответа. Если у ученика есть незавершённые домашние задания, под полем для ввода ответа отображается список домашних заданий с указанием дедлайна, пунктов задания, тем задач для каждого пункта, количества решённых задач в пункте задания и общего количества задач в пункте задания. Также отображается список доступных тем с чекбоксами, показывающими, включена ли тема в пул для генерации примеров. Текст примера содержит только материал по всем темам, чекбоксы которых отмечены;
2. Ученик обновляет страницу;
3. Приложение отображает страницу (см. шаг 1) с тем же примером;
4. Ученик вводит ответ в числовое поле ответа и убирает фокус с поля ввода или нажимает на кнопку Enter/Return на клавиатуре;
   1. Ответ ученика верный;
   2. Ответ ученика неверный.
5. i. Приложение подсвечивает границы поля ввода ответа зелёным, показывая правильность ответа, и отображает кнопку перехода к следующему примеру рядом с решённым примером. Если решённый пример по критериям подходит под тему одного или нескольких пунктов из домашних заданий, то счётчики прогресса соответствующих пунктов увеличиваются на 1;

ii. Приложение посвечивает границы поля ввода красным цветом.

1. i. Ученик нажимает кнопку перехода к следующему примеру;

ii. Ученик нажимает кнопку выхода или ссылку на другую страницу;

iii. Ученик вводит новый ответ в текстовое поле и убирает фокус с поля

ввода либо нажимает на Enter/Return;

iv. Ученик задаёт с помощью чекбоксов новую комбинацию тем и нажимает

кнопку перехода к следующему примеру;

v. Ученик обновляет страницу.

1. i. Переход к шагу 1;

ii. Приложение отображает соответствующую страницу;

iii. Переход к шагу 4;

iv. Переход к шагу 1 с учётом новой комбинации тем;

v. Переход к шагу 1. Если пример не был решён верно, пример остаётся тем

же.

**Решение учеником домашнего задания.**

Действующее лицо: ученик.

Предусловие: ученик авторизован, открыта страница с примером.

Сценарий:

1. Ученик кликает по одному из заголовков заданий, отображаемых под полем ввода ответа;
2. Система приводит чекбоксы напротив тем заданий в соответствие с условиями домашнего задания и генерирует новый пример (если не подходит текущий) по условиям домашнего задания;
3. Переход к началу сценария 1.

Просмотр учеником истории своих решений.

Действующее лицо: ученик.

Предусловие: ученик авторизован.

Сценарий:

1. Ученик нажимает на надпись "История" в верхней части экрана;
2. Система отображает решения ученика в табличном виде. Имеются столбцы: дата и время отправки решения, текст примера, используемые в примере темы, потраченное на решение время, отправленный ответ, вердикт системы (верно/неверно);
3. Ученик кликает на "Типы задач" в заголовке таблицы и в списке тем выбирает "Умножение" и "Деление";
4. В отображаемой таблице остаются только решения примеров, содержащих умножение или деление;
5. Ученик кликает на "Время ответа" в заголовке таблицы;
6. В отображаемой таблице решения сортируются по потраченному на решение времени.

**Просмотр учеником статистики.**

Действующее лицо: ученик.

Предусловие: ученик авторизован.

Сценарий:

1. Ученик нажимает на надпись "Статистика" в верхней части экрана;
2. Система отображает гистограмму, показывающую количество отправленных решений ученика по каждой теме за всё время с момента регистрации, число правильных и неправильных среди этих решений;
3. Ученик вводит в поля ввода даты под гистограммой даты, за которые желает посмотреть статистику;

i. В указанный промежуток времени учеником был решён хоть один пример;

ii. В указанный промежуток времени учеником не был решён ни один пример.

1. i. Система отображает в гистограмме только данные, полученные в

указанный временной интервал;

ii. Система отображает вместо гистограммы сообщение об отсутствии

данных.

**Публикация учителем домашнего задания.**

Действующее лицо: учитель.

Предусловие: учитель авторизован.

Сценарий:

1. Учитель нажимает на кнопку добавления нового задания в верхней части страницы;
2. Система отображает меню настройки нового задания;
3. Учитель выбирает из списка имеющихся классов классы, для которых будет добавлено задание; указывает в поле ввода даты и времени дедлайн задания;
4. Система отображает элементы настройки примеров в пункте задания;
5. Учитель указывает в числовом поле ввода требуемое количество примеров (минимум 1), выбирает темы примеров, которые необходимо будет решить;
6. Если учитель хочет добавить ещё один пункт в задание, учитель нажимает на кнопку добавления нового пункта в задание. Возврат к шагу 5;
7. Учитель нажимает на кнопку "Опубликовать задание".
8. i. Все поля заполнены, дедлайн задания находится в будущем;

ii. Все остальные случаи.

1. i. Система выводит сообщение об успешной публикации задания и

отображает главную страницу учителя;

ii. Поле ввода с неверной информацией подсвечивается красным и

отображается стандартный текст ошибки ввода на веб-странице; возврат

к редактированию данных.

**Создание класса учителем.**

Действующее лицо: учитель.

Предусловие: учитель авторизован.

Сценарий:

1. Учитель нажимает на надпись "Классы" в верхней части страницы;
2. Система отображает список классов в табличном виде и элементы управления для создания нового класса;
3. Учитель вводит название класса в текстовое поле и нажимает на кнопку создания класса;
4. Система отображает обновлённый список классов и копирует ссылку для вступления в класс в буфер обмена;
5. Учитель отправляет ученикам ссылку-приглашение вне приложения.

**Поступление ученика в класс.**

Действующее лицо: ученик.

Предусловие: ученик авторизован.

Сценарий:

1. Ученик кликает на ссылку-приглашение вне системы;
2. Система в открывшейся вкладке отображает запрос на подтверждение вступления в класс с указанием названия класса и фамилии и имени учителя;
3. Ученик нажимает "Подтвердить";
4. Система отображает главную страницу ученика. Под полем ввода ответа теперь отображаются, кроме прочих, все задания класса, в который ученик только что вступил.

**Просмотр учителем статистики по ученикам.**

Действующее лицо: учитель.

Предусловие: учитель авторизован.

Сценарий:

1. Учитель нажимает на надпись "Классы" в верхней части экрана;
2. Система отображает список классов и сведения по ним в табличном виде: название, количество учеников, домашнее задание и срок его выполнения, количество ответов на примеры домашнего задания, количество сдавших ДЗ учеников. Ниже на странице отображается ссылка-приглашение для вступления в класс;

i. Если у учителя нет ни одного класса, вместо таблицы отображается стандартный текст с приглашением добавить класс.

1. Учитель нажимает на название одного из классов;
2. Система отображает список учеников класса в табличном виде: фамилию и имя ученика, время последней активности, прогресс выполнения заданного учителем домашнего задания, время последнего отвлечения (длинной паузы в решении) и т.п. Список метрик, которые должны собираться для ученика, будет окончательно утверждён на стадии разработки прототипа;

i. Если в классе нет учеников, вместо таблицы отображается текст с приглашением отправить ссылку для вступления в класс ученикам.

1. Учитель нажимает на надпись "Последняя активность" в заголовке таблицы;
2. Ученики в таблице сортируются по убыванию временной метки последней активности;
3. Учитель нажимает на фамилию одного из учеников;
4. Система отображает страницу статистики ученика (см. сценарий "Просмотр учеником статистики", пункт 2);
5. Учитель нажимает на надпись "История" в верхней части экрана;
6. Система отображает решения ученика (см. сценарий "Просмотр учеником истории своих решений", пункт 2);
7. Учитель нажимает на заголовок столбца "Время ответа";
8. Записи в таблице сортируются по времени, потраченному учеником на решение.

**Регистрация нового пользователя.**

Действующее лицо: ученик или учитель.

Предусловие: открыта страница входа в систему.

Сценарий:

1. Пользователь нажимает на кнопку "Зарегистрироваться";
2. Система отображает форму регистрации;
3. Пользователь выбирает роль (ученик/учитель) с помощью переключателя и заполняет текстовые поля: имя, фамилия, адрес электронной почты, пароль:

i. Имя и фамилия не пусты и состоят только из букв, адрес электронной почты не пуст, имеет действительный формат и не используется в качестве логина другого пользователя, пароль не пуст и удовлетворяет требованиям безопасности (требования к паролю будут сформулированы на этапе разработки прототипа);

ii. Все остальные случаи.

1. i. Система отображает сообщение об успешном создании нового

пользователя;

ii. Поле ввода с неверной информацией подсвечивается красным и

отображается стандартный текст ошибки ввода на веб-странице.

Возврат к шагу 2.

1. Пользователь закрывает сообщение;
2. Система отображает форму входа в систему.

**Вход пользователя в систему.**

Действующее лицо: администратор, ученик или учитель.

Предусловие: открыта страница входа в систему.

Сценарий:

1. Пользователь вводит логин и пароль в соответствующие поля и нажимает на кнопку "Войти". Для администратора логин и пароль задаются разработчиком;

i. Адрес электронной почты имеет верный формат и уже используется в системе в качестве логина, пароль соответствует введённому адресу электронной почты;

ii. Все остальные случаи.

1. i. Система отображает главную страницу пользователя (для

администратора ‒ страницу логов). В правом верхнем углу

отображаются имя и фамилия пользователя;

ii. Поле ввода с неверной информацией подсвечивается красным и

iii. отображается стандартный текст ошибки ввода на веб-странице. Возврат

к предусловию.

Работа администратора с логами.

Действующее лицо: администратор.

Предусловие: администратор авторизован, открыта главная страница логов.

Сценарий:

1. Система отображает логи в виде таблицы. Как минимум в таблице должны быть столбцы: уровень логирования (FINEST/DEBUG/INFO/WARNING/ERROR/CRITICAL), временная метка, содержание сообщения;
2. Пользователь выбирает из выпадающего списка фильтров уровней логирования "Все", вводит в полях фильтрации по дате и времени интересующие временные точки начала и конца наблюдений, вводит в текстовом поле поиска логин ученика или учителя:

i. В системе хранятся логи, соответствующие фильтрам;

ii. В системе нет логов, соответствующих фильтрам.

1. i. Система отображает в табличном виде логи всех уровней за указанный

промежуток времени, у которых в теле сообщения встречается

указанный логин пользователя;

ii. Система отображает сообщение о том, что логи не найдены, например:

"Нет подходящих результатов. Проверьте временной отрезок или

попробуйте другой запрос. Помните, что если логи старше нескольких

дней и не предшествуют критической ошибке, скорее всего, они уже

удалены."

1. Пользователь выбирает из выпадающего списка фильтров уровней логирования "INFO", вводит в полях фильтрации по дате и времени интересующие временные точки начала и конца наблюдений, вводит в текстовом поле поиска произвольный текст;
2. Система отображает сообщение аналогично п. 3.2, если логи не найдены, иначе отображается таблица с логами уровня INFO и важнее за указанный промежуток времени, содержащие в сообщении все токены введённого текста ‒ части, разделённые пробельными символами (не обязательно подряд).

**Работа администратора с историей действий.**

Действующее лицо: администратор.

Предусловие: администратор авторизован, открыта главная страница логов.

Сценарий:

1. Администратор нажимает на надпись "История действий" в верхней части экрана;
2. Система отображает в табличном виде действия учеников и учителей с указанием даты и времени совершения действия, логина пользователя, роли пользователя (ученик или учитель), типа действия и подробностей. По умолчанию действия отсортированы по времени совершения ‒ свежие наверху;
3. Администратор нажимает на надпись "Тип пользователя" в заголовке таблицы, выбирает в выпадающем списке "Ученик";
4. Система отображает в таблице только действия учеников;
5. Администратор вводит в текстовое поле рядом со словом "Действие" в заголовке таблицы текст "Редактирование ответа";
6. Система отображает в таблице только редактирование учениками своих ответов;
7. Администратор вводит некоторый текст в текстовое поле ввода рядом со словом "Сообщение" в заголовке таблицы;
8. Система отображает только действия редактирования ответов, которые в описании содержат введённый текст.

**3. МОДЕЛЬ ДАННЫХ**

**3.1. Нереляционная модель данных**

**Графическое представление.**

Графическое представление нереляционной модели данных представлено на рис. 2.

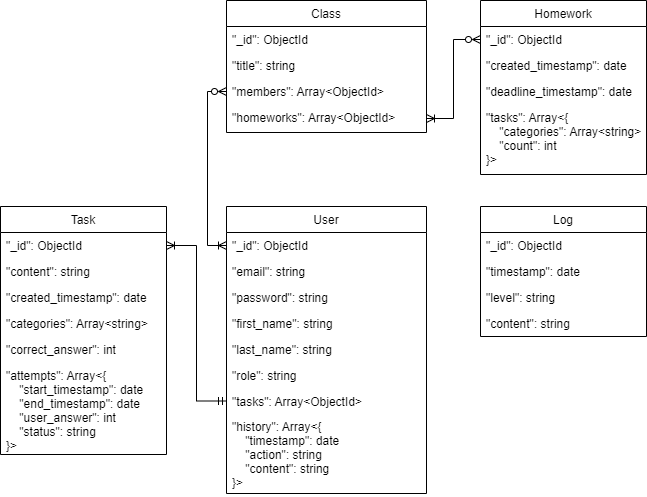


Рисунок 2 – Графическое представление нереляционной модели данных

**Описание назначений коллекций, типов данных и сущностей.**

*Коллекция «Users»:*

Назначение: хранит в себе данные пользователей (сущности User).

Содержание полей:

* email: String - адрес электронной почты (логин);
* password: String - пароль;
* first\_name: String - имя;
* last\_name: String - фамилия;
* role: String - роль (pupil, teacher, administrator);
* tasks: Array<ObjectId> - массив \_id задач, созданных для данного пользователя;
* history: Array<Object> - массив сохранённых действий пользователя:
  + timestamp: Date - дата действия;
  + action: String - класс действия;
  + content: String - информация о действии.

*Коллекция «Classes»:*

Назначение: содержит в себе данные созданных классов (сущности Class).

Содержание полей:

* title: String - название;
* members: Array<ObjectId> - массив \_id участников класса;
* homeworks: Array<ObjectId> - массив \_id домашних заданий.

*Коллекция «Homeworks»:*

Назначение: cодержит информацию о заданных домашних заданиях (сущности Homework).

Содержание полей:

* created\_timestamp: Date - дата создания;
* deadline\_timestamp: Date - дедлайн;
* tasks: Array<Object> - массив задач, необходимых для выполнения:
  + categories: Array<String> - массив категорий задачи (addition, subtraction, multiplication, division);
  + count: Number - необходимое количество выполненных задач.

*Коллекция «Tasks»:*

Назначение: хранит в себе данные сгенерированных задач (сущности Task).

Содержание полей:

* content: String - формулировка задачи;
* created\_timestamp: Date - дата создания;
* categories: Array<String> - массив категорий задачи (addition, subtraction, multiplication, division);
* correct\_answer: Number - верный ответ на данную задачу;
* attempts: Array<Object> - массив попыток решения задачи:
  + start\_timestamp: Date - дата начала попытки;
  + end\_timestamp: Date - дата окончания попытки;
  + user\_answer: Number - введённый ответ;
  + status: String - статус попытки (correct, not correct, in progress).

*Коллекция «Logs»:*

Назначение: содержит информацию о системных сообщениях (сущности Log).

Содержание полей:

* timestamp: Date - дата события;
* level: String - уровень логирования (FINEST, DEBUG, INFO, WARNING, ERROR, CRITICAL);
* content: String - содержание лога.

*Описание типов связей:*

User и Class имеют тип связи "многие ко многим". У пользователя может как ни состоять в классе (например, если он только зарегистрировался) или, если пользователь является учителем, состоять в нескольких классах.

User и Task имеют тип связи "один ко многим". Задача обязательно принадлежит только одному пользователю, а к пользователю может относиться как одна задача (если он ещё не решал задачи), так и несколько.

Class и Homework имеют тип связи "многие ко многим". Класс может не иметь домашних заданий (если они ни разу не были заданы), или же иметь несколько. В свою очередь домашнее задание может быть задано одному или нескольким классам сразу.

**Оценка удельного объема информации, хранимой в модели.**

*Документ «User»:*

* \_id: 12 байт;
* email: 30 байт;
* password: 30 байт;
* first\_name: 50 байт;
* last\_name: 50 байт;
* role: 13 байт;
* tasks: байт, где – количество созданных для данного пользователя задач;
* history: байт, где – количество сохранённых действий пользователя:
  + timestamp: 8 байт;
  + action: 50 байт;
  + content: 200 байт.

Чистый объём документа User: байт.

Фактический объём документа User: байт.

*Документ «Class»:*

* \_id: 12 байт;
* title: 30 байт;
* members: , где – количество участников класса;
* homeworks: , где – количество присвоенных домашних заданий.

Чистый объём документа Class: байт.

Фактический объём документа Class: байт.

*Документ «Homework»:*

* \_id: 12 байт;
* created\_timestamp: 8 байт;
* deadline\_timestamp: 8 байт;
* tasks: , где – количество различных задач:
  + categories: байт, где – количество категорий, использующихся в задаче;
  + count: 4 байта.

Чистый объём документа Homework: байт.

Фактический объём документа Homework: байт.

*Документ «Task»:*

* \_id: 12 байт;
* content: 20 байт;
* created\_timestamp: 8 байт;
* categories: байт, где – количество категорий, использующихся в задаче;
* correct\_answer: 4 байта;
* attempts: байт, где – количество попыток решения задачи:
  + start\_timestamp: 8 байт;
  + end\_timestamp: 8 байт;
  + user\_answer: 4 байта;
  + status: 11 байт.

Чистый объём документа Task: байт.

Фактический объём документа Task: .байт.

*Документ «Log»:*

* \_id: 12 байт;
* timestamp: 8 байт;
* level: 8 байт;
* content: 200 байт.

Чистый объём документа Log: 216 байт.

Фактический объём документа Log: 228 байт.

**Избыточность модели.**

Пусть – число классов. В каждом классе = 15 участников, = 10 домашних заданий. У каждого участника: = 20 действий и = 20 решённых задач, в каждой из которых по = 3 категории и = 2 попытки. У каждого домашнего задания: = 3 пункта, в каждом из которых = 3 категории. Число системных логов = 100.

Чистый объём данных:

байт.

Фактический объём данных:

байт.

Избыточность: .

**Направление роста модели при увеличении количества объектов каждой сущности.**

В результате анализа модели данных был получен вывод, что направление роста модели – линейное.

**Примеры запросов к модели для выполнения сценариев.**

*Восстановление задания при авторизации:*

const user\_id = ObjectId('637cef8a42b25ba8ee08ece8')

const categories = ['addition', 'division']

const task\_ids = (await schema.users.findOne({'\_id': user\_id}, {'tasks': 1})).tasks

const result = await schema.tasks.findOne({'\_id': {$in: task\_ids},

'categories': categories,

'attempts.status': 'in

progress'})

Количество запросов: 2.

Количество коллекций: 2.

*Вставка попытки решения:*

const task\_id = ObjectId('637de40e230fffbc9b34414e')

const user\_answer = 10

const correct\_answer = (await schema.tasks.findOne({'\_id': task\_id},

{'\_id': 0, 'correct\_answer': 1})).correct\_answer

const status = user\_answer == correct\_answer ? 'correct' : 'not correct'

await schema.tasks.updateOne({'\_id': task\_id},

{$set: {"attempts.$[attempt].end\_timestamp":

Date.now(),

"attempts.$[attempt].status":

status,

"attempts.$[attempt].user\_answer":

user\_answer}},

{"arrayFilters": [{"attempt.status": 'in

progress'}]})

if (status == 'not correct')

await schema.tasks.updateOne({'\_id': task\_id},

{$push: {"attempts": {'start\_timestamp':

Date.now(),

'status': 'in

progress'}}})

Количество запросов: 2 или 3 (создание новой попытки со статусом 'in progress', если пользователь ввёл неверный ответ).

Количество коллекций: 1.

*Создание нового задания:*

const class\_ids = [ObjectId('637cf20192bec933530fc362'),

ObjectId('637cf4044d77a3dc40b1e37b'),

ObjectId('637cfef64d77a3dc40b1e3a5')]

const homework = new schema.homeworks({created\_timestamp: Date.now(),

deadline\_timestamp:

Date.parse('2025-11-22T16:30:29.791+00:00'),

tasks: [{categories: ['addittion',

'subtraction'], count: 5},

{categories:

['multiplication'], count: 3},

{categories: ['addittion',

'division'], count: 8}]})

await homework.save()

await schema.classes.updateMany({'\_id': {$in: class\_ids}}, {$push: {'homeworks': homework.\_id}})

Количество запросов: 2.

Количество коллекций: 2.

*Получение всех учеников класса:*

const class\_id = ObjectId('637cf20192bec933530fc362')

const members = (await schema.classes.findOne({'\_id': class\_id}, {'\_id': 0, 'members': 1})).members

const result = await schema.users.find({'\_id': {$in: members}, 'role': 'pupil'})

Количество запросов: 2.

Количество коллекций: 2.

*Удаление класса:*

const class\_id = ObjectId('637d31e9eb3701aff0787c0c')

const homework\_ids = (await schema.classes.findOne({'\_id': class\_id}, {'\_id': 0, 'homeworks': 1})).homeworks

if (homework\_ids.length > 0){

const response = (await schema.classes.aggregate([

{$unwind: '$homeworks'},

{$group: {'\_id': '$homeworks',

'count': {$sum: 1}}},

{$match: {"$expr":

{"$in": ["$\_id", homework\_ids]},

"count": 1}},

{$project: {'\_id': 1}} ])).map(obj => obj.\_id)

await schema.homeworks.deleteMany({'\_id': {$in: response}})

}

await schema.classes.deleteOne({'\_id': class\_id})

Количество запросов: 2 или 4, в зависимости от того, были ли у класса домашние задания (удаление домашних заданий, которые были присвоены только этому классу).

Количество коллекций: 1 или 2 (по причине, описанной в количестве запросов).

*Получение актуального домашнего задания и задач, которые были сделаны в этот промежуток:*

const user\_id = ObjectId('637cef8a42b25ba8ee08ece8')

const homework\_ids = (await schema.classes.findOne({'members': user\_id}, {'\_id': 0, 'homeworks': 1})).homeworks

const homework = await schema.homeworks.findOne({'\_id': {$in: homework\_ids},

'deadline\_timestamp': {$gte: (new Date).toISOString()}},

{'\_id': 0})

if (homework != null){

const task\_ids = (await schema.users.findOne({'\_id': user\_id},

{'\_id': 0, 'tasks': 1})).tasks

const result = await schema.tasks.aggregate([

{$match: {'\_id': {$in: task\_ids}}},

{$unwind: {'path': '$attempts'}},

{$match: {'attempts.status': 'correct',

'attempts.end\_timestamp':

{$gte: homework.created\_timestamp,

$lte: homework.deadline\_timestamp}}},

{$project: {'\_id': '$\_id',

'categories':'$categories'}}])

}

Количество запросов: 2 или 4 (в зависимости от того, найдено ли домашнее задание или нет).

Количество коллекций: 2 или 3 (по причине, описанной в количестве запросов).

*Создание нового пользователя (ученик):*

const user = new schema.users({email: 'mahalichev.n@gmail.com',

password: 'mahalichev321',

first\_name: 'Никита',

last\_name: 'Махаличев',

role: 'pupil'})

await user.save()

Количество запросов: 1.

Количество коллекций: 1.

*Создание нового пользователя (учитель):*

const user = new schema.users({email: 'eyswald@gmail.com',

password: 'michail\_good\_teacher',

first\_name: 'Михаил',

last\_name: 'Эйсвальд',

role: 'teacher'})

await user.save()

Количество запросов: 1.

Количество коллекций: 1.

*Получение пользователя по адресу почты:*

const email = 'mahalichev.n@gmail.com' const result = await schema.users.findOne({'email': email})

Количество запросов: 1.

Количество коллекций: 1.

*Добавление ученика в класс:*

const user\_id = ObjectId('637ceff0484241578e5eb04e')

const class\_id = ObjectId('637cf20192bec933530fc362')

await schema.classes.updateOne({'members': user\_id}, {$pull: {'members': user\_id}})

await schema.classes.updateOne({'\_id': class\_id}, {$push: {'members': user\_id}})

Количество запросов: 2.

Количество коллекций: 1.

*Получение истории действий пользователя:*

const user\_id = ObjectId('637cef8a42b25ba8ee08ece8')

const result = (await schema.users.findOne({'\_id': user\_id}, {'\_id': 0, 'history': 1})).history

Количество запросов: 1.

Количество коллекций: 1.

*Получение истории действий всех пользователей:*

const result = await schema.users.find({'history': {$type: 'array', $ne: []}}, {'history': 1})

Количество запросов: 1.

Количество коллекций: 1.

*Фильтрация системных логов по уровням логирования:*

const filter = ["DEBUG", "INFO"]

const result = await schema.logs.find({'level': {$in: filter}})

Количество запросов: 1.

Количество коллекций: 1.

**3.2. Аналог – реляционная модель данных**

**Графическое представление.**

Графическое представление реляционной модели данных см. на рис. 3.

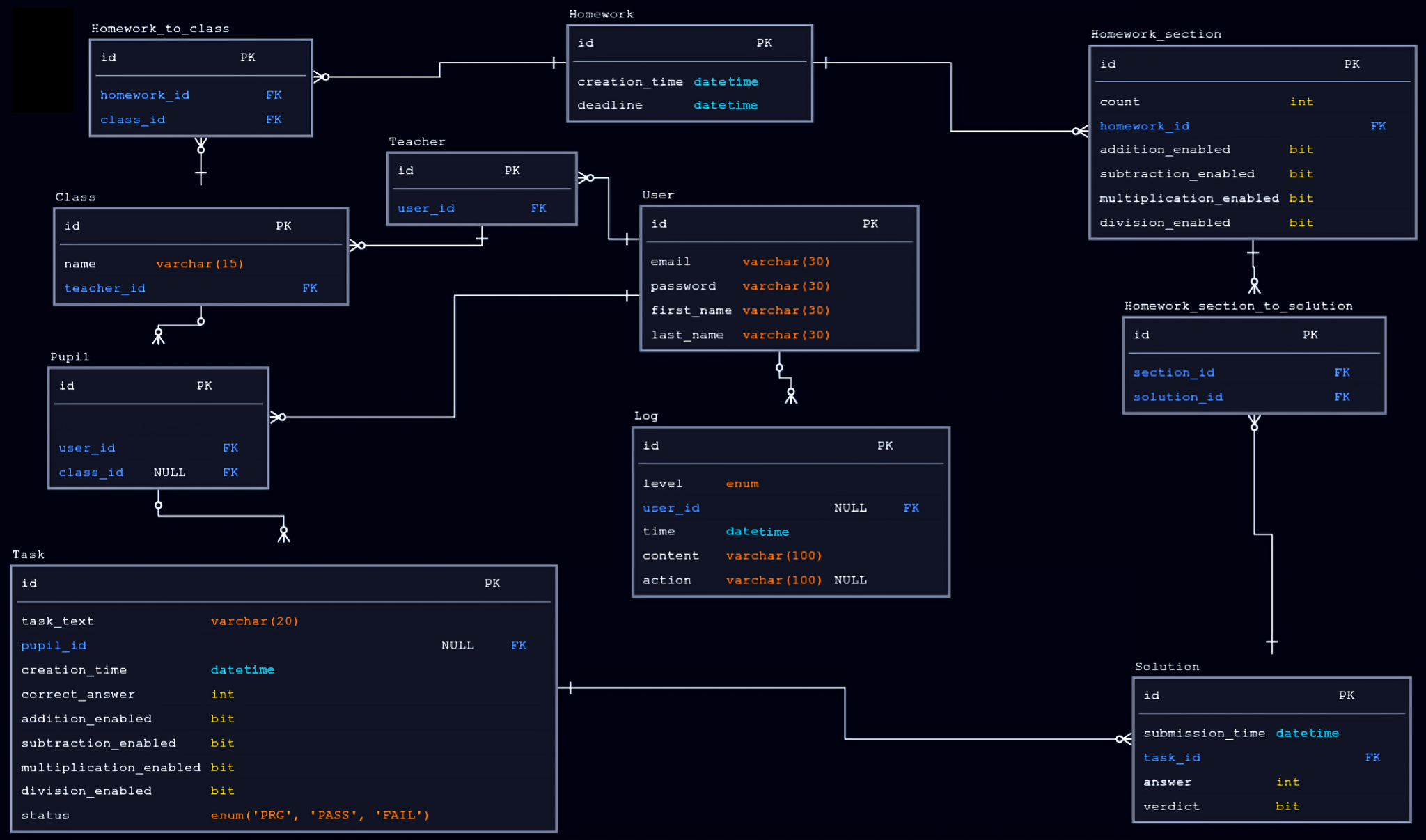


Рисунок 3 – Графическое представление реляционной модели данных

**Описание назначений коллекций, типов данных и сущностей.**

Все связи между таблицами являются необязательными и множественными со стороны, хранящей id главной сущности. Такое обозначение отражает не структуру предметной области, а структуру БД: с точки зрения синтаксиса БД наличие для каждой главной сущности хотя бы одной второстепенной сущности не гарантируется.

*Таблица «User»:*

Таблица хранит данные зарегистрированного пользователя, которые вводятся при регистрации как ученика, так и учителя:

* email: VARCHAR(30) ‒ адрес электронной почты пользователя (служит логином);
* password: VARCHAR(30) ‒ пароль;
* first\_name: VARCHAR(25) ‒ имя пользователя;
* last\_name: VARCHAR(25) ‒ фамилия пользователя.

*Таблица «Pupil»:*

Таблица хранит ссылки на данные учеников в таблице User и идентификатор класса, в котором состоит ученик. Эти данные вынесены в отдельную таблицу только для "проверки типов": на уровне самой базы данных присутствует защита от добавления пользователя-учителя в класс в качестве пользователя-ученика некорректным запросом; операции только с учениками не требуют дополнительного WHERE:

* FK user\_id: INT ‒ id пользователя;
* FK class\_id: INT ‒ id класса.

*Таблица «Teacher»:*

Таблица хранит ссылку на данные пользователя для каждого учителя. Как и таблица "Pupil", существует только для "проверки типов" и возможности хранения учителей отдельно от учеников:

* FK user\_id: INT ‒ id пользователя.

*Таблица «Class»:*

Таблица хранит имя класса и ссылку на учителя класса:

* FK teacher\_id: INT ‒ id учителя класса;
* name: VARCHAR(15) ‒ имя класса.

*Таблица «Homework»:*

Таблица хранит общие для всего домашнего задания характеристики: дату и время создания, дату и время дедлайна.

* creation\_time: DATETIME ‒ дата и время создания;
* deadline: DATETIME ‒ дедлайн.

*Таблица «Homework\_to\_class»:*

Таблица используется для реализации отношения "многие ко многим" между классами и домашними заданиями (учитель может задать ДЗ нескольким своим классам).

* FK homework\_id: INT ‒ идентификатор домашнего задания;
* FK class\_id: INT ‒ идентификатор класса.

*Таблица «Homework\_section»:*

Таблица хранит данные о разделе ДЗ.

* count: INT ‒ количество примеров, которые требуется решить в рамках раздела;
* FK homework\_id: INT ‒ идентификатор домашнего задания, к которому относится раздел;
* addition\_enabled: BIT ‒ является ли сложение одной из тем примера;
* subtraction\_enabled: BIT ‒ является ли вычитание одной из тем примера;
* multiplication\_enabled: BIT ‒ является ли умножение одной из тем примера;
* division\_enabled: BIT ‒ является ли деление одной из тем примера.

*Таблица «Task»:*

Таблица хранит данные о сгенерированном примере.

* FK pupil\_id: INT ‒ идентификатор ученика, в сессию которого был сгенерирован пример. Может быть NULL, если пользователь не авторизован;
* task\_text: VARCHAR ‒ текст примера;
* creation\_time: DATETIME ‒ дата и время генерации задания;
* correct\_answer: INT ‒ правильный ответ (всегда является целым числом);
* addition\_enabled: BIT ‒ является ли сложение одной из тем примера;
* subtraction\_enabled: BIT ‒ является ли вычитание одной из тем примера;
* multiplication\_enabled: BIT ‒ является ли умножение одной из тем примера;
* division\_enabled: BIT ‒ является ли деление одной из тем примера;
* status: ENUM("PRG", "PASS", "FAIL") ‒ текущий статус примера (не решён, решён правильно хотя бы раз, решён неправильно).

*Таблица «Solution»:*

Таблица хранит данные об отправленном решении.

* FK task\_id: INT ‒ Идентификатор примера, к которому относится решение;
* submission\_time: DATETIME ‒ Дата и время отправки ответа;
* answer: INT ‒ Ответ ученика;
* verdict: BIT ‒ Индикатор правильности ответа, чтобы не сравнивать при каждом запросе ответ ученика с эталонным ответом.

*Таблица «Homework\_section\_to\_solution»:*

Таблица используется для моделирования отношения между решениями и разделами ДЗ "многие ко многим".

* FK solution\_id: INT ‒ идентификатор решения;
* FK section\_id: INT ‒ идентификатор раздела домашнего задания.

*Таблица «Log»:*

Таблица используется для хранения логов.

* FK user\_id: INT ‒ идентификатор пользователя, частью истории действий которого запись является (или NULL, если запись не является чатью истории действий)
* level: ENUM("FINEST", "DEBUG", "INFO", "WARNING", "ERROR", "CRITICAL") ‒ уровень логирования;
* time: DATETIME ‒ дата и время сообщения;
* content: VARCHAR(100) ‒ содержание сообщения;
* action: VARCHAR(25) ‒ вид действия пользователя, если применимо; иначе NULL.

**Оценка удельного объема информации, хранимой в модели.**

*Таблица «User»:*

* id ‒ 4 байт;
* email ‒ 30 байт;
* password ‒ 30 байт;
* first\_name ‒ 50 байт;
* last\_name ‒ 50 байт;

Фактический объём на одну запись: 164 байт.

Чистый объём на одну запись: 160 байт.

*Таблица «Pupil»:*

* id ‒ 4 байт;
* user\_id ‒ 4 байт;
* class\_id ‒ 4 байт;

Фактический объём на одну запись: 12 байт.

Чистый объём на одну запись: 0 байт.

*Таблица «Teacher»:*

* id ‒ 4 байт;
* user\_id ‒ 4 байт;

Фактический объём на одну запись: 8 байт.

Чистый объём на одну запись: 0 байт.

*Таблица «Class»:*

* id ‒ 4 байт;
* teacher\_id ‒ 4 байт;
* name ‒ 30 байт;

Фактический объём на одну запись: 38 байт.

Чистый объём на одну запись: 30 байт.

*Таблица «Homework»:*

* id ‒ 4 байт;
* creation\_time ‒ 8 байт;
* deadline ‒ 8 байт;

Фактический объём на одну запись: 20 байт.

Чистый объём на одну запись: 16 байт.

*Таблица «Homework\_to\_class»:*

* id ‒ 4 байт;
* homework\_id ‒ 4 байт;
* class\_id ‒ 4 байт;

Фактический объём на одну запись: 12 байт.

Чистый объём на одну запись: 0 байт.

*Таблица «Homework\_section»:*

* id ‒ 4 байт;
* count ‒ 4 байт;
* homework\_id ‒ 4 байт;
* addition\_enabled ‒ 1 байт;
* subtraction\_enabled ‒ 1 байт;
* multiplication\_enabled ‒ 1 байт;
* division\_enabled ‒ 1 байт;

Фактический объём на одну запись: 16 байт.

Чистый объём на одну запись: 8 байт.

*Таблица «Task»:*

* id ‒ 4 байт;
* pupil\_id ‒ 4 байт;
* task\_text ‒ 20 байт;
* creation\_time ‒ 8 байт;
* correct\_answer ‒ 4 байт;
* addition\_enabled ‒ 1 байт;
* subtraction\_enabled ‒ 1 байт;
* multiplication\_enabled ‒ 1 байт;
* division\_enabled ‒ 1 байт;
* status ‒ 1 байт;

Фактический объём на одну запись: 45 байт.

Чистый объём на одну запись: 37 байт.

*Таблица «Solution»:*

* id ‒ 4 байт;
* task\_id ‒ 4 байт;
* submission\_time ‒ 8 байт;
* answer ‒ 4 байт;
* verdict ‒ 1 байт;

Фактический объём на одну запись: 21 байт.

Чистый объём на одну запись: 12 байт.

*Таблица «Homework\_section\_to\_solution»:*

* id ‒ 4 байт;
* solution\_id ‒ 4 байт;
* section\_id ‒ 4 байт;

Фактический объём на одну запись: 12 байт.

Чистый объём на одну запись: 0 байт.

*Таблица «Log»:*

* id ‒ 4 байт;
* user\_id ‒ 4 байт;
* level ‒ 1 байт;
* time ‒ 8 байт;
* content ‒ 200 байт;
* action ‒ 50 байт;

Фактический объём на одну запись: 267 байт.

Чистый объём на одну запись: 259 байт.

**Расчёт избыточности модели.**

Используем обозначения для количества данных каждого типа из аналогичного пункта для нереляционной БД. Оттуда же возьмём предполагаемые значения переменных. К тому же пусть ‒ число зарегистрированных в системе учителей, и пусть учителей вчетверо меньше, чем классов.

Фактический объём всей БД равен:

байт.

С учётом предполагаемых соотношений переменных размер БД равен байт.

Чистый объём БД равен:

байт.

С учётом предполагаемых соотношений переменных размер БД равен байт.

Избыточность модели равна .

**Направление роста модели при увеличении количества объектов каждой сущности.**

Как можно видеть из выражений размера модели от количества данных, при увеличении количества объектов каждой отдельной сущности размер БД увеличивается линейно.

**Примеры запросов к БД.**

*Восстановление задания при авторизации (@id = id ученика):*

SELECT task\_text FROM Task JOIN Pupil ON Pupil.id = Task.pupil\_id WHERE pupil\_id = @id AND status = "PRG";

Запросы: 1.

Задействованные отношения: 2.

*Вставка попытки решения:*

INSERT INTO Solution (task\_id, answer, verdict)

SELECT task\_id FROM Task ORDER BY creation\_time DESC LIMIT 1, 42, 1;

Запросы: 2.

Задействованные отношения: 2.

*Создание нового задания (@now – текущее время; @deadline – отметка времени дедлайна):*

INSERT INTO Homework (creation\_time, deadline)

VALUES (@now, @deadline);

INSERT INTO Homework\_section (homework\_id, count, addition\_enabled, subtraction\_enabled, multiplication\_enabled, division\_enabled)

SELECT id FROM Homework WHERE creation\_time = @now AND deadline = @deadline ORDER BY id DESC LIMIT 1,

20, 0, 1, 1, 0;

Запросы: 3.

Задействованные отношения: 2.

*Получение всех учеников класса (@id = id класса):*

SELECT \* FROM User JOIN Pupil ON User.id = Pupil.user\_id WHERE class\_id = @id;

Запросы: 1.

Задействованные отношения: 2.

*Удаление класса (@id = id класса):*

# Для class\_id в Homework\_to\_class настроен параметр ON DELETE CASCADE;

# Для homework\_id в Homework\_section настроен параметр ON DELETE CASCADE;

DELETE FROM Class

WHERE id = @id;

DELETE FROM Homework

WHERE id NOT IN

(SELECT DISTINCT Homework.id FROM

Homework JOIN Homework\_to\_class ON Homework.id = Homework\_to\_class.homework\_id);

Запросы: 3.

Задействованные отношения: 4.

*Получение примеров, решённых во время активности задания (@id = id задания):*

SELECT \* FROM Task

JOIN Solution ON Task.id = Solution.task\_id

JOIN Homework\_section\_to\_solution ON

Homework\_section\_to\_solution.solution\_id = Solution.id

JOIN Homework\_section ON Homework\_section\_to\_solution.section\_id =

Homework\_section.id

JOIN Homework ON Homework\_section.homework\_id = Homework.id

WHERE Homework.id = @id

AND DATEDIFF(millisecond, Solution.submission\_time,

Homework.deadline) < 0

AND DATEDIFF(millisecond, Solution.submission\_time,

Homework.creation\_date) > 0

AND Solution.verdict = "PASS";

Запросы: 1.

Задействованные отношения: 5.

*Создание нового пользователя (Ученик):*

INSERT INTO User (email, password, first\_name, last\_name) VALUES ("john.doe@example.com", "2dfd511684ac00b8", "John", "Doe");

INSERT INTO Pupil (user\_id)

SELECT id FROM User WHERE email = "john.doe@example.com" AND password = "2dfd511684ac00b8" AND first\_name = "John" AND last\_name = "Doe" ORDER BY id DESC LIMIT 1;

Запросы: 3.

Задействованные отношения: 2.

*Создание нового пользователя (Учитель):*

INSERT INTO User (email, password, first\_name, last\_name) VALUES ("john.doe@example.com", "2dfd511684ac00b8", "John", "Doe");

INSERT INTO Teacher (user\_id)

SELECT id FROM User WHERE email = "john.doe@example.com" AND password = "2dfd511684ac00b8" AND first\_name = "John" AND last\_name = "Doe" ORDER BY id DESC LIMIT 1;

Запросы: 3.

Задействованные отношения: 2.

*Получение пользователя по адресу почты (@email – почтовый адрес):*

SELECT \* FROM User where email = @email;

Запросы: 1.

Задействованные отношения: 1.

*Добавление ученика в класс (@class – id класса, @pupil – id ученика):*

UPDATE Pupil

SET class\_id = @class

WHERE id = @pupil;

Запросы: 1.

Задействованные отношения: 1.

*Получение истории действий пользователя (@id - id пользователя):*

SELECT \* FROM Log

WHERE user\_id = @id;

Запросы: 1.

Задействованные отношения: 1.

*Получение истории действий всех пользователей:*

SELECT \* FROM Log

WHERE user\_id IS NOT NULL;

Запросы: 1.

Задействованные отношения: 1.

*Фильтрация системных логов по уровням логирования ("DEBUG", "INFO"):*

SELECT \* FROM Log

WHERE user\_id IS NULL AND level IN ("DEBUG", "INFO");

Запросы: 1.

Задействованные отношения: 1.

**3.3. Сравнение моделей**

Если допустить рассмотренное соотношение количества разных сущностей, то характеристики представленных моделей соотносятся следующим образом:

* Размер SQL и NoSQL моделей зависит от объёма входных данных схожим образом ‒ объём базы данных линейно зависит от количества сущностей разных типов.
* Реляционная БД занимает меньше места и медленнее растёт с увеличением числа классов:
  + Нереляционная база данных занимает

байт;

* + Реляционная база данных занимает

байт;

* При этом избыточность реляционной БД больше.
* Реализации сценариев использования реляционной БД задействуют не меньше коллекций, чем реализации тех же сценариев средствами нереляционной БД.
* По количеству необходимых для реализации сценария запросов сложно выделить превосходящую схему: для некоторых сценариев использования количество запросов в реляционную и нереляционную БД одинаково, где-то по этому показателю выигрывает реляционная БД, где-то ‒ нереляционная. Однако нереляционная БД включает дополнительную серверную логику между запросами в некоторых сценариях (яркий пример: if в сценарии удаления класса).

Однозначный вывод о превосходстве одной БД над другой сделать сложно: на примерах выше лишь подтверждается классическое соотношение ‒ документо-ориентированные базы данных занимают больше места, но быстрее ищут данные; но даже эти различия выражены не очень ярко. Возможно, в таком случае для выбора схемы хранения данных стоит использовать следующее соображение: если специализированный подход не даёт выраженного преимущества, возможно, стоит использовать универсальный подход. Таким образом, для предполагаемой структуры данных из представленных схем предположительно лучше подойдёт реляционная.

**4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ**

**4.1. Краткое описание**

Back-end сделан с использованием фреймворка express на основе платформы Node.js.

В качестве базы данных используется MongoDB.

Front-end представляет из себя web-приложение, использующее React. С его помощью можно удобно взаимодействовать с базой данных.

Для автоматизации развертывания и взаимодействия вышеуказанных элементов используется программное обеспечение Docker.

Инструкция по развёртыванию приложения представлена в приложении А.

**4.2. Страницы экранов приложения**

Экраны приложения представлены на рисунках ниже.

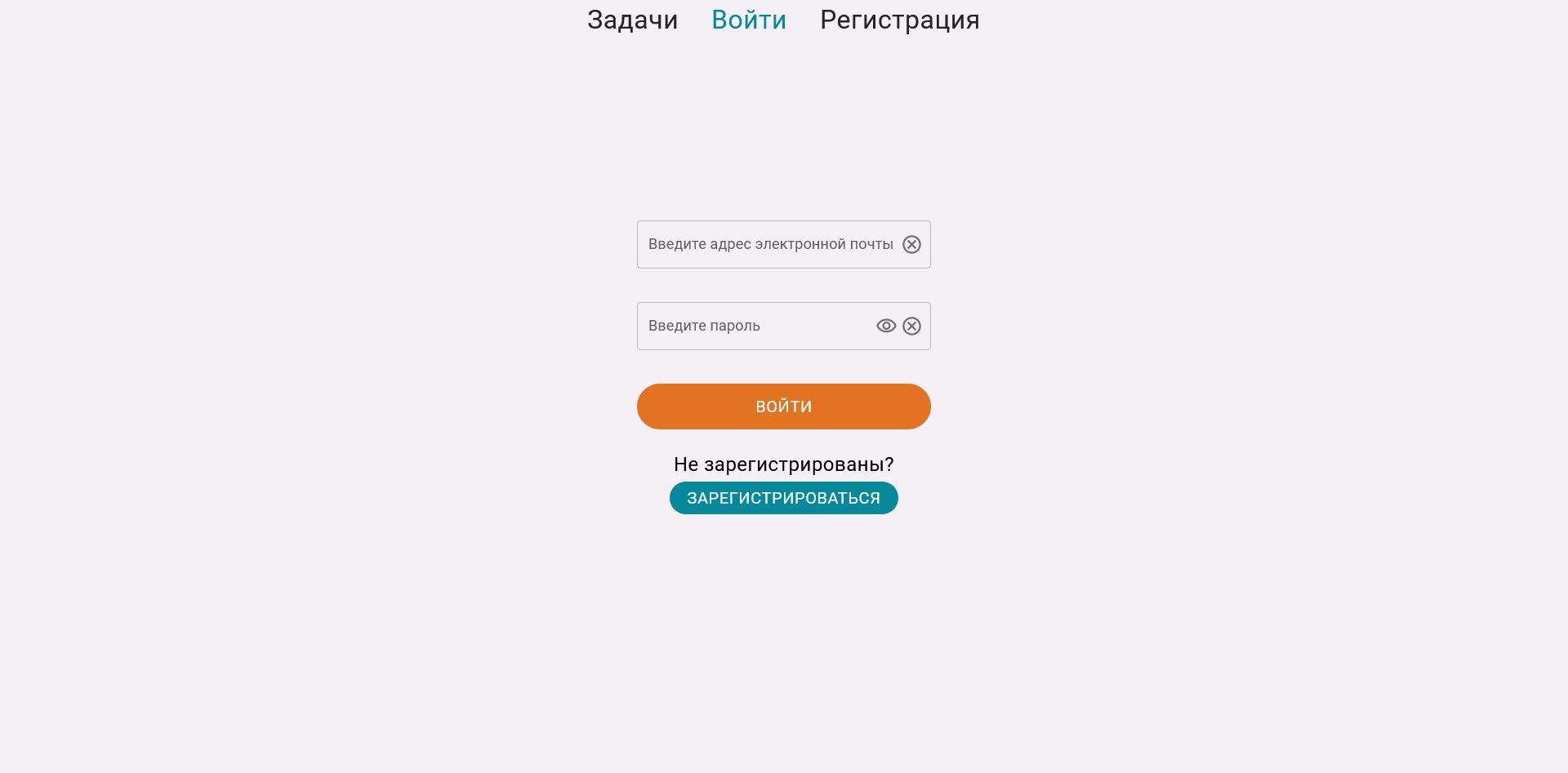


Рисунок 4 – Страница авторизации

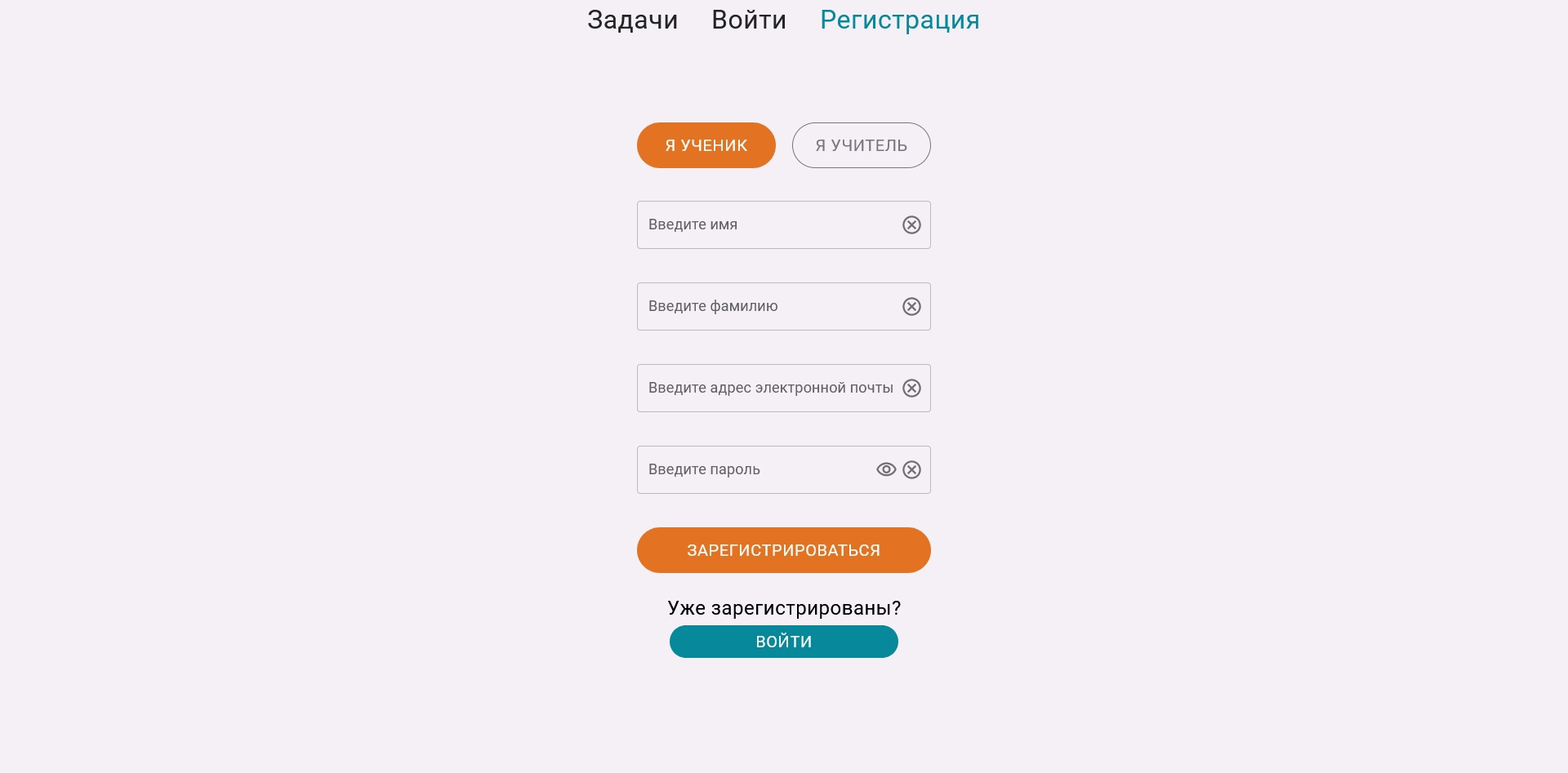


Рисунок 5 – Страница регистрации

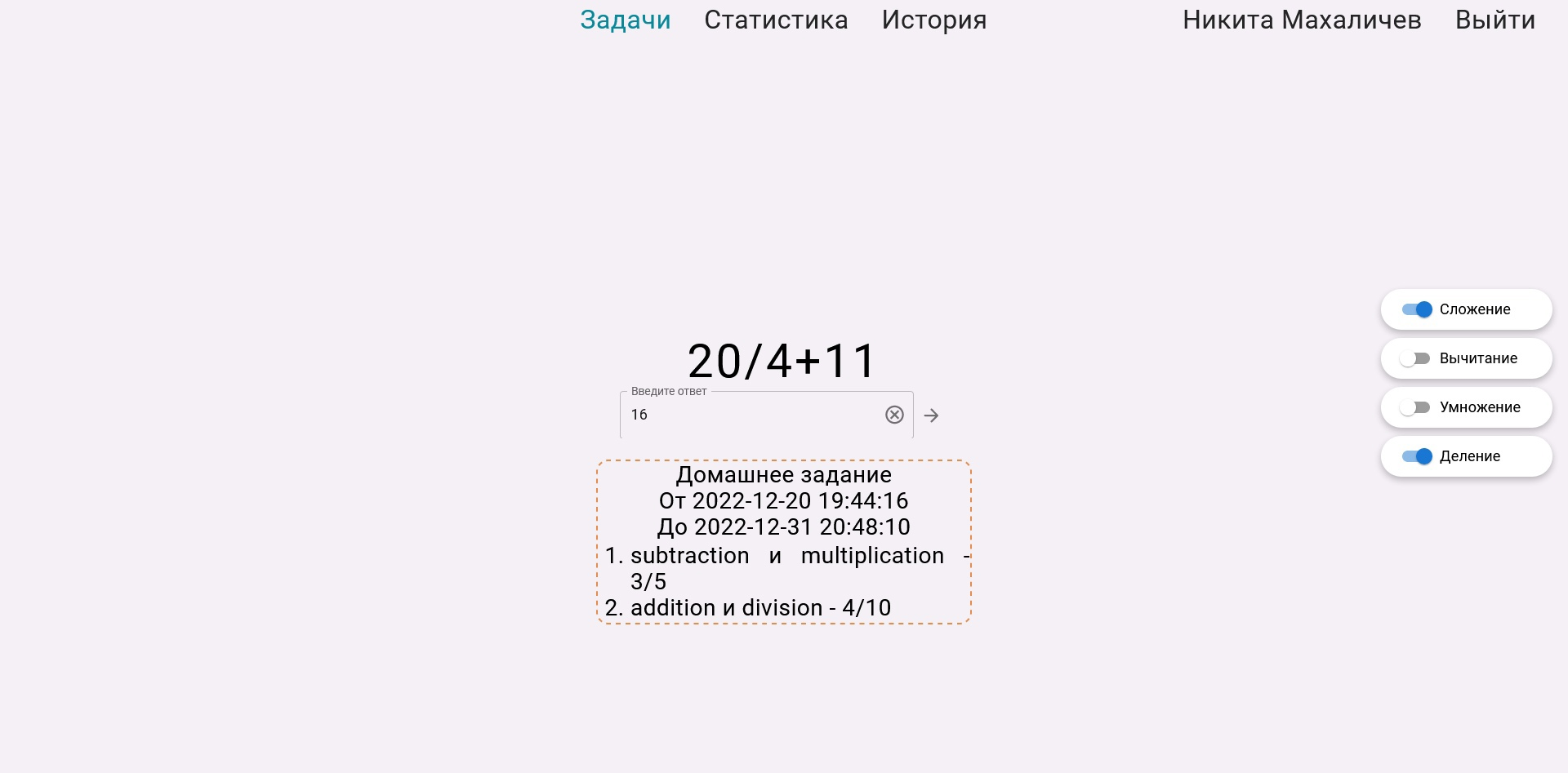


Рисунок 6 – Страница решения примеров

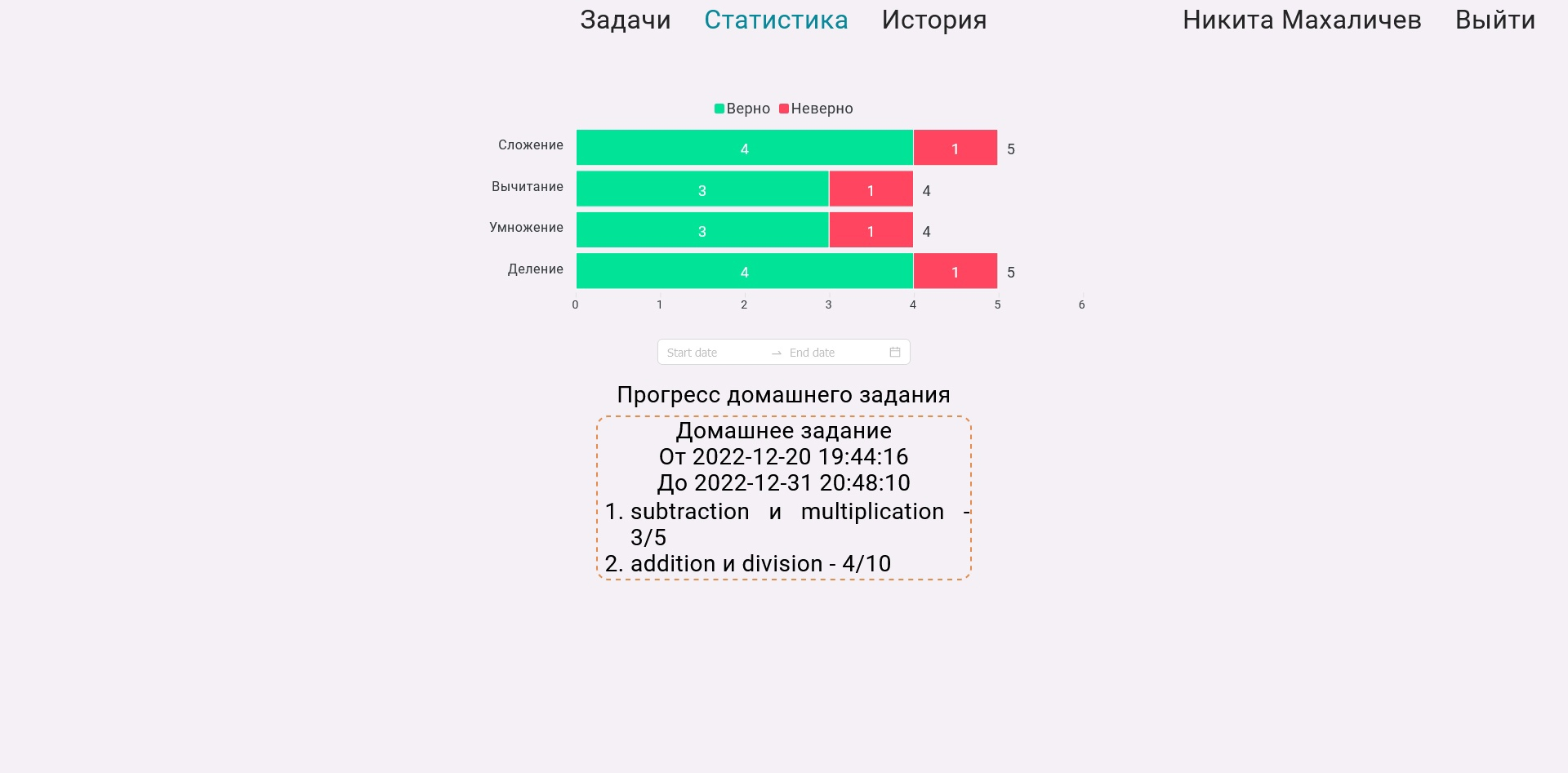


Рисунок 7 – Страница статистики ученика

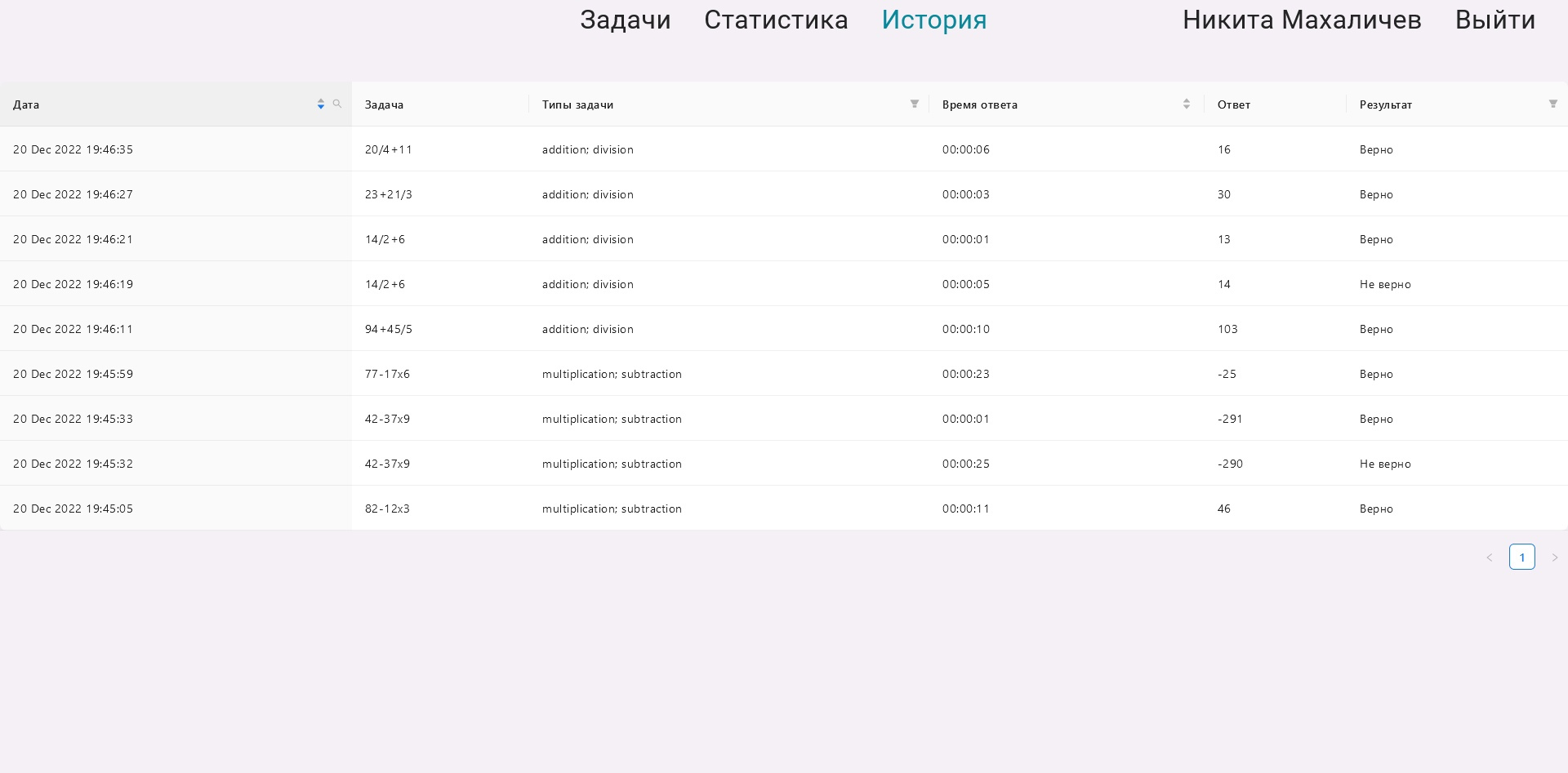


Рисунок 8 – Страница истории решений ученика

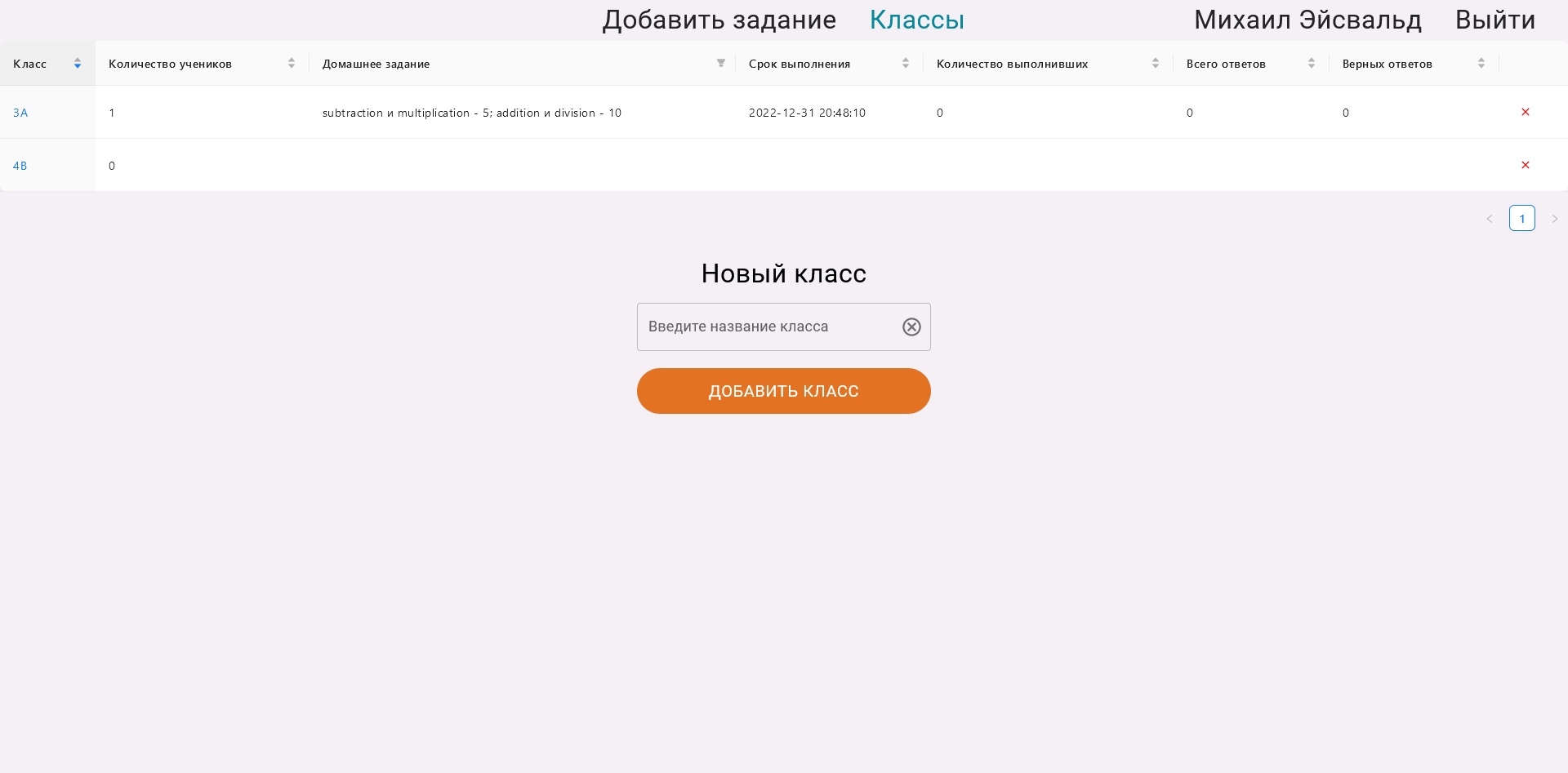


Рисунок 9 – Страница классов учителя

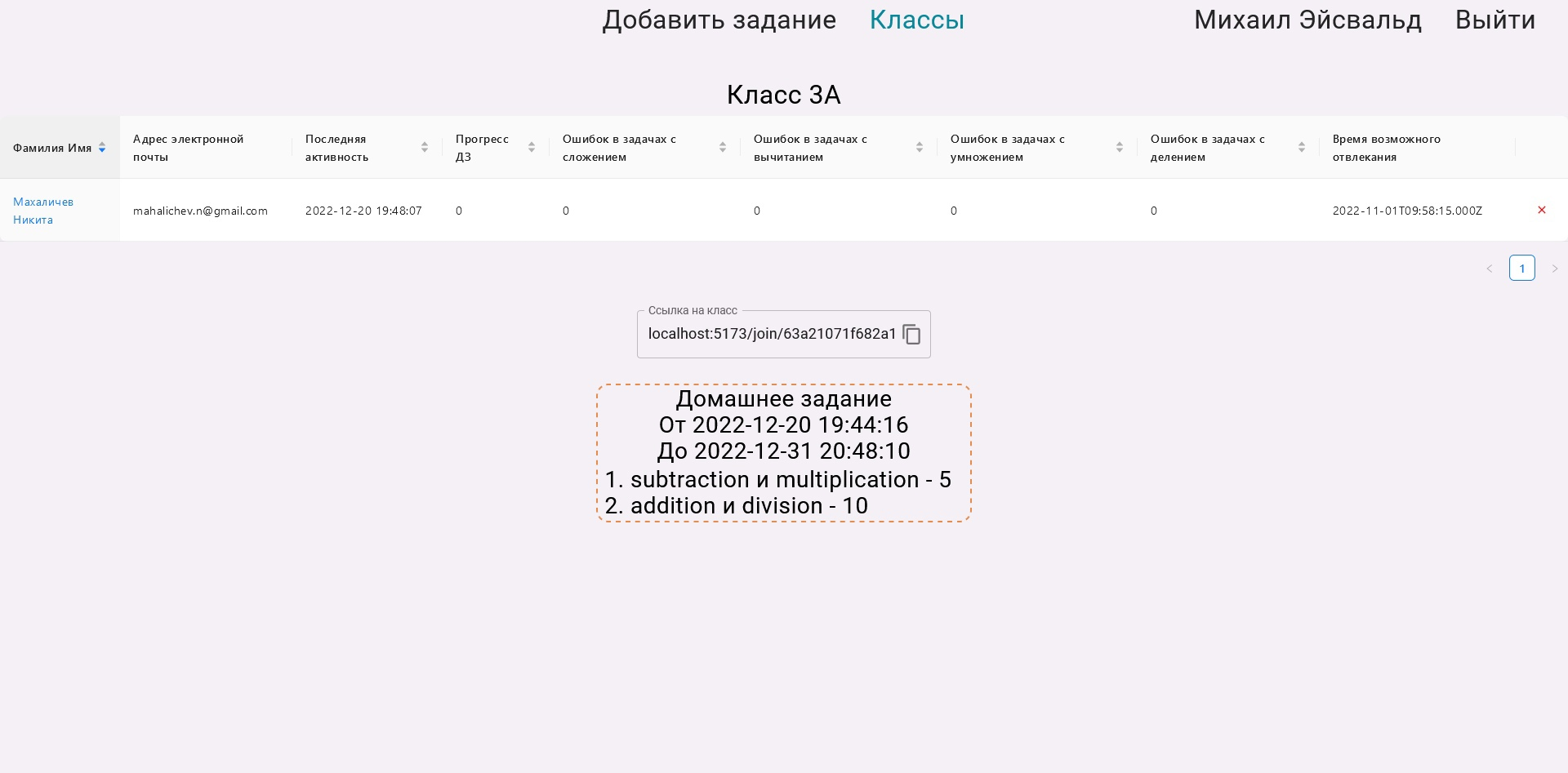


Рисунок 10 – Страница класса

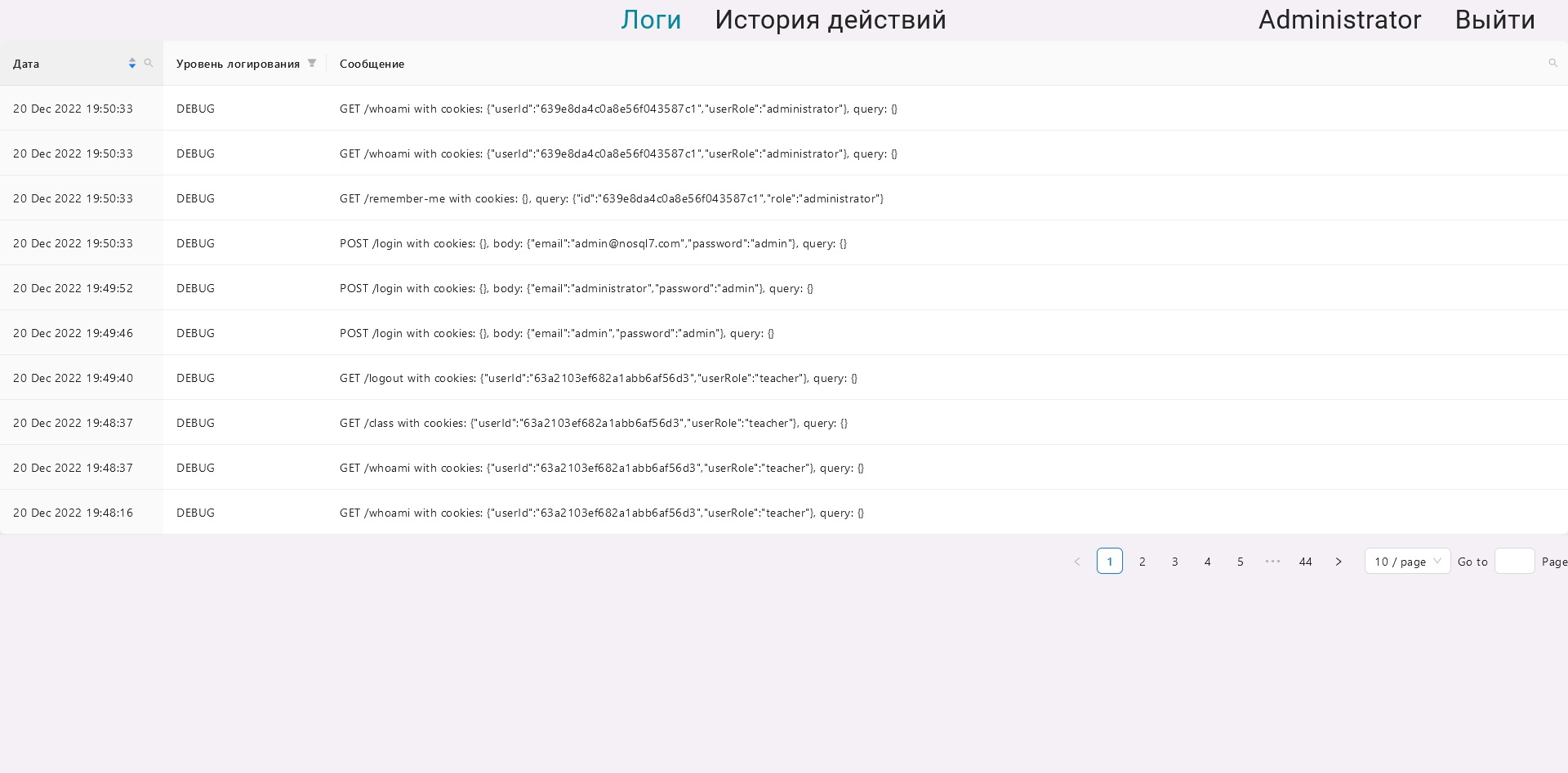


Рисунок 11 – Страница логов

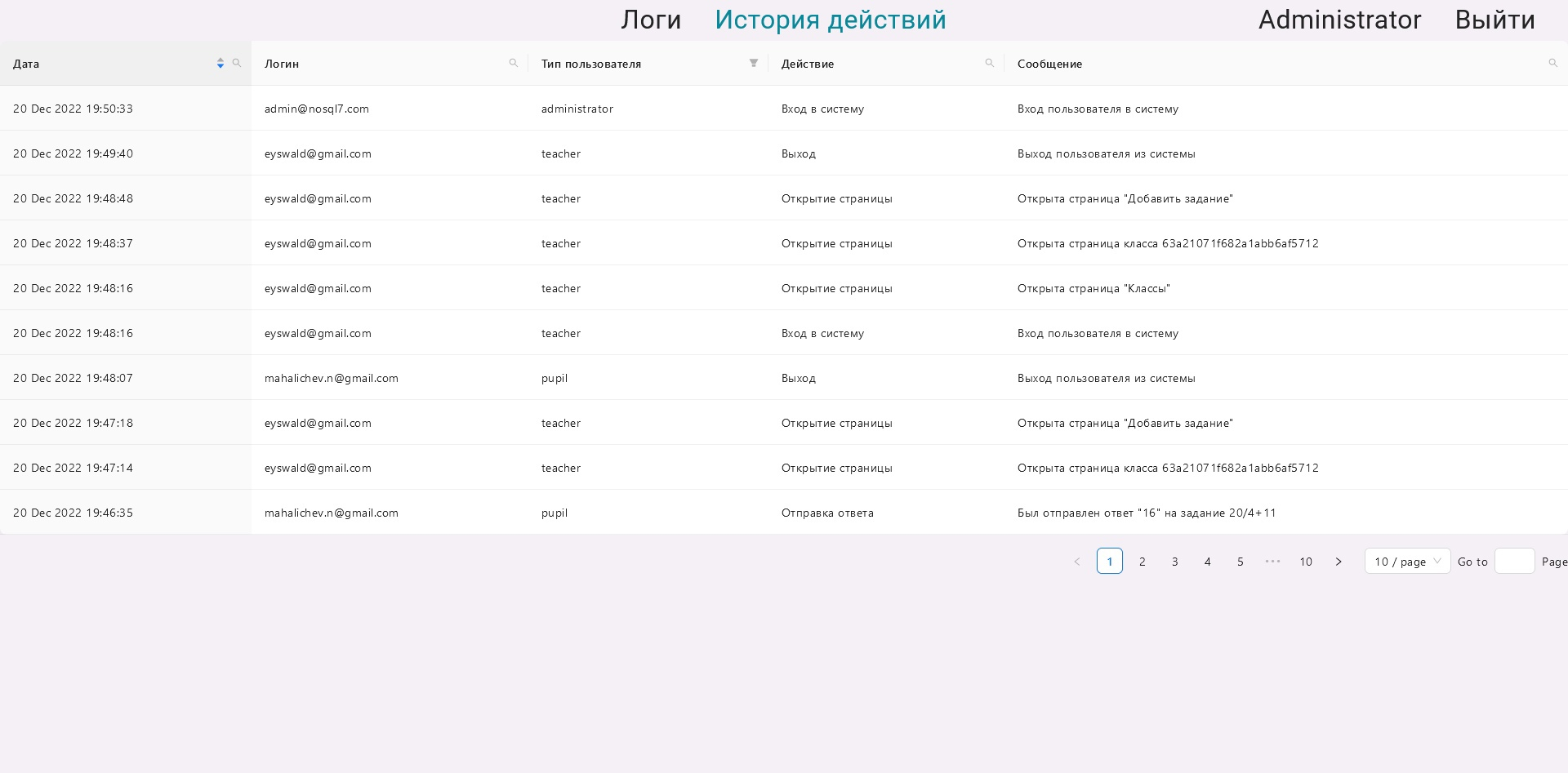


Рисунок 12 – Страница истории действий

**ВЫВОД**

**Достигнутые результаты**

В ходе работы было разработано web-приложение «Система автоматической проверки задач по математике», позволяющее пользователям взаимодействовать с базой данных: просмотр содержимого СУБД с помощью таблиц, добавление новых элементов, также была реализована регистрация и авторизация.

**Недостатки и пути для улучшения полученного решения**

Реализован только базовый функционал – добавление элемента и просмотр данных. Улучшением решения послужит фильтрация и сортировка данных БД для получения исчерпывающей статистики решения задач как для учителей, так и для учеников.

Нет функционала импорта/экспорта данных, из-за чего для выполнения этих действий придется использовать инструмент mongoexport на стороне сервера. Решением данного недостатка может послужить реализация данных функций в виде запросов на сервер с помощью клиентского интерфейса администратора.

**Будущее развитие решения**

* Доработка приложения до состояния версии с полным функционалом: сортировка, фильтрация, импорт и экспорт данных;
* Увеличение разнообразия категорий задач, которые могут генерироваться;
* Разработка мобильной версии сервиса.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. MongoDB: The Developer Data Platform // URL: https://www.mongodb.com/ (дата обращения: 15.12.2022).
2. React JavaScript-библиотека для создания пользовательских интерфейсов // URL: https://ru.reactjs.org/ (дата обращения: 15.12.2022).
3. Vite: Next Generation Frontend Tooling // URL: https://vitejs.dev/ (дата обращения: 15.12.2022).
4. Express.js // URL: https://expressjs.com/ (дата обращения: 15.12.2022).
5. Mongoose // URL: https://mongoosejs.com/ (дата обращения: 15.12.2022).

**ПРИЛОЖЕНИЕ А**

**ИНСТРУКЦИЯ ПО РАЗВЁРТЫВАНИЮ ПРИЛОЖЕНИЯ**

1. Скачать проект из репозитория по ссылке https://github.com/moevm/nosql2h22-math;
2. Запустить сборку командой  
   docker-compose build --no-cache && docker-compose up;
3. Открыть приложение в браузере по адресу 127.0.0.1:5173.