МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**

**(национальный исследовательский университет)**

Институт № 7 «Интеллектуальные и робототехнические системы»

Кафедра 703

**Курсовая работа**

**по курсу «Технологии сетевого взаимодействия»**

**По теме: “Клиент-Серверное приложение”**

**Вариант 5**

Работу выполнил:

Cтудент группы М7О-506C-20

Ибрагимов Тимур Румильевич\_\_\_\_\_\_\_\_\_\_\_\_

Работу проверил:

Старший преподаватель кафедры 703

Барчев Николай Борисович\_\_\_\_\_\_\_\_\_\_\_\_

Дата сдачи отчета: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Москва**

**2024**

Оглавление

[**Задание** 3](#_Toc185961129)

[**Псевдокод** 4](#_Toc185961130)

[**Сведения о программной реализации** 8](#_Toc185961131)

[**Инструкция пользователя** 13](#_Toc185961132)

[**Тестирование** 14](#_Toc185961133)

[**Коды состояния ответа HTTP** 14](#_Toc185961134)

[**Тестирование через Swagger Fast Api:** 14](#_Toc185961135)

[**Листсинг** 32](#_Toc185961136)

[**main.py** 32](#_Toc185961137)

[**app.py** 32](#_Toc185961138)

[**database.py** 33](#_Toc185961139)

[**student.py** 34](#_Toc185961140)

[**student.py** 37](#_Toc185961141)

[**get\_pwd\_name.html** 38](#_Toc185961142)

[**std\_add.html** 39](#_Toc185961143)

[**std\_del.html** 41](#_Toc185961144)

[**pwd\_add.html** 42](#_Toc185961145)

[**pwd\_del.html** 44](#_Toc185961146)

[**main.html** 45](#_Toc185961147)

# **Задание**

1. Подготовить серверное приложение с API, реализующим следующие методы взаимодействия с клиентами, содержащих:

* сведения о студентах: состоит из атрибутов, каждая из которых включает три поля – фамилии, группы и номера по списку;
* сведения о паролях для работы на ЭВМ: состоит из атрибутов, каждая из которых включает три поля – группы, номера по списку и пароля.

2. В процессе проектирования предусмотреть необходимые по смыслу задания проверки корректности данных, а также адекватное задаче взаимодействие с пользователем.

**В обязательном порядке использовать языковые средства организации программных единиц.**

3. Программы подготовить с использованием средств одной из реализаций языков программирования **C++ или Python.**

# **Псевдокод**

* **Выдача страниц сайта**

**Начало**

Вернуть HTML-шаблон запрошенной страницы

**Конец**

* **Добавление новой записи студента**

**Начало**

Получить данные из запроса: «Имя», «Номер группы», «Номер по списку»

Запросить из базы запись о студенте в данной группе с таким же номером

Если запись существует

Выдать ошибку, перезапись невозможна

Иначе если имя состоит из букв

Если номер по списку – натуральное число

Добавить запись в базу

Вернуть сообщение о добавлении записи

Иначе

Выдать ошибку номера по списку

Иначе

Выдать ошибку имени

**Конец**

* **Удаление записи о студенте**

**Начало**

Получить данные о номере группы и номере по списку из запроса

Получить ID записи о студенте в данной группе с таким же номером

Удалить запись о пароле студента

Удалить запись по ID если существует

Если удаление успешно

Вернуть сообщение об успешном удалении

Иначе

Выдать ошибку удаления

**Конец**

* **Получение всех записей о студентах**

**Начало**

Запросить из базы записи о студентах

Если записи получены

Вернуть записи

Иначе

Выдать ошибку об отсутствии записей

**Конец**

* **Получение записи о студенте**

**Начало**

Получить данные о номере группы и номере по списку из запроса

Получить ID записи о студенте в данной группе с таким же номером

Запросить из базы запись по ID если существует

Если получена запись

Вернуть сообщение об успешном запросе и данные

Иначе

Выдать ошибку получения данных

**Конец**

* **Добавление новой записи пароля**

**Начало**

Получить данные из запроса: «Номер группы», «Номер по списку», «Пароль»

Запросить из базы запись о студенте в данной группе с таким же номером

Если запись существует

Запросить из базы запись о пароле в данной группе с таким же номером

Если запись существует

Выдать ошибку перезапись невозможна

Иначе если пароль длиннее 8 символов

Добавить в базу запись о пароле

Вернуть сообщение об успешной записи

Иначе

Выдать ошибку длины пароля

Иначе

Выдать ошибку отсутствия записи студента

**Конец**

* **Удаление записи о студенте**

**Начало**

Получить данные о номере группы и номере по списку из запроса

Удалить запись о пароле в данной группе с таким же номером если существует

Если удаление успешно

Вернуть сообщение об успешном удалении

Иначе

Выдать ошибку удаления

**Конец**

* **Получение записи о пароле**

**Начало**

Получить данные о номере группы и номере по списку из запроса

Получить запись о пароле в данной группе с таким же номером если существует

Если получена запись

Вернуть сообщение об успешном запросе и данные

Иначе

Выдать ошибку получения данных

**Конец**

# **Сведения о программной реализации**

При написании программы использовался язык программирования Python3.9. Среда разработки – Notepad++, версия 8.5.7.

Системные требования для запуска программы – ОС Windows/linux, ОЗУ- 1 гб, 10 гб свободного места на жёстком диске.

**Краткое описание программных единиц:**

1. **Функция** index(request: Request) -> HTMLResponse

Функция, возвращающая шаблон главной страницы в ответ на запрос

Входные параметры: request – запрос, полученный сервером

Выходные параметры: HTML страница запрошенная у сервера

1. **Функция** std\_add (request: Request) -> HTMLResponse

Функция, возвращающая шаблон страницы добавленяи студента в базу в ответ на запрос

Входные параметры: request – запрос, полученный сервером

Выходные параметры: HTML страница запрошенная у сервера

1. **Функция** std\_del (request: Request) -> HTMLResponse

Функция, возвращающая шаблон страницы удаления студента из базы в ответ на запрос

Входные параметры: request – запрос, полученный сервером

Выходные параметры: HTML страница запрошенная у сервера

1. **Функция** pwd\_add (request: Request) -> HTMLResponse

Функция, возвращающая шаблон страницы добавления пароля в базу в ответ на запрос

Входные параметры: request – запрос, полученный сервером

Выходные параметры: HTML страница запрошенная у сервера

1. **Функция** pwd\_del (request: Request) -> HTMLResponse

Функция, возвращающая шаблон страницы удаления пароля из базы в ответ на запрос

Входные параметры: request – запрос, полученный сервером

Выходные параметры: HTML страница запрошенная у сервера

1. **Функция** get\_pwd\_name (request: Request) -> HTMLResponse

Функция, возвращающая шаблон страницы получения данных для входа в ответ на запрос

Входные параметры: request – запрос, полученный сервером

Выходные параметры: HTML страница запрошенная у сервера

1. **Функция** isletter(name: str) -> bool

Функция проверки строки на содержание не букв

Входные параметры: name – проверяемая строка

Выходные параметры: True если строка содержит только буквы, иначе False

1. **Функция** student\_helper(student) -> dict

Функция формирования записи в формате записи о студенте в базе данных

Входные параметры: student – не отформатированные данные

Выходные параметры: отформатированный список

1. **Функция**

Функция формирования записи в формате записи о пароле в базе данных

Входные параметры: password – не отформатированные данные

Выходные параметры: отформатированный список

1. **Функция** get\_student\_id(group: str, num: int) -> str

Функция получения id записи студента из базы по номеру группы и номеру по списку

Входные параметры:

* + - group – строка содержащая номер группы
    - num – номер в группе по списку

Выходные параметры: строка ID записи о студенте, если существует, иначе None

1. **Функция** retrieve\_students() -> list()

Функция получения всех записей о студентах из базы

Входные параметры: отсутствуют

Выходные параметры: массив записей о студентах из базы данных

1. **Функция** add\_student(student\_data: dict) -> dict

Функция добавления записи студента в базу

Входные параметры: отформатированные данные о студенте

Выходные параметры: результат операции запись, если добавлена, иначе None

1. **Функция** retrieve\_student(id: str) -> dict

Функция получения записи студента из базы по ID

Входные параметры: ID записи в базе данных

Выходные параметры: запись, если существует, иначе None

1. **Функция** delete\_student(id: str) -> bool

Функция удаления записи студента из базы по ID

Входные параметры: ID записи в базе данных

Выходные параметры: True если запись удалена, False, если не существовала

1. **Функция** retrieve\_password(group: str, num: int) -> dict

Функция получения записи пароля из базы по номеру группы и номеру по списку

Входные параметры:

* + - group – строка содержащая номер группы
    - num – номер в группе по списку

Выходные параметры: запись пароля если существует, иначе None

1. **Функция** add\_password(password\_data: dict) -> dict

Функция добавления записи пароля в базу

Входные параметры: отформатированные данные о пароле

Выходные параметры: результат операции – добавленная запись

1. **Функция** delete\_password(group: str, num: int) -> bool

Функция удаления записи пароля из базы по номеру группы и номеру по списку

Входные параметры:

* + - group – строка содержащая номер группы
    - num – номер в группе по списку

Выходные параметры: True если запись удалена, False, если не существовала

1. **Функция** add\_student\_data(student: StudentSchema = Body(...)) -> dict

Функция добавления студента

Входные параметры: данные о студенте в формате JSON

Выходные параметры: результат операции записанные данные или ошибка

1. **Функция** get\_students() -> dict

Функциям получения списка студентов

Входные параметры: отсутствуют

Выходные параметры: результат операции полученные данные или ошибка

1. **Функция** get\_student\_data(group\_num: str) -> dict

Функция получения конкретного студента

Входные параметры: group\_num – JSON строка, содержащая данные о номере группы и номере по списку

Выходные параметры: результат операции полученные данные или ошибка

1. **Функция** delete\_student\_data(group\_num: GroupNumRequest) -> dict

Функция удаления студента

Входные параметры: group\_num – JSON строка, содержащая данные о номере группы и номере по списку

Выходные параметры: результат операции удаленные данные или ошибка

1. **Функция** add\_password\_data(password: PasswordSchema = Body(...)) -> dict

Функция добавления пароля студенту

Входные параметры: данные о пароле в формате JSON

Выходные параметры: результат операции записанные данные или ошибка

1. **Функция** get\_password\_data(group\_num: str) -> dict

Функция получения пароля студента

Входные параметры: group\_num – JSON строка, содержащая данные о номере группы и номере по списку

Выходные параметры: результат операции полученные данные или ошибка

1. **Функция** delete\_password\_data(group\_num: GroupNumRequest) -> dict

Функция удаления пароля студента

Входные параметры: group\_num – JSON строка, содержащая данные о номере группы и номере по списку

Выходные параметры: результат операции удаленные данные или ошибка

1. **Класс** StudentSchema

Класс, содержащий данные о студенте, полученные сервером

Поля:

* + - fullname: str – Имя студента
    - group: str - Номер группы
    - num: int - Номер по списку группы

1. **Класс** PasswordSchema

Класс, содержащий данные о пароле, полученные сервером

Поля:

* + - group: str - Номер группы
    - num: int - Номер по списку группы
    - password: str – Пароль студента

1. **Класс** StudentSchema

Класс, содержащий данные о номере группы и номере по списку, полученные сервером

Поля:

* + - groupNum: str– Композитные данные о номере группы и номере по списку

1. **HTML-шаблон “main.html”**

Шаблон содержит главную страницу приложения.

Содержит:

* Навигационную панель
* Кнопку получения записей студентов из базы
* Поле вывода полученных записей

1. **HTML-шаблон “std\_add.html”**

Шаблон содержит страницу добавления записи о студенте.

Содержит:

* Поля ввода данных записи о студенте
* Кнопка подтверждения добавления записи

1. **HTML-шаблон “std\_del.html”**

Шаблон содержит страницу удаления записи о студенте.

Содержит:

* Поле ввода композитных данных о номере группы и номере по списку удаляемого студента
* Кнопка подтверждения добавления записи

1. **HTML-шаблон “pwd\_add.html”**

Шаблон содержит страницу добавления записи о пароле.

Содержит:

* Поля ввода данных записи о пароле
* Кнопка подтверждения добавления записи

1. **HTML-шаблон “pwd\_del.html”**

Шаблон содержит страницу удаления записи о пароле.

Содержит:

* Поле ввода композитных данных о номере группы и номере по списку удаляемого пароля
* Кнопка подтверждения удаления записи

1. **HTML-шаблон “get\_pwd\_name.html”**

Шаблон содержит страницу получение данных для входа по номеру группы и номеру в группе.

Содержит:

* Поле ввода композитных данных о номере группы и номере по списку запрашиваемого набора данных
* Кнопка подтверждения запроса набора данных
* Поле вывода имени
* Поле вывода пароля

# **Инструкция пользователя**

**Инструкция по запуску приложения**

Для запуска приложения требуется:

1. Запустить файл cmd.bat
2. Убедиться, что контейнер mongodb запущен на локальном устройстве на порту 27017:27017
3. Выполнить команду curse/Scripts/activate
4. Выполнить команду pip install -r requirements.txt
5. Выполнить команду python app/main.py
6. Перейти на сайт <http://localhost:8000>

**Инструкция по работе с сайтом**

На главной странице сайта имеется кнопка «обновить список студентов», которая выводит все записи о студентах, хранящиеся в базе на данный момент.

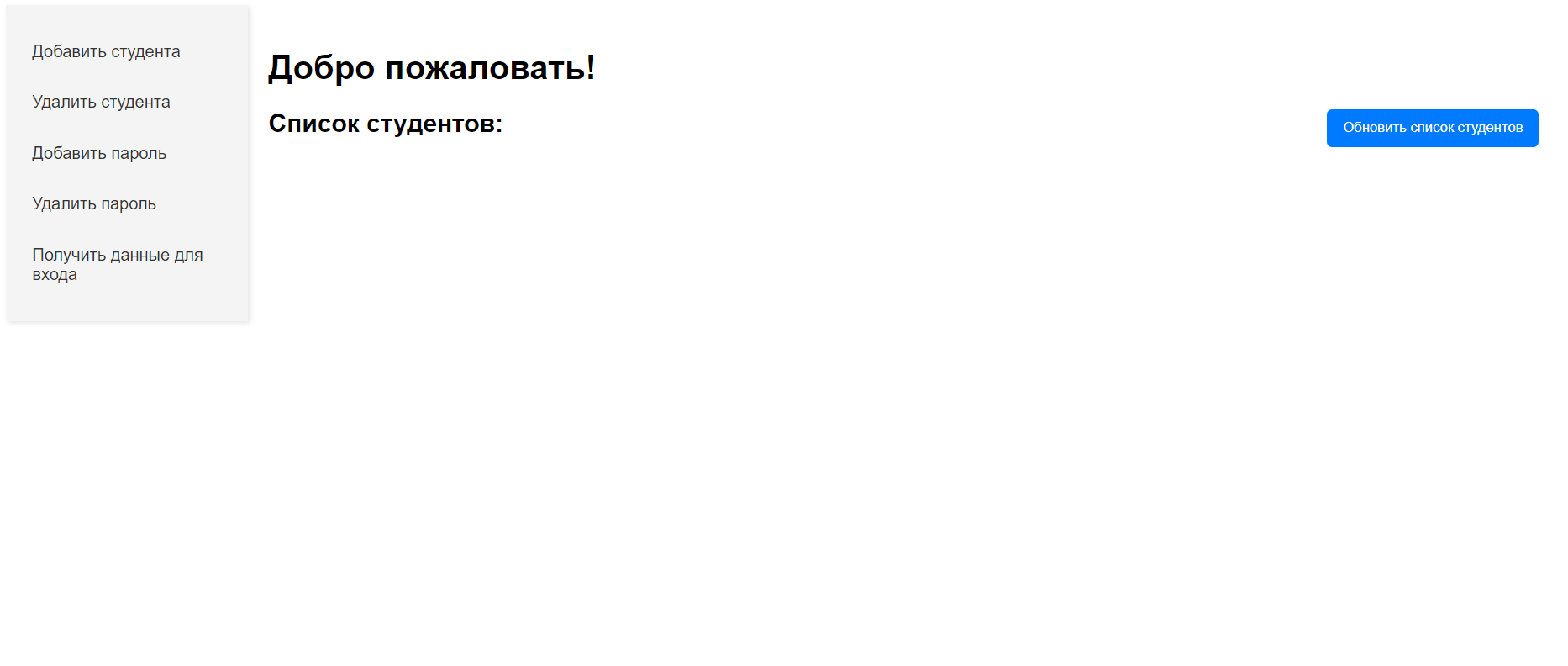


Рис 1. – Главная страница до получения списка.

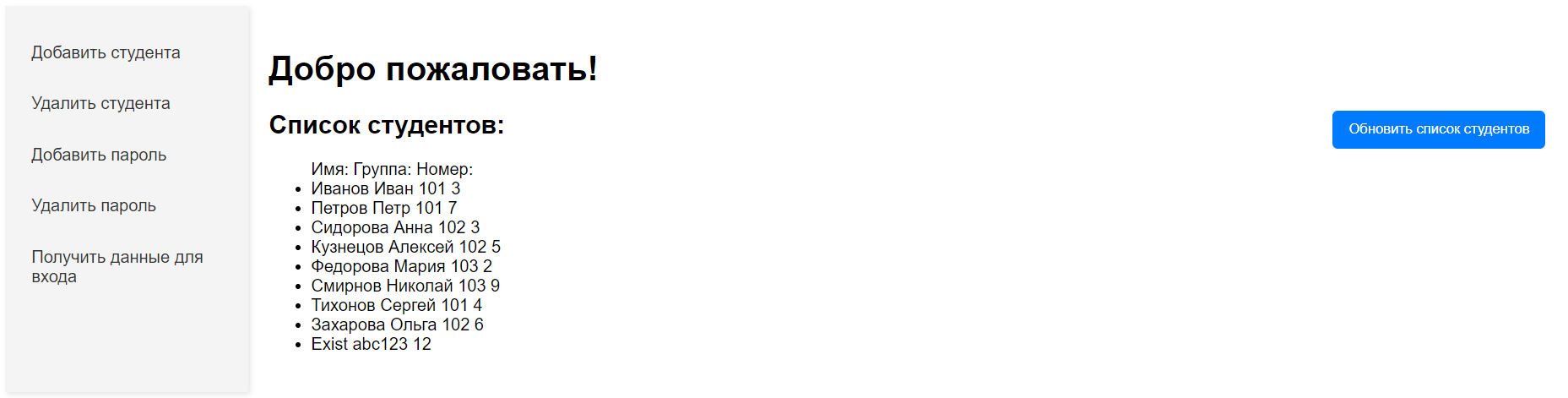


Рис 2. – Главная страница после получения текущего списка студентов.

Также в правой части главной страницы расположено поле навигации. Для добавления записи о студенте требуется перейти во вкладку «добавить студента».

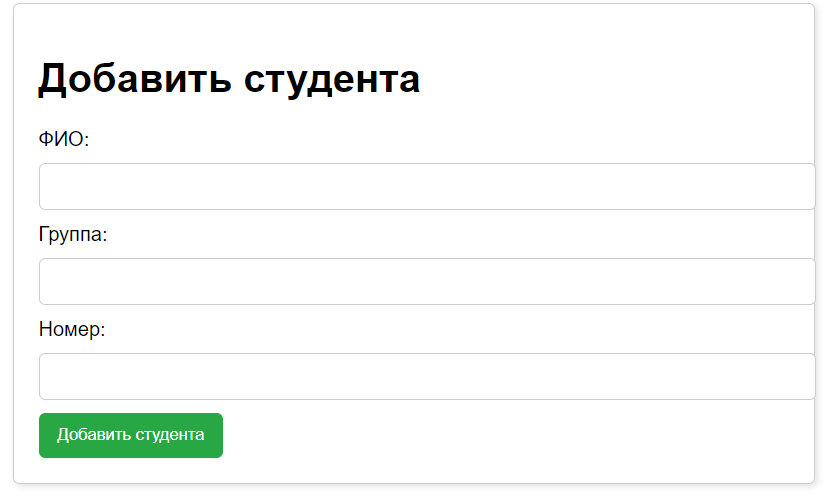


Рис 3. – Страница добавления студентов.

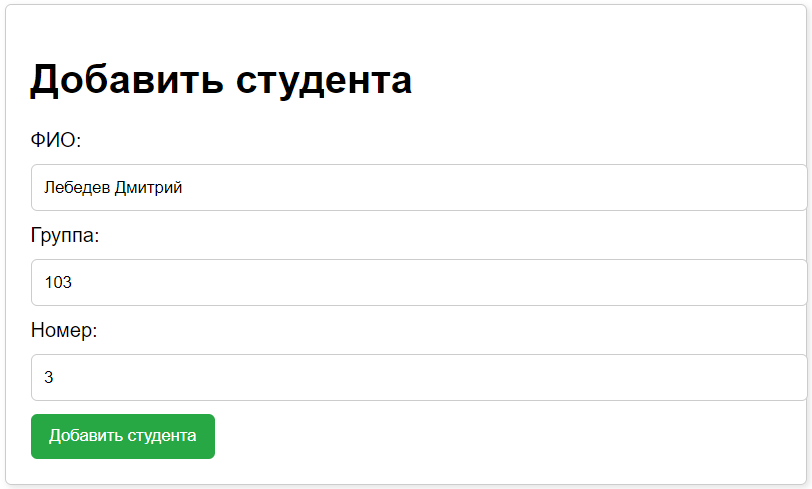


Рис 4. – Заполненная форма добавления студента.

В случае успешного добавления пользователь получит уведомление.

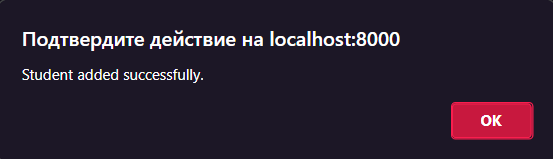


Рис 5. – Уведомление о успешном добавлении студента в базу.

При попытке ввода некорректных данных пользователь получит уведомление.

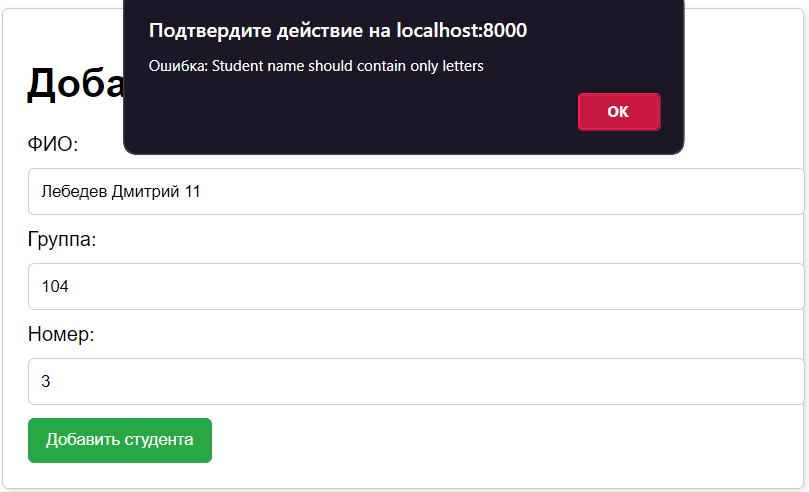


Рис 6. – Уведомление об ошибке имени при добавлении студента в базу.

Пользователь не может ввести что-то кроме чисел в поле ввода номера по списку, но при вводе отрицательного номера получит уведомление.

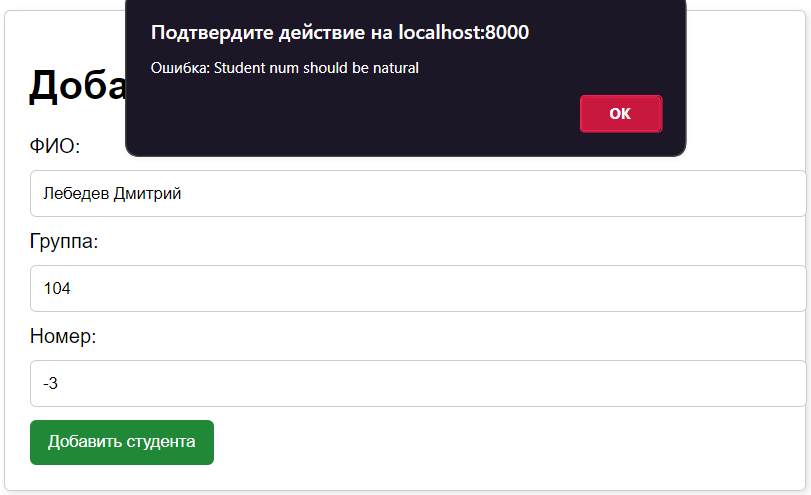


Рис 7. – Уведомление об ошибке номера при добавлении студента в базу.

Для удаления записи о студенте требуется перейти во вкладку «удалить студента».

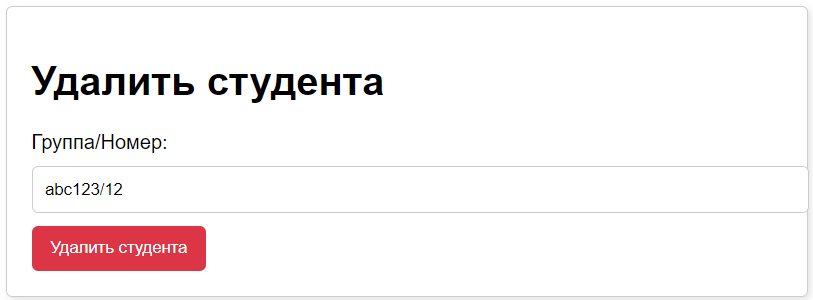


Рис 8. – Страница удаления студентов.

При успешном удалении студента пользователь получит уведомление.

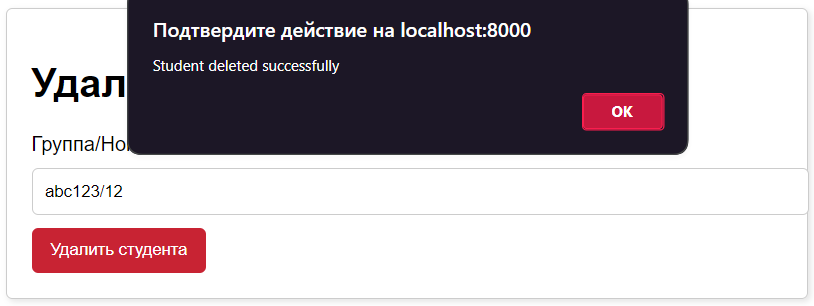


Рис 9. – Уведомление о успешном удалении студента из базы.

Если при удалении студента возникнет ошибка пользователь получит уведомление.

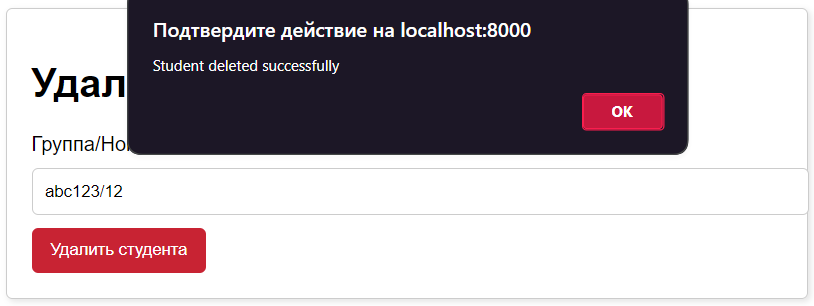


Рис 10. – Уведомление об ошибке при удалении студента из базы.

Для добавления записи о пароле требуется перейти во вкладку «добавить пароль».

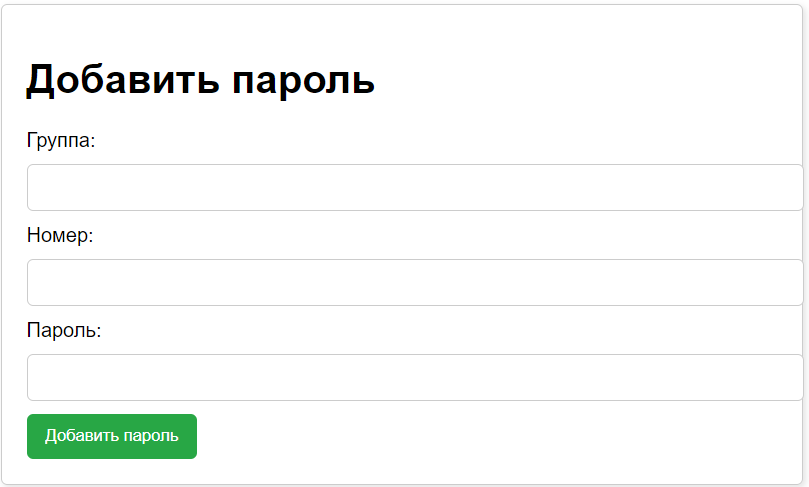


Рис 11. – Страница добавления паролей.

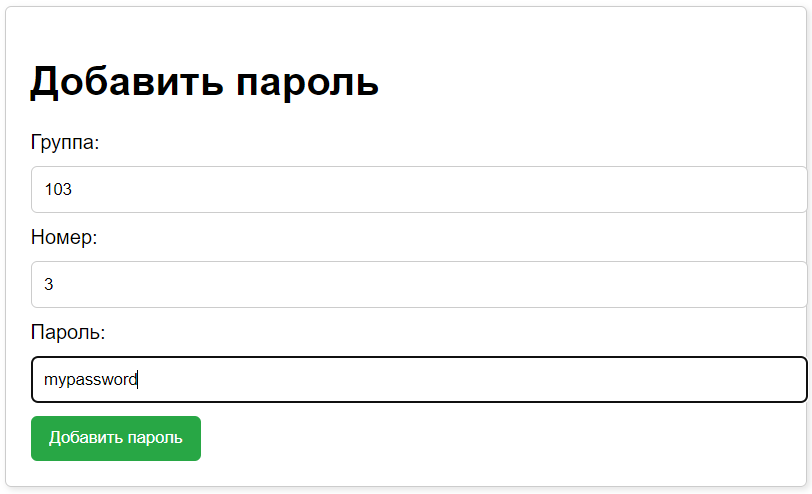


Рис 12. – Заполненная форма добавления паролей.

В случае успешного добавления пользователь получит уведомление.

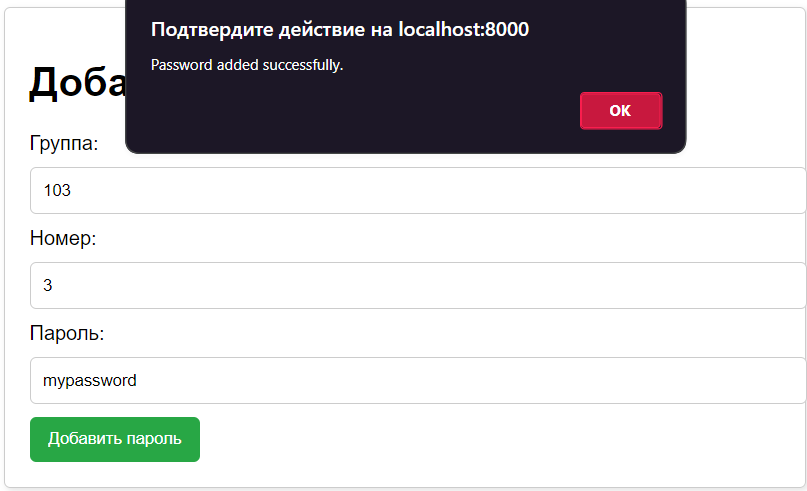


Рис 13. – Уведомление о успешном добавлении студента в базу.

При попытке ввода некорректных данных пользователь получит уведомление.

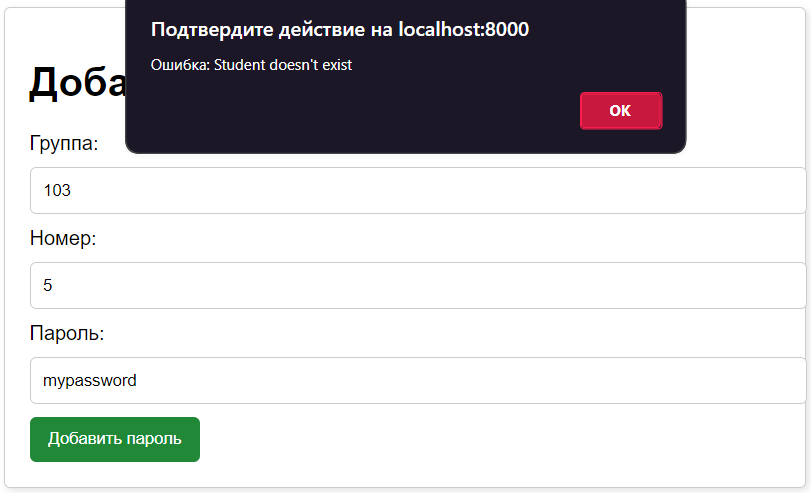


Рис 14. – Уведомление об ошибке студента при добавлении пароля.

При вводе короткого пароля получит уведомление.

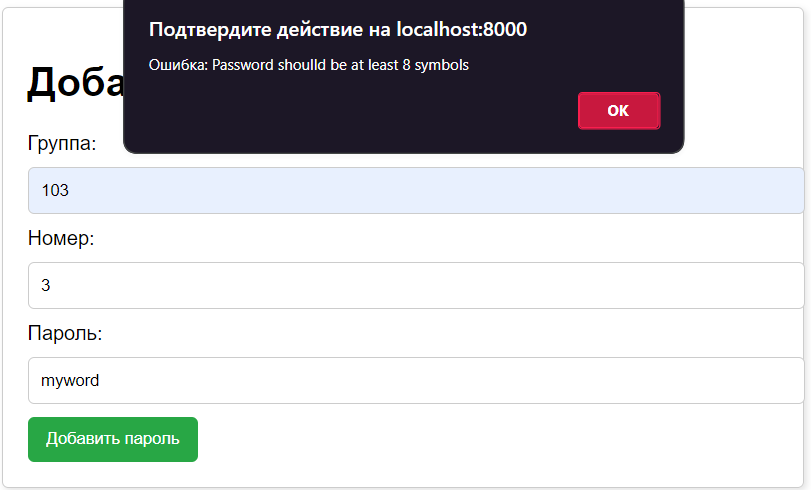


Рис 15. – Уведомление о слишком коротком пароле.

Для удаления записи о пароле требуется перейти во вкладку «удалить студента».

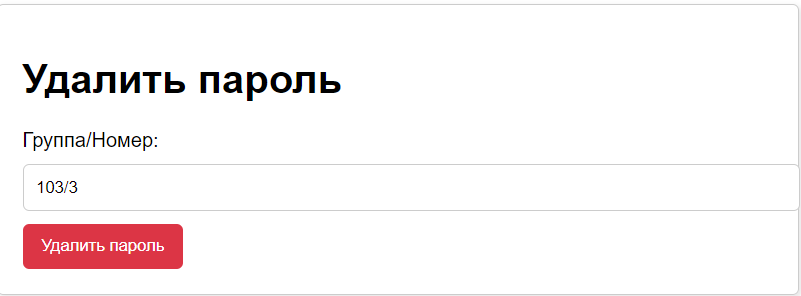


Рис 16. – Страница удаления паролей.

При успешном удалении пароля пользователь получит уведомление.

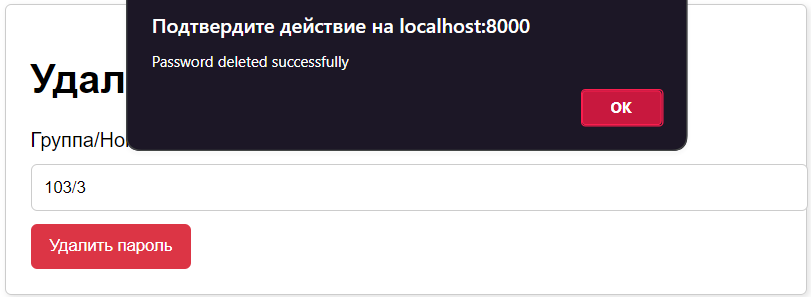


Рис 17. – Уведомление о успешном удалении пароля из базы.

Если при удалении пароля возникнет ошибка пользователь получит уведомление.

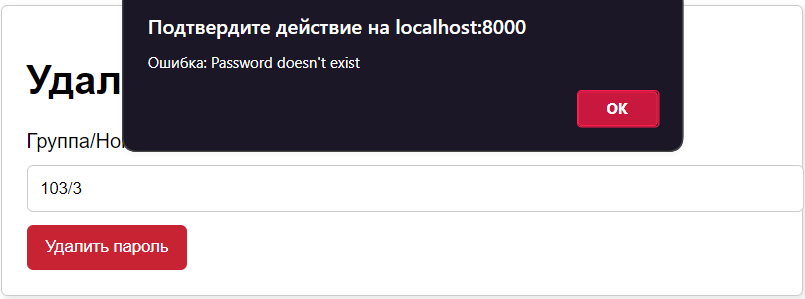


Рис 18. – Уведомление об ошибке при удалении пароля из базы.

Для получения данных для входа требуется перейти во вкладку «Получить данные для входа».

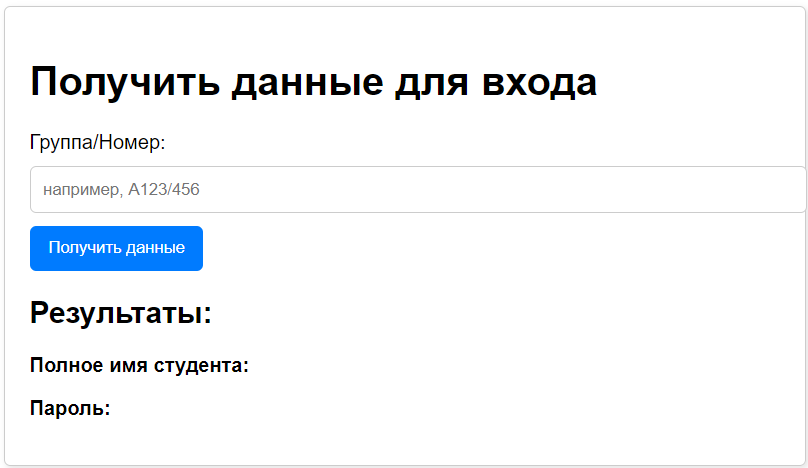


Рис 19. – Страница получения данных для входа.

При корректном вводе пользователь получит набор данных для входа.

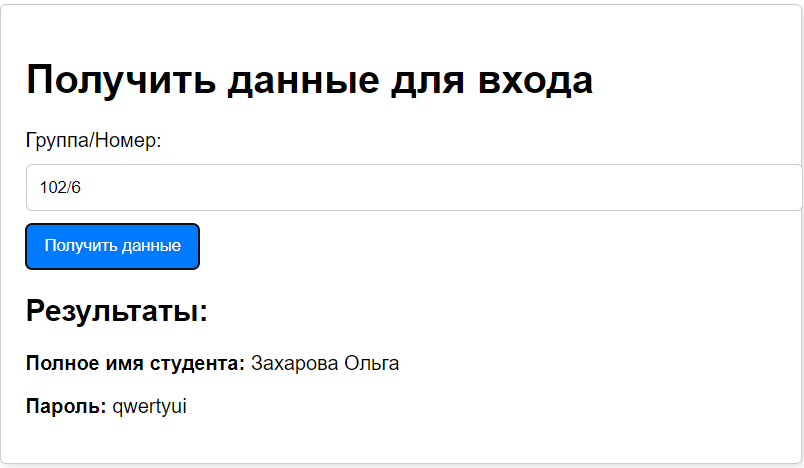


Рис 20. – Данные для пользователя при корректном вводе

В случае если студенту не задан пароль, пользователь получит уведомление.

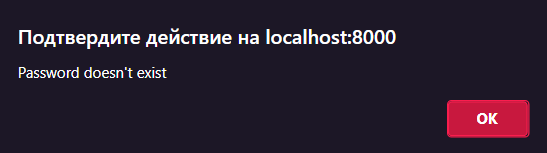


Рис 21. – Уведомление об отсутствии пароля у студента.

В случае если студент не существует, пользователь получит уведомление.

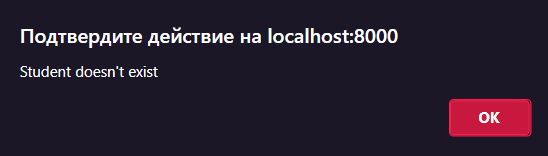


Рис 22. – Уведомление об отсутствии студента в базе.

# **Тестирование**

**Коды состояния ответа HTTP**

|  |  |
| --- | --- |
| 200 | Запрос успешно выполнен. Если клиентом были запрошены какие-либо данные, то они находятся в заголовке и/или теле сообщения |
| 404 | Сервер не может найти запрошенный ресурс. В браузере это означает, что URL-адрес не распознан. В API это также может означать, что адрес правильный, но ресурс не существует. |
| 422 | Сервер успешно принял запрос, может работать с указанным видом данных, однако имеется какая-то логическая ошибка, из-за которой невозможно произвести операцию над ресурсом |
| 500 | Произошла ошибка в ядре серверного приложения. Как правило это говорит о возникновении необработанного исключения. |

**Тестирование через Swagger Fast Api:**

**Функция получения записей о студентах из базы**

Запрос при не пустой базе:

* Ожидаемое поведение: получены данные о записях о студентах из базы
* Фактическое поведение: от сервера получена JSON строка, содержащая все записи о студентах, которые на данный момент содержатся в базе данных. Сервер вернул код 200, что говорит о том, что запрос был успешно выполнен

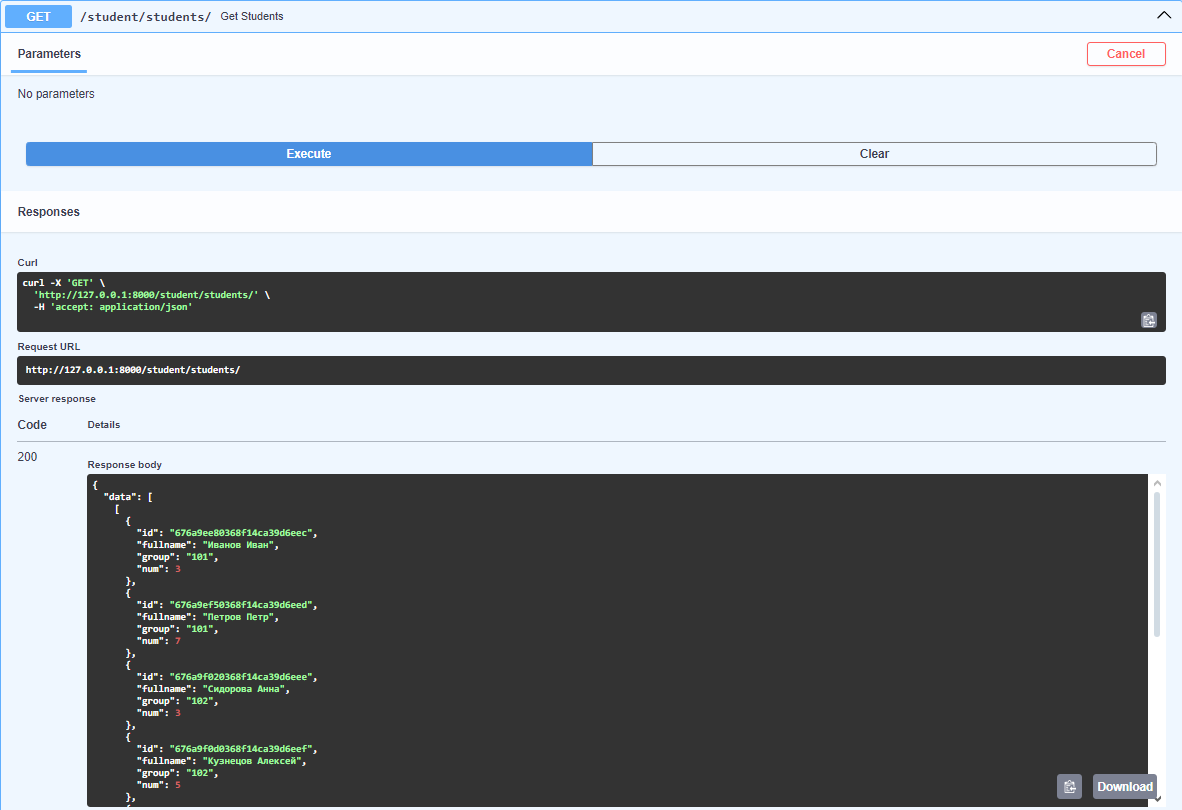


Рисунок 23. – Успешное получение записей о студентах из БД

Запрос при пустой базе:

* Ожидаемое поведение: получена JSON строка, содержащая описание ошибки с кодом ошибки от сервера 404
* Фактическое поведение: от сервера получена JSON строка, содержащая детали ошибки «Empty list returned», которая говорит о том, что из базы не было получено ни одной записи. Сервер вернул код 404.

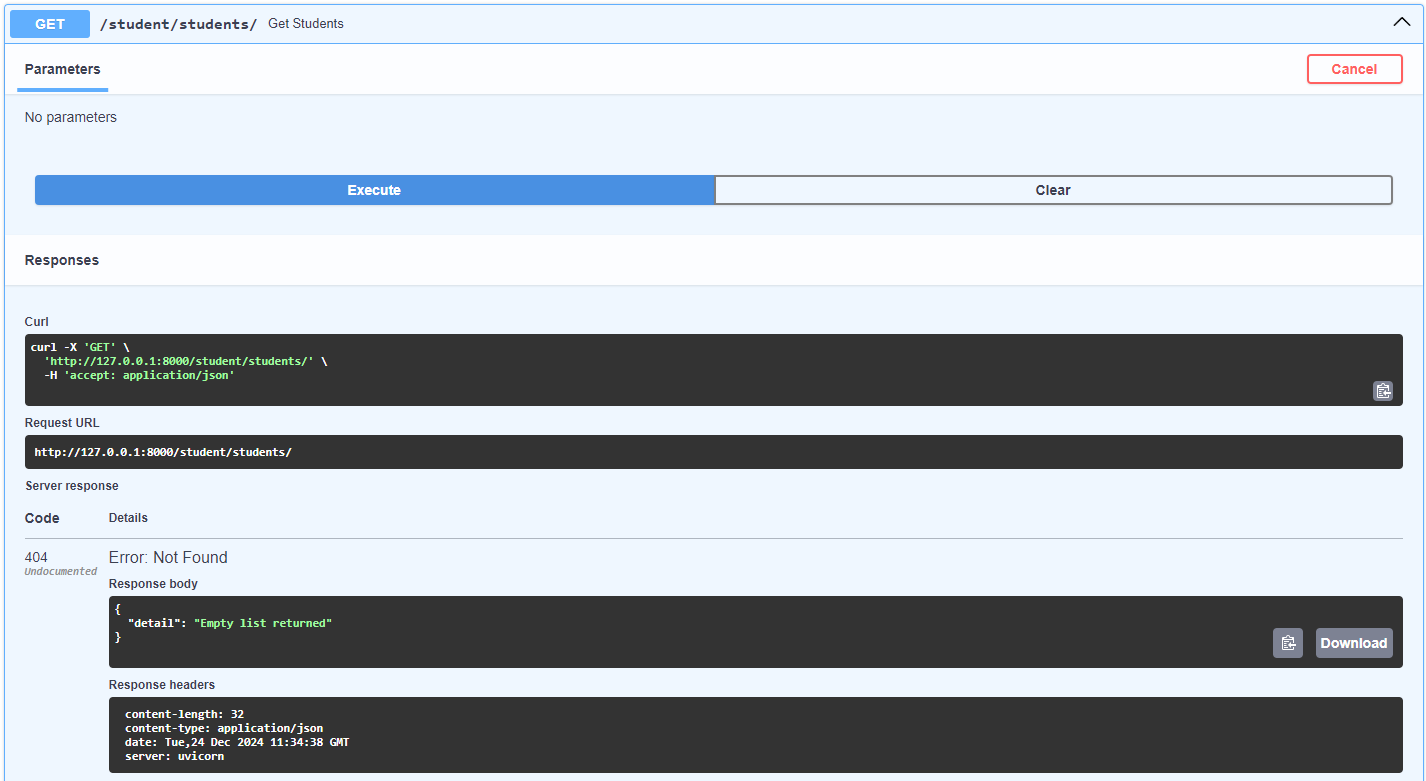


Рисунок 23. – В базе данных не содержится ни одной записи о студентах

**Функция добавления студента**

Добавление записи о студенте с корректными полями:

* Ожидаемое поведение: получена JSON строка, содержащая данные о добавленной записи и подтверждение успешного создания записи. Сервер вернул код 200.
* Фактическое поведение: от сервера получена JSON строка, содержащая данные, введенные в поля ввода при создании записи. Сервер вернул код 200 и подтверждение создания записи в базе.

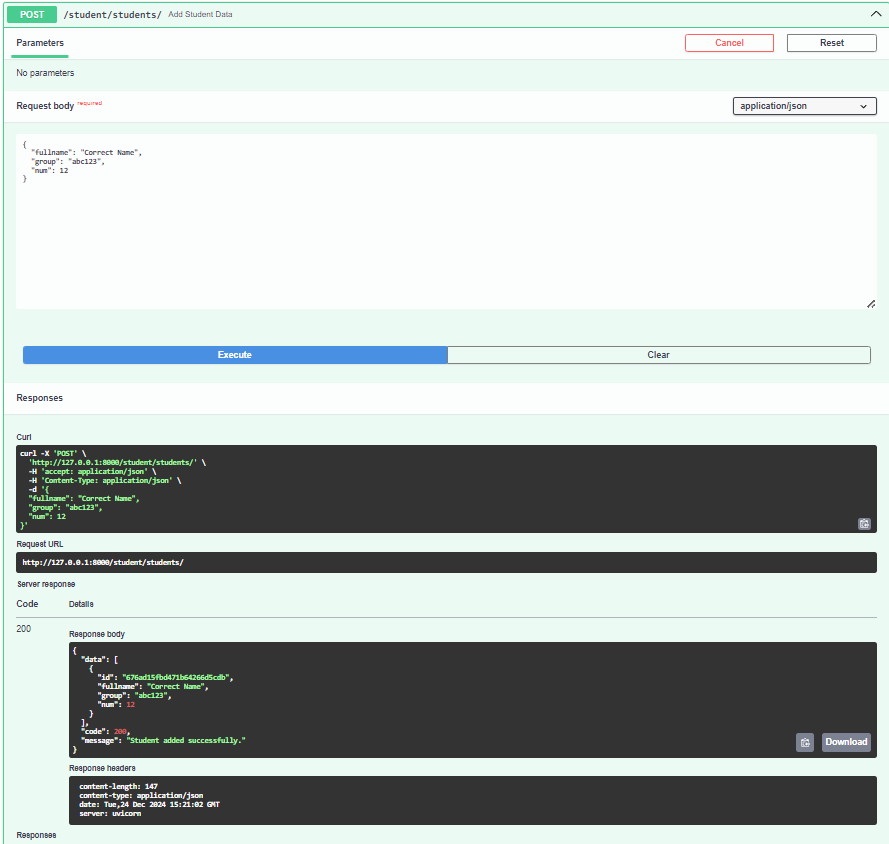


Рисунок 24. – Данные о студенте успешно добавлены в базу

Добавление записи о студенте с повторяющимися данными:

* Ожидаемое поведение: получена JSON строка, содержащая данные описание ошибки с кодом ошибки от сервера 422.
* Фактическое поведение: от сервера получена JSON строка, содержащая детали ошибки «Student already exists». От сервера получен код ошибки 422.

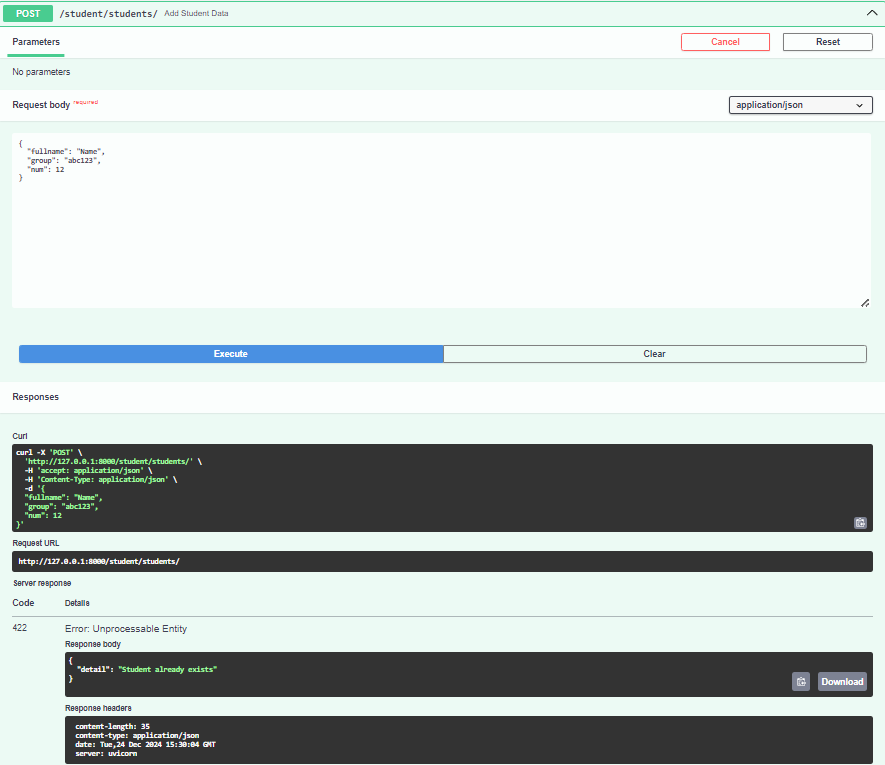


Рисунок 25. – Запись о студенте уже существует

Добавление записи о студенте с некорректным полем имени:

* Ожидаемое поведение: получена JSON строка, содержащая данные описание ошибки с кодом ошибки от сервера 422.
* Фактическое поведение: от сервера получена JSON строка, содержащая детали ошибки «Student name should contain only letters». От сервера получен код ошибки 422.

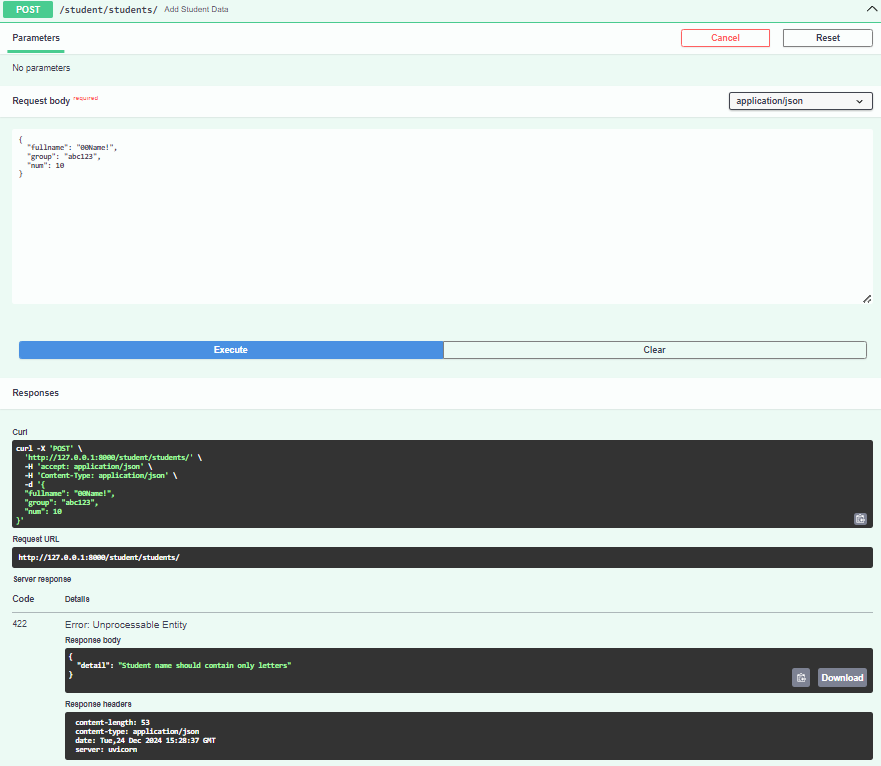


Рисунок 26. – Имя студента должно содержать только буквы

Добавление записи о студенте с некорректными полем номера по списку:

* Ожидаемое поведение: получена JSON строка, содержащая данные описание ошибки с кодом ошибки от сервера 422.
* Фактическое поведение: от сервера получена JSON строка, содержащая детали ошибки «Input should be a valid integer, unable to parse string as an integer». От сервера получен код ошибки 422.

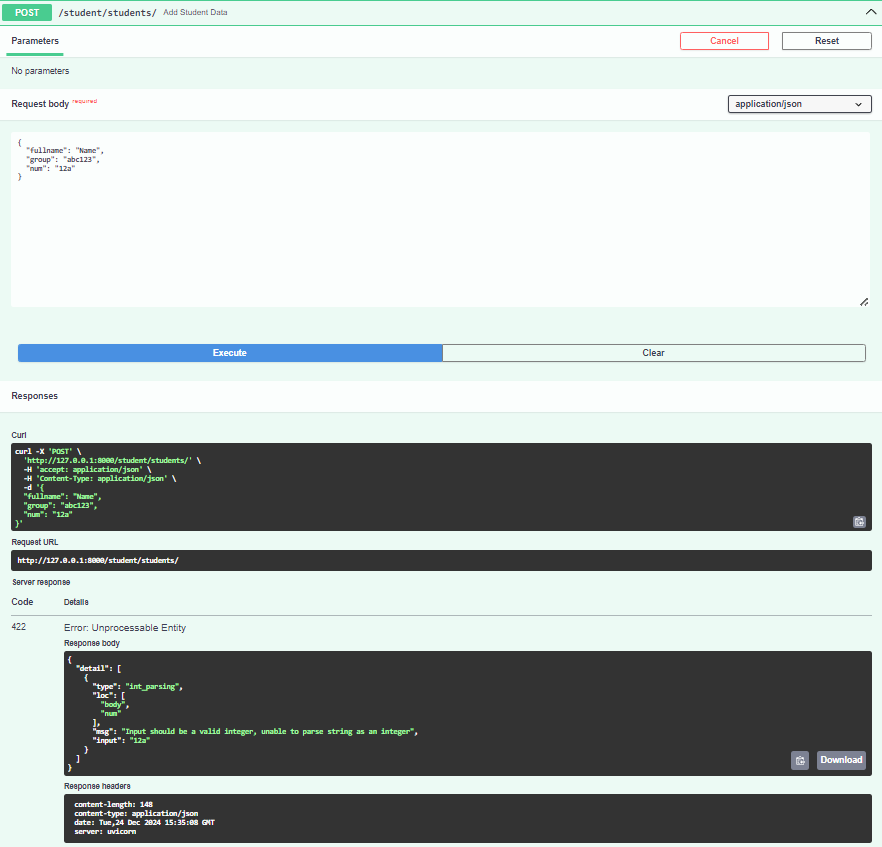


Рисунок 27. – Номер по списку должен быть числом

**Функция удаления студента**

Удаление существующей записи о студенте:

* Ожидаемое поведение: получена JSON строка, содержащая данные о успешном удалении записи. Сервер вернул код 200.
* Фактическое поведение: от сервера получена JSON строка, содержащая подтверждение успешного удаления записи. Сервер вернул код 200.

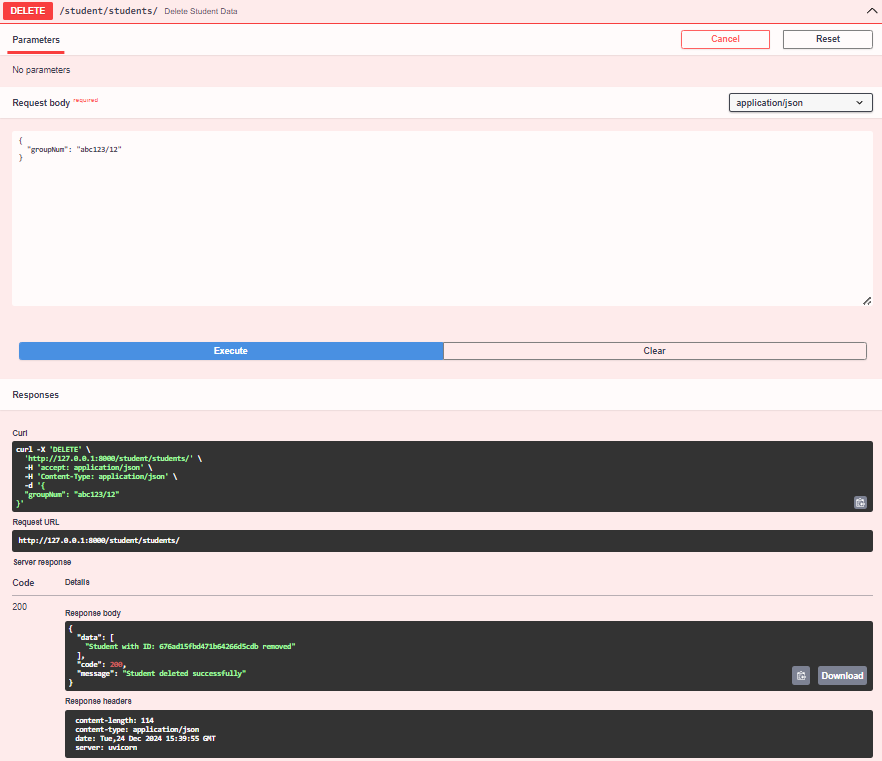
****

Рисунок 28. – Успешное удаление записи о студенте

Удаление несуществующей записи о студенте:

* Ожидаемое поведение: получена JSON строка, содержащая описание ошибки с кодом ошибки от сервера 404
* Фактическое поведение: от сервера получена JSON строка, содержащая детали ошибки «Student doesn't exist», которая говорит о том, что при попытке удаления в базе не было обнаружено подходящей записи. Сервер вернул код 404.

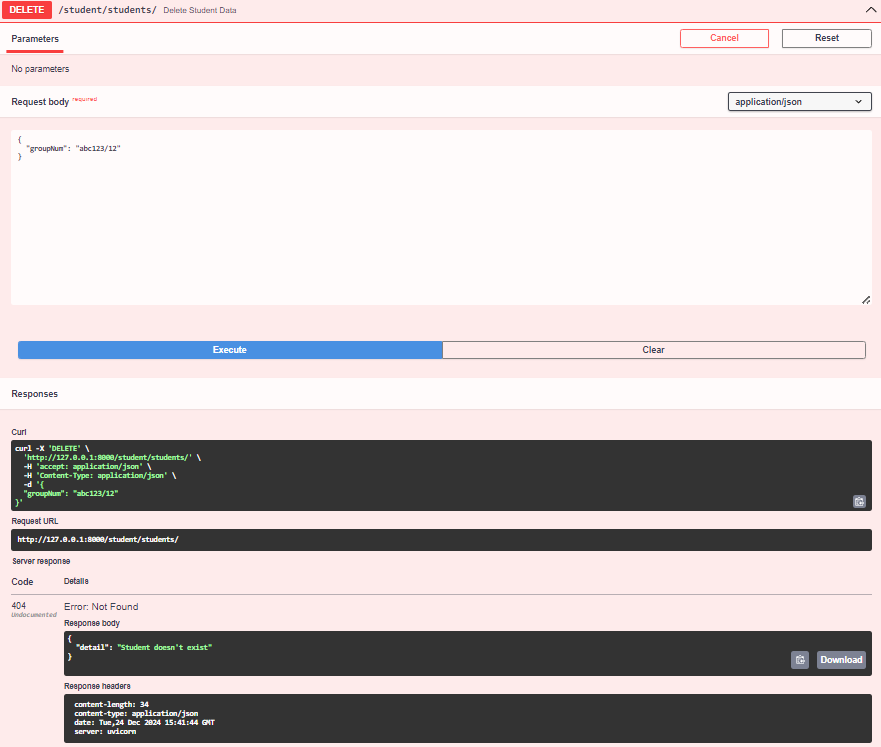


Рисунок 29. – Попытка удаления несуществующей записи о студенте

**Функция добавления пароля студенту**

Добавление записи о пароле с корректными полями:

* Ожидаемое поведение: получена JSON строка, содержащая данные о добавленной записи и подтверждение успешного создания записи. Сервер вернул код 200.
* Фактическое поведение: от сервера получена JSON строка, содержащая данные, введенные в поля ввода при создании записи. Сервер вернул код 200 и подтверждение создания записи в базе.

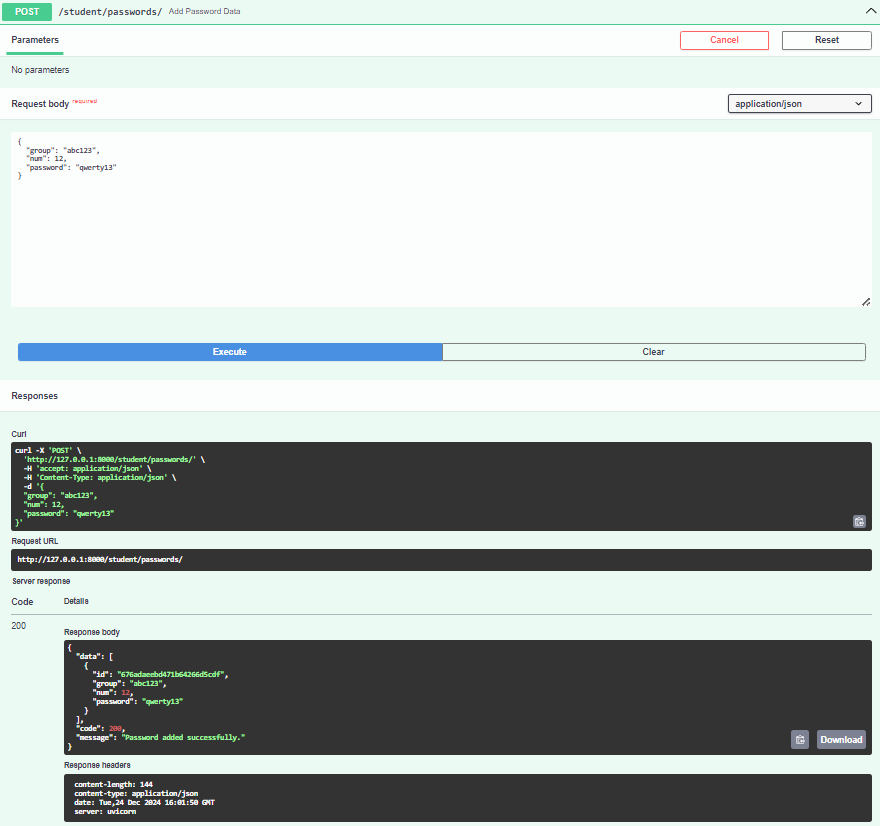


Рисунок 30. – Данные о пароле успешно добавлены в базу

Добавление записи о студенте с повторяющимися данными:

* Ожидаемое поведение: получена JSON строка, содержащая данные описание ошибки с кодом ошибки от сервера 422.
* Фактическое поведение: от сервера получена JSON строка, содержащая детали ошибки «Password already exists». От сервера получен код ошибки 422.

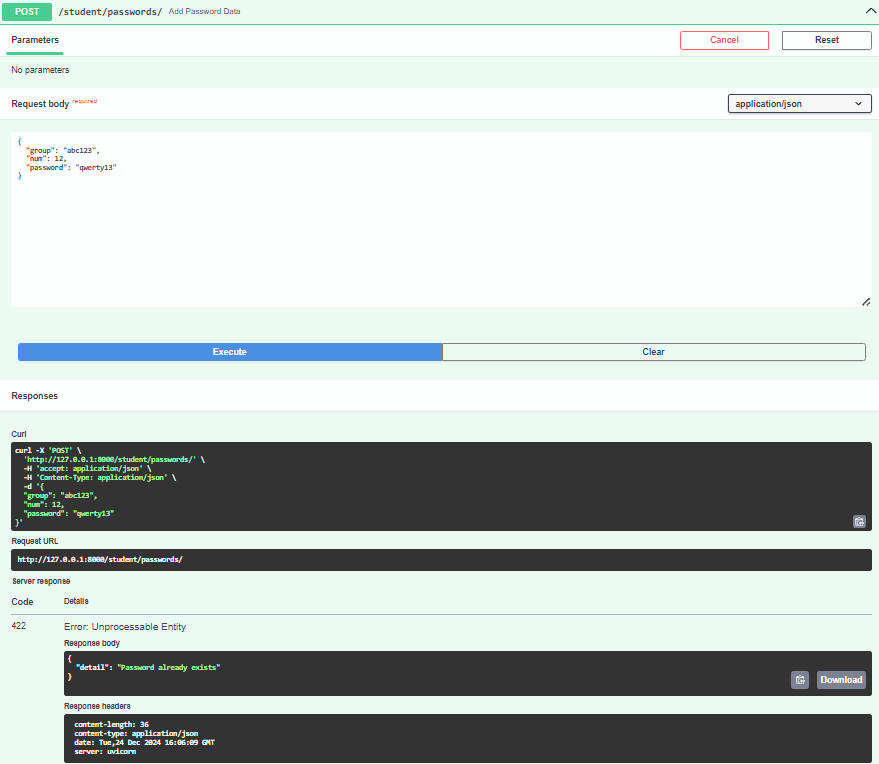


Рисунок 31. – Запись о пароле уже существует

Добавление записи о пароле с коротким полем пароля:

* Ожидаемое поведение: получена JSON строка, содержащая данные описание ошибки с кодом ошибки от сервера 422.
* Фактическое поведение: от сервера получена JSON строка, содержащая детали ошибки «Student name should contain only letters». От сервера получен код ошибки 422.

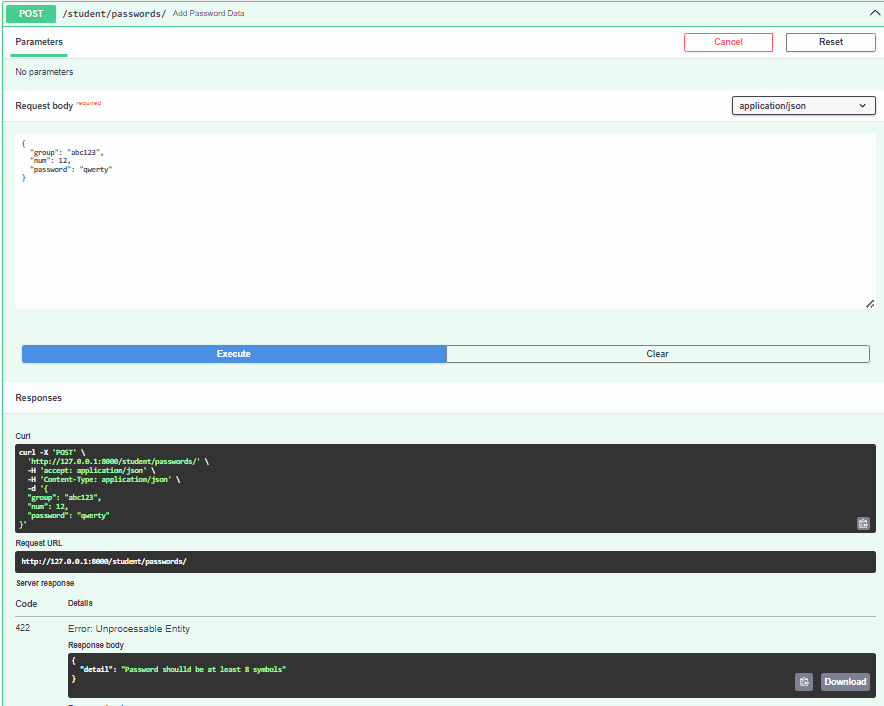


Рисунок 32. – Пароль должен быть длиннее 8 символов

**Функция удаления пароля студента**

Удаление существующей записи о пароле:

* Ожидаемое поведение: получена JSON строка, содержащая данные о успешном удалении записи. Сервер вернул код 200.
* Фактическое поведение: от сервера получена JSON строка, содержащая подтверждение успешного удаления записи. Сервер вернул код 200.

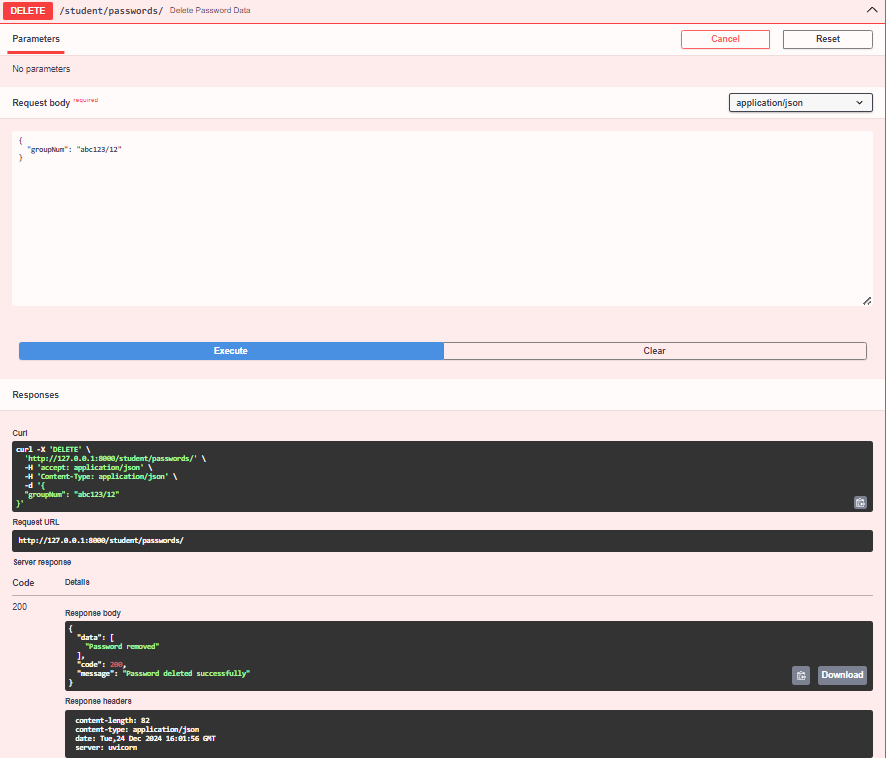
****

Рисунок 33. – Успешное удаление записи о пароле

Удаление несуществующей записи о пароле:

* Ожидаемое поведение: получена JSON строка, содержащая описание ошибки с кодом ошибки от сервера 404
* Фактическое поведение: от сервера получена JSON строка, содержащая детали ошибки «Password doesn't exist», которая говорит о том, что при попытке удаления в базе не было обнаружено подходящей записи. Сервер вернул код 404.

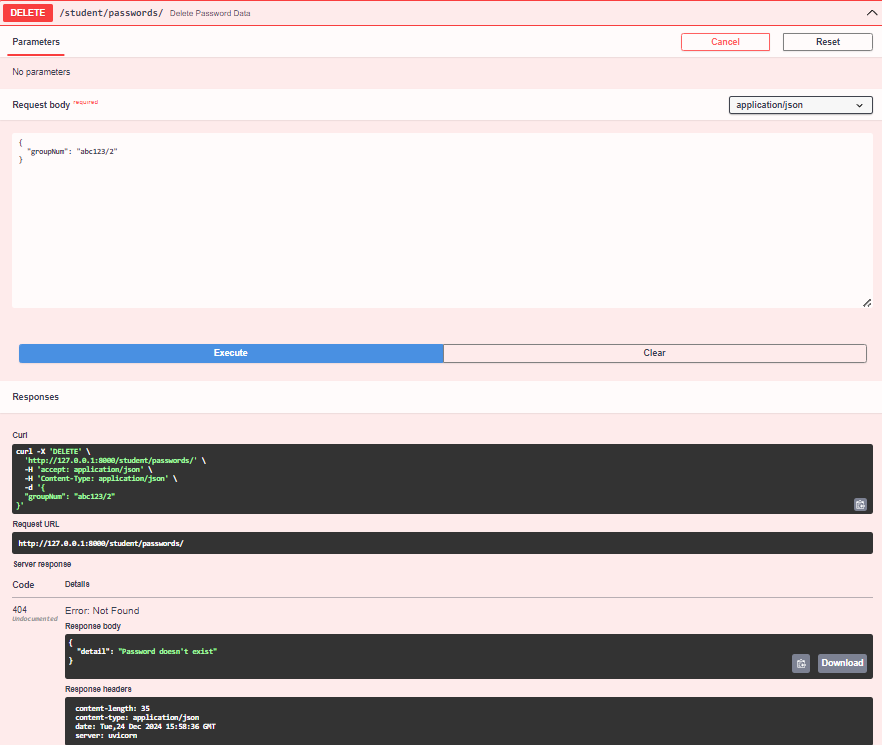


Рисунок 34. – Попытка удаления несуществующей записи о пароле

**Функция поиска данных о студенте**

Запрос существующей записи о студенте:

* Ожидаемое поведение: получена JSON строка, содержащая данные о запрошенном студенте. Сервер вернул код 200.
* Фактическое поведение: от сервера получена JSON строка, содержащая подтверждение успешного запроса и данные о запрошенном студенте. Сервер вернул код 200.

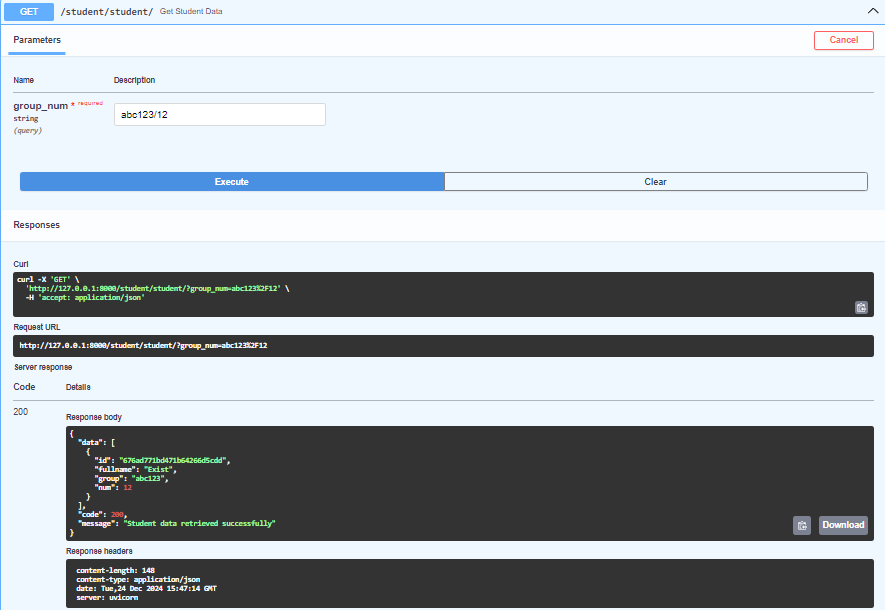
****

Рисунок 35. – Успешный запрос записи о студенте

Запрос несуществующей записи о студенте:

* Ожидаемое поведение: получена JSON строка, содержащая описание ошибки с кодом ошибки от сервера 404
* Фактическое поведение: от сервера получена JSON строка, содержащая детали ошибки «Student doesn't exist», которая говорит о том, что при запросе в базе не было обнаружено подходящей записи. Сервер вернул код 404.

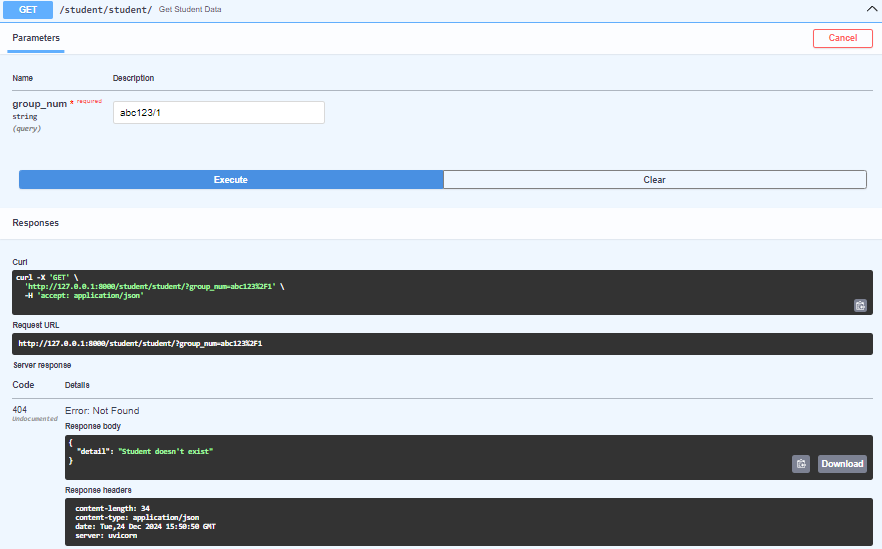


Рисунок 36. – Попытка запроса несуществующей записи о студенте

**Функция поиска данных о пароле**

Запрос существующей записи о пароле:

* Ожидаемое поведение: получена JSON строка, содержащая данные записи о запрошенном пароле. Сервер вернул код 200.
* Фактическое поведение: от сервера получена JSON строка, содержащая подтверждение успешного запроса и данные о запрошенном пароле. Сервер вернул код 200.

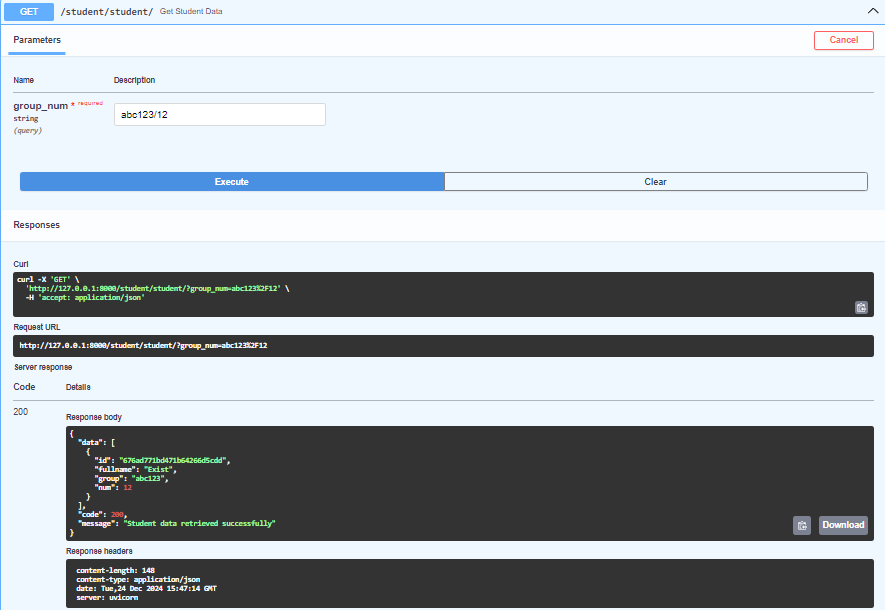
****

Рисунок 37. – Успешный запрос записи о пароле

Запрос несуществующей записи о пароле:

* Ожидаемое поведение: получена JSON строка, содержащая описание ошибки с кодом ошибки от сервера 404
* Фактическое поведение: от сервера получена JSON строка, содержащая детали ошибки «Password doesn't exist», которая говорит о том, что при запросе в базе не было обнаружено подходящей записи. Сервер вернул код 404.

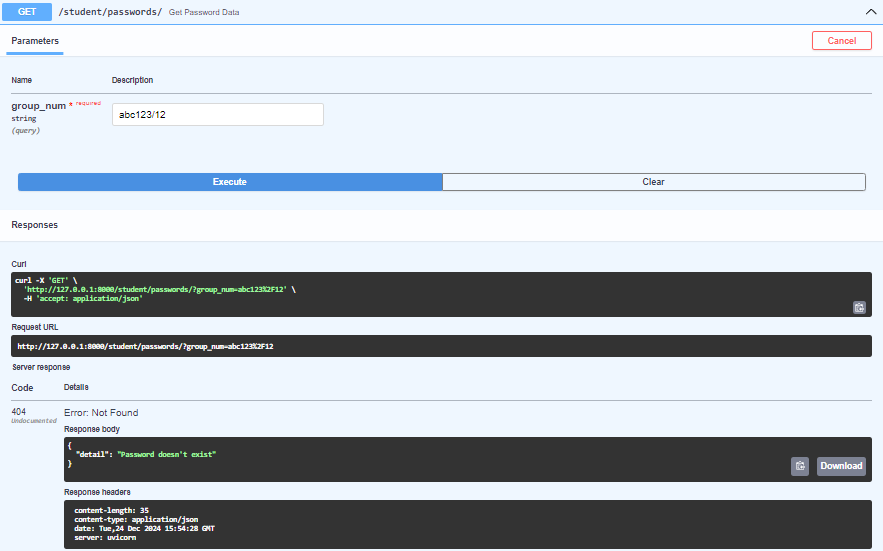


Рисунок 38. – Попытка запроса несуществующей записи о пароле

Запрос несуществующей записи о студенте:

* Ожидаемое поведение: получена JSON строка, содержащая описание ошибки с кодом ошибки от сервера 404
* Фактическое поведение: от сервера получена JSON строка, содержащая детали ошибки «Student doesn't exist», которая говорит о том, что при запросе в базе не было обнаружено подходящей записи. Сервер вернул код 404.

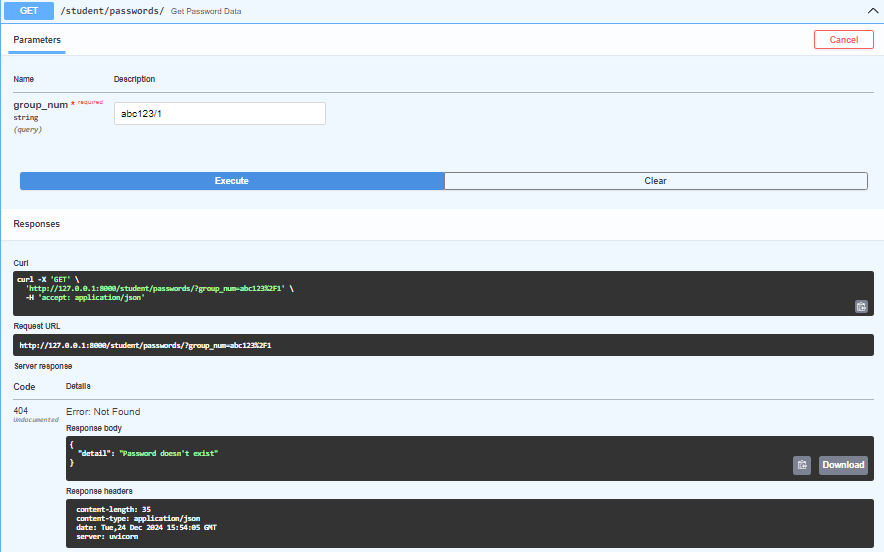


Рисунок 39. – Попытка запроса несуществующей записи о студенте

# **Листсинг**

**main.py**

import uvicorn

if \_\_name\_\_ == "\_\_main\_\_":

uvicorn.run("server.app:app", host="0.0.0.0", port=8000, reload=True)

**app.py**

from fastapi import FastAPI, Request

from fastapi.templating import Jinja2Templates

from fastapi.staticfiles import StaticFiles

from fastapi.responses import HTMLResponse

import pathlib

from .routes.student import router as StudentRouter

app = FastAPI()

app.include\_router(StudentRouter, tags=["Student"], prefix="/student")

BASE\_DIR = pathlib.Path(\_\_file\_\_).parent.parent

templates = Jinja2Templates(directory=[

BASE\_DIR / "front" / "templates",

])

app.mount("/static", StaticFiles(directory= (BASE\_DIR / "front" / "static")), name="static")

"""

@app.get("/", tags=["Root"])

async def read\_root():

return {"message": "Welcome to this fantastic app!"}

"""

#Маршрутизация выдачи главной страницы

@app.get('/', response\_class=HTMLResponse)

async def index(request: Request) -> HTMLResponse:

response = templates.TemplateResponse("main.html", {"request": request}) # Передаем request

return response

#Маршрутизация выдачи страницы добавления студента

@app.get("/std\_add.html", response\_class=HTMLResponse)

async def std\_add(request: Request) -> HTMLResponse:

return templates.TemplateResponse("std\_add.html", {"request": request})

#Маршрутизация выдачи страницы удаления студента

@app.get("/std\_del.html", response\_class=HTMLResponse)

async def std\_del(request: Request) -> HTMLResponse:

return templates.TemplateResponse("std\_del.html", {"request": request})

#Маршрутизация выдачи страницы добавления пароля

@app.get("/pwd\_add.html", response\_class=HTMLResponse)

async def pwd\_add(request: Request) -> HTMLResponse:

return templates.TemplateResponse("pwd\_add.html", {"request": request})

#Маршрутизация выдачи страницы удаления пароля

@app.get("/pwd\_del.html", response\_class=HTMLResponse)

async def pwd\_del(request: Request) -> HTMLResponse:

return templates.TemplateResponse("pwd\_del.html", {"request": request})

#Маршрутизация выдачи страницы получения данных для входа

@app.get("/get\_pwd\_name.html", response\_class=HTMLResponse)

async def get\_pwd\_name(request: Request) -> HTMLResponse:

return templates.TemplateResponse("get\_pwd\_name.html", {"request": request})

**database.py**

import motor.motor\_asyncio

import string

from bson.objectid import ObjectId

#Инициализация базы данных

MONGO\_DETAILS = "mongodb://localhost:27017"

client = motor.motor\_asyncio.AsyncIOMotorClient(MONGO\_DETAILS)

database = client.curse

#получение таблиц из базы данных

student\_collection = database.get\_collection("students\_collection")

password\_collection = database.get\_collection("passwords\_collection")

# helpers

#Функция проверки строки на буквы

def isletter(name: str) -> bool:

for n in name:

if n.isdigit() or n in string.punctuation:

return False

return True

#Функции преобразования в формат записи в базе

def student\_helper(student) -> dict:

return {

"id": str(student["\_id"]),

"fullname": student["fullname"],

"group": student["group"],

"num": student["num"],

}

def password\_helper(password) -> dict:

return {

"id": str(password["\_id"]),

"group": password["group"],

"num": password["num"],

"password": password["password"],

}

#Функция получения id записи студента из базы по номеру группы и номеру по списку

async def get\_student\_id(group: str, num: int) -> str:

student = await student\_collection.find\_one({"group": group, "num": num})

try: #проверяем что запись успешно найдена

return student\_helper(student)["id"]

except:

return None

#Функция получения всех записей о студентах из базы

async def retrieve\_students() -> list():

students = []

async for student in student\_collection.find():

students.append(student\_helper(student))

return students

#Функция добавления записи студента в базу

async def add\_student(student\_data: dict) -> dict:

student = await student\_collection.insert\_one(student\_data)

new\_student = await student\_collection.find\_one({"\_id": student.inserted\_id})

return student\_helper(new\_student)

#Функция получения записи студента из базы по ID

async def retrieve\_student(id: str) -> dict:

student = await student\_collection.find\_one({"\_id": ObjectId(id)})

try: #проверяем что запись успешно найдена

return student\_helper(student)

except:

return None

#Функция удаления записи студента из базы по ID

async def delete\_student(id: str) -> bool:

student = await student\_collection.find\_one({"\_id": ObjectId(id)})

if student: #Удаляем только если запись существует

await student\_collection.delete\_one({"\_id": ObjectId(id)})

return True

#Функция получения записи пароля из базы по номеру группы и номеру по списку

async def retrieve\_password(group: str, num: int) -> dict:

password = await password\_collection.find\_one({"group": group, "num": num})

try: #проверяем что запись успешно найдена

return password\_helper(password)

except:

return None

#Функция добавления записи пароля в базу

async def add\_password(password\_data: dict) -> dict:

password = await password\_collection.insert\_one(password\_data)

new\_password = await password\_collection.find\_one({"\_id": password.inserted\_id})

return password\_helper(new\_password)

#Функция удаления записи пароля из базы по номеру группы и номеру по списку

async def delete\_password(group: str, num: int) -> bool:

password = await password\_collection.find\_one({"group": group, "num": num})

if password: #Удаляем только если запись существует

await password\_collection.delete\_one({"group": group, "num": num})

return True

**student.py**

from fastapi import APIRouter, Body, HTTPException

from fastapi.encoders import jsonable\_encoder

from pydantic import ValidationError

from ..database import (

add\_student,

delete\_student,

retrieve\_student,

retrieve\_students,

#update\_student,

get\_student\_id,

retrieve\_password,

add\_password,

delete\_password,

isletter,

)

from ..models.student import (

#ErrorResponseModel,

ResponseModel,

StudentSchema,

#UpdateStudentModel,

PasswordSchema,

GroupNumRequest,

)

router = APIRouter()

#Маршрутизация добавления студента

@router.post("/students/", response\_description="Student data added into the database")

async def add\_student\_data(student: StudentSchema = Body(...)) -> dict:

student = jsonable\_encoder(student)

student\_chk = await get\_student\_id(student["group"], student["num"])

if student\_chk: #Если запись о студенте существует возвращаем ошибку

raise HTTPException(status\_code=422, detail="Student already exists")

else:

if isletter(student["fullname"].strip(' ')): #Имя должно состоять только из букв и пробелов

if student["num"] > 0: #номер по списку должен быть натуральным числом

student = jsonable\_encoder(student)

new\_student = await add\_student(student)

return ResponseModel(new\_student, "Student added successfully.")

else: #Сообщаем о некорректном номере

raise HTTPException(status\_code=422, detail="Student num should be a number")

else: #Сообщаем о некорректном имени

raise HTTPException(status\_code=422, detail="Student name should contain only letters")

#Маршрутизация получения списка студентов

@router.get("/students/", response\_description="Students retrieved")

async def get\_students() -> dict:

students = await retrieve\_students()

if students: #Если в базе есть запись возвращаем, иначе выдаем ошибку

return ResponseModel(students, "Students data retrieved successfully")

raise HTTPException(status\_code=404, detail="Empty list returned")

#Маршрутизация получения конкретного студента

@router.get("/student/", response\_description="Student data retrieved")

async def get\_student\_data(group\_num: str) -> dict:

group, num = group\_num.split('/')

id = await get\_student\_id(group, int(num)) #Получаем id записи студента

student = await retrieve\_student(id)

if student: #Если в базе есть запись возвращаем, иначе выдаем ошибку

return ResponseModel(student, "Student data retrieved successfully")

raise HTTPException(status\_code=404, detail="Student doesn't exist")

#Маршрутизация удаления студента

@router.delete("/students/", response\_description="Student data deleted from the database")

async def delete\_student\_data(group\_num: GroupNumRequest) -> dict:

group\_num = group\_num.groupNum

group, num = group\_num.split('/')

id = await get\_student\_id(group, int(num)) #Получаем id записи студента

deleted\_password = await delete\_password(group, int(num)) #Удаляем запись о пароле

deleted\_student = await delete\_student(id) #Удаляем запись о студенте

if deleted\_student: #Если удалили успешно сообщаем, иначе возвращаем ошибку

return ResponseModel(

"Student with ID: {} removed".format(id), "Student deleted successfully"

)

raise HTTPException(status\_code=404, detail="Student doesn't exist")

#Маршрутизация добавления пароля студенту

@router.post("/passwords/", response\_description="Password data added into the database")

async def add\_password\_data(password: PasswordSchema = Body(...)) -> dict:

password = jsonable\_encoder(password)

student\_chk = await get\_student\_id(password["group"], password["num"])

if student\_chk: #Если запись о студенте существует

password\_chk = await retrieve\_password(password["group"], password["num"])

if password\_chk: #Если запись о пароле существует возвращаем ошибку

raise HTTPException(status\_code=422, detail="Password already exists")

else:

if len(password["password"]) > 8: #номер по списку должен быть натуральным числом

new\_password = await add\_password(password) #Добавляем запись о пароле в базу

return ResponseModel(new\_password, "Password added successfully.")

else: #Сообщаем о некорректном номере

raise HTTPException(status\_code=422, detail="Password shoulld be at least 8 symbols")

else: #Сообщаем об отсутствии записи о студенте

raise HTTPException(status\_code=404, detail="Student doesn't exist")

#Маршрутизация получения пароля студента

@router.get("/passwords/", response\_description="Password data retrieved")

async def get\_password\_data(group\_num: str) -> dict:

group, num = group\_num.split('/')

password = await retrieve\_password(group, int(num))

if password: #Если запись о пароле существует возвращаем, иначе выдаем ошибку

return ResponseModel(password, "Password data retrieved successfully")

raise HTTPException(status\_code=404, detail="Password doesn't exist")

#Маршрутизация удаления пароля студента

@router.delete("/passwords/", response\_description="Password data deleted from the database")

async def delete\_password\_data(group\_num: GroupNumRequest) -> dict:

group\_num = group\_num.groupNum

group, num = group\_num.split('/')

deleted\_password = await delete\_password(group, int(num)) #Удаляем запись о пароле

if deleted\_password: #Если удалили успешно сообщаем, иначе возвращаем ошибку

return ResponseModel(

"Password removed", "Password deleted successfully"

) #Иначе выдаем ошибку

raise HTTPException(status\_code=404, detail="Password doesn't exist")

**student.py**

from typing import Optional

from pydantic import BaseModel, Field, validator

#модель для студентов

class StudentSchema(BaseModel):

fullname: str = Field(...) #Имя

group: str = Field(...) #Номер группы

num: int = Field(...) #Номер по списку группы

class Config:

schema\_extra = {

"example": {

"fullname": "John Doe",

"group": "М7О-105С-24",

"num": "9",

}

}

# Модель для паролей

class PasswordSchema(BaseModel):

group: str #Номер группы

num: int #Номер по списку группы

password: str #Пароль

#модель для композитных данных group/num

class GroupNumRequest(BaseModel):

groupNum: str

#Функция формирования ответа на запрос

def ResponseModel(data, message) -> dict:

return {

"data": [data],

"code": 200,

"message": message,

}

**get\_pwd\_name.html**

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Получить данные для входа</title>

<style>

body {

font-family: Arial, sans-serif;

}

.container {

max-width: 600px;

margin: auto;

padding: 20px;

border: 1px solid #ccc;

border-radius: 5px;

box-shadow: 2px 2px 5px rgba(0,0,0,0.1);

}

input[type="text"] {

width: 100%;

padding: 10px;

margin: 10px 0;

border: 1px solid #ccc;

border-radius: 5px;

}

button {

padding: 10px 15px;

background-color: #007bff;

color: white;

border: none;

border-radius: 5px;

cursor: pointer;

}

button:hover {

background-color: #0056b3;

}

.result {

margin-top: 20px;

}

</style>

</head>

<body>

<div class="container">

<h1>Получить данные для входа</h1>

<form id="getDataForm">

<label for="groupNum">Группа/Номер:</label>

<input type="text" id="groupNum" name="groupNum" placeholder="например, A123/456" required>

<button type="submit">Получить данные</button>

</form>

<div class="result">

<h2>Результаты:</h2>

<p><strong>Полное имя студента:</strong> <span id="studentName"></span></p>

<p><strong>Пароль:</strong> <span id="password"></span></p>

</div>

</div>

<script>

document.getElementById('getDataForm').addEventListener('submit', function(event) {

event.preventDefault();

const groupNum = document.getElementById('groupNum').value;

// Запрос данных студента

fetch('/student/student/?group\_num=' + encodeURIComponent(groupNum), {

method: 'GET',

headers: {

'Content-Type': 'application/json',

},

})

.then(response => {

if (!response.ok) {

return response.json().then(err => { throw new Error(err.detail); });

document.getElementById('studentName').innerText = '';

document.getElementById('password').innerText = '';

}

return response.json();

})

.then(studentData => {

document.getElementById('studentName').innerText = studentData.data[0].fullname;

// Запрос данных пароля

return fetch('/student/passwords/?group\_num=' + encodeURIComponent(groupNum), {

method: 'GET',

headers: {

'Content-Type': 'application/json',

},

})

})

.then(response => {

if (!response.ok) {

return response.json().then(err => { throw new Error(err.detail); });

document.getElementById('password').innerText = '';

}

return response.json();

})

.then(passwordData => {

document.getElementById('password').innerText = passwordData.data[0].password;

})

.catch(error => {

alert(error.message);

});

});

</script>

</body>

</html>

**std\_add.html**

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Добавить студента</title>

<style>

body {

font-family: Arial, sans-serif;

}

.container {

max-width: 600px;

margin: auto;

padding: 20px;

border: 1px solid #ccc;

border-radius: 5px;

box-shadow: 2px 2px 5px rgba(0,0,0,0.1);

}

input[type="text"], input[type="number"] {

width: 100%;

padding: 10px;

margin: 10px 0;

border: 1px solid #ccc;

border-radius: 5px;

}

button {

padding: 10px 15px;

background-color: #28a745;

color: white;

border: none;

border-radius: 5px;

cursor: pointer;

}

button:hover {

background-color: #218838;

}

</style>

</head>

<body>

<div class="container">

<h1>Добавить студента</h1>

<form id="addStudentForm">

<label for="fullname">ФИО:</label>

<input type="text" id="fullname" name="fullname" required>

<label for="group">Группа:</label>

<input type="text" id="group" name="group" required>

<label for="num">Номер:</label>

<input type="number" id="num" name="num" required>

<button type="submit">Добавить студента</button>

</form>

</div>

<script>

document.getElementById('addStudentForm').addEventListener('submit', function(event) {

event.preventDefault();

const fullname = document.getElementById('fullname').value;

const group = document.getElementById('group').value;

const num = document.getElementById('num').value;

fetch('/student/students/', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ fullname, group, num }),

})

.then(response => {

if (!response.ok) {

return response.json().then(err => { throw new Error(err.detail); });

}

return response.json();

})

.then(data => {

alert(data.message);

})

.catch(error => alert('Ошибка: ' + error.message));

});

</script>

</body>

</html>

**std\_del.html**

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Удалить студента</title>

<style>

body {

font-family: Arial, sans-serif;

}

.container {

max-width: 600px;

margin: auto;

padding: 20px;

border: 1px solid #ccc;

border-radius: 5px;

box-shadow: 2px 2px 5px rgba(0,0,0,0.1);

}

input[type="text"] {

width: 100%;

padding: 10px;

margin: 10px 0;

border: 1px solid #ccc;

border-radius: 5px;

}

button {

padding: 10px 15px;

background-color: #dc3545;

color: white;

border: none;

border-radius: 5px;

cursor: pointer;

}

button:hover {

background-color: #c82333;

}

</style>

</head>

<body>

<div class="container">

<h1>Удалить студента</h1>

<form id="deleteStudentForm">

<label for="groupNum">Группа/Номер:</label>

<input type="text" id="groupNum" name="groupNum" placeholder="например, A123/456" required>

<button type="submit">Удалить студента</button>

</form>

</div>

<script>

document.getElementById('deleteStudentForm').addEventListener('submit', function(event) {

event.preventDefault();

const groupNum = document.getElementById('groupNum').value;

fetch('/student/students/', {

method: 'DELETE',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({groupNum}),

})

.then(response => {

if (!response.ok) {

return response.json().then(err => { throw new Error(err.detail); });

}

return response.json();

})

.then(data => {

alert(data.message);

})

.catch(error => alert('Ошибка: ' + error.message));

});

</script>

</body>

</html>

**pwd\_add.html**

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Добавить пароль</title>

<style>

body {

font-family: Arial, sans-serif;

}

.container {

max-width: 600px;

margin: auto;

padding: 20px;

border: 1px solid #ccc;

border-radius: 5px;

box-shadow: 2px 2px 5px rgba(0,0,0,0.1);

}

input[type="text"], input[type="number"] {

width: 100%;

padding: 10px;

margin: 10px 0;

border: 1px solid #ccc;

border-radius: 5px;

}

button {

padding: 10px 15px;

background-color: #28a745;

color: white;

border: none;

border-radius: 5px;

cursor: pointer;

}

button:hover {

background-color: #218838;

}

</style>

</head>

<body>

<div class="container">

<h1>Добавить пароль</h1>

<form id="addPasswordForm">

<label for="group">Группа:</label>

<input type="text" id="group" name="group" required>

<label for="num">Номер:</label>

<input type="number" id="num" name="num" required>

<label for="password">Пароль:</label>

<input type="text" id="password" name="password" required>

<button type="submit">Добавить пароль</button>

</form>

</div>

<script>

document.getElementById('addPasswordForm').addEventListener('submit', function(event) {

event.preventDefault();

const group = document.getElementById('group').value;

const num = document.getElementById('num').value;

const password = document.getElementById('password').value;

fetch('/student/passwords/', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ group, num, password }),

})

.then(response => {

if (!response.ok) {

return response.json().then(err => { throw new Error(err.detail); });

}

return response.json();

})

.then(data => {

alert(data.message);

})

.catch(error => alert('Ошибка: ' + error.message));

});

</script>

</body>

</html>

**pwd\_del.html**

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Удалить пароль</title>

<style>

body {

font-family: Arial, sans-serif;

}

.container {

max-width: 600px;

margin: auto;

padding: 20px;

border: 1px solid #ccc;

border-radius: 5px;

box-shadow: 2px 2px 5px rgba(0,0,0,0.1);

}

input[type="text"] {

width: 100%;

padding: 10px;

margin: 10px 0;

border: 1px solid #ccc;

border-radius: 5px;

}

button {

padding: 10px 15px;

background-color: #dc3545;

color: white;

border: none;

border-radius: 5px;

cursor: pointer;

}

button:hover {

background-color:#c82333;

}

</style>

</head>

<body>

<div class="container">

<h1>Удалить пароль</h1>

<form id="deletePasswordForm">

<label for="groupNum">Группа/Номер:</label>

<input type="text" id="groupNum" name="groupNum" placeholder="например, A123/456" required>

<button type="submit">Удалить пароль</button>

</form>

</div>

<script>

document.getElementById('deletePasswordForm').addEventListener('submit', function(event) {

event.preventDefault();

const groupNum = document.getElementById('groupNum').value;

fetch('/student/passwords/', {

method: 'DELETE',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({groupNum}),

})

.then(response => {

if (!response.ok) {

return response.json().then(err => { throw new Error(err.detail); });

}

return response.json();

})

.then(data => {

alert(data.message);

})

.catch(error => alert('Ошибка: ' + error.message));

});

</script>

</body>

</html>

**main.html**

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Главная страница</title>

<style>

body {

display: flex;

font-family: Arial, sans-serif;

}

nav {

width: 200px;

background-color: #f4f4f4;

padding: 15px;

box-shadow: 2px 2px 5px rgba(0,0,0,0.1);

}

nav a {

display: block;

margin: 10px 0;

padding: 10px;

text-decoration: none;

color: #333;

border-radius: 5px;

transition: background-color 0.3s;

}

nav a:hover {

background-color: #ddd;

}

.content {

padding: 20px;

flex-grow: 1;

}

.refresh-button {

float: right;

padding: 10px 15px;

background-color: #007bff;

color: white;

border: none;

border-radius: 5px;

cursor: pointer;

}

.refresh-button:hover {

background-color: #0056b3;

}

.student-list {

margin-top: 20px;

}

</style>

</head>

<body>

<nav>

<a href="/std\_add.html">Добавить студента</a>

<a href="/std\_del.html">Удалить студента</a>

<a href="/pwd\_add.html">Добавить пароль</a>

<a href="/pwd\_del.html">Удалить пароль</a>

<a href="/get\_pwd\_name.html">Получить данные для входа</a>

</nav>

<div class="content">

<h1>Добро пожаловать!</h1>

<button class="refresh-button" onclick="refreshStudents()">Обновить список студентов</button>

<div class="student-list" id="studentList">

<h2>Список студентов:</h2>

<ul id="students"></ul>

</div>

</div>

<script>

function refreshStudents() {

fetch('/student/students/', {

method: 'GET',

headers: {

'Content-Type': 'application/json',

},

}).then(response => {

if (!response.ok) {

return response.json().then(err => { throw new Error(err.detail); });

}

return response.json();

}).then(data => {

const studentList = document.getElementById('students');

studentList.innerHTML = 'Имя: ' + '\tГруппа: ' + '\tНомер: '; // Очистка списка перед обновлением

data.data[0].forEach(student => {

const li = document.createElement('li');

li.textContent = student.fullname + '\t' + student.group + '\t' + student.num;

studentList.appendChild(li);

});

})

.catch(error => console.error('Ошибка:', error));

}

</script>

</body>

</html>