МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Нереляционные базы данных»

Тема: Телеграм-бот для вывода данных из таблиц

Студентка гр. 0382 _____ Чегодаева Е.А.

Студент гр. 0382 _____ Сергеев Д.А.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2023

ЗАДАНИЕ

Студентка Чегодаева Е.А.

Студент Сергеев Д.А.

Группа 0382

Тема работы: Телеграм-бот для вывода данных из таблиц

Исходные данные:

Разработать web-приложение “Панель администрирования” для управления данными телеграм-бота: добавление и редактирование таблиц, управление пользователями, просмотр статистики обращений к боту.

Содержание пояснительной записки:

1. «Содержание»
2. «Введение»
3. «Сценарий использования»
4. «Модель данных»
5. «Разработка приложения»
6. «Вывод»
7. «Будущее развитие решения»
8. «Приложения»

Предполагаемый объем пояснительной записки:

Не менее 30 страниц.

Дата выдачи задания: 26.09.2023

Дата сдачи реферата: 21.12.2023

Дата защиты реферата: 21.12.2023

Студентка гр. 0382 Чегодаева Е.А.

Студент гр. 0382 _____ Сергеев Д.А.

Преподаватель _____ Заславский М.М.

АННОТАЦИЯ

Разработано приложение, являющееся инструментом для администрирования telegram-бота, которое осуществляет вывод данных из таблиц. Приложение позволяет отслеживать статистику обращений к боту, просматривать сведения о студентах, зарегистрированных в системе, а также осуществлять контроль над таблицами в формате docs.google. Хранение данных в “Панели администрирования” реализовано посредством документно-ориентированной базы данных — MongoDB. Приложение поддерживает опцию массового импорта/экспорта в целях возможности проведения резервного копирования данных.

SUMMARY

Has been developed an application, which used as a tool for administering a Telegram bot, that retrieves data from tables. The application allows tracking statistics of interactions with the bot, viewing information about students registered in the system, and controlling tables in Google Docs format. Data storage in the "Administration Panel" is implemented using a document-oriented database - MongoDB. The application supports the option of mass import/export for the purpose of data backup.

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

1.1. Актуальность решаемой проблемы

Таблицы — удобный способ предоставления информации об успеваемости студентов. В учебный период приходится контролировать изменения во многих таблицах по разным предметам, с учетом того, что преподаватели могут хранить ссылки на таблицы в разных источниках. Также, чаще всего, таблицы содержат данные всех студентов, что можно отнести к нарушению конфиденциальности. Для упрощения этой задачи можно использовать мессенджер telegram, поддерживающий автоматизированную систему ведения диалога (бот), который будет предоставлять студенту сведения об его успеваемости.

1.2. Постановка задачи

Создать бота, который сможет выполнять запрос “покажи мне строку по запросу А из таблицы Б”. От студентов скрыта вся таблица успеваемости и демонстрируются только собственные данные студента. Приложение должно обладать: веб-интерфейсом для администрирования и настроек, страницей со списком подключенных таблиц, страницей для настройки отдельной таблицы, статистикой доступа. Реализовать функциональность массового импорта/экспорта данных.

1.3. Предлагаемое решение

Реализация первого этапа разработки решения заключается в создании web-приложения для администрирования системы. Система должна базироваться на базе данных MongoDB и включать всю требуемую функциональность.

1.4. Качественные требования к решению

Разработать web-приложение, предоставляющее пользователю практичный интерфейс для взаимодействия и управления таблицами. Все параметры для функционирования системы должны храниться в удаленной базе данных. Система должна обладать возможностями создания и внедрения резервной копии данных.

2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

2.1. Макет UI

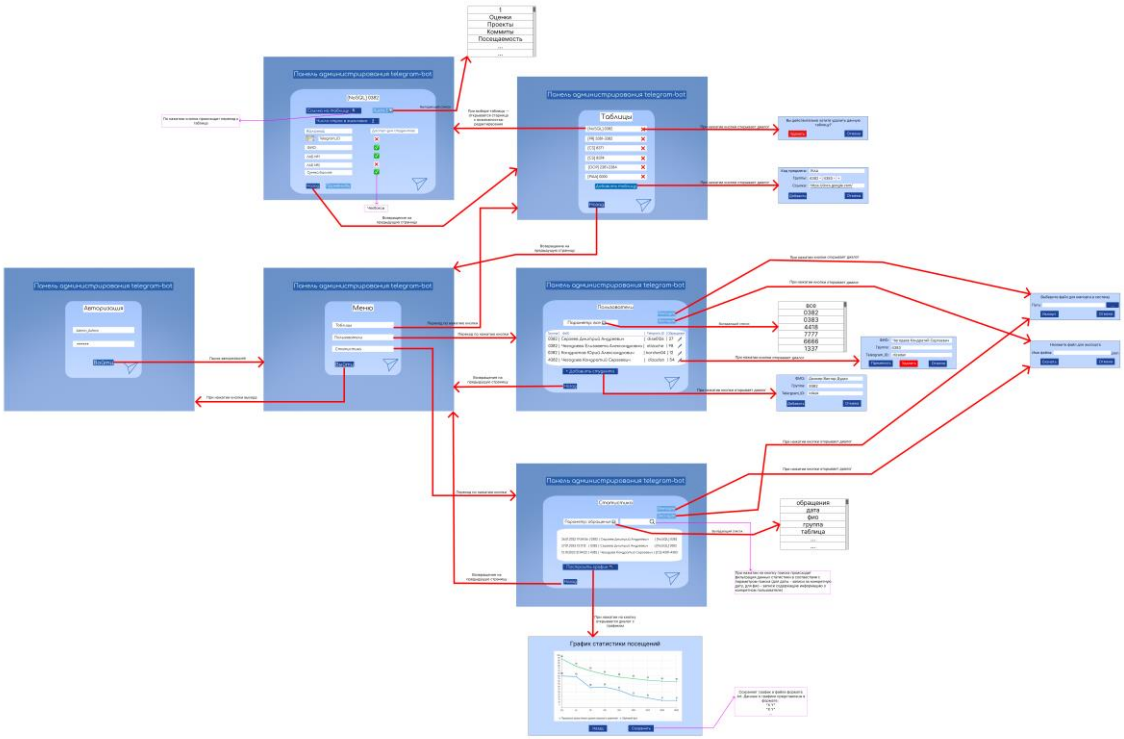

Рисунок 1 — Макет панели администрирования

Рисунок 2 — Макет взаимодействия с telegram-bot

2.2. Сценарии использования для задачи

I. **Telegram-bot** — Получение студентом данных из таблицы:

1) Задать команду */start* боту в телеграмм;
2) Если данные о пользователе есть в системе – бот пришлёт сообщение о возможных таблицах для просмотра;

2*) Если данных о пользователе в системе нет – будет получено сообщение, уведомляющее об этом, и предложен путь решения этой проблемы;

3) Пользователь выбирает интересующую из доступных таблиц и пишет её (указанный) номер;

4) Бот присылает доступные данному студенту колонки таблицы и значения в них;
5) Далее бот предлагает выбрать другую таблицу командой /tables;
6) В случае выполнения пользователем п. 5 вновь реализуется алгоритм, начиная с п. 2 .

II. **Панель администрирования** — Добавление таблицы в систему:

1) Администратор авторизируется;
2) Открывается меню с возможными вариантами взаимодействия в панели администрования;
3) Выбирается кнопка *«Таблицы»*;
4) Открывается страница со всеми таблицами системы;
5) Внизу страницы необходимо нажать кнопку *«Добавить таблицу»;*
6) При нажатии - открывается диалоговое окно, которое требует внесение определённых сведений о таблице:
   - Код предмета (сокращённое название)
   - Группы (одна или несколько)
   - Ссылка на docs.google (таблица)
   - Сведения о листах
7) После введения всех данных следует нажать кнопку «*Добавить*» для сохранения таблицы в системе (или *«Отменить»* в ином случае)*;*
8) Следом, диалоговое окно закрывается.

III. **Панель администрирования** — Изменить поля таблицы, доступные студентам:

1) Администратор авторизируется;
2) Открывается меню с возможными вариантами взаимодействия в панели администрования;
3) Выбирается кнопка *«Таблицы»*;
4) Открывается страница со всеми таблицами системы;

5) Необходимо выбрать таблицу, в которую необходимо внести изменения;
6) Открывается страница листа конкретной таблицы, с содержащимися в ней колонками и статусом (“*доступно для студентов*” // “*недоступно для студентов*”);
7) Посредством “*галочки*” или “*крестика*” в чекбоксе изменяется статус доступа;
8) После завершения редактирования статуса одной или нескольких колонок следует нажать кнопку «*Применить»;*

IV. **Панель администрирования** — Просмотр графики статистики обращения по конкретному диапазону дат:

1) Администратор авторизируется;
2) Открывается меню с возможными вариантами взаимодействия в панели администрирования;
3) Выбирается кнопка *«Статистика»*;
4) Открывается страница со всеми обращениями;
5) Администратор настраивает параметры, соответствующие запросу – в данном случае:
    - Необходимо ввести диапазон дат в формате “*ОТ*” и “*ДО*”
    - Нажать на кнопку «*Применить фильтры*»
6) После настройки будут выведены все обращения к боту, в соответствующем диапазоне;
7) По фильтрованным данным при нажатии кнопки «Построить график» будет открыто диалоговое окно с графиком;
8) График можно сохранить или вернуться назад – к странице со статистикой;

V. **Панель администрирования** с Добавление студента в систему:

1) Администратор авторизируется;

2) Открывается меню с возможными вариантами взаимодействия в панели администрирования;
3) Выбирается кнопка *«Пользователи»*;
4) Открывается страница со всеми пользователями;
5) Внизу страницы необходимо нажать кнопку *«+ Добавить студента»;*
6) При нажатии - открывается диалоговое окно, которое требует внесение определённых сведений о студенте:
    - ФИО
    - Группа
    - Telegram_ID
7) После введения всех данных следует нажать кнопку «*Добавить*» для сохранения в системе (или «*Отменить*» в ином случае);
8) Следом, диалоговое окно закрывается.

VI. **Панель администрирования** — Массовый импорт:

1) Администратор авторизируется;
2) Открывается меню с возможными вариантами взаимодействия в панели администрирования;
3) Выбирается кнопка *«Импорт»*;
4) При нажатии - открывается диалоговое окно, которое требует внесения пути файла для импорта в систему;
5) После введения всех данных следует нажать кнопку *«Импорт»* для сохранения в системе (или «*Отменить*» в ином случае);
6) Следом, диалоговое окно закрывается.

2.3. Вывод о том, какие операции будут преобладать

Исходя из сценариев использования, можно сделать вывод о том, что в приложении преобладают запросы записи для панели администрирования и чтения в telegram-bot.

3. МОДЕЛЬ ДАННЫХ

3.1. Нереляционная модель данных

1. Графическое представление

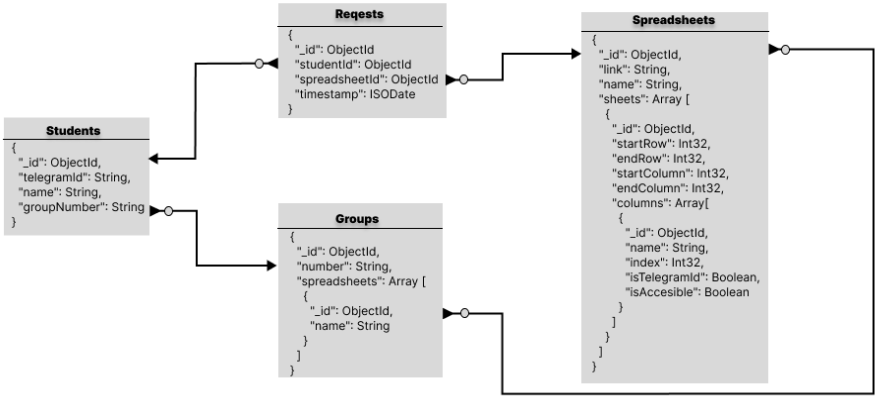

Рисунок 3 — Схема для нереляционной модели БД

2. Описание назначений коллекций, типов данных и сущностей

Коллекция “Students” содержит документы, в которых хранится информация о каждом студенте в системе:

- *_id* — Идентификатор студента;
- *telegramId* — ID в телеграм;
- *name* — ФИО;
- *groupNumber* — Группа (номер группы, согласно встроенной коллекции "Groups").

Коллекция “Requests” содержит документы, в которых хранится информация о каждом обращении к телеграм-боту:

- *_id* — Идентификатор обращения;
- *student* — Студент (идентификатор, согласно коллекции "Students");
- *spreadsheetId* — Таблица, к которой был совершен запрос (идентификатор таблицы, согласно "Spreadsheets");
- *timestamp* — Время обращения.

Коллекция “Groups” содержит информацию о группах в системе:

- *_id* — Идентификатор группы;
- *number* — Номер группы;
- *spreadsheets* — Массив, содержащий информацию о таблицах, которые привязаны к конкретной группе (*spreadsheets._id* и *name* - Идентификатор и Название таблицы соответственно).

Коллекция “Spreadsheets” содержит документы, в которых хранится информация о существующих в системе таблицах:

- *_id* — Идентификатор таблицы;
- *link* — Ссылка на таблицу;
- *name* — Имя таблицы;
- *sheets* — Массив (*подколлекция*), в котором хранится информация о листах текущей таблицы:
    - *sheets._id* — Идентификатор листа;
    - *startRow* — Номер первой строки;
    - *endRow* — Номер последней строки;
    - *startColumn* — Номер первого столбца;
    - *endColumn* — Номер последнего столбца;
    - *columns* — Массив (*подколлекция*), хранящий данные о колонках на текущем листе:
        - *_id* — Идентификатор колонки;
        - *name* — Название столбца;
        - *index* — Порядковый номер колонки (слева направо);
        - *isTelegramId* — Флаг, определяющий содержит ли в данная колонка telegram_ID студента;
        - *isAccesible* — Флаг, определяющий доступ колонки для студента.

3. Оценка удельного объема информации

Пусть n — количество таблиц одной группы;

m — количество листов в одной таблице;

k — количество колонок на одном листе.

Таблица 1 — Типы данных каждой коллекции

	Students	Requests	Groups	Spreadsheets	Weight
ObjectId	1	3	1 + n	1 + m + m*k	12 bytes
String	3	-	1 + n	2 + m * k	1 byte per symbol + 1
ISODate	-	1	-	-	8 bytes
Int32	-	-	-	4m + m * k	4 bytes
Boolean	-	-	-	2(m*k)	2 bytes

Предположим, что число студентов в одной группе Ns; групп на 1 потоке МОЕВМ — Ng; всего курсов (курс == поток) — 6 (бакалавриат+магистратура) => $6*Ns*Ng$ студентов;

На одном курсе Np предметов => $6*Np$ таблиц на одну группу => $6*Np*Ng$ — таблиц всего.

Пусть каждая таблица имеет 2 листа с 10-ю колонками.

Сделаем предположение о том, что в день поступает около 3 обращений от ОДНОГО студента к боту в telegram => $3*6*Ns*Ng$ обращений ото всех студентов.

Итог за 1 месяц представлен в таблице 2.

Таблица 2 — Объём коллекции

	Students	Requests	Groups	Spreadsheets
Volume	$6*Ns*Ng$	$31*(3*(6*(Ns*Ng)))$	$6*Ng$	$6*Np*Ng$
Abbreviation	T_{ST}	T_{RQ}	T_{GR}	T_{SS}

Предположим, что максимальная длина одной строки 255 символов.

Удельный объем всех коллекций базы данных представлен в таблице 3.

Расчётные выкладки:

Groups: $((12+256) * (1+Np)) * T_{GR} = 268 * (1+Np) * T_{GR}$

Spreadsheets: $(12 * (1+2+2*10)+256(2+2 * 10) +4 * (2*4+2*10)+ 2(2 * (2 * 10))) * T_{SS} = 6100 * T_{SS}$

Таблица 3 — Удельный объём коллекций

	Students	Requests	Groups	Spreadsheets
Weight	$780 * T_{ST}$	$44 * T_{RQ}$	$268 * (1+ Np) * T_{GR}$	$6100 * T_{SS}$

Итоговая формула:

$W = [780 * 6 * Ns * Ng] + [44 * 31 * (3 * (6 * (Ns * Ng)))] + [268 * (1+Np) * 6 * Ng] + [6100 * 6 * Np * Ng]$

Пусть Ng (число групп) равно 5 — среднее число групп на одном потоке;

Пусть Np (число предметов на курсе) равно 15 — среднее число предметов за 2 семестра. =>

$W = [780 * 6 * Ns * 5] + [44 * 31 * (3 * (6 * (Ns * 5)))] + [268 * (1+15) * 6 * 5] + [6100 * 6 * 15 * 5] = 146160\,Ns + 2873640$ (*)

Избыточность модели:

В коллекции “Groups” содержатся данные о таблицах (ID и название) — хранить названия избыточно, но это позволяет упростить вывод доступных таблиц студенту, что является частым случаем => оправдывается дублирование.

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

На основании выражения (*) можно судить о том, что рост числа студентов приводит к возрастанию числа групп, что в свою очередь - вызывает увеличение количества таблиц, а также и рост обращений. Таким образом,

можно говорить о *линейном росте* содержимого коллекций в зависимости от числа студентов, занесённых в систему.

Чистый объём данных:

Таблица 4 — Чистый объём коллекций

	Students	Requests	Groups	Spreadsheets
Weight	$780 * T_{ST}$	$44 * T_{RQ}$	$(256 + 12(1 + Np)) * T_{GR}$	$6100 * T_{SS}$

Так как дублирование имеется только в коллекции “Groups” => сравнение удельного и чистого объема имеет смысл только для этой коллекции:

$$k_{duplication} = \frac{268 * (1 + Np) * T_{GR}}{(256 + 12 * (1 + Np)) * T_{GR}}$$

Пусть $Np = 9$:

$k_{duplication} = 7.1$ — коэффициент дублирования в построенной модели.

Значение коэффициента дублирования в построенной модели напрямую зависит от заданной длины строки — в ходе расчёта длина строки объективно превышает ожидаемые значения (для запаса), ввиду этого коэффициент имеет высокое значение.

4. Запросы к модели, с помощью которых реализуются сценарии использования

I. Получение студентом данных из таблицы:

- `db.Students.find({telegramId: "dizadan"}) -> (!!!)`
- `db.Groups.find({number: (!!!)}) -> (...)`
- `db.Spreadsheets.find({name: "(...)", "sheets.columns.isTelegramId": true})`

=> Используется 3 коллекции, 3 запроса

II. Добавление таблицы в систему:

- *db.Spreadsheets.insertOne({ link: "doodle.doc", name: "[OOP] 3383", sheets: [{_id: new ObjectId(), startRow: 0, endRow: 2, startColumn: 0, endColumn: 5, columns: [{_id: new ObjectId(), name: "ФИО", index: 0, isTelegramId: false, isAccesible: true},{_id: new ObjectId(), name: "TelegramId", index: 1, isTelegramId: true, isAccesible: false}],}], })*

=> Используется 1 коллекция, 1 запрос

III. Изменить поля таблицы, доступные студентам:

- *db.Spreadsheets.find({name: "[NoSQL] 0382"})*
- *db.Spreadsheets.updateOne({name: "[NoSQL] 0382"},{$set: {"sheets.$[testo].columns.$[elem].isAccesible": false}},{arrayFilters: [{"testo._id": ObjectId("aaa527df62d4a2a9cab8713a")},{"elem._id": ObjectId("fff527df62d4a2a9cab8713a")}]})*

=> Используется 1 коллекция, 2 запроса

IV. Просмотр графики статистики обращения по конкретному диапазону дат:

- *db.Students.find({name: "Chegodaeva Elizaveta Alexandrovna"}) -> (!!!)*
- *db.Requests.find({timestamp: "22.10.2023", studentId: "(!!!)"})*

=> Используется 2 коллекции, 2 запроса

V. Добавление студента в систему:

- *db.Students.insertOne({"telegramId": "dise0126","name": "Sergeev Dmitriy Andreevich","groupNumber": "3383" })*

=> Используется 1 коллекция, 1 запрос

VI. Массовый импорт:

- *db.Students.deleteMany({})*
- *db.Students.insertMany({students})*

=> Используется 1 коллекция, 2 запроса

3.2. Реляционная модель данных

1. Графическое представление

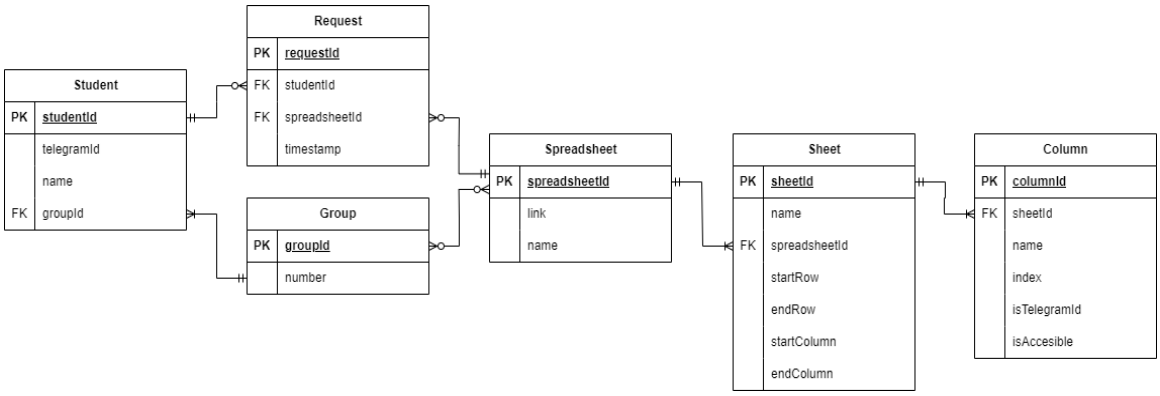

Рисунок 4 — Схема для реляционной модели БД

2. Описание назначений коллекций, типов данных и сущностей

Таблица “Student” хранит информацию о студентах: идентификатор, Телеграм-идентификатор, ФИО, идентификатор группы.

Таблица “Group” хранит информацию о группах: идентификатор, номер группы.

Таблица “Request” хранит информацию о запросах студентов к таблицам: идентификатор, идентификатор студента, идентификатор таблицы, дата и время запроса.

Таблица “Spreadsheet” хранит информацию о таблицах: идентификатор, ссылка на таблицу, название таблицы.

Таблица “Sheet” хранит информацию о листах таблицы: идентификатор, название листа, диапазон расположения данных на листе (первая строка,

последняя строка, первый столбец, последний столбец), идентификатор таблицы.

Таблица “Column” хранит информацию о колонках из диапазона данных листа: идентификатор, название колонки, идентификатор листа, порядковый номер колонки (слева направо), флаг доступности, флаг содержания Телеграм-идентификатора.

3. Оценка удельного объема информации

Будем считать, что символ строка занимает 512 байт (символ — 2 байта, максимальная длина строки — 256 байт), идентификаторы представлены 8-байтовыми целыми числами, прочие целочисленные атрибуты занимают 4 байта, дата и время занимают 8 байт, булевый тип занимает 1 байт.

Тогда размеры сущностей: Student — $8 + 512 + 512 + 8 = 1040$ байт, Group — $8 + 4 = 12$, Request — $8 + 8 + 8 + 8 = 32$ байта, Spreadsheet — $8 + 512 + 512 = 1032$ байта, Sheet — $8 + 512 + 4 + 4 + 4 + 4 + 8 = 544$ байта, Column — $8 + 8 + 512 + 4 + 1 + 1 = 532$ байта. Также для реализации связи Many-To-Many между Spreadsheet и Group понадобится связующая таблица с двумя идентификаторами (16 байт).

Если каждая таблица имеет 2 листа с 10-ю колонками, то совокупный вес Spreadsheet равен $1032 + 2 * 544 + 20 * 532 = 12760$ байт.

Также, учитывая предположение, что у каждой группы 6 таблиц получаем дополнительно $16 * 6 = 96$ байт для каждой группы.

Используя обозначения, введённые для нереляционной модели:

Таблица 5 — Удельный объём таблиц

	Students	Requests	Groups	Spreadsheets
Weight	$1040 * T_{ST}$	$32 * T_{RQ}$	$108 * T_{GR}$	$12760 * T_{SS}$

Итоговая формула:

$W = [1040 * 6 * Ns * Ng] + [32 * 31 * (3 * (6 * (Ns * Ng)))] + [108 * 6 * Ng] + [12760 * 6 * Np * Ng]$

Пусть Ng, Np — аналогично нереляционной модели =>

$W = [1040 * 6 * Ns * 5] + [32 * 31 * (3 * (6 * (Ns * 5)))] + [108 * 6 * 5] + [12760 * 6 * 15 * 5] = 120480\,Ns + 5745240$

При увеличении количества студентов объем модели будет расти линейно.

Избыточность модели:

Рассмотрим - какие данные избыточны в модели:

- sheetId дублируется в Column, увеличивая объем на $8*C$ байт, где C - количество колонок
- spreadsheetId дублируется в Sheet, увеличивая объем на $8*S$ байт, где S - количество листов
- groupId и spreadsheetId используются для создания связи, увеличивая объем при добавлении группы в худшем случае на $16*T$ байт, где T - количество таблица, и при добавлении таблицы в худшем случае на $6*G$ байт

Допустим, что количество студентов в группе — 30, а предметов на потоке 10. Тогда объем данных без дублирования будет равен:

$$\left(1040 + 31 * 3 * 32 + 1/30 * 12 + 1/3 * (12760 - 22 * 8)\right) * T_{ST},$$

а с дублированием:

$$(1040 + 31 * 3 * 32 + 1/30 * 108 + 1/3 * 12760) * T_{ST}.$$

Получаем коэффициент дублирования: $K_{duplication} = 1.0075$

4. Запросы к модели, с помощью которых реализуются сценарии использования

I. Получение студентом данных из таблицы:

- *SELECT * FROM Student WHERE Student.telegramId = <telegramId>* — проверка наличия пользователя
- *SELECT * FROM Group AS g JOIN SpreadsheetGroup AS sg ON g.groupId = sg.groupId JOIN Spreadsheet AS s ON sg.spreadsheetId = s.spreadsheetId WHERE g.groupId = <groupId>* — запрос доступных таблиц
- *SELECT * FROM Spreadsheet AS ss JOIN Sheet AS s ON ss.spreadsheetId = s.spreadsheetId JOIN Column AS c ON c.sheetId = s.sheetId WHERE ss.spreadsheetId = <spreadsheetId>* — запрос данных о таблице

=> Используется 3 коллекции, 3 запроса

II. Добавление таблицы в систему:

- *INSERT INTO Column (...) VALUES (...), ...* — добавление информации о колонках
- *INSERT INTO Sheet (...) VALUES (...), ...* — добавление информации о листах
- *INSERT INTO Spreadsheet (...) VALUES (...), ...* — добавление информации о листах

=> Используется 3 коллекции, 3 запроса

III. Изменить поля таблицы, доступные студентам:

- *SELECT * FROM Spreadsheet AS ss JOIN Sheet AS s ON ss.spreadsheetId = s.spreadsheetId JOIN Column AS c ON c.sheetId = s.sheetId WHERE ss.spreadsheetId = <spreadsheetId>* — запрос данных о колонках
- *INSERT INTO Column (...) VALUES (...), ...* — изменение информации о доступности колонок

=> Используется 2 коллекции, 2 запроса

IV. Просмотр графики статистики обращения по конкретному диапазону дат:

- *SELECT * FROM Request WHERE <условие запроса>* — получение

=> Используется 1 коллекция, 1 запрос

V. Добавление студента в систему:

- *INSERT INTO Column (...) VALUES (...), ...* — изменение информации о доступности колонок
- *INSERT INTO Student (...) VALUES (...)* — добавление информации о студенте

=> Используется 2 коллекции, 2 запроса

3.3. Сравнение моделей

1. Удельный объем информации

На основе данных, представленных в таблицах 3 и 5 — сущности обоих моделей совпадают, исходя из этого:

- Сущность Students: удельный объём реляционной модели превышает значение нереляционной

  780 байт (NoSQL) vs 1040 (SQL)
- Сущность Requests: объём нереляционной модели превышает значение реляционной

  44 байт (NoSQL) vs **32 (SQL)**
- Сущность Groups: в данном случае, ввиду особенностей реализации сущности в MongoDB объём представлен формулой, имеющей зависимость от количества предметов в отличии от варианта для SQL. Основываясь на поверхностной оценке, можно сказать о том, что удельный объём реляционной модели ниже значения нереляционной

  (256+12*(1+ Np)) байт (NoSQL) vs **108 (SQL)**

- Сущность Spreadsheets: удельный объём нереляционной модели меньше объёма реляционной

  6100 байт (NoSQL) vs 12760 (SQL)

Итоговые формулы удельного объёма совпадают для обоих моделей.

2. Запросы

Количество запросов для совершения сценариев в зависимости от числа объектов в БД и прочих параметров:

- Получение студентом данных из таблицы: число запросов — равно

  3 (NoSQL) vs 3 (SQL)
- Добавление таблицы в систему: количество запросов для нереляционной модели меньше, чем для реляционной

  1 (NoSQL) vs 3 (SQL)
- Изменение доступности колонок студентам: число запросов — равно

  2 (NoSQL) vs 2 (SQL)
- Просмотр статистики обращений: количество запросов для нереляционной модели больше, чем для реляционной

  2 (NoSQL) vs **1 (SQL)**
- Добавление студентов: число запросов — равно

  1 (NoSQL) vs 1 (SQL)

Количество задействованных коллекций:

- Получение студентом данных из таблицы: число коллекций — равно

  3 (NoSQL) vs 3 (SQL)
- Добавление таблицы в систему: количество коллекций для нереляционной модели меньше, чем для реляционной

  1 (NoSQL) vs 3 (SQL)
- Изменение доступности колонок студентам: число задействованных коллекций для нереляционной модели меньше, чем для реляционной

1 (NoSQL) vs 2 (SQL)

- Просмотр статистики обращений: количество коллекций для реляционной модели меньше, чем для нереляционной

  2 (NoSQL) vs **1 (SQL)**

- Добавление студентов: число задействованных коллекций для нереляционной модели меньше, чем для реляционной

  1 (NoSQL) vs 2 (SQL)

3. Вывод

На основе оценки, представленной выше, можно сделать вывод о том, что для данного проекта:

- В рамках удельного объёма - модели примерно одинаковы (стоит отметить, что коэффициент дублирования в нереляционной модели значительно превышает значение реляционной модели);
- По критерию числа запросов к БД - модели примерно одинаковы;
- По критерию количества задействованных коллекций - лучшие показатели у нереляционной модели.

Заметим, что нереляционная реализация (**MongoDB**) позволяет уменьшить число сущностей за счёт вложенности, что приведёт к возможности совершать меньше обращений к БД.

4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

4.1. Краткое описание

В ходе выполнения работы было реализована база данных, веб-приложение и сервер:

- Сервер представляет собой python-приложение, которое получает запросы от клиента, делает запрос к базе данных и возвращает требуемую информацию.
- Клиент — это web-приложение, которое делает запросы к серверу, а затем отображает полученную информацию в доступном и понятном виде.
- База данных — синтетический набор данных, а также команды для взаимодействия.

4.2. Использованные технологии

- Сервер: FastAPI (Python 3.9)
- Клиент: React (JavaScript ES6), HTML, CSS
- База данных: MongoDB
- Развёртывание приложения: Docker Compose

4.3. Снимки экрана приложения

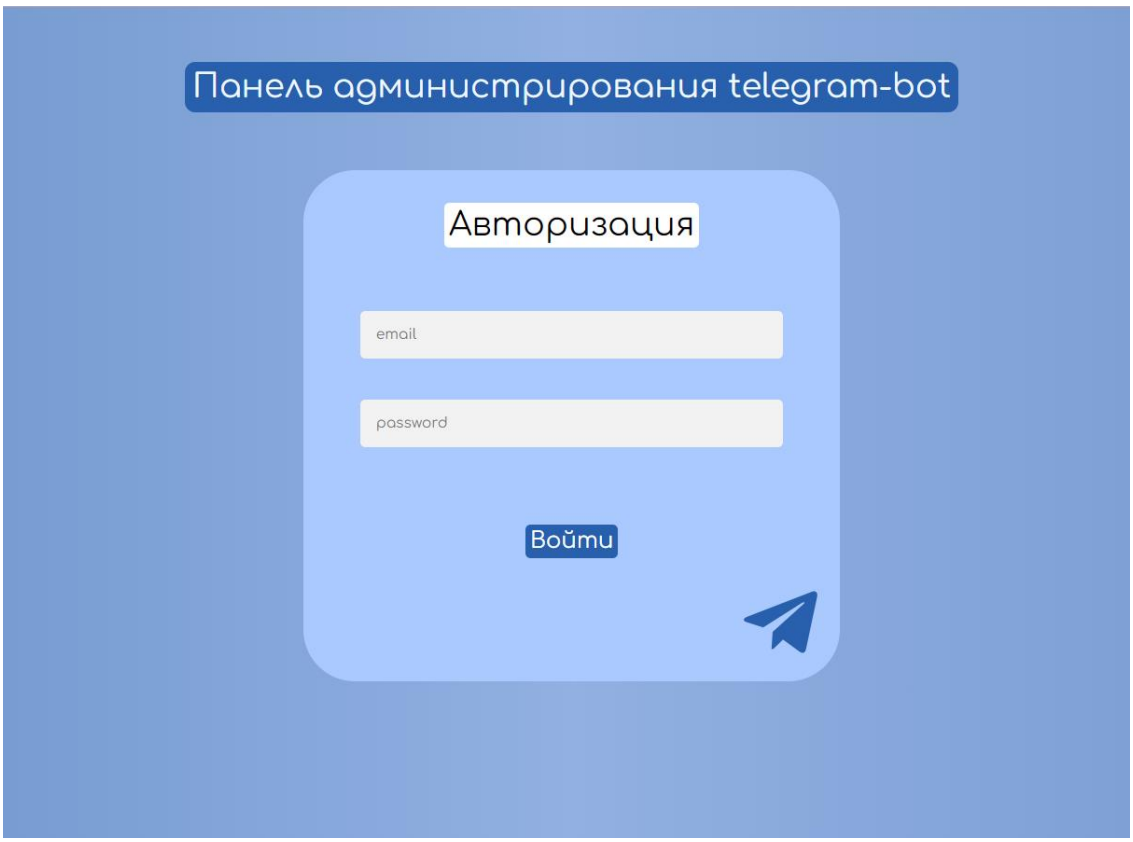

Рисунок 5 — Страница «Авторизация»

Рисунок 6 — Страница «Меню»

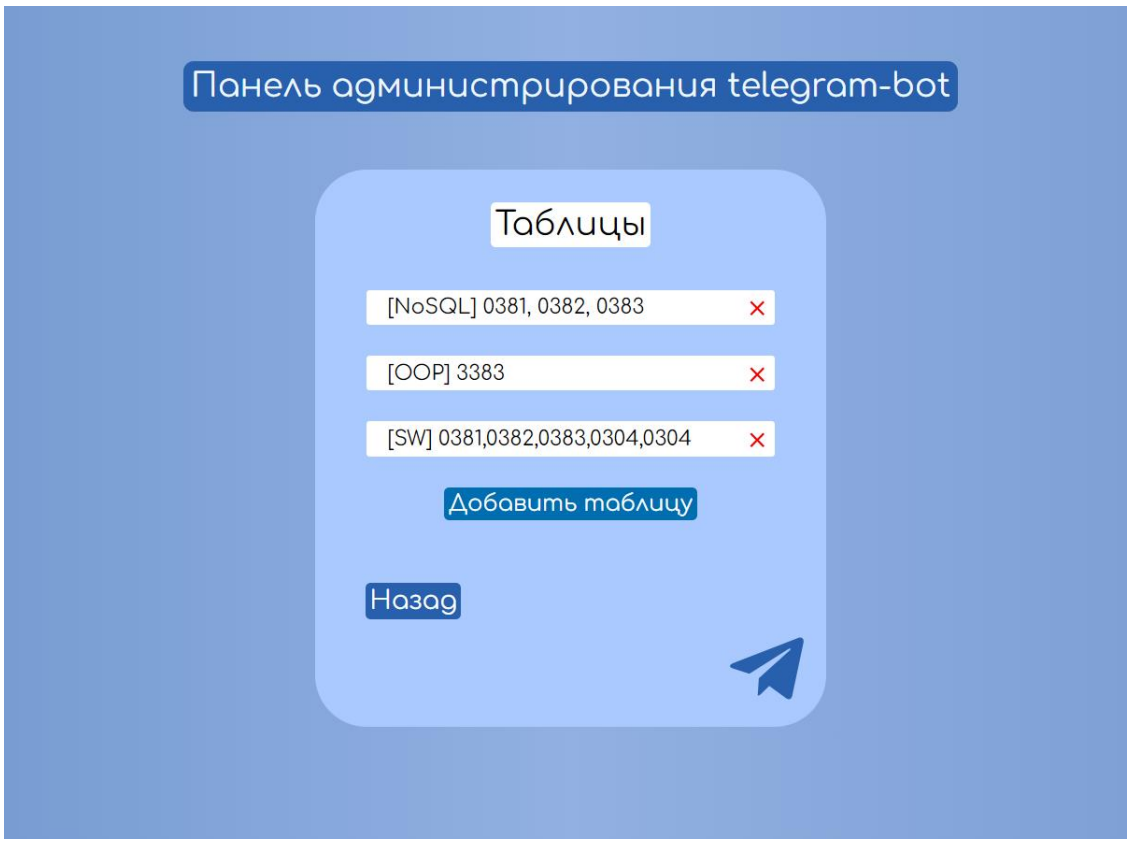

Рисунок 7 — Страница «Таблицы»

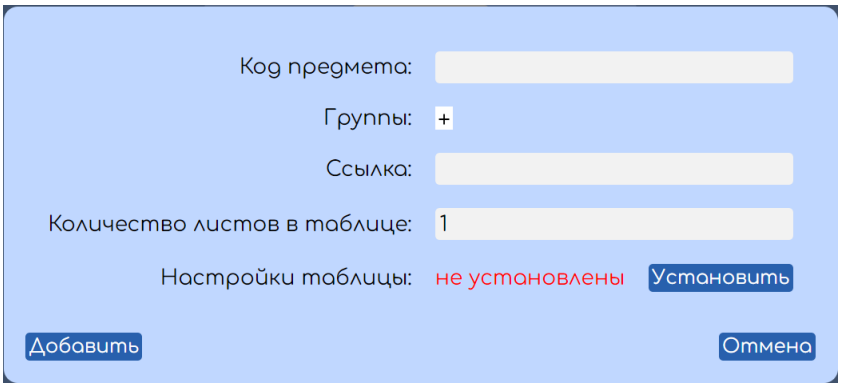

Рисунок 8 — Диалоговое окно: добавление таблицы

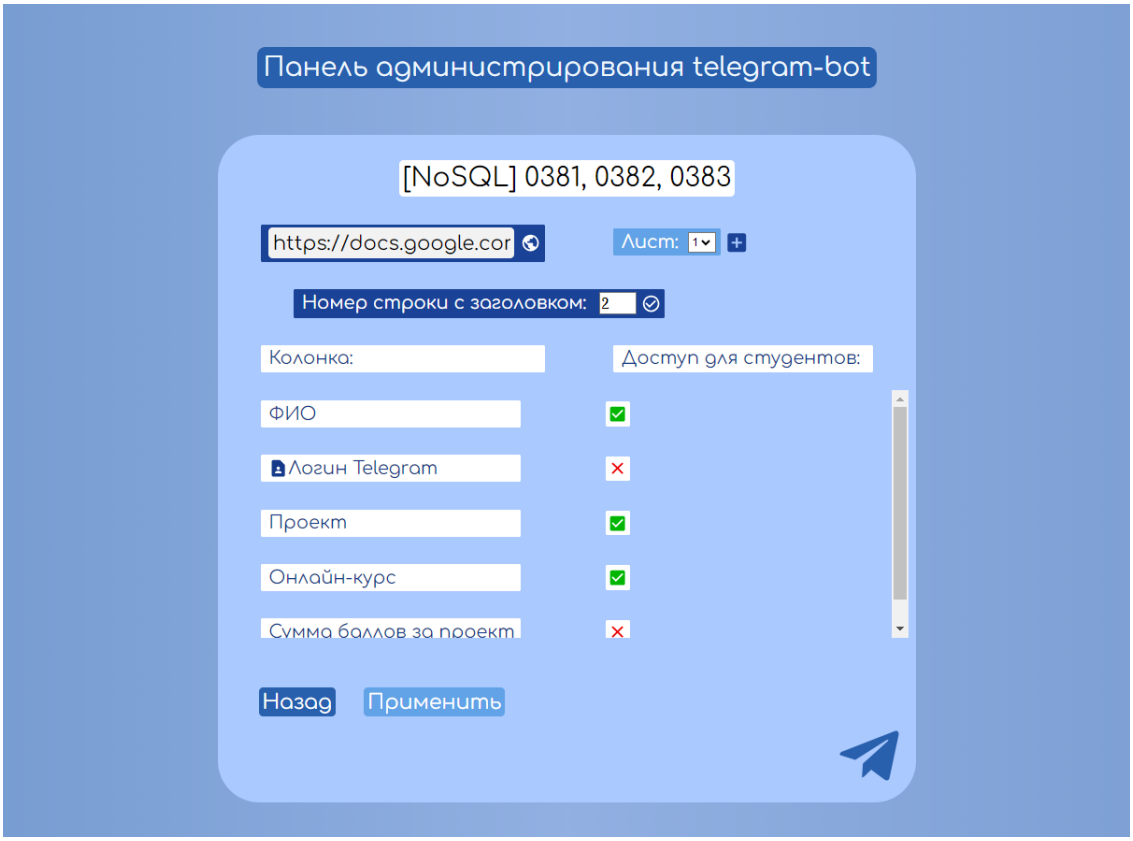

Рисунок 9 — Страница «Конкретная таблица»

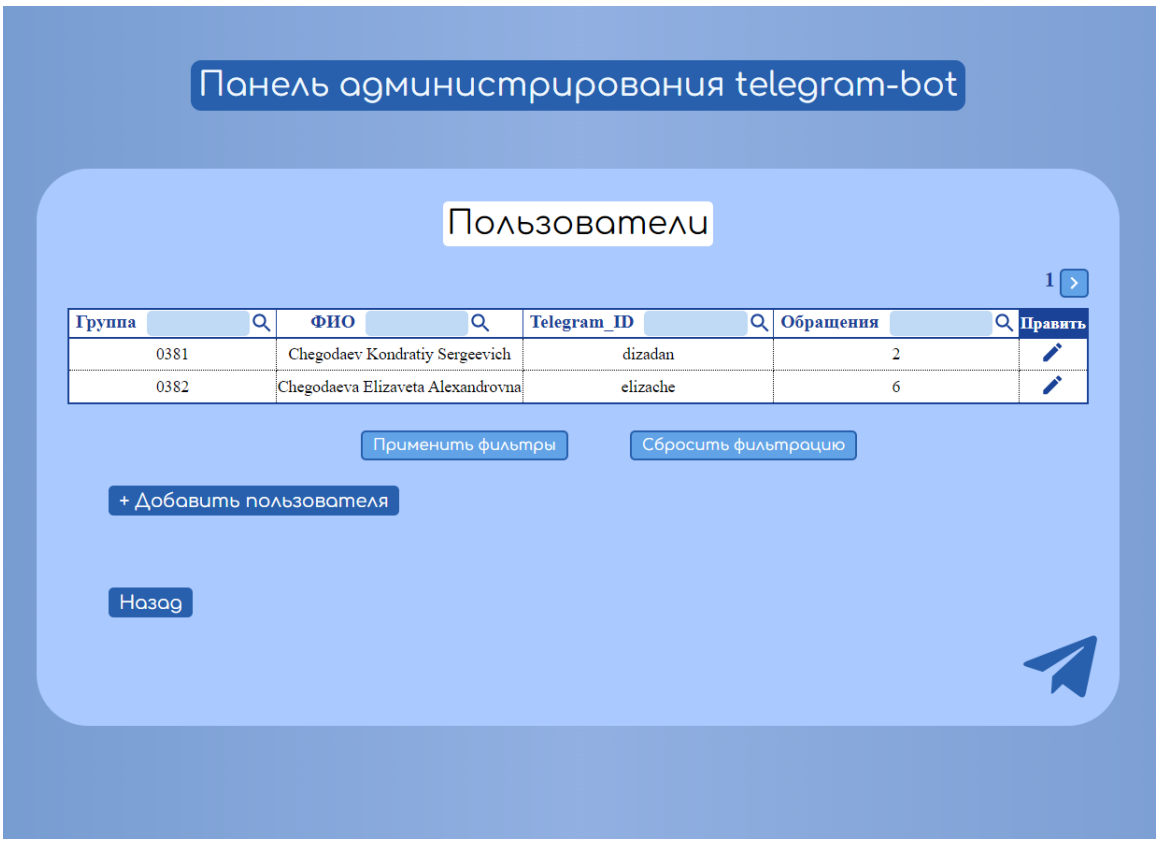

Рисунок 10 — Страница «Пользователи»

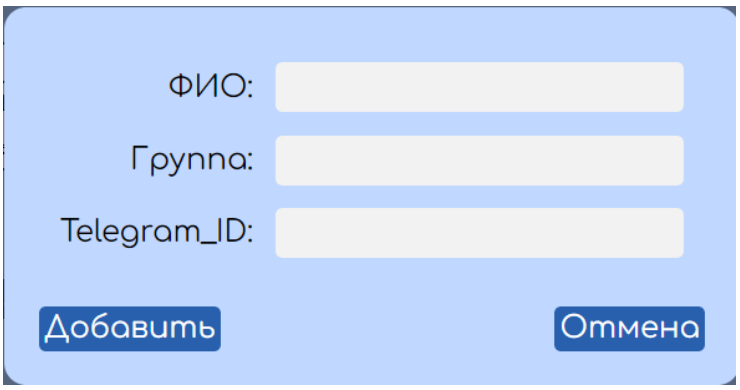

Рисунок 11 — Диалоговое окно: добавление пользователя

Рисунок 12 — Страница «Статистика»

Рисунок 13 — Диалоговое окно: График статистики посещений

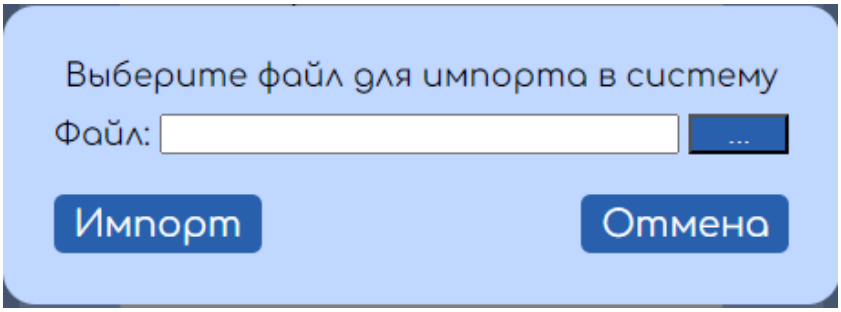

Рисунок 14 — Диалоговое окно: массовый импорт

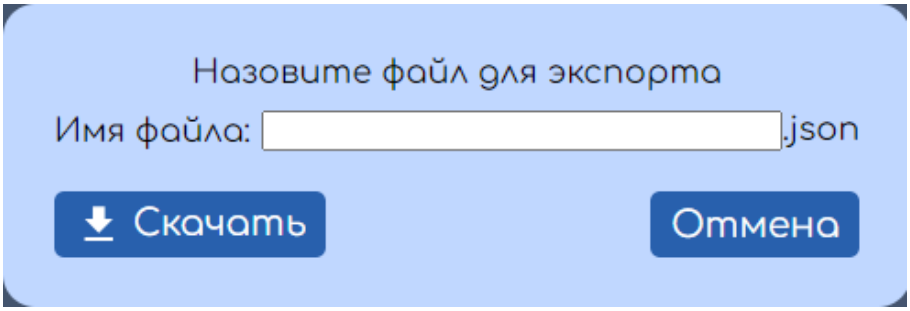

Рисунок 15 — Диалоговое окно: массовый экспорт

5. ВЫВОДЫ

5.1. Достигнутые результаты

В ходе работы было разработано web-приложение «Панель администрирования», позволяющее взаимодействовать с таблицами (добавлять, редактировать и удалять), подключать к системе новых пользователей и редактировать/удалять имеющиеся профили, просматривать статистику обращений, также реализована возможность массового импорта/экспорта данных.

5.2. Недостатки и пути для улучшения полученного решения

В текущем состоянии приложение поддерживает только формат google-таблиц, но для хранения таблиц могут использоваться и другие сервисы.

Массовый импорт/экспорт производится в формате json, для расширения универсальности приложения следует добавить и другие форматы представления данных (csv и прочие).

6. БУДУЩЕЕ РАЗВИТИЕ РЕШЕНИЯ

На основе реализованного приложения — разработать Telegram-бота, который будет принимать запросы пользователей, отправлять их на сервер, а затем отображать ответ сервера в диалоге с пользователем.

Предполагается, что Telegram-бот будет обладать следующими характеристиками:

1. Для каждого нового пользователя проводится авторизация, в ходе которой проверяется наличие Telegram Id пользователя в базе данных системы;
2. Пользователю предоставляется список доступных ему таблиц;
3. При запросе пользователя о выводе данных с конкретной таблицы бот демонстрирует данные только текущего пользователя, а не всех студентов;
4. Выводятся поля таблицы, отмеченные как доступные;
5. Данные выводятся со всех листов таблицы.

7. ПРИЛОЖЕНИЯ

7.1. Документация по сборке и развертыванию приложения

1. Клонировать репозиторий (ссылка на репозиторий приведена в списке литературы);
2. Открыть терминал и перейти в директорию с репозиторием;
3. Выполнить команду: “*docker-compose build*” для сборки приложения;
4. Выполнить команду: “*docker-compose up*” для развёртывания приложения;
5. Открыть приложение в браузере по адресу 127.0.0.1:3000.

7.2. Инструкция для пользователя

1. Авторизоваться, введя логин и пароль в соответствующие поля и нажав кнопку “Вход”;
2. Для того чтобы перейти к взаимодействию с пользователями, в меню выберите — “Пользователи”. Далее осуществится переход на страницу “Пользователи”, где можно просмотреть, добавить, удалить и редактировать пользователей;
3. Для того чтобы перейти к взаимодействию с таблицами, в меню выберите — “Таблицы”. Далее осуществится переход на страницу “Таблицы”, где можно просмотреть, добавить *(стоит учесть тот факт, что при добавлении ссылки на таблицу - следует сделать таблицу доступной для просмотра)* или удалить таблицы, а также перейти в форму редактирования конкретной таблицы;
4. Для того чтобы просмотреть статистику обращение к Telegram-боту, в меню выберите — “Статистика”. Далее осуществится переход на страницу “Статистика”, где можно просмотреть обращения каждого пользователя, отфильтровать по выбранному(ым) параметру(ам), а также построить визуальное отображение данных в виде графика (который можно скачать);

5. Для того чтобы осуществить массовый импорт/экспорт, в меню выберите соответствующую опцию и следуйте инструкциям в диалоговом окне.

8. ЛИТЕРАТУРА

1. GitHub-репозиторий: https://github.com/moevm/nosql2h23-tg-table

2. Документация FastApi: https://fastapi.tiangolo.com/learn/
3. Документация React: https://react.dev/learn
4. Документация MongoDB: https://www.mongodb.com/docs/
5. Документация Docker: https://docs.docker.com/