**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

**Курсовая Работа**

**по дисциплине «Введение в нереляционные базы данных»**

**Тема: Информационная система оценок и аналитика проблемных студентов**

|  |  |  |
| --- | --- | --- |
| Студент гр. 2381 |  | Ильясов М.Р. |
| Студент гр. 2381 |  | Комосский Е.А. |
| Студент гр. 2381 |  | Кузнецов И.И. |
| Студент гр. 2381 |  | Мавликаев И.С. |
| Студент гр. 2381 |  | Рыжиков И.А. |
|  |  |  |
| Преподаватель |  | Заславский М.М. |

Санкт-Петербург

2025

# Задание на курсовую работу

Студент Ильясов М.Р. 2381

Студент Комосский Е.А. 2381

Студент Кузнецов И.И. 2381

Студент Мавликаев И.С. 2381

Студент Рыжиков И.А. 2381

Тема работы: Информационная система оценок и аналитика проблемных студентов

Исходные данные:

Задача - организовать сервис, где будут агрегироваться все действия студента в ВУЗе (учебные активности, посещаемость ВУЗа, участие в социальной активности ….), и на их основе будут строится прогнозы по успеваемости, рисках для студента / группы / потока.

Содержание пояснительной записки:

«Содержание», «Введение», «Сценарии использования», «Модель данных», «Разработанное приложение», «Выводы», «Приложения», «Литература»

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 05.02.2025

Дата сдачи реферата: 29.05.2025

Дата защиты реферата: 29.05.2025

|  |  |  |
| --- | --- | --- |
| Студент гр. 2381 |  | Ильясов М.Р. |
| Студент гр. 2381 |  | Комосский Е.А. |
| Студент гр. 2381 |  | Кузнецов И.И. |
| Студент гр. 2381 |  | Мавликаев И.С. |
| Студент гр. 2381 |  | Рыжиков И.А. |
|  |  |  |
| Преподаватель |  | Заславский М.М. |

# Аннотация

Разработан прототип информационной системы, предназначенной для централизованного хранения и просмотра данных о студентах: персональные сведения, академические показатели, принадлежность к группам и учёт действий пользователей. MVP включает FastAPI‑backend, React + Vite‑frontend и кластер MongoDB (реплика‑сет). На текущий момент реализованы базовые CRUD‑операции, фильтрация. Модули прогнозирования успеваемости и массового импорта находятся в плане развития и в данную версию не входят.

Summary

A prototype information system was developed for centralized storage and viewing of student data: personal information, academic performance, group membership, and user activity logs. The MVP includes a FastAPI backend, React + Vite frontend, and a MongoDB cluster (replica set). To date, basic CRUD operations and filtering have been implemented. Modules for predicting academic performance and mass import are in the development plan and are not included in this version.

Содержание

[Задание на курсовую работу 2](#_Toc199443917)

[Аннотация 4](#_Toc199443918)

[Введение 6](#_Toc199443919)

[1. Просмотр статистики и истории действий 7](#_Toc199443920)

[2. Редактирование данных (массовые изменения) 8](#_Toc199443921)

[3. Импорт данных 9](#_Toc199443922)

[4. Удаление данных 10](#_Toc199443923)

[5. Сообщение об ошибке 10](#_Toc199443924)

[6. Настройка данных (категории, вычисляемые поля) 11](#_Toc199443925)

[7. Экспорт данных 12](#_Toc199443926)

[Макет UI 13](#_Toc199443927)

[Модель данных 15](#_Toc199443928)

[Нереляционная модель 15](#_Toc199443929)

[Описание назначений коллекций, типов данных и сущностей 17](#_Toc199443930)

[Заключение 22](#_Toc199443931)

[Список используемой литературы 23](#_Toc199443932)

[Приложение А Исходный код программы 24](#_Toc199443933)

# Введение

**Актуальность решаемой проблемы**

Современные образовательные учреждения сталкиваются с необходимостью оперативного анализа учебной активности студентов для своевременного выявления проблем в обучении и принятия обоснованных управленческих решений. Актуальность данной задачи обусловлена ростом объёмов образовательных данных и потребностью в их интеграции из различных источников: посещаемость занятий, оценки, участие в социальной и научной деятельности.

**Постановка задачи**

В рамках курсовой работы поставлена задача разработки информационной системы, позволяющей агрегировать данные о действиях студентов в процессе обучения, обеспечивать их централизованное хранение и использовать для построения прогнозов по успеваемости и оценке рисков как для отдельных студентов, так и для учебных групп и потоков.

**Предлагаемое решение**

Предлагаемое решение представляет собой веб-сервис, основанный на базе данных MongoDB, с интерфейсами для работы с пользователями, студентами, преподавателями и учебными дисциплинами, а также с модулем аналитики, предусматривающим расширение для прогнозирования успеваемости.

**Качественные требования к решению**

К качественным требованиям к системе относятся:

* масштабируемость (возможность обработки больших объёмов данных),
* расширяемость архитектуры,
* удобство пользовательского интерфейса,
* корректность хранения и отображения информации,
* возможность подключения дополнительных источников данных в будущем.

СЦЕНАРИЙ ИСПОЛЬЗОВАНИЯ

## 1. Просмотр статистики и истории действий

**Акторы:**

* Студент
* Староста
* Преподаватель
* Администратор

**Предусловия:**

* Пользователь имеет доступ к системе.
* Данные о статистике и истории действий загружены.

**Основной поток:**

1. Пользователь заходит в систему.
2. Открывает раздел статистики.
3. Выбирает интересующий его период или группу (если доступно).
4. Получает визуализацию данных (графики, проценты, средний балл и т. д.).

**Альтернативные потоки:**

* Если данные отсутствуют, отображается сообщение о недоступности информации.
* Если у пользователя нет прав на просмотр, выводится уведомление об ограничении доступа.

**Дополнительно для студента:**

* У студента есть более подробный профиль, содержащий расширенную информацию о нём.
* Другие пользователи (например, преподаватели и администраторы) могут просматривать профиль студента.

**Общая таблица статистики пользователей:**

* Отображается таблица со статистикой всех пользователей.
* Минимальный набор фильтров: группа, год рождения (если применимо), вероятность отчисления, средний и предсказанный баллы.
* Если интерфейс позволяет, дополнительно: посещаемость, оценки, активности.
* Возможность сортировки и фильтрации данных по ключевым категориям.

**Отображение сложных данных:**

* Для сложных данных в таблице применяется механизм раскрывающихся блоков (аккордеонов), аналогично блокам кода в вебе.
* В общем виде отображаются ключевые метрики (например, средний балл, посещаемость, вероятность отчисления).
* При раскрытии блока показываются детализированные данные (оценки по предметам, активности, индивидуальные показатели).

## 2. Редактирование данных (массовые изменения)

**Акторы:**

* Администратор
* Преподаватель (ограниченное редактирование)

**Предусловия:**

* Пользователь обладает правами на редактирование.
* Данные доступны для изменения.

**Основной поток:**

1. Пользователь открывает раздел редактирования.
2. Выбирает группу данных для редактирования.
3. Вносит изменения (например, обновляет оценки, посещаемость).
4. Подтверждает изменения.

**Альтернативные потоки:**

* Если пользователь пытается редактировать данные, загруженные кем-то другим, выводится предупреждение.
* Если пользователь не имеет прав на редактирование, действие блокируется.

## 3. Импорт данных

**Акторы:**

* Администратор
* Староста (для своей группы)
* Преподаватель (свои данные)

**Предусловия:**

* Данные подготовлены для загрузки (формат CSV или иные поддерживаемые структуры).

**Основной поток:**

1. Пользователь заходит в раздел загрузки данных.
2. Выбирает файл или вводит данные вручную.
3. Подтверждает загрузку.
4. Система проверяет корректность данных.
5. Данные сохраняются в системе.

**Альтернативные потоки:**

* Если файл содержит ошибки, пользователь получает уведомление о некорректных данных.
* Если у пользователя нет прав на загрузку, операция блокируется.

## 4. Удаление данных

**Акторы:**

* Администратор

**Предусловия:**

* Удаление разрешено только для данных, загруженных самим пользователем в течение ограниченного времени.

**Основной поток:**

1. Пользователь открывает список загруженных данных.
2. Выбирает данные для удаления.
3. Подтверждает удаление.

**Альтернативные потоки:**

* Если данные старше разрешённого периода, удаление невозможно.
* Если у пользователя нет прав на удаление, выводится сообщение об ошибке.

## 5. Сообщение об ошибке

**Акторы:**

* Студент
* Староста
* Преподаватель

**Предусловия:**

* Ошибка выявлена в загруженных данных.

**Основной поток:**

1. Пользователь открывает форму сообщения об ошибке.
2. Выбирает тип ошибки (например, неверные данные, отсутствие данных).
3. Вводит комментарий.
4. Отправляет запрос.
5. Система уведомляет ответственного (администратора или старосту).

**Альтернативные потоки:**

* Если ошибка не связана с системой, пользователю предлагается обратиться к преподавателю напрямую.

**Типы ошибок:**

1. Общая ошибка сайта – оформляется в виде формы обратной связи.
2. Ошибка в одной записи – должна либо появляться при наведении рядом с данными, либо быть всегда видимой.

## 6. Настройка данных (категории, вычисляемые поля)

**Акторы:**

* Администратор

**Предусловия:**

* Пользователь имеет права на изменение структуры данных.

**Основной поток:**

1. Пользователь заходит в настройки данных.
2. Добавляет новые категории или вычисляемые поля.
3. Определяет параметры обработки (например, автоматический расчёт среднего балла).
4. Сохраняет изменения.

**Альтернативные потоки:**

* Если изменение может повлиять на существующие данные, система запрашивает подтверждение.
* Если формат некорректен, отображается сообщение об ошибке.

## 7. Экспорт данных

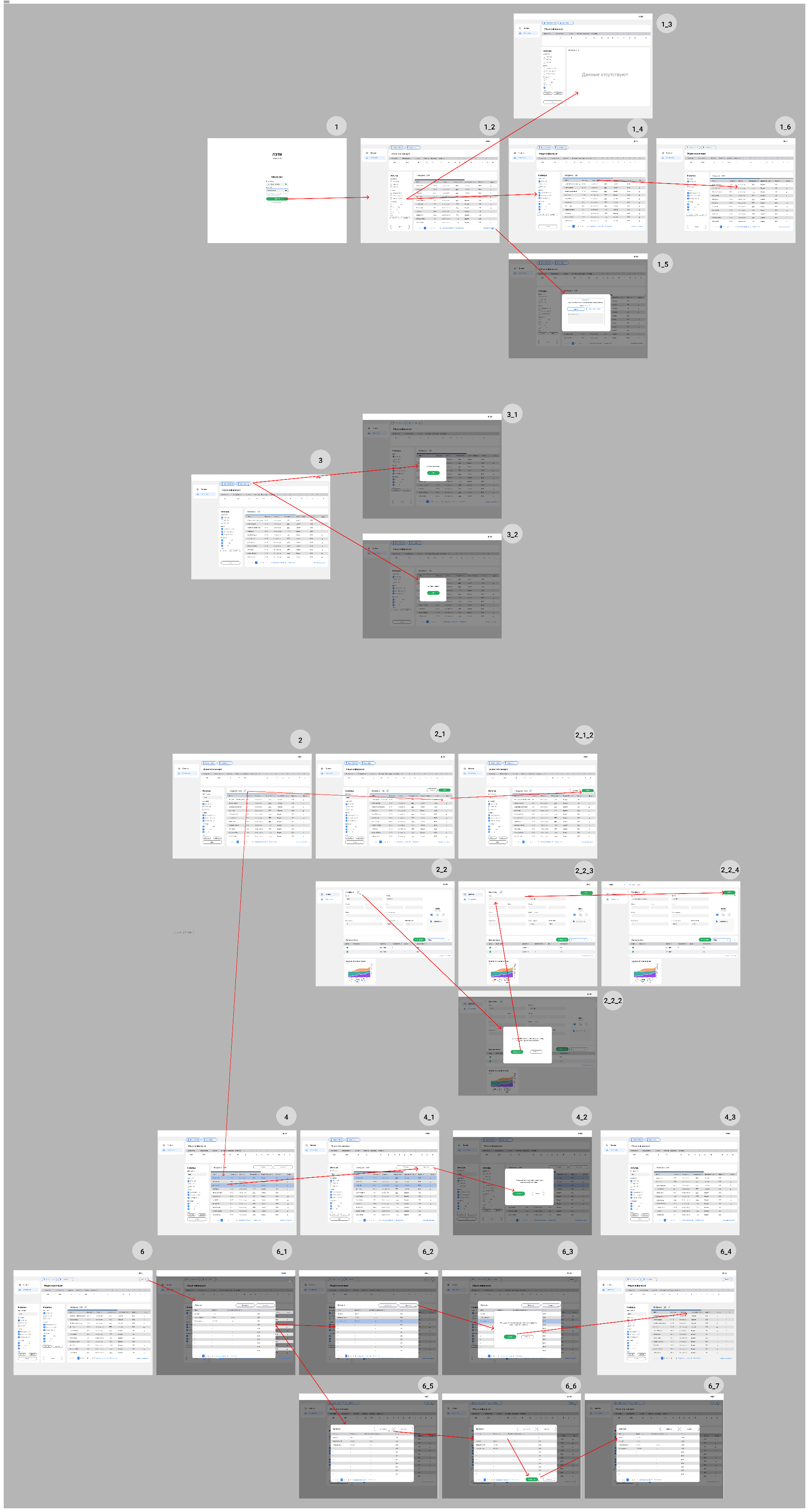
**Акторы:**

* Администратор

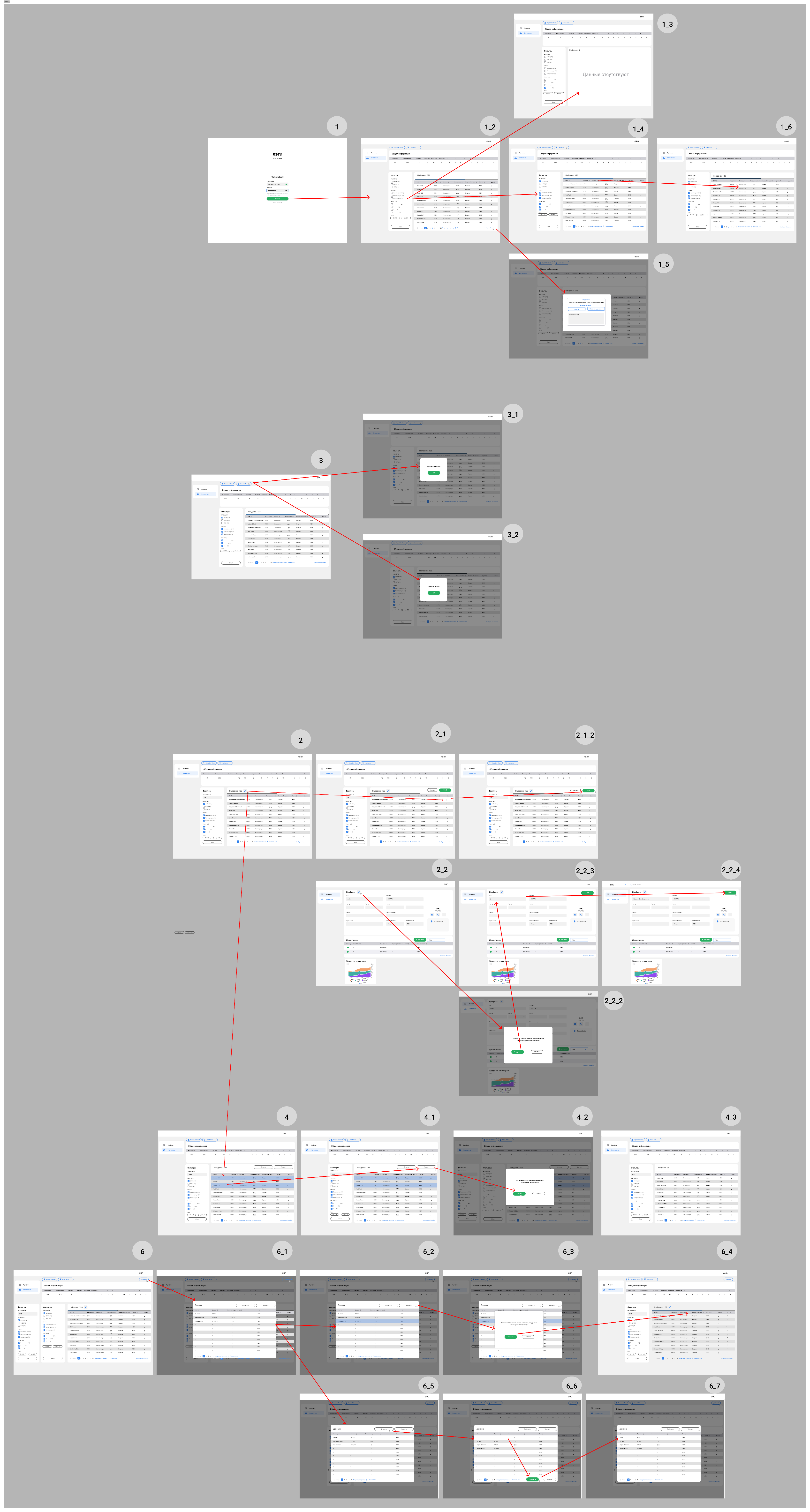
**Предусловия:**

* Пользователь имеет права на выгрузку данных.
* Основной поток:
* Пользователь заходит в список загруженных данных.
* Выставляет необходимые фильтры.
* Начинает экспорт.
* Данные выгружаются в формате csv/json.

# Макет UI



1. Макет UI. Часть 1



1. Макет UI. Часть 2



1. Макет UI. Часть 3

# Модель данных

## Нереляционная модель

"User": {

"\_id": "ObjectId",

"email": "string",

"login": "string",

"password\_hash": "string",

"first\_name": "string",

"middle\_name": "string",

"last\_name": "string",

"active": "bool",

"role\_id": "string"

},

"Student": {

"\_id": "ObjectId",

"user\_id": "ObjectId",

"birth\_date": "date",

"admission\_year": "int",

"student\_type": {

"type": "string",

"enum": ["bachelor", "master", "aspirant", "specialist"]

},

"course": "int",

"program\_name": {

"type": "string",

"enum": ["Theoretical math", "Applied physics", "Computer science"]

},

"faculty": {

"type": "string",

"enum": ["Math", "Physics", "IT"]

},

"group\_name": {

"type": "string",

"enum": ["2323", "1421", "3501"]

},

"funding\_type": {

"type": "string",

"enum": ["budget", "contract"]

},

"statistic": {

"average\_score": "decimal",

"attendance\_percent": "decimal",

"calculation\_date": "date",

"count\_activities": "int",

"exlusion\_probability": "decimal",

"subjects": [

{

"subject\_id": "ObjectId",

"total\_lessons": "int",

"attendance\_lessons": "int",

"year": "int",

"season": {

"type": "string",

"enum": ["autumn", "spring"]

},

"prediction\_score": "decimal",

"score": "decimal",

"grade\_value": {

"type": "string",

"enum": ["pass", "fail", "5", "4", "3", null]

}

}

]

}

},

"Log": {

"\_id": "ObjectId",

"user\_id": "ObjectId",

"action\_type": "string",

"action\_date": "date",

"ip\_address": "string",

"affected\_entity": "string",

"entity\_id": "ObjectId",

"description": "string",

"role": "string"

},

"Teacher": {

"\_id": "ObjectId",

"user\_id": "ObjectId",

"assigned\_groups": ["string"],

"assigned\_subjects": ["ObjectId"]

},

"SubjectMeta": {

"\_id": "ObjectId",

"subject\_name": "string",

"description": "string",

"grade\_type": {

"type": "string",

"enum": ["pass/fail", "exam"]

},

"is\_activity": "bool"

}

}

### Описание назначений коллекций, типов данных и сущностей

| **Коллекция** | **Назначение** |
| --- | --- |
| User | Учетная запись: email, логин, пароль, роль |
| Student | Студенческий профиль + вложенная статистика |
| Teacher | Преподаватель: доступные группы и предметы |
| SubjectMeta | Метаинформация о предмете: название, тип, описание |
| Log | История действий пользователя |

**User**

* \_id (ObjectId, уникальный id) — **12 байт**
* email (string) — **~30 байт**
* login (string) — **~15 байт**
* password\_hash (string) — **~60 байт**
* first\_name (string) — **~15 байт**
* middle\_name (string) — **~15 байт**
* last\_name (string) — **~15 байт**
* active (bool) — **1 байт**
* role\_id (string) — **~10 байт**

Итого ~150 байт на пользователя.

**Student**

* \_id (ObjectId) — **12 байт**
* user\_id (ObjectId) — **12 байт**
* birth\_date (date) — **8 байт**
* admission\_year (int) — **4 байта**
* student\_type (string, enum) — **~10 байт**
* course (int) — **4 байта**
* program\_name (string, enum) — **~20 байт**
* faculty (string, enum) — **~10 байт**
* group\_name (string, enum) — **~6 байт**
* funding\_type (string, enum) — **~8 байт**
* statistic — **60 байт**
  + average\_score (decimal) — **16 байт**
  + attendance\_percent (decimal) — **16 байт**
  + calculation\_date (date) — **8 байт**
  + count\_activities (int) — **4 байта**
  + exlusion\_probability (decimal) — **16 байт**
* subjects[] (array of objects, ~50 предметов) — **~3400 байт**
  + Один элемент массива — **~68 байт**
    - subject\_id (ObjectId) — **12 байт**
    - total\_lessons (int) — **4 байта**
    - attendance\_lessons (int) — **4 байта**
    - year (int) — **4 байта**
    - season (string, enum) — **~6 байт**
    - prediction\_score (decimal) — **16 байт**
    - score (decimal) — **16 байт**
    - grade\_value (string, enum) — **~6 байт**

Итого ~5 038 байт на студента.

**Log**

* \_id (ObjectId) — **12 байт**
* user\_id (ObjectId) — **12 байт**
* action\_type (string) — **~10 байт**
* action\_date (date) — **8 байт**
* ip\_address (string) — **~15 байт**
* affected\_entity (string) — **~15 байт**
* entity\_id (ObjectId) — **12 байт**
* description (string) — **~50 байт**
* role (string) — **~10 байт**

Итого ~150 байт на лог.

**Teacher**

* \_id (ObjectId) — **12 байт**
* user\_id (ObjectId) — **12 байт**
* assigned\_groups (array of strings, ~10 групп) — **~60 байт**
* assigned\_subjects (array of ObjectId, ~2 предмета) — **24 байта**

Итого ~110 байт на преподавателя.

**SubjectMeta**

* \_id (ObjectId) — **12 байт**
* subject\_name (string) — **~25 байт**
* description (string) — **~100 байт**
* grade\_type (string, enum) — **~10 байт**
* is\_activity (bool) — **1 байт**

Итого ~160 байт на предмет.

**Оценка объема информации, хранимой в модели**

**Общие параметры**

Мы оценим размер каждой коллекции и выразим итог через N, учитывая:

* **N** — количество студентов.
* **U = N** — количество пользователей (один к одному).
* **T = N / 20** — количество преподавателей (в среднем 1 на 20 студентов).
* **L = N × 5** — количество логов (допустим, в среднем 5 записей на студента).
* **S = 150** — количество уникальных предметов (SubjectMeta), фиксировано.

**Размер объектов**

| **Объект** | **Размер одного экземпляра** | **Кол-во** | **Формула размера** |
| --- | --- | --- | --- |
| **User** | ~150 байт | = N | 150 × N |
| **Student** | ~5038 байт | = N | 5038 × N |
| **Teacher** | ~110 байт | = N / 20 | 110 × N / 20 = 5.5 × N |
| **Log** | ~150 байт | = 5 × N | 150 × 5 × N = 750 × N |
| **SubjectMeta** | ~160 байт | = 150 | 24 000 байт (константа) |

**Общая формула размера в байтах**

Общий\_объем(N) =

150 × N // User

+ 5 038 × N // Student

+ 5.5 × N // Teacher

+ 750 × N // Log

+ 24 000 // SubjectMeta (константа)

Или упрощённо:

Общий\_объем(N) ≈ 5 943.5 × N + 24 000 байт

**Примеры**

| **Кол-во студентов (N)** | **Общий объем (байт)** | **Примерно в МБ** |
| --- | --- | --- |
| 1 000 | 5 971 500 | ~5.7 МБ |
| 10 000 | 59 566 00 | ~56.7 МБ |
| 100 000 | 595 566 000 | ~566.8 МБ |

**Избыточность данных (отношение между фактическим объемом модели и «чистым» объемом данных)**

**Чистый объём данных**

**User**

* login (string) — **~15 байт**
* password\_hash (string) — **~60 байт**
* email (string) — **~30 байт**
* first\_name (string) — **~15 байт**
* middle\_name (string) — **~15 байт**
* last\_name (string) — **~15 байт**

Итого: **~150 байт на пользователя**

**Student**

* user\_id (ObjectId) — **12 байт**
* birth\_date (date) — **8 байт**
* admission\_year (int) — **4 байта**
* student\_type (string, enum) — **~10 байт**
* course (int) — **4 байта**
* program\_name (string, enum) — **~20 байт**
* faculty (string, enum) — **~10 байт**
* group\_name (string, enum) — **~6 байт**
* funding\_type (string, enum) — **~8 байт**
* statistic — **60 байт**
  + average\_score (decimal) — **16 байт**
  + attendance\_percent (decimal) — **16 байт**
  + calculation\_date (date) — **8 байт**
  + count\_activities (int) — **4 байта**
  + exlusion\_probability (decimal) — **16 байт**
* subjects[] (array of objects, ~50 предметов) — **~2 000 байт**
  + Один элемент массива — **~40 байт**
    - subject\_id (ObjectId) — **12 байт**
    - attendance\_lessons (int) — **4 байта**
    - year (int) — **4 байта**
    - season (string, enum) — **~6 байт**
    - score (decimal) — **16 байт**
    - grade\_value (string, enum) — **~6 байт**

Итого: ~2 152 байта на одного студента

**Teacher**

* user\_id (ObjectId) — **12 байт**
* assigned\_groups (array of strings, ~10 групп) — **~60 байт**
* assigned\_subjects (array of ObjectId, ~2 предмета) — **24 байта**

Итого: **~96 байт на одного преподавателя**  
→ В среднем 1 преподаватель на 20 студентов: 96 × N / 20 = 4.8 × N

**Log**

* user\_id (ObjectId) — **12 байт**
* action\_type (string) — **~10 байт**
* action\_date (date) — **8 байт**
* affected\_entity (string) — **~10 байт**
* entity\_id (ObjectId) — **12 байт**

Итого: **~52 байта × 5 записей на студента = 260 × N**

**SubjectMeta**

* subject\_name (string) — **~25 байт**
* description (string) — **~100 байт**
* grade\_type (string) — **~10 байт**
* is\_activity (bool) — **1 байт**

Итого: **~160 байт × 150 = 24 000 байт (константа)**

Размер объектов

| **Объект** | **Размер одного экземпляра** | **Кол-во** | **Формула размера** |
| --- | --- | --- | --- |
| **User** | ~150 байт | = N | 150 × N |
| **Student** | ~2 152 байт | = N | 2 152 × N |
| **Teacher** | ~96 байт | = N / 20 | 4.8 × N |
| **Log** | ~52 байта × 5 = 260 байт | = 5 × N | 260 × N |
| **SubjectMeta** | ~160 байт | = 150 | 24 000 байт (константа) |

**Общая формула "чистого" объёма в байтах**

Чистый\_объем(N) =

150 × N // User

+ 2 152 × N // Student

+ 4.8 × N // Teacher

+ 260 × N // Log

+ 24 000 // SubjectMeta

Или:

Чистый\_объем(N) = 2 566.8 × N + 24 000 байт

**Избыточность модели**

Избыточность = Общий\_объем / Чистый\_объем

Избыточность = (5 943.5 × N + 24 000) / (2 566.8 × N + 24 000)

**Примеры избыточности для разных значений N**

| **Кол-во студентов (N)** | **Избыточность** |
| --- | --- |
| 1 000 | 2.30 |
| 10 000 | 2.31 |
| 100 000 | 2.32 |
| N→∞N→∞ | 2.32 |

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

Так как в модели все данные выражаются через N, то при увеличении количества объектов каждой сущности, модель будет расти линейно.

При этом, стоит учитывать, что несмотря на то, что в модели есть массивы объектов, они не влияют на линейный рост, так как количество элементов в них фиксировано и ограниченно сверху (например, 50 предметов в учебном плане).

Таким образом, при увеличении количества объектов в модели, она будет расти линейно, что позволяет легко масштабировать систему.

**Примеры данных для нереляционной базы данных**

**User**

{

"\_id": { "$oid": "661adfa5e13f1a1234567890" },

"email": "ivan.ivanov@example.com",

"login": "ivanivan",

"password\_hash": "$2b$10$....",

"first\_name": "Иван",

"middle\_name": "Иванович",

"last\_name": "Иванов",

"active": true,

"role\_id": "student"

}

**Student**

{

"\_id": { "$oid": "661ae05de13f1a1234567891" },

"user\_id": { "$oid": "661adfa5e13f1a1234567890" },

"birth\_date": { "$date": "2003-06-21T00:00:00Z" },

"admission\_year": 2021,

"student\_type": "bachelor",

"course": 3,

"program\_name": "Прикладная математика и информатика",

"faculty": "IT",

"group\_name": "2323",

"funding\_type": "budget",

"statistic": {

"average\_score": 4.2,

"attendance\_percent": 87.5,

"calculation\_date": { "$date": "2025-04-10T00:00:00Z" },

"count\_activities": 12,

"exlusion\_probability": 0.08,

"subjects": [

{

"subject\_id": { "$oid": "661ae1d9e13f1a12345678a0" },

"attendance\_lessons": 28,

"year": 2024,

"season": "autumn",

"score": 5,

"grade\_value": "5"

},

{

"subject\_id": { "$oid": "661ae1d9e13f1a12345678a1" },

"attendance\_lessons": 22,

"year": 2025,

"season": "spring",

"score": null,

"grade\_value": null

}

]

}

}

**Teacher**

{

"\_id": { "$oid": "661ae1fbe13f1a1234567892" },

"user\_id": { "$oid": "661ae1fbe13f1a1234567893" },

"assigned\_groups": ["2323", "2330"],

"assigned\_subjects": [

{ "$oid": "661ae1d9e13f1a12345678a0" },

{ "$oid": "661ae1d9e13f1a12345678a1" }

]

}

**Log**

{

"\_id": { "$oid": "661ae30fe13f1a1234567894" },

"user\_id": { "$oid": "661ae1fbe13f1a1234567893" },

"action\_type": "update",

"action\_date": { "$date": "2025-04-12T14:42:00Z" },

"ip\_address": "192.168.1.101",

"affected\_entity": "Student",

"entity\_id": { "$oid": "661ae05de13f1a1234567891" },

"description": "Изменена посещаемость по предмету ID 661ae1d9e13f1a12345678a0",

"role": "teacher"

}

**SubjectMeta**

{

"\_id": { "$oid": "661ae1d9e13f1a12345678a0" },

"subject\_name": "Введение в нереляционные базы данных",

"description": "Курс по основам работы с NoSQL базами данных",

"grade\_type": "exam",

"is\_activity": false

}

**Примеры запросов для нереляционной базы данных**

1. Получить данные по одному студенту
2. async def get\_student\_by\_user\_id(db, user\_id: str):
3. student = await db.Student.find\_one({ "user\_id": ObjectId(user\_id) })
4. return student

* Коллекции: Student
* Масштаб: O(1)

1. Получить полные данные по студенту вместе с User (логин, email, ФИО)
2. async def get\_full\_student\_info(db, user\_id: str):
3. pipeline = [
4. {
5. "$match": { "user\_id": ObjectId(user\_id) }
6. },
7. {
8. "$lookup": {
9. "from": "User",
10. "localField": "user\_id",
11. "foreignField": "\_id",
12. "as": "user"
13. }
14. },
15. {
16. "$unwind": "$user"
17. },
18. {
19. "$project": {
20. "\_id": 1,
21. "birth\_date": 1,
22. "admission\_year": 1,
23. "student\_type": 1,
24. "course": 1,
25. "program\_name": 1,
26. "faculty": 1,
27. "group\_name": 1,
28. "funding\_type": 1,
29. "statistic": 1,
30. "user.email": 1,
31. "user.first\_name": 1,
32. "user.middle\_name": 1,
33. "user.last\_name": 1
34. }
35. }
36. ]
37. cursor = db.Student.aggregate(pipeline)
38. return await cursor.to\_list(length=1)

Коллекции: Student, User

Использует: $lookup, $project

Масштаб: O(1)

Получить всех всех студентов постранично, отсортированных по exlusion\_probability по возростанию.

async def get\_students\_sorted(db, page: int = 0, page\_size: int = 20):

now = datetime.utcnow()

current\_year = now.year

cursor = db.Student.find({

"admission\_year": { "$lte": current\_year },

"course": { "$lt": 5 }, # фильтр: ещё учится

"statistic.exlusion\_probability": { "$ne": None }

}).sort(

"statistic.exlusion\_probability", 1

).skip(

page \* page\_size

).limit(

page\_size

)

return [doc async for doc in cursor]

* Коллекции: Student
* Использует: find, sort, skip, limit
* Масштаб: O(N), требует индекс

1. Преподаватель заходит и видит список студентов из своих групп,  
   с только теми предметами, которые он сам ведёт.
2. async def get\_students\_for\_teacher(db, teacher\_user\_id: str):
3. teacher = await db.Teacher.find\_one({ "user\_id": ObjectId(teacher\_user\_id) })
4. if not teacher:
5. return []
6. groups = teacher["assigned\_groups"]
7. subject\_ids = teacher["assigned\_subjects"]
8. pipeline = [
9. {
10. "$match": {
11. "group\_name": { "$in": groups }
12. }
13. },
14. {
15. "$project": {
16. "user\_id": 1,
17. "group\_name": 1,
18. "faculty": 1,
19. "program\_name": 1,
20. "statistic.average\_score": 1,
21. "statistic.attendance\_percent": 1,
22. "statistic.subjects": {
23. "$filter": {
24. "input": "$statistic.subjects",
25. "as": "subj",
26. "cond": { "$in": ["$$subj.subject\_id", subject\_ids] }
27. }
28. }
29. }
30. }
31. ]
32. cursor = db.Student.aggregate(pipeline)
33. return await cursor.to\_list(length=None)

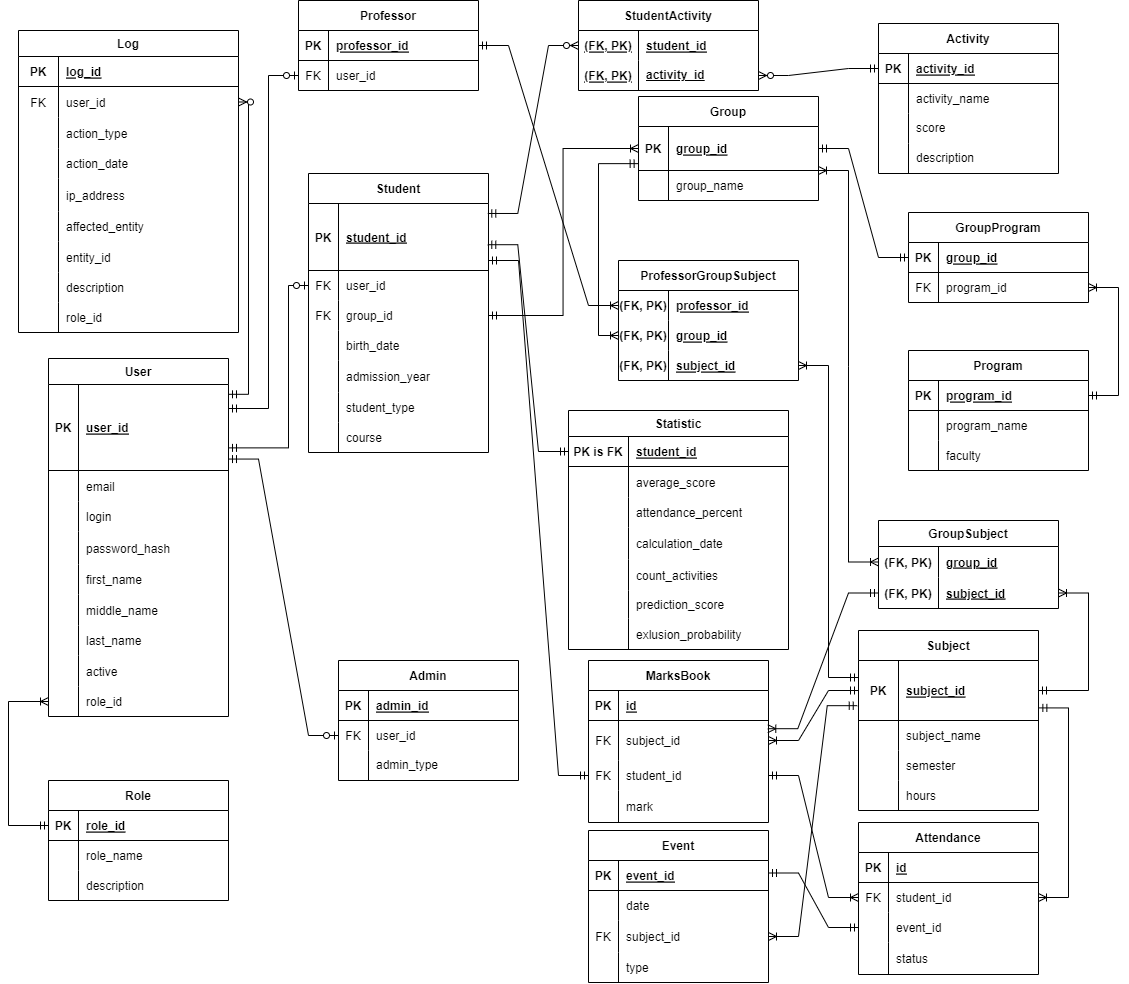
* Коллекции: Teacher, Student
* Использует: find, $filter, $project
* Масштаб: O(N/20), фильтрация по group\_name, subject\_id
  1. Фильтрация студентов по группе/факультету/типу финансирования
* async def filter\_students(db, group\_name=None, faculty=None, funding\_type=None):
* filters = {}
* if group\_name:
* filters["group\_name"] = group\_name
* if faculty:
* filters["faculty"] = faculty
* if funding\_type:
* filters["funding\_type"] = funding\_type
* cursor = db.Student.find(filters)
* return [doc async for doc in cursor]

Коллекции: Student

Использует: find, комбинированные фильтры

Масштаб: зависит от выборки

## Реляционная модель



1. ER-диаграмма реляционной модели

**Описание назначений таблиц, типов данных и сущностей**

**Таблица User**

Хранит информацию о пользователях.

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| user\_id | int | Уникальный ID (auto-increment) |
| email | string | Название электронной почты |
| login | string | Логин |
| password\_hash | string | Хеш пароля |
| first\_name | string | Имя |
| middle\_name | string | Отчество |
| last\_name | string | Фамилия |
| active | bool | Активная иили нет учётная запись |
| role\_id | int | Роль пользователя ("student" / "professor" / "admin") |

**Таблица Student**

Хранит информацию о студентах.

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| student\_id | int | Уникальный ID (auto-increment) |
| user\_id | int | Уникальный ID пользователя |
| group\_id | int | ID группы |
| birth\_date | datetime | Дата рождения |
| admission\_year | int | Год поступления |
| student\_type | string | Тип студента - ('bachelor'/'master'/'aspirant'/'specialist') |
| course | int | Номер курса |
| program\_name | string | Название программы обучения |

**Таблица Professor**

Хранит информацию о преподавателях.

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| professor\_id | int | ID преподавателя |
| user\_id | int | ID пользователя |
| position | string | Должность |

**Таблица Admin**

Хранит информацию об администраторах.

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| admin\_id | int | ID администратора |
| user\_id | int | ID пользователя |
| admin\_type | string | Тип администратора |

**Таблица Log**

Таблица логов.

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| log\_id | int | ID записи (auto-increment) |
| user\_id | int | Уникальный ID пользователя |
| action\_type | string | Тип действия ("elit" / "add" / "delete") |
| action\_date | datetime | Время действия |
| ip\_address | string | IP-адрес |
| affected\_entity | string | Сущность, которая редактировалась |
| entity\_id | int | ID сущности |
| description | string | Описание |
| role\_id | string | Роль пользователя ("student" / "professor" / "admin") |

**Таблица Group**

Таблица с распределением студентов по группам.

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| group\_id | int | Уникальный ID группы |
| group\_name | string | Номер группы |

**Таблица Subject**

Таблица с информацией о предмете.

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| subject\_id | int | Уникальный ID (auto-increment) |
| subject\_name | string | Название предмета |
| semester | int | Номер семестра |
| hours | int | Количество часов |

**Таблица Program**

Таблица программ обучения.

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| program\_id | int | ID учебной программы |
| program\_name | string | Название программы |
| faculty | categorial | Название факультета |

**Таблица Role**

Хранит список ролей пользователей (admin, student, professor и т.д.)

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| role\_id | int | ID роли |
| role\_name | string | Название роли |
| description | string | Описание |

**Таблица GroupProgram**

Таблица соответствия между группами и программами.

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| group\_id | int | ID учебной группы |
| program\_id | int | ID учебной программы |

**Таблица GroupSubject**

Учебный план, у каких групп какие предметы.

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| group\_id | int | ID группы |
| subject\_id | int | ID предмета |

**Таблица Activity**

Таблица с инфориацией об активностях, проводиых в ВУЗе.

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| activity\_id | int | ID активности |
| activity\_name | string | Название активности |
| score | int | Количество баллов, в которое можно оценить активность |
| description | string | Описание активности |

**Таблица StudentActivity**

Информация о том, какие студенты занимаются какими активностями.

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| student\_id | int | ID студента |
| activity\_id | int | ID активности |

**Таблица MarksBook**

Зачетная книжка студента.

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| id | int | ID записи |
| subject\_id | int | ID предмета |
| student\_id | int | ID студента |
| mark | int | Оценка за предмет |

**Таблица Statistic**

Хранит сжатую статистику о студенте.

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| student\_id | int | Уникальный ID (auto-increment) |
| average\_score | decimal | Средний балл |
| attendance\_percent | decimal | Процент посещаемости |
| calculation\_date | datetime | Дата вычисления статистики |
| count\_activities | int | Количество активностей |
| prediction\_score | decimal | Предсказанный средний балл |
| exlusion\_probability | decimal | Вероятность отчисления |

**Таблица Event**

Хранит информацию о событии.

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| event\_id | int | ID записи |
| subject\_id | int | ID предмета |
| date | datetime | Время проведения занятия |
| type | string | Тип пары ("lecture"/"practice") |

**Таблица Attendance**

Хранит посещаемость лекций и практик студентами.

| **Поле** | **Тип** | **Описание** |
| --- | --- | --- |
| id | int | ID записи |
| student\_id | int | ID студента |
| event\_id | int | ID события |
| status | bool | 1 - пара посещена, 0 - не посещена |

**Примеры данных для реляционной базы данных**

**Таблица User**

| **user\_id** | **email** | **login** | **password\_hash** | **first\_name** | **middle\_name** | **last\_name** | **active** | **role\_id** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0001 | vasiliev101@mail.ru | hagr89 | fdh647h9 | Andrew | Andreevich | Vasiliev | 0 | student |
| 0002 | ivanovstep@mail.ru | user1 | 23746dy5 | Stepan | Alekseevich | Ivanov | 0 | student |

**Таблица Student**

| **student\_id** | **user\_id** | **group\_id** | **birth\_date** | **admission\_year** | **student\_type** | **course** | **program\_name** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0001 | 0001 | 2323 | 2004-04-02 | 2022 | bachelor | 3 | Theoretical math |
| 0002 | 0002 | 2323 | 2004-09-30 | 2022 | bachelor | 3 | Theoretical math |

**Таблица Professor**

| **professor\_id** | **user\_id** | **position** |
| --- | --- | --- |
| 0001 | 0003 | dean |
| 0002 | 0004 | professor |

**Таблица Admin**

| **admin\_id** | **user\_id** | **admin\_type** |
| --- | --- | --- |
| 0001 | 0005 | head\_admin |
| 0002 | 0006 | admin |

**Таблица Log**

| **log\_id** | **user\_id** | **action\_type** | **action\_date** | **ip\_address** | **affected\_entity** | **entity\_id** | **description** | **role\_id** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0001 | 0005 | add | 2025-02-15 15:36:45 | 192.168.0.103 | "Event" | 0001 | Добавление 2 строк | 0001 |

**Таблица Subject**

| **subject\_id** | **subject\_name** | **semester** | **hours** |
| --- | --- | --- | --- |
| 0001 | History | 2 | 64 |
| 0002 | Math statistic | 4 | 64 |
| 0003 | English | 1 | 58 |
| 0004 | Functional Analysis | 6 | 64 |

**Таблица MarksBook**

| **id** | **subject\_id** | **student\_id** | **mark** |
| --- | --- | --- | --- |
| 0001 | 0008 | 0007 | 5 |
| 0002 | 0005 | 0007 | 3 |
| 0003 | 0019 | 0007 | 4 |

**Таблица Attendance**

| **id** | **student\_id** | **event\_id** | **status** |
| --- | --- | --- | --- |
| 0001 | 0001 | 0534 | 1 |
| 0002 | 0001 | 0635 | 0 |

**Таблица Group**

| **group\_id** | **group\_name** |
| --- | --- |
| 2381 | 2381 |

**Таблица Activity**

| **activity\_id** | **activity\_name** | **score** | **description** |
| --- | --- | --- | --- |
| 0001 | math olympiad | 1 | Winter math olympiad |

**Таблица Event**

| **event\_id** | **subject\_id** | **date** | **type** |
| --- | --- | --- | --- |
| 0001 | 0004 | 2025-04-21 | lecture |
| 0002 | 0004 | 2025-04-22 | practice |

**Таблица Role**

| **role\_id** | **role\_name** | **description** |
| --- | --- | --- |
| 0001 | head\_admin | Главный администратор |

**Таблица ProfessorGroupSubject**

| **professor\_id** | **group\_id** | **subject\_id** |
| --- | --- | --- |
| 0001 | 2387 | 0006 |

**Таблица Program**

| **program\_id** | **program\_name** | **faculty** |
| --- | --- | --- |
| 0011 | Program engineering | FCTI |

**Таблица GroupProgram**

| **group\_id** | **program\_id** |
| --- | --- |
| 2304 | 0011 |
| 2305 | 0011 |
| 2381 | 0012 |

**Таблица GroupSubject**

| **group\_id** | **subject\_id** |
| --- | --- |
| 2300 | 0075 |
| 2304 | 0045 |
| 2305 | 0023 |

**Таблица StudentActivity**

| **student\_id** | **activity\_id** |
| --- | --- |
| 0002 | 0001 |

**Примеры запросов для реляционной базы данных**

1. Получить данные по одному студенту
2. SELECT
3. s.\*,
4. stat.average\_score, stat.attendance\_percent, stat.calculation\_date,
5. stat.count\_activities, stat.prediction\_score, stat.exlusion\_probability,
6. sbj.subject\_id, sbj.subject\_name, sbj.semester, sbj.hours,
7. mb.mark
8. FROM Student s
9. JOIN Statistic stat ON stat.student\_id = s.student\_id
10. LEFT JOIN MarksBook mb ON mb.student\_id = s.student\_id
11. LEFT JOIN Subject sbj ON sbj.subject\_id = mb.subject\_id
12. WHERE s.user\_id = $1;

* Таблицы: Student, Statistic, MarksBook, Subject
* Масштаб: **O(1)** по Student, **O(k)** по количеству предметов  
  (где k — предметы конкретного студента, в среднем ≈ 50)

1. Получить полные данные по студенту вместе с User (логин, email, ФИО)

SELECT

s.\*,

u.email, u.first\_name, u.middle\_name, u.last\_name,

stat.average\_score, stat.attendance\_percent, stat.calculation\_date,

stat.count\_activities, stat.prediction\_score, stat.exlusion\_probability,

sbj.subject\_id, sbj.subject\_name, sbj.semester, sbj.hours,

mb.mark

FROM Student s

JOIN "User" u ON u.user\_id = s.user\_id

LEFT JOIN Statistic stat ON stat.student\_id = s.student\_id

LEFT JOIN MarksBook mb ON mb.student\_id = s.student\_id

LEFT JOIN Subject sbj ON sbj.subject\_id = mb.subject\_id

WHERE s.user\_id = $1;

* Таблицы: Student, User, Statistic, MarksBook, Subject
* Масштаб: **O(1)** по Student, **O(k)** по количеству предметов  
  (где k — предметы конкретного студента, в среднем ≈ 50)

1. Получить всех всех студентов постранично, отсортированных по exlusion\_probability по возростанию.

SELECT

s.student\_id,

s.user\_id,

s.group\_id,

s.course,

s.program\_name,

stat.average\_score,

stat.exlusion\_probability

FROM Student s

LEFT JOIN Statistic stat ON stat.student\_id = s.student\_id

ORDER BY stat.exlusion\_probability ASC

LIMIT $1 OFFSET $2;

* Таблицы: Student, Statistic
* Масштаб: **O(N)** (где N — общее количество студентов, зависит от LIMIT + OFFSET)

1. Преподаватель заходит и видит список студентов из своих групп,  
   с только теми предметами, которые он сам ведёт.

SELECT

s.student\_id,

s.group\_id,

s.course,

s.program\_name,

u.first\_name,

u.last\_name,

subj.subject\_id,

subj.subject\_name,

mb.mark

FROM ProfessorGroupSubject pgs

JOIN Student s ON s.group\_id = pgs.group\_id

JOIN "User" u ON u.user\_id = s.user\_id

JOIN MarksBook mb ON mb.student\_id = s.student\_id AND mb.subject\_id = pgs.subject\_id

JOIN Subject subj ON subj.subject\_id = mb.subject\_id

WHERE pgs.professor\_id = $1

ORDER BY s.group\_id, subj.subject\_name;

* Таблицы: ProfessorGroupSubject, Student, User, MarksBook, Subject
* Масштаб: **O(k × m)**, где k — студентов в группах, m — предметов от преподавателя

1. Фильтрация студентов по группе/факультету/типу финансирования

SELECT

s.student\_id,

s.group\_id,

g.group\_name,

s.program\_name,

p.faculty,

s.course

FROM Student s

JOIN "Group" g ON g.group\_id = s.group\_id

JOIN GroupProgram gp ON gp.group\_id = g.group\_id

JOIN Program p ON p.program\_id = gp.program\_id

WHERE

($1 IS NULL OR g.group\_name = $1)

AND ($2 IS NULL OR p.faculty = $2)

AND ($3 IS NULL OR s.student\_type = $3);

* Таблицы: Student, Group, GroupProgram, Program
* Масштаб: **O(N)** (зависит от количества студентов)
* Поддержка фильтрации по всем 3 параметрам (каждый — необязателен)

**Оценка объема информации, хранимой в модели SQL**

**Общие параметры для SQL**

Для оценки объёма данных примем:

* **N** — количество студентов
* **U = N** — количество пользователей (1:1 со студентами)
* **T = N / 20** — количество преподавателей
* **L = 5 × N** — количество логов
* **S = 150** — количество предметов
* **M = 50** — среднее число записей в MarksBook на студента
* **A = 10** — количество активностей на студента
* **E = 200** — общее число событий (Event)
* **G = N / 25** — количество групп
* **P = 20** — количество программ
* \*\*PGS = T × 2\*\* — записи в ProfessorGroupSubject`

**Размер объектов реляционной базы данных**

| **Таблица** | **Размер одного экземпляра** | **Кол-во** | **Формула размера** |
| --- | --- | --- | --- |
| **User** | ~150 байт | N | 150 × N |
| **Student** | ~974 байта | N | 974 × N |
| **Professor** | ~14 байт | N / 20 | 0.7 × N |
| **Log** | ~36 байт × 5 | 5 × N | 180 × N |
| **Subject** | ~33 байта | 150 | 4 950 (константа) |
| **Program** | ~40 байт | 20 | 800 (константа) |
| **MarksBook** | ~8 байт | 50 × N | 400 × N |
| **Statistic** | ~60 байт | N | 60 × N |
| **Activity** | ~50 байт | 10 × N | 500 × N |
| **StudentActivity** | ~8 байт | 10 × N | 80 × N |
| **Group** | ~20 байт | N / 25 | 0.8 × N |
| **GroupProgram** | ~8 байт | N / 25 | 0.8 × N |
| **GroupSubject** | ~8 байт | 150 | 1 200 (константа) |
| **Event** | ~16 байт | 200 | 3 200 (константа) |
| **Attendance** | ~8 байт | 10 × N | 80 × N |
| **ProfessorGroupSubject** | ~12 байт | 2 × T | 1.2 × N |

**Общая формула объема в байтах**

Общий\_объем(N) =

150 × N // User

+ 974 × N // Student

+ 0.7 × N // Professor

+ 180 × N // Log

+ 400 × N // MarksBook

+ 60 × N // Statistic

+ 500 × N // Activity

+ 80 × N // StudentActivity

+ 80 × N // Attendance

+ 1.2 × N // ProfessorGroupSubject

+ 0.8 × N // Group

+ 0.8 × N // GroupProgram

+ 4 950 // Subject

+ 800 // Program

+ 1 200 // GroupSubject

+ 3 200 // Event

= 2 487.5 × N + 10 150 байт

**Примеры**

| **Кол-во студентов (N)** | **Общий объем (байт)** | **Примерно в МБ** |
| --- | --- | --- |
| 1 000 | 2 497 650 | ~2.4 МБ |
| 10 000 | 24 885 150 | ~23.7 МБ |
| 100 000 | 248 760 150 | ~237.2 МБ |

**Избыточность данных SQL**

**Чистый объём данных SQL**

**User**

* login (string) — **~15 байт**
* password\_hash (string) — **~60 байт**
* email (string) — **~30 байт**
* first\_name (string) — **~15 байт**
* middle\_name (string) — **~15 байт**
* last\_name (string) — **~15 байт**

Итого: **~150 байт на пользователя**

**Student**

* user\_id (int, FK) — **4 байта**
* birth\_date (date) — **8 байт**
* admission\_year (int) — **4 байта**
* student\_type (string, enum) — **~10 байт**
* course (int) — **4 байта**
* program\_name (string) — **~20 байт**
* group\_id (int) — **4 байта**
* statistic — **60 байт**
  + average\_score (decimal) — **16 байт**
  + attendance\_percent (decimal) — **16 байт**
  + calculation\_date (date) — **8 байт**
  + count\_activities (int) — **4 байта**
  + exlusion\_probability (decimal) — **16 байт**
* marksbook (50 предметов) — **50 × 4 (subject\_id) + 4 (mark)** ≈ **~400 байт**

Итого: **~514 байт + 60 + 400 = ~974 байта на студента**

**Professor**

* user\_id (int, FK) — **4 байта**
* position (string) — **~10 байт**

Итого: **~14 байт на преподавателя**  
→ В среднем 1 преподаватель на 20 студентов → 14 × N / 20 = 0.7 × N

**Log**

* user\_id (int) — **4 байта**
* action\_type (string) — **~10 байт**
* action\_date (datetime) — **8 байт**
* affected\_entity (string) — **~10 байт**
* entity\_id (int) — **4 байта**

Итого: **~36 байт × 5 записей = 180 байт / студент**

**Subject**

* subject\_name (string) — **~25 байт**
* semester (int) — **4 байта**
* hours (int) — **4 байта**

Итого: **~33 байта × 150 = 4 950 байт (константа)**

**Program**

* program\_name (string) — **~25 байт**
* faculty (string) — **~15 байт**

Итого: **~40 байт × ~20 программ = ~800 байт** → можно считать **константой**

**Размер объектов SQL**

| **Объект** | **Размер одного экземпляра** | **Кол-во** | **Формула размера** |
| --- | --- | --- | --- |
| **User** | ~150 байт | = N | 150 × N |
| **Student** | ~974 байт | = N | 974 × N |
| **Professor** | ~14 байт | = N / 20 | 0.7 × N |
| **Log** | ~180 байт | = 5 × N | 180 × N |
| **Subject** | ~33 байта | = 150 | 4 950 (константа) |
| **Program** | ~40 байт | = ~20 | 800 (константа) |

**Общая формула чистого объёма sql**

Чистый\_объем(N) =

150 × N // User

+ 974 × N // Student

+ 0.7 × N // Professor

+ 180 × N // Log

+ 4 950 // Subject

+ 800 // Program

Или:

Чистый\_объем(N) = 1 304.7 × N + 5 750 байт

**Избыточность реляционной модели**

Избыточность = Общий\_объем / Чистый\_объем

Подставим формулы:

Общий\_объем(N) = 2 487.5 × N + 10 150

Чистый\_объем(N) = 1 304.7 × N + 5 750

Избыточность = (2 487.5 × N + 10 150) / (1 304.7 × N + 5 750)

**Примеры избыточности для разных значений N реляционной БД**

| **Кол-во студентов (N)** | **Избыточность** |
| --- | --- |
| 1 000 | **1.91** |
| 10 000 | **1.91** |
| 100 000 | **1.91** |
| NtoinftyNtoinfty | **1.91** |

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

Реляционная модель масштабируется линейно по количеству студентов N, поскольку практически все основные сущности (такие как User, Student, Statistic, MarksBook, Log, StudentActivity, Attendance) прямо или косвенно связаны со студентом.

**Сравнение моделей**

**Удельный объём информации**

| **Параметр** | **NoSQL** | **SQL** |
| --- | --- | --- |
| Формула объёма | 2 566.8 × N + 24 000 байт | 2 487.5 × N + 10 150 байт |
| Объём при N = 10 000 | 25 692 000 байт (~24.5 МБ) | 24 885 150 байт (~23.7 МБ) |
| Избыточность | ~2.32 | ~1.91 |

**Запросы по отдельным юзкейсам**

| **Юзкейс** | **NoSQL** | **SQL** |
| --- | --- | --- |
| **1. Получить одного студента** | 1 find\_one(Student) Коллекции: 1 **Сложность:** O(1) | 1 SELECT с JOIN (Statistic, MarksBook, Subject) Таблицы: 4 **Сложность:** O(k) |
| **2. Студент + User** | 1 aggregate с $lookup (User) Коллекции: 2 **Сложность:** O(1) | 1 SELECT с JOIN (User, Statistic, MarksBook, Subject) Таблицы: 5 **Сложность:** O(k) |
| **3. Студенты постранично по exlusion\_probability** | 1 find().sort().skip().limit() Коллекции: 1 **Сложность:** O(log N + P) *(при индексе)* | 1 SELECT + JOIN(Statistic) + ORDER BY + LIMIT/OFFSET Таблицы: 2 **Сложность:** O(log N + P) |
| **4. Преподаватель → студенты по группам и предметам** | 1 find\_one(Teacher) + 1 aggregate(Student) с $filter Коллекции: 2 **Сложность:** O(N/20 × m) | 1 SELECT с JOIN (ProfessorGroupSubject, Student, User, MarksBook, Subject) Таблицы: 5 **Сложность:** O(k × m) |
| **5. Фильтрация по группе, факультету, финансированию** | 1 find(Student) с комбинированными фильтрами Коллекции: 1 **Сложность:** O(fN) *(зависит от фильтров)* | 1 SELECT с JOIN (Group, GroupProgram, Program) Таблицы: 4 **Сложность:** O(fN) |

**Примечания:**

* k — количество предметов у одного студента (≈ 50)
* m — количество предметов, читаемых преподавателем
* P — размер страницы (limit)
* fN — фильтрованное подмножество студентов

NoSQL использует меньшее количество коллекций, особенно если данные вложены. Подходит для прямого доступа, но сложные выборки требуют aggregate.

SQL объединяет все данные в одном SELECT с JOIN, что позволяет выразительно формулировать сложные запросы, особенно при работе с аналитикой.

**Вывод**

Обе модели — SQL и NoSQL — демонстрируют линейный рост объёма при увеличении числа студентов и справляются с базовыми операциями за 1–2 запроса, но SQL обеспечивает большую выразительность в аналитических сценариях благодаря JOIN-запросам, тогда как NoSQL проще и эффективнее при работе с вложенными структурами. SQL-модель оказывается чуть менее избыточной (~1.91 против ~2.32), а разница в реальном объёме хранения при масштабе до 100 000 студентов не превышает 5–10%, что делает выбор модели скорее вопросом архитектурных предпочтений и типов запросов.

# Разработанное приложение. Прототип

**Задачи, решённые в MVP**

* разработать схему NoSQL‑БД и минимальный набор коллекций;
* реализовать CRUD‑эндпоинты для пользователей, студентов, преподавателей и логов;
* обеспечить фильтрацию и пагинацию на стороне сервера;
* создать интерфейсы создания, просмотра и редактирования записей;
* автоматизировать развёртывание в Docker‑среде.

## Архитектура решения

**2.1 Диаграмма компонентов**

React‑Frontend → FastAPI‑Backend → MongoDB (replica‑set)

1. **Слои и их ответственность**

|  |  |
| --- | --- |
| **Слой** | **Ответственность** |
| **Frontend (UI)** | React‑компоненты, маршрутизация (React Router v6), формы, таблицы, базовая валидация. |
| **FastAPI‑Backend** | Роутинг HTTP, JWT‑аутентификация (сквозной), бизнес‑логика и валидация через Pydantic, доступ к Mongo. |
| **Repository layer** | Инкапсулированный доступ к MongoDB (motor), CRUD, агрегирующие пайплайны. |
| **Database (MongoDB)** | Документы пользователей, студентов, преподавателей, логов; индексы на email, user\_id, датах логов. |
| **Infrastructure** | Docker Compose (dev/ prod), кастомный образ Mongo с реплика‑сетом, Nginx для статики фронта. |

## Реализация back‑end

* **Стек**: Python 3.11, FastAPI 0.110, Motor 3.x, Pydantic v2.
* **Дерево кода**
* backend/src/
* ├── main.py # точка входа FastAPI
* ├── handlers/ # роутеры: users, students, teachers, logs
* ├── repositories/ # UserRepository, TeacherRepository …
* ├── models/ # Pydantic‑отражения коллекций
* ├── schemas/ # DTO для входных/выходных данных
* ├── db/ # init replica‑set, dependency get\_db

└── utils/security.py # bcrypt‑хеширование, JWT‑токены

* **Реализованные эндпоинты**

|  |  |  |
| --- | --- | --- |
| Метод | Путь | Описание |
| GET | /users | Список пользователей с фильтрами и пагинацией |
| POST | /users | Создание пользователя (admin‑only) |
| PATCH | /users/{id} | Изменение полей пользователя |
| GET | /students | Выборка студентов по группе/факультету |
| … |  | (аналогично teachers, logs) |

* **Особенности**
  + Кастомный тип PyObjectID – единообразная конвертация строковых id.
  + Разграничение ролей (admin, teacher, student) через зависимости FastAPI.
  + Exports CSV: эндпоинт формирует text/csv из текущей выборки.

## Реализация front‑end

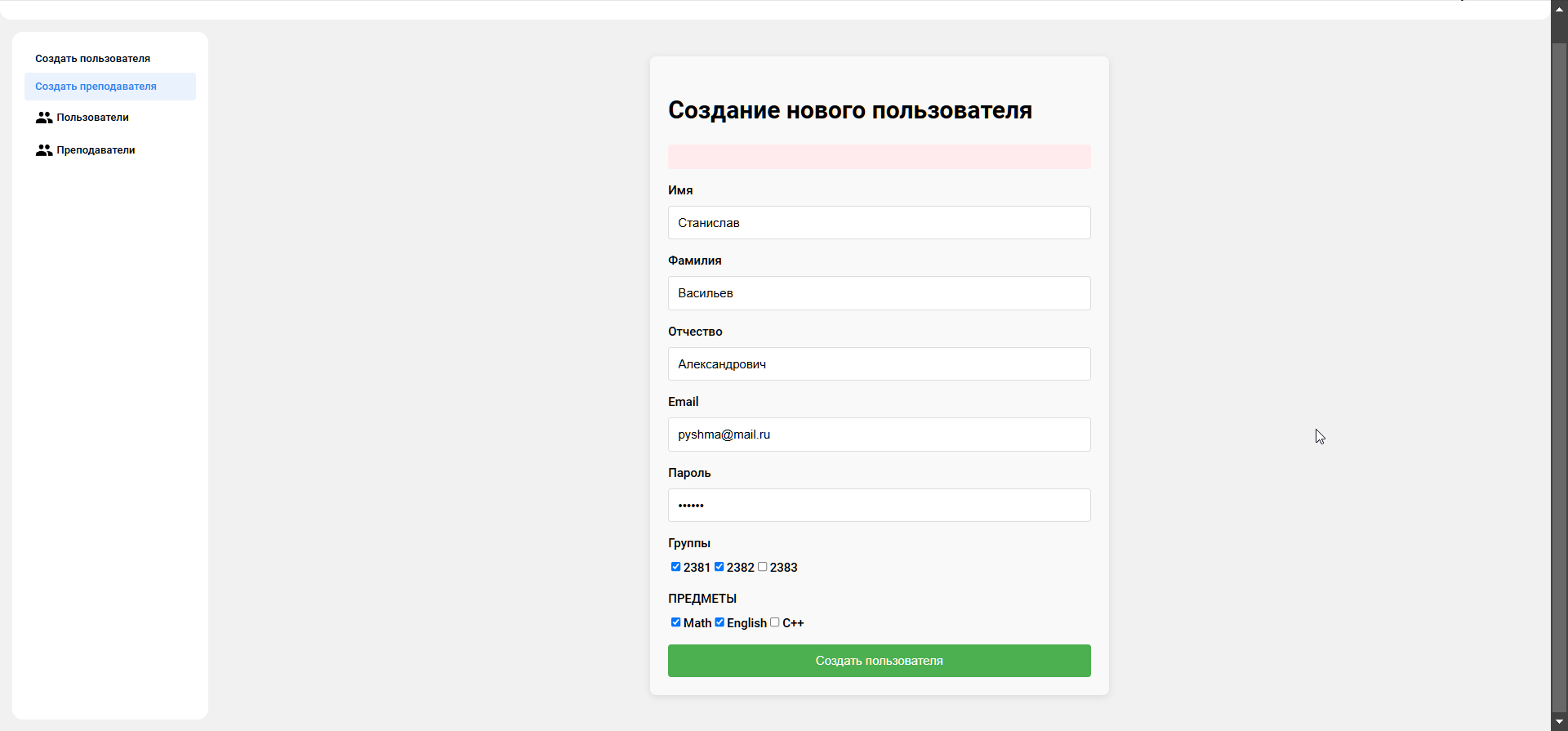
* **Стек**: React 18, Vite 5, CSS Modules.
* **Страницы**: CreateUser, CreateTeacher, UpdateUser, Profile, StatsMain (таблица с фильтрами), TeacherList.
* **Hooks**: useUsers, useTeachers, useStudents – инкапсулируют fetch, состояние loading / error и дебаунс фильтров.
* **Функционал**
  + Таблица с серверной пагинацией, сортировкой и checkbox‑фильтрами.
  + Формы создания/обновления с инлайновой валидацией.

## Инфраструктура

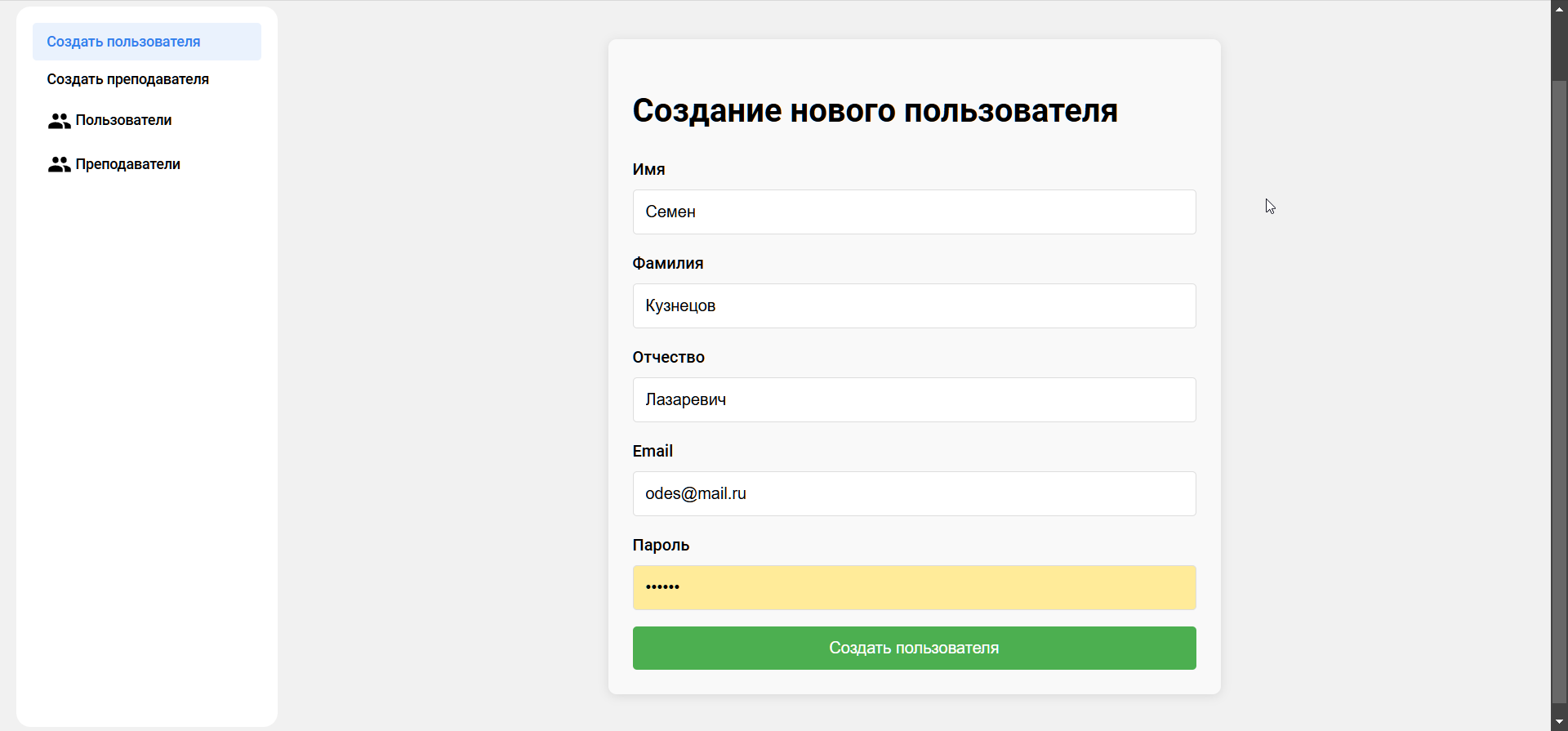
* **Docker Compose**: сервисы frontend, backend, mongo-rs, nginx.
* **Mongo образ**: Dockerfile + docker-entrypoint.sh, который — если кластер ещё не инициализирован — создаёт реплика‑сет и пользователя root.
* **Развёртывание**: достаточно docker compose up -d с предварительно заполненным .env (пример в репозитории).
* **CI/CD**: на момент защиты отсутствует; сборка образов выполняется вручную.

## Внешний вид системы

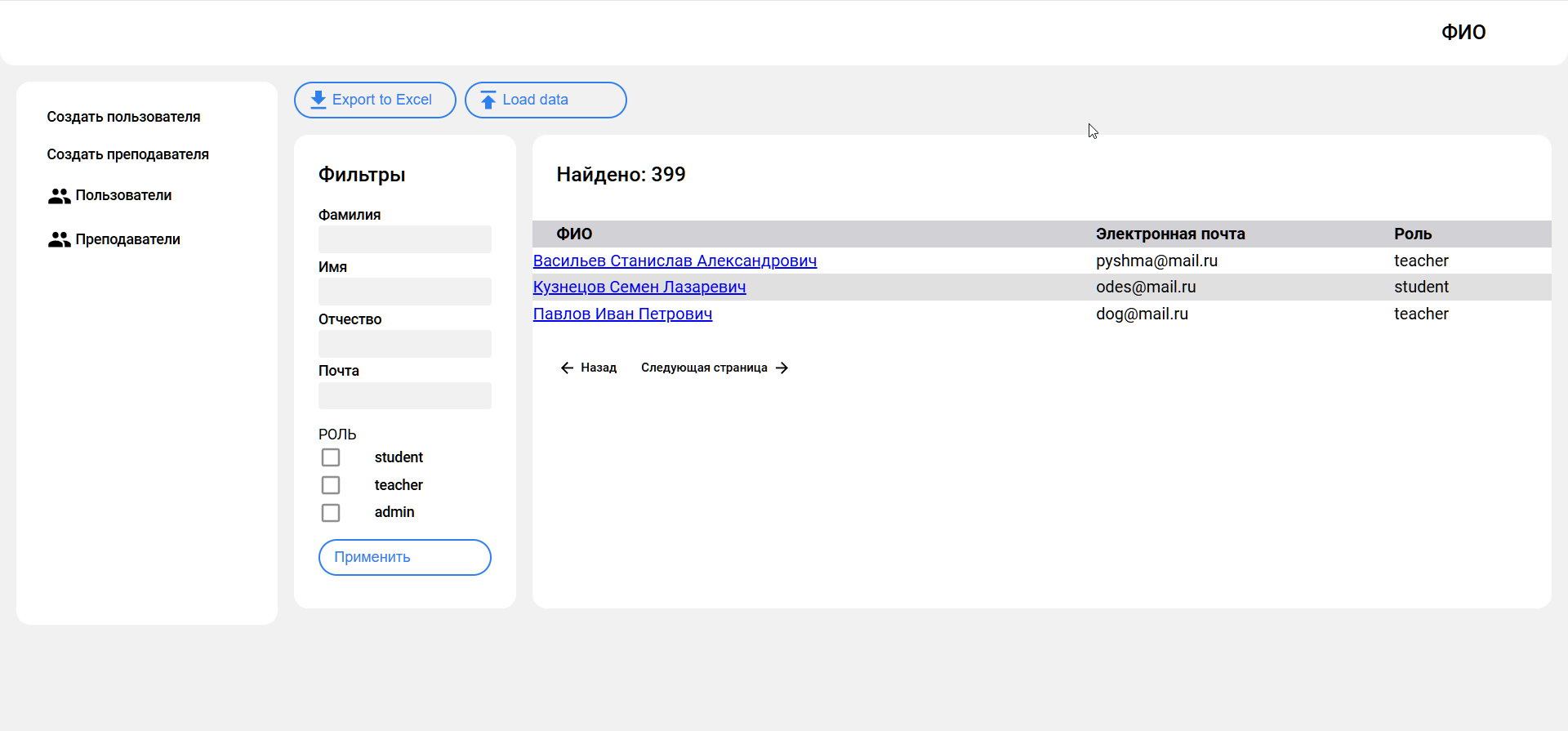
Ключевые элементы интерфейса: боковое меню, таблица пользователей с фильтрами слева, формы создания/редактирования, кнопки экспорта и обновления данных.



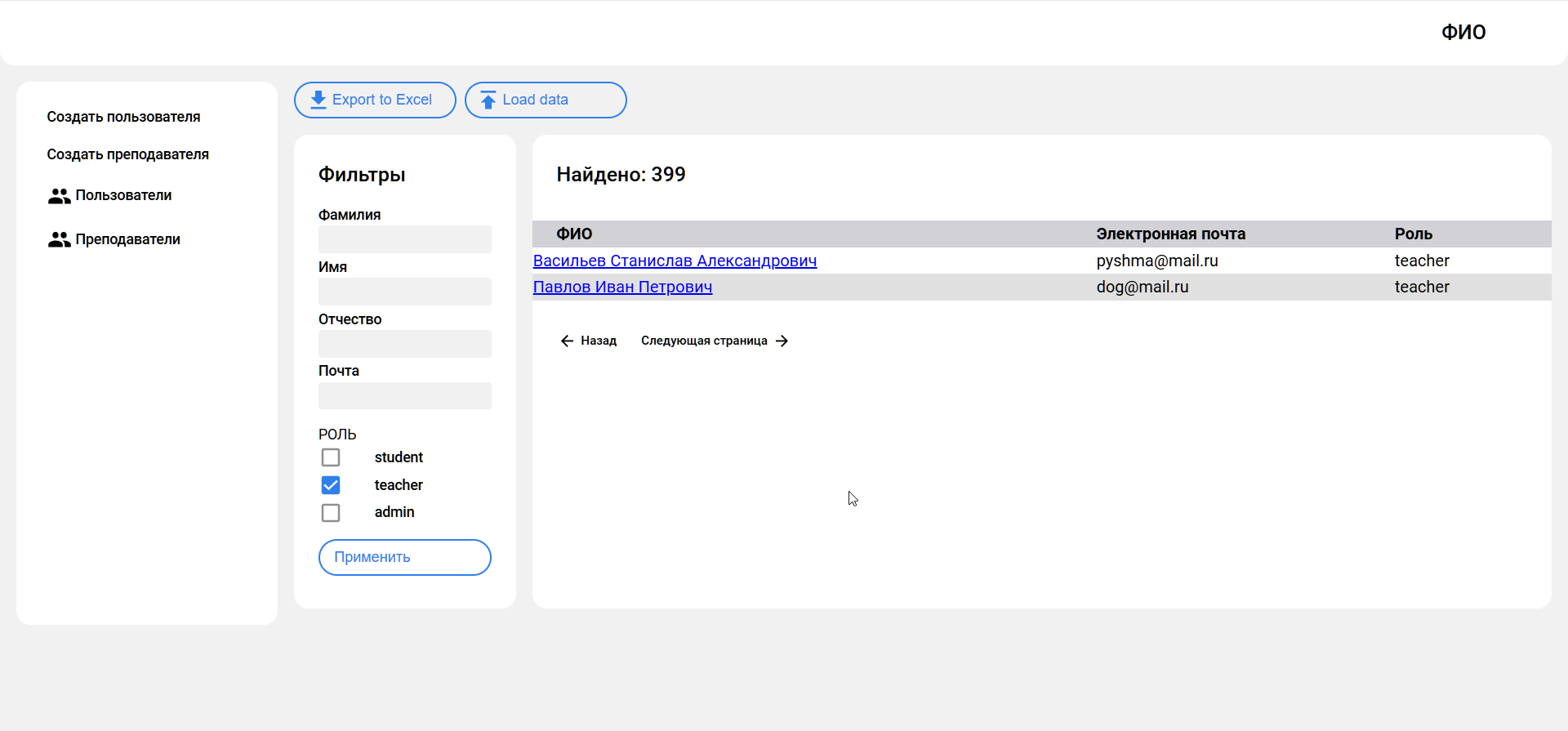
1. Добавление нового преподавателя



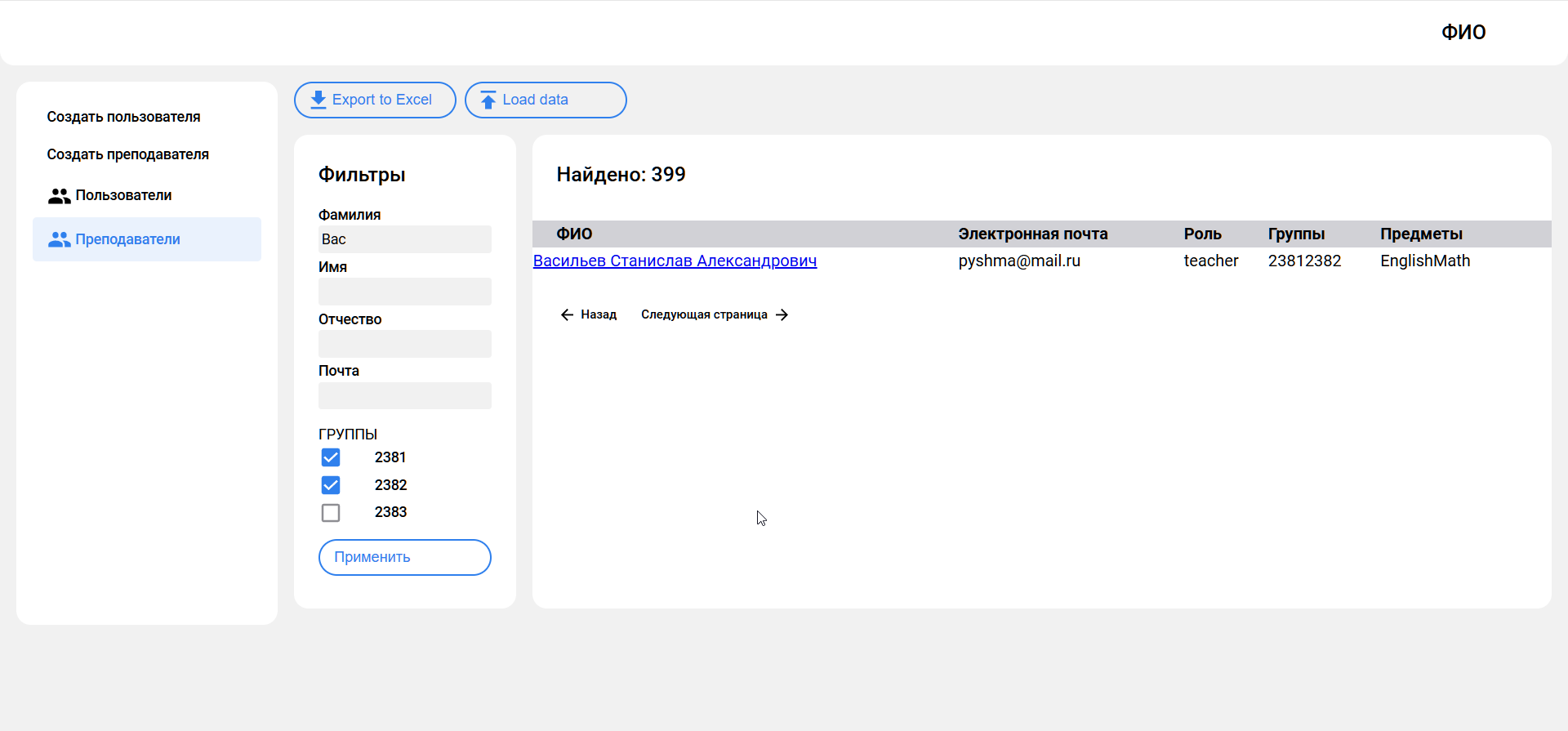
1. Создание нового пользователя



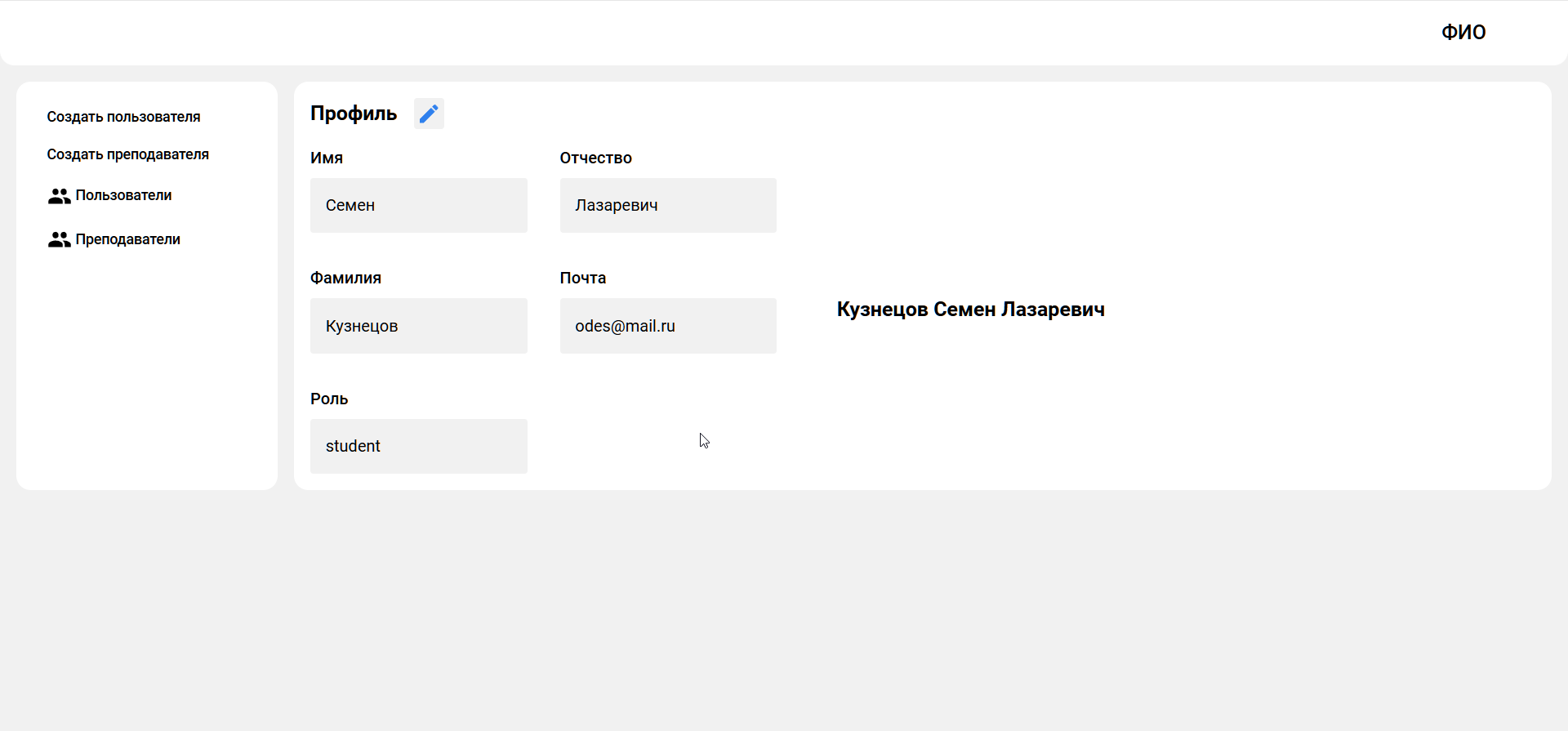
1. Список пользователй



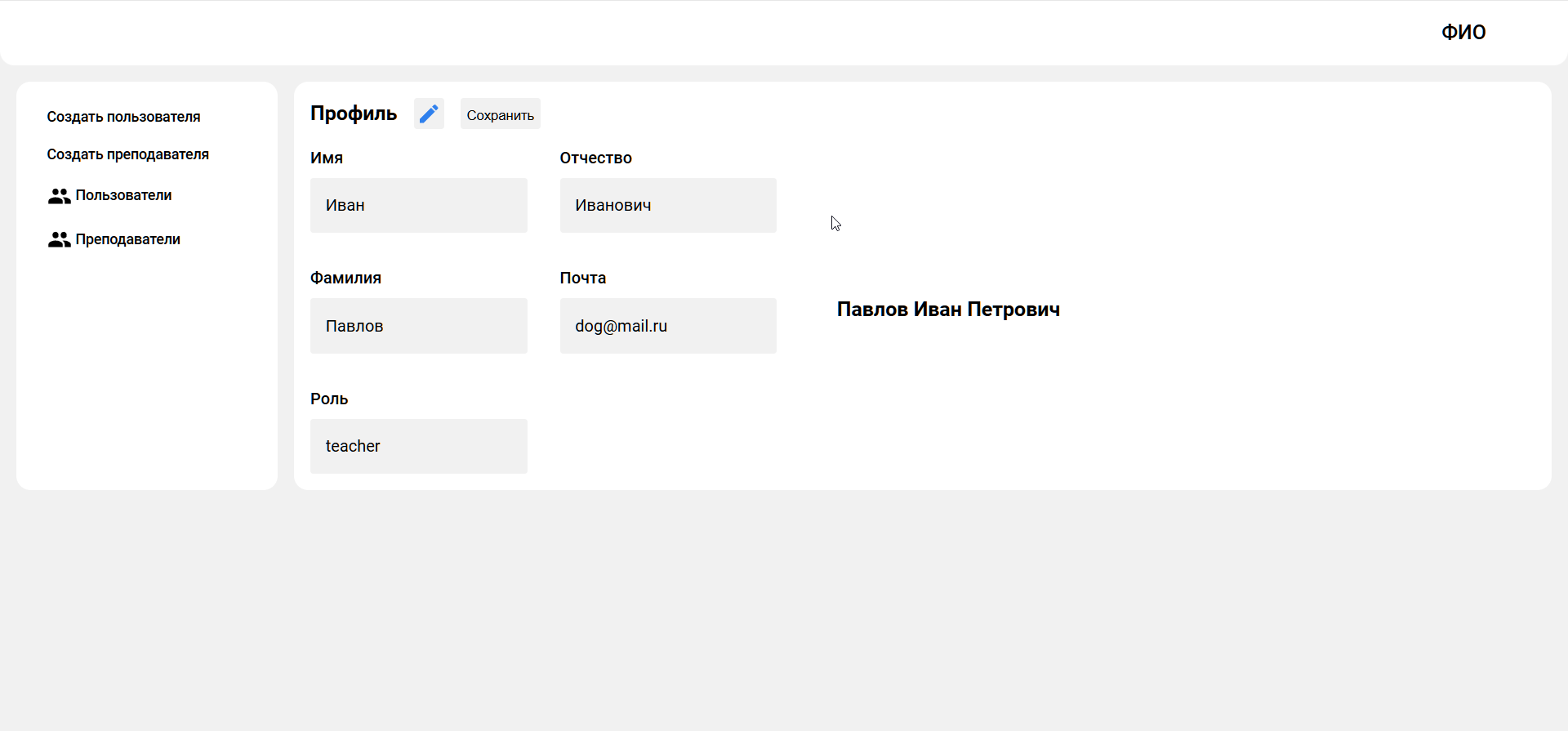
1. Список пользователей. Фильтрация



1. Список преподавателей. Фильтрация



1. Страница пользователя



1. Страница пользователя. Редактирование

**Выводы и планы развития**

Создан работоспособный MVP, позволяющий:

* создавать и редактировать записи пользователей, студентов, преподавателей;
* просматривать списки с фильтрацией и экспортом;
* логировать действия в коллекции Log.

Планируется:

1. Добавить массовый импорт CSV/Excel и bulk‑update.
2. Внедрить модуль аналитики (прогноз риска отчисления).
3. Покрыть критические участки кода тестами и настроить CI в GitHub Actions.
4. Улучшить UX таблиц (индикаторы пагинации, пустой результат).

# Выводы

Большая часть поставленных задач реализована: спроектирована NoSQL-модель хранения данных (MongoDB), разработаны основные интерфейсы для работы с пользователями, студентами, преподавателями и предметами. В систему заложены возможности для дальнейшего расширения и аналитической обработки данных, включая прогнозирование успеваемости. Сервис частично реализует заявленную цель, демонстрируя работоспособную архитектуру и потенциал для дальнейшего развития.