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

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Введение в нереляционные базы данных»

Тема: Сервис-агрегатор для поиска свободных мест на занятиях йоги

Студентка гр. 1303 _____ Андреева Е.А.

Студент гр. 1303 _____ Бутыло Е.А.

Студент гр. 1303 _____ Ягодаров М.А.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2024

ЗАДАНИЕ

Студенты

Андреева Е.А.

Бутыло Е.А.

Ягодаров М.А.

Группа 1303

Тема работы: Сервис-агрегатор для поиска свободных мест на занятиях йоги

Исходные данные:

Необходимо реализовать веб-приложение для поиска свободных мест на занятиях йоги с использованием СУБД MongoDB.

Содержание пояснительной записки:

Введение

Сценарий использования

Модель данных

Разработка приложения

Выводы

Приложение

Литература

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 01.09.2024

Дата сдачи реферата: 24.12.2024

Дата защиты реферата: 24.12.2024

Студентка гр. 1303	_____	Андреева Е.А.
Студент гр. 1303	_____	Бутыло Е.А.
Студент гр. 1303	_____	Ягодаров М.А.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

В рамках ИДЗ разработано веб-приложение, представляющее собой сервис для записи в студии йоги. Приложение включает функционал для администратора: просмотр всех данных приложения, их добавление; и функционал для обычных пользователей: просмотр и запись на доступные занятия, удаление записи. Для администратора реализована система фильтрации и поиска всех данных приложения по различным доступным параметрам.

Для разработки использовались следующие технологии: Vue.js, TypeScript, Go, MongoDB.

SUMMARY

The IDH developed a web application, which is a service for recording in yoga studios. The application includes functionality for the administrator: viewing all the application data, adding them; and functionality for ordinary users: viewing and signing up for available classes, deleting the record. For the administrator the system of filtering and searching all the application data by various available parameters is implemented.

The following technologies were used for development: Vue.js, TypeScript, Go, MongoDB.

СОДЕРЖАНИЕ

	Введение	6
1.	Сценарии использования	7
2.	Модель данных	23
3.	Разработанное приложение	46
4.	Выводы	50
5.	Приложения	51
6.	Литература	52

ВВЕДЕНИЕ

Актуальность проблемы:

Современные студии йоги сталкиваются с трудностями записи клиентов из-за непрозрачного расписания, неудобных способов бронирования и отсутствия автоматизации. Это снижает удобство для клиентов и увеличивает нагрузку на администраторов. Веб-приложение для записи упрощает процесс, повышая удовлетворенность клиентов и эффективность управления студией.

Постановка задачи:

Разработать веб-приложение для поиска свободных мест на занятиях йоги, которое позволяет:

- Просматривать расписание занятий студий йоги
- Записываться на занятия, где есть свободные места
- Отменять запись на занятие
- Просматривать все данные приложения (администраторам)

Предлагаемое решение:

Создание распределенной системы на основе:

- MongoDB для хранения данных
- Backend на Go для обработки запросов
- Frontend на Vue.js для визуализации

Качественные требования к решению:

- Удобный пользовательский интерфейс
- Оперативное оповещение о превышениях
- Возможность анализа исторических данных

1. СЦЕНАРИЙ ИСПОЛЬЗОВАНИЯ

Макет UI (см. Рисунок 1-3)

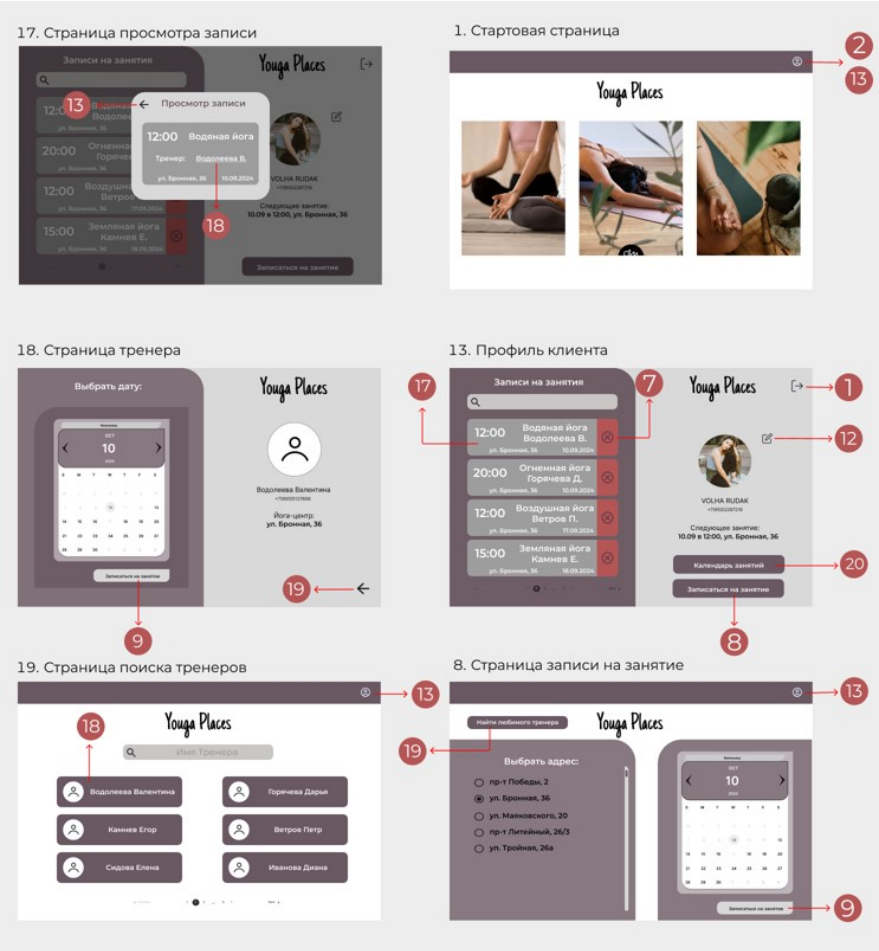

Рисунок 1 – Макет UI

Рисунок 2 – Макет UI

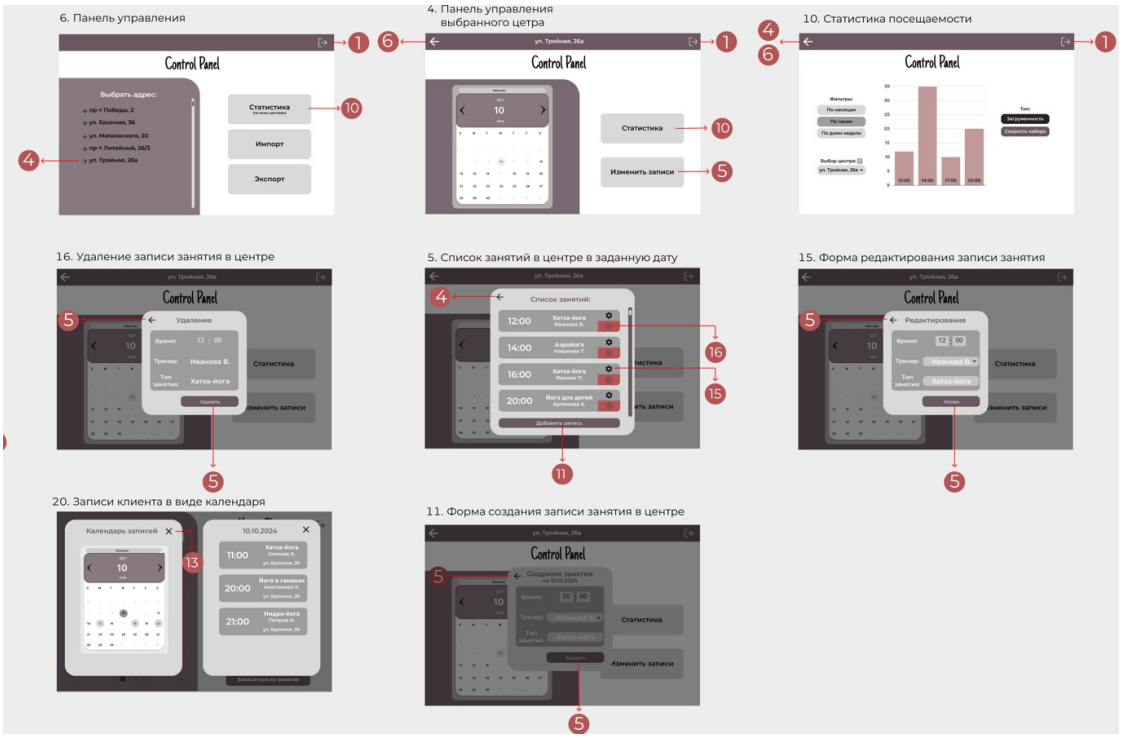

Рисунок 3 – Макет UI

Сценарии использования для задачи.

1) “Авторизация”.

Действующие лица:

- Неавторизованный пользователь.

Предусловия:

- Пользователь должен находиться на главной странице (страница 1).
- Пользователь не должен быть авторизован.
- Пользователь должен быть зарегистрирован в системе.
- У пользователя есть действующий логин (номер телефона) и пароль.

Основной сценарий:

- Пользователь открывает страницу авторизации (страница 2).
- Пользователь вводит свой зарегистрированный номер телефона.
- Пользователь вводить свой пароль.
- Пользователь нажимает кнопку "Log In".

Результат основного сценария:

- Система авторизует пользователя и перенаправляет на страницу его профиля (страница 13), либо на страницу панели управления (страница 6), если данный аккаунт соответствует требуемым правам.

Альтернативный сценарий:

- Неверный номер телефона.
- Неверный пароль.

Результат альтернативного сценария:

- Система уведомляет пользователя, что указанный номер телефона не зарегистрирован в системе, пользователь может повторно ввести номер или перейти к регистрации.
- Система уведомляет пользователя о том, что пароль введён неправильно, пользователь может ввести пароль повторно.

2) “Регистрация”.

Действующие лица:

- Неавторизованный пользователь.

Предусловия:

- Пользователь должен находиться на главной странице (страница 1).
- Пользователь не должен быть авторизован.

Основной сценарий:

- Пользователь открывает страницу авторизации (страница 2).
- Пользователь открывает страницу регистрации (страница 3).
- Пользователь вводит действующий номер телефона (уникальный).
- Пользователь вводит свои имя и фамилию, которые будут отображаться на сайте.
- Пользователь вводит пароль.
- Пользователь нажимает кнопку "Sign Up".

Результат основного сценария:

- Система регистрирует пользователя и перенаправляет на страницу его профиля (страница 13).

Альтернативный сценарий:

- Неправильный формат номера телефона.
- Пользователь с данным номером телефона уже есть в системе.
- Некорректный пароль.

Результат альтернативного сценария:

- Система уведомляет пользователя об ошибке с введенным номером телефона. Пользователь корректирует номер и снова нажимает "Sign Up".
- Система уведомляет пользователя, что введенный номер телефона пользователя уже существует в системе, пользователь вводит другой номер и нажимает "Sign Up".
- Система уведомляет пользователя, что пароль для учётной записи в целях безопасности должен быть длиннее 7 символов и содержать хотя бы одну прописную и одну строчную букву и одну цифру. Пользователь корректирует пароль и сохраняет его.

3) “Редактирование информации профиля пользователя”.

Действующие лица:

- Авторизованный пользователь.

Предусловия:

- Пользователь должен находиться на странице своего профиля (страница 13).

Основной сценарий:

- Пользователь открывает страницу редактирования информации профиля (страница 12).
- Пользователь загружает новую фотографию профиля.
- Пользователь вводит своё новое имя.
- Пользователь нажимает на кнопку сохранения нового имени.
- Пользователь вводить свой новый пароль.
- Пользователь нажимает на кнопку сохранения нового пароля.

Результат основного сценария:

- Система обновляет данные пользователя.

Альтернативный сценарий:

- Некорректное имя.
- Некорректный пароль.

Результат альтернативного сценария:

- Система уведомляет пользователя, что указанное имя не должно быть пустой строкой и состоять из хотя бы одной буквы. Пользователь корректирует имя и сохраняет его.
- Система уведомляет пользователя, что пароль для учётной записи в целях безопасности должен быть длиннее 7 символов и содержать хотя бы одну прописную и одну строчную букву и одну цифру. Пользователь корректирует пароль и сохраняет его.

4) “Отмена записи клиента на занятие”.

Действующие лица:

- Авторизованный пользователь.

Предусловия:

- Пользователь должен находиться на странице своего профиля (страница 13).

Основной сценарий:

- Пользователь находит искомое занятие в списке записей, отсортированных по времени, переключаясь между частями списка с помощью пагинации.
- Пользователь нажимает на кнопку отмены записи и попадает на соответствующую форму (страница 7).
- Пользователь нажимает на кнопку "Отменить" для подтверждения отмены записи в форме, после того как убедился в корректности данных.

Результат основного сценария:

- Система удаляет пользователя из списка пользователей записанных на конкретное занятие. Система удаляет запись на занятие из профиля пользователя и перенаправляет на страницу профиля (страница 13).

Альтернативный сценарий:

- После нажатия на кнопку отмены занятия пользователь, ознакомившись с данными формы отмены, понял, что выбрал не искомую запись.

Результат альтернативного сценария:

- Пользователь нажимает на кнопку "Назад" и ищет нужную запись на занятие.

5) “Запись клиента на занятие”.

Действующие лица:

- Авторизованный пользователь.

Предусловия:

- Пользователь должен находиться на странице своего профиля (страница 13).

Основной сценарий:

- Пользователь нажимает на кнопку "Записаться на занятие" и попадает на страницу записи (страница 8).

- Пользователь настраивает фильтры, такие как центр йоги и дату, где и когда будет проходить занятие йогой.
- Пользователь нажимает кнопку "Записаться на занятие" и появляется список записей (страница 9).
- Пользователь выбирает из списка запись на занятие подходящую ему по времени, разновидности и тренеру.
- Пользователь нажимает на кнопку "Записаться" и видит форму подтверждения (страница 14).
- Пользователь ознакомляется с данными формы записи и убеждается, что все данные корректны.
- Пользователь нажимает на кнопку "Записаться".

Результат основного сценария:

- Система записывает пользователя на выбранное занятие, обновляет данные занятия и переадресует пользователя в профиль (страница 13), где он может найти новую запись.

Альтернативный сценарий:

- Пользователь выбирает некорректную дату.
- Пользователь не смог подобрать занятие в выбранную дату.
- Пользователь ознакомился с данными формы подтверждения записи и понял, что выбрал не нужное занятие.

Результат альтернативного сценария:

- Система уведомляет пользователя, что необходимо выбрать дату не ранее сегодняшнего дня. Пользователь корректирует дату и нажимает кнопку "Записаться на занятие".
- Пользователь нажимает на кнопку назад и выбирает другую дату.
- Пользователь нажимает на кнопку назад и выбирает другую запись.

6) “Запись клиента на занятие к определённому тренеру”.

Действующие лица:

- Авторизованный пользователь.

Предусловия:

- Пользователь должен находиться на странице тренера (страница 18).

Основной сценарий:

- Пользователь выбирает дату.
- Пользователь нажимает кнопку "Записаться на занятие" и появляется список записей (страница 9).
- Пользователь выбирает из списка запись на занятие подходящую ему по времени, разновидности.
- Пользователь нажимает на кнопку "Записаться" и видит форму подтверждения (страница 14).
- Пользователь ознакомляется с данными формы записи и убеждается, что все данные корректны.
- Пользователь нажимает на кнопку "Записаться".

Результат основного сценария:

- Система записывает пользователя на выбранное занятие, обновляет данные занятия и переадресует пользователя в профиль (страница 13), где он может найти новую запись.

Альтернативный сценарий:

- Пользователь выбирает некорректную дату.
- Пользователь не смог подобрать занятие в выбранную дату.
- Пользователь ознакомился с данными формы подтверждения записи и понял, что выбрал не нужное занятие.

Результат альтернативного сценария:

- Система уведомляет пользователя, что необходимо выбрать дату не ранее сегодняшнего дня. Пользователь корректирует дату и нажимает кнопку "Записаться на занятие".
- Пользователь нажимает на кнопку назад и выбирает другую дату.
- Пользователь нажимает на кнопку назад и выбирает другую запись.

7) “Статистика посещаемости”.

Действующие лица:

- Авторизованный пользователь (администратор).

Предусловия:

- Пользователь должен находиться на странице общей панели управления (страница 6), либо определённого центра (страница 4).

Основной сценарий:

- Пользователь нажимает на кнопку "Статистика" и попадает на страницу с соответствующей статистикой (страница 8), если со страницы 6 - информация по всем центрам, если со страницы 4 - по определённому центру.
- Пользователь настраивает фильтр, за какое время он хочет увидеть статистику (неделя, месяц, год).

Результат основного сценария:

- Система выводит статистику посещаемости в виде графика в зависимости от выбранного отрезка времени.

8) “Создание занятия в определённом центре”.

Действующие лица:

- Авторизованный пользователь (администратор).

Предусловия:

- Пользователь должен находиться на странице панели управления определённого центра (страница 4).

Основной сценарий:

- Пользователь, с помощью фильтра "datepicker", выбирает дату, занятие в которую он хочет создать.
- Пользователь нажимает на кнопку "Изменить записи" и выводится список занятий в выбранную дату (страница 5).
- Пользователь нажимает на кнопку "Добавить запись" и попадает в форму создания (страница 11).
- Пользователь задаёт время занятия.
- Пользователь задаёт тренера занятия.
- Пользователь задаёт тип занятия.
- Пользователь нажимает на кнопку "Создать".

Результат основного сценария:

- Система создаёт новое занятие в определённом центре в выбранную дату с полученной от пользователя информацией. Переадресует пользователя на страницу со списком занятий (страница 5).

Альтернативный сценарий:

- Пользователь выбирает некорректную дату.
- Пользователь захотел создать занятие в другую дату.
- Пользователь ввёл некорректное время для занятия в форму создания.

Результат альтернативного сценария:

- Система уведомляет пользователя, что необходимо выбрать дату не ранее сегодняшнего дня. Пользователь корректирует дату и нажимает кнопку "Изменить записи".
- Пользователь нажимает на кнопку назад и выбирает другую дату.
- Система уведомляет, что время для занятия должно быть в рамках рабочего времени, в случае, если занятие создаётся на текущий рабочий день, то не должно быть ранее текущего времени. Пользователь корректирует время и продолжает заполнять форму создания.

9) “Редактирование занятия в определённом центре”.

Действующие лица:

- Авторизованный пользователь (администратор).

Предусловия:

- Пользователь должен находиться на странице панели управления определённого центра (страница 4).

Основной сценарий:

- Пользователь, с помощью фильтра "datepicker", выбирает дату, занятие в которую он хочет изменить.
- Пользователь нажимает на кнопку "Изменить записи" и выводится список занятий в выбранную дату (страница 5).
- Пользователь нажимает на кнопку редактирования и попадает в соответствующую форму (страница 15).

- Пользователь корректирует время занятия.
- Пользователь корректирует тренера занятия.
- Пользователь корректирует тип занятия.
- Пользователь нажимает на кнопку "Готово".

Результат основного сценария:

- Система обновляет данные занятия в определённом центре в выбранную дату с полученной от пользователя информацией. Переадресует пользователя на страницу со списком занятий (страница 5).

Альтернативный сценарий:

- Пользователь выбирает некорректную дату.
- Пользователь захотел изменить занятие в другую дату.
- Пользователь ввёл некорректное время для занятия в форму редактирования.

Результат альтернативного сценария:

- Система уведомляет пользователя, что необходимо выбрать дату не ранее сегодняшнего дня. Пользователь корректирует дату и нажимает кнопку "Изменить записи".
- Пользователь нажимает на кнопку назад и выбирает другую дату.
- Система уведомляет, что время для занятие должно быть в рамках рабочего времени, в случае, если занятие редактируется в границах текущего рабочего дня, то не должно быть ранее текущего времени. Пользователь корректирует время и продолжает заполнять форму редактирования.

10) “Удаление занятия в определённом центре”.

Действующие лица:

- Авторизованный пользователь (администратор).

Предусловия:

- Пользователь должен находиться на странице панели управления определённого центра (страница 4).

Основной сценарий:

- Пользователь, с помощью фильтра "datepicker", выбирает дату, занятие в которую он хочет удалить.
- Пользователь нажимает на кнопку "Изменить записи" и выводится список занятий в выбранную дату (страница 5).
- Пользователь нажимает на кнопку удаления и попадает в соответствующую форму (страница 16).
- Пользователь ознакомляется с данными формы удаления и убеждается, что все данные корректны..
- Пользователь нажимает на кнопку "Удалить".

Результат основного сценария:

- Система отменяет запись для всех пользователей, записавшихся на выбранное занятие. Система удаляет данное занятие в определённом центре в выбранную дату. Переадресует пользователя на страницу со списком занятий (страница 5).

Альтернативный сценарий:

- Пользователь выбирает некорректную дату.
- Пользователь захотел удалить занятие в другую дату.
- Пользователь ознакомился с данными формы удаления занятия и понял, что выбрал не нужное занятие.

Результат альтернативного сценария:

- Система уведомляет пользователя, что необходимо выбрать дату не ранее сегодняшнего дня. Пользователь корректирует дату и нажимает кнопку "Изменить записи".
- Пользователь нажимает на кнопку назад и выбирает другую дату.
- Пользователь нажимает на кнопку назад и выбирает другое занятие.

11) “Экспорт”.

Действующие лица:

- Авторизованный пользователь (администратор).

Предусловия:

- Пользователь должен находиться на странице общей панели управления (страница 6).

Основной сценарий:

- Пользователь нажимает кнопку “Экспорт”.

Результат основного сценария:

- Система формирует файл формата JSON со всей информацией о занятиях по всем центрам, у пользователя начинается загрузка сформированного файла.

12) “Импорт”.

Действующие лица:

- Авторизованный пользователь (администратор).

Предусловия:

- Пользователь должен находиться на странице общей панели управления (страница 6).

Основной сценарий:

- Пользователь нажимает кнопку “Импорт”.
- Пользователь выбирает файл формата JSON в открывшемся проводнике его операционной системы и нажимает кнопку подтверждения.

Результат основного сценария:

- Система считывает полученный файл и обновляет расписание занятий по всем центрам.

Альтернативный сценарий:

- Выбранный файл не формата JSON.
- Выбранный файл превышает максимально допустимый размер (100 МБ).
- Система не смогла валидировать полученный файл.

Результат альтернативного сценария:

- Система уведомляет пользователя о несоответствующем формате выбранного файла. Пользователь нажимает кнопку “Импорт” и выбирает другой файл.

- Система уведомляет пользователя о превышении максимально допустимого размера файла. Пользователь нажимает кнопку “Импорт” и выбирает другой файл.
- Система уведомляет пользователя о неудачной валидации файла. Пользователь нажимает кнопку “Импорт” и выбирает другой файл.

13) ”Просмотр записи клиента на занятие”.

Действующие лица:

- Авторизованный пользователь.

Предусловия:

- Пользователь должен находиться на странице своего профиля (страница 13).

Основной сценарий:

- Пользователь находит искомое занятие в списке записей, отсортированных по времени, переключаясь между частями списка с помощью пагинации.
- Пользователь нажимает на ячейку своей записи и попадает на соответствующую форму (страница 17).

Результат основного сценария:

- Система выводит пользователю информацию о выбранном занятии.

14) ”Просмотр записей клиента в виде календаря”.

Действующие лица:

- Авторизованный пользователь.

Предусловия:

- Пользователь должен находиться на странице своего профиля (страница 13).

Основной сценарий:

- Пользователь нажимает на кнопку "Календарь занятий" и попадает на соответствующую форму (страница 20).
- Пользователь нажимает на выбранную дату.

Результат основного сценария:

- Система выводит пользователю информацию о занятиях в выбранную дату.

Альтернативный сценарий:

- Пользователь выбирает некорректную дату.

Результат альтернативного сценария:

- Система уведомляет пользователя, что необходимо выбрать дату не ранее сегодняшнего дня. Пользователь корректирует дату.

15) ”Поиск тренера”.

Действующие лица:

- Авторизованный пользователь.

Предусловия:

- Пользователь должен находиться на странице записи на занятие (страница 8).

Основной сценарий:

- Пользователь нажимает на кнопку "Найти любимого тренера" и попадает на соответствующую страницу (страница 19).
- Пользователь вводит им тренера.

Результат основного сценария:

- Система выводит пользователю информацию об искомом тренере.

Альтернативный сценарий:

- Пользователь вводит имя несуществующего тренера.
- Пользователь не ввел имя тренера.

Результат альтернативного сценария:

- Система уведомляет пользователя, что в по запросу ничего не найдено.
- Система выводит информацию о всех тренерах.

16) ”Просмотр профиля тренера”.

Действующие лица:

- Авторизованный пользователь.

Предусловия:

- Пользователь должен находиться на странице поиска тренеров (страница 19).
- На странице отображается минимум одна ячейка с информацией о тренере.

Основной сценарий:

- Пользователь нажимает на ячейку с информацией о тренере.

Результат основного сценария:

- Система выводит страницу тренера (страница 18).

Вывод об операциях:

В системе будут преобладать операции чтения данных, так как основной функционал направлен на:

- Постоянный мониторинг показаний
- Визуализацию данных через графики
- Анализ статистики
- Просмотр превышений

Операции записи будут происходить реже и включают:

- Импорт данных
- Добавление заметок к превышениям
- Управление пользователями

2. МОДЕЛЬ ДАННЫХ

1. Нереляционная модель

Графическое представление модели

Client – Рисунок 4.

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "type": "object",
        "required": ["_id", "name", "phone", "password", "updated_at",
"created_at"],
    "properties": {
        "_id": {
            "bsonType": "objectId"
        },
        "name": {
            "type": "string"
        },
        "phone": {
            "type": "string",
            "pattern": "^\\+7\\([0-9]{3}\\)[0-9]{3}\\-[0-9]{4}$"
        },
        "password": {
            "type": "string",
            "minLength": 8
        },
        "picture_uri": {
            "type": "string",
            "format": "uri",
            "default": "https://cdn.example.com"
        },
        "birth_date": {
            "bsonType": "date",
            "default": "2000-01-01"
        },
        "gender": {
            "type": "string",
            "enum": ["M", "F"],
            "default": "F"
        },
        "updated_at": {
            "bsonType": "date"
        },
        "created_at": {
            "bsonType": "date"
        },
        "classes": {
            "bsonType": "array",
            "items": {
                "bsonType": "objectId"
            },
            "default": []
        }
    }
}
```

Рисунок 4 – Client NoSQL scheme

Trainer – Рисунок 5.

Рисунок 5 – Trainer NoSQL scheme

Studio – Рисунок 6.

Рисунок 6 – Studio NoSQL scheme

Class – Рисунок 7.

Рисунок 7 – Class NoSQL scheme

Описание назначений коллекций, типов данных и сущностей

Client - клиент студий йоги (~1300 байт):

_id (bsonType.objectId - 12 байт) - уникальный идентификатор (id) клиента

name (string - в среднем 50 байт) - имя клиента

phone (string - 15 байт) - номер телефона клиента

password (string - в среднем 12 байт) - пароль клиента для входа в свой профиль

picture_uri (string - в среднем 50 байт) - ссылка на изображение клиента

birth_date (bsonType.date - 8 байт) - дата рождения

gender (string - 1 байт) - пол

updated_at (bsonType.date - 8 байт) - когда клиент обновлял профиль в последний раз

created_at (bsonType.date - 8 байт) - когда клиент создал профиль

classes (bsonType.array - в среднем 1152 байт (каждая запись удаляется из БД через 12 месяцев, в месяц в среднем 8 записей, то есть 96 записей за год)) - массив идентификаторов (id) записей на занятия данного клиента

Trainer - тренер студии йоги (~13000 байт):

_id (bsonType.objectId - 12 байт) - уникальный идентификатор (id) тренера

name (string - в среднем 50 байт) - имя тренера

phone (string - 15 байт) - номер телефона тренера

studio_id (bsonType.objectId - 12 байт) - идентификатор студии, в которой работает тренер

picture_uri (string - в среднем 50 байт) - ссылка на изображение тренера

birth_date (bsonType.date - 8 байт) - дата рождения

gender (string - 1 байт) - пол

updated_at (bsonType.date - 8 байт) - когда клиент обновлял профиль в последний раз

created_at (bsonType.date - 8 байт) - когда клиент создал профиль

classes (bsonType.array - в среднем 12960 байт (каждая запись удаляется из БД через 12 месяцев, в месяц в среднем 90 записей, то есть 1080

записей за год)) - массив идентификаторов (id) записей на занятия данного тренера

Studio - студия йоги (~52000 байт):

_id (bsonType.objectId - 12 байт) - уникальный идентификатор (id) студии

address (string - в среднем 100 байт) - адрес студии

classes (bsonType.array - в среднем 51840 байт (каждая запись удаляется из БД через 12 месяцев, в месяц в среднем 360 записей, то есть 4320 записей за год)) - массив идентификаторов (id) записей на занятия в данной студии

trainers (bsonType.array - ~48 байт (~4 тренера на 1 студию))

Class - занятие (~160 байт):

_id (bsonType.objectId - 12 байт) - уникальный идентификатор (id) занятия

class_name (string - в среднем 15 байт) - название занятия

time (bsonType.date - 8 байт) - время начала занятия

studio_id (bsonType.objectId - 12 байт) - идентификатор студии, в которой проводится занятие

trainer_id (bsonType.objectId - 12 байт) - идентификатор тренера, который проводит занятие

clients (bsonType.array - ~96 байт (в среднем 8 клиентов на 1 запись))

Оценка объема информации, хранимой в модели

Пусть n - количество студий йоги. Каждый тренер проводит по ~3 занятия в день. В год студия работает ~350 дней. На каждое занятие записывается ~8 клиентов. Итого:

$$V(n) = \left(52000 + \left(\frac{4320 \cdot 8}{96} \cdot 1300\right) + (4 \cdot 13000) + (4320 \cdot 160)\right)n = 1\,263\,200n$$

Избыточность данных

В client поле classes избыточно (1152 байт). В trainer поле studio_id избыточно (12 байт). В class поля trainer_id, studio_id избыточны (24 байт).

Итого:

$$V_{clean}(n) = \left(52000 + \left(\frac{4320 \cdot 8}{96} \cdot (1300 - 1152)\right) + \left(4 \cdot (13000 - 12)\right) + \left(4320 \cdot (160 - 24)\right)\right)n = 744\,752$$

Избыточность как отношение объема модели к “чистому” объему:

$$\frac{V(n)}{V_{clean}(n)} = \frac{1\,263\,200n}{744\,752} \approx 1.696$$

Направление роста модели при увеличении количества объектов каждой сущности

Рост модели напрямую зависит от увеличения количества объектов следующих сущностей:

- Рост количества занятий линейно увеличивает размер документов студий, тренеров.
- Рост количества тренеров линейно увеличивает размер документов студий.

Примеры данных

Client


```
{
    "_id" : "671e6188c4140ba749707d2f",
    "name": "Elizaveta Andreeva",
    "phone": "+7(999)99-9999",
    "gender": "F",
    "birth_date": "2000-01-01",
    "created_at": "2040-10-28T23:58:18Z",
    "updated_at": "2040-10-29T23:58:18Z",
    "password": "11111111",
    "picture_uri": "https://cdn.example.com",
    "classes": ["671e6196bc9ec2a1455fda9a"]
}
```


Class


```
{
    "_id" : "671e6196bc9ec2a1455fda9a",
    "class_name": "Yoga",
    "time": "2040-10-28T23:58:18Z",
    "studio_id": "671e6196bc9ec2a1455fda91",
    "trainer_id": "671e6196bc9ec2a1455fda92",
    "clients": [
        "671e6188c4140ba749707d2f",
        "271e6188c4140ba749707d2f"
    ]
}
```


Studio


```
{
    "_id" : "671e6196bc9ec2a1455fda9a",
    "address": "Saint-Petersburg, str. Bronnaya 22",
    "classes": [
        "671e6188c4140ba749707d2f",
        "271e6188c4140ba749707d2f"
```

```
    ],
    "trainers": [
        "331e6188c4140ba749707d2f",
        "261e6188c4140ba749707d2f"
    ]
}
```

Trainer

```
{
    "_id" : "671e6188c4140ba749707d2f",
    "name": "Elizaveta Andreeva",
    "phone": "+7(999)99-9999",
    "gender": "F",
    "birth_date": "2000-01-01",
    "created_at": "2040-10-28T23:58:18Z",
    "updated_at": "2040-10-29T23:58:18Z",
    "studio_id": "331e6188c4ff0ba749707d2f",
    "picture_uri": "https://cdn.example.com",
    "classes": [
        "671e6196bc9ec2a1455fda9a"
    ]
}
```

Примеры запросов

Названия коллекций:

```
const (
    studios  = "studios"
    clients  = "clients"
    trainers = "trainers"
    classes  = "classes"
)
```

Структуры существ:

```
type Gender string

const (
    GenderFemale Gender = "F"
    GenderMale   Gender = "M"
)

type Person struct {
    ID         bson.ObjectID   `bson:"_id,omitempty"`
    Name       string          `bson:"name"`
    Phone      string          `bson:"phone"`
    PictureURI string          `bson:"picture_uri"`
    BirthDate  time.Time       `bson:"birth_date"`
    Gender     Gender          `bson:"gender"`
    Classes    []bson.ObjectID `bson:"classes"`
    CreatedAt  time.Time       `bson:"created_at"`
    UpdatedAt  time.Time       `bson:"updated_at"`
}
```

```
type Client struct {
    Person   `bson:",inline"`
    Password string `bson:"password"`
}

type Trainer struct {
    Person   `bson:",inline"`
    StudioID bson.ObjectID `bson:"studio_Id"`
}

type Studio struct {
    ID       bson.ObjectID   `bson:"_id,omitempty"`
    Address  string          `bson:"address"`
    Classes  []bson.ObjectID `bson:"classes"`
    Trainers []bson.ObjectID `bson:"trainers"`
}

type Class struct {
    ID        bson.ObjectID   `bson:"_id,omitempty"`
    Name      string          `bson:"name"`
    Time      time.Time       `bson:"time"`
    StudioID  bson.ObjectID   `bson:"studio_id"`
    TrainerID bson.ObjectID   `bson:"trainer_id"`
    Clients   []bson.ObjectID `bson:"clients"`
}
```

Добавление клиента:

```
func (r Repository) InsertClient(client Client) bson.ObjectID {
    collection := r.mg.Database("yoga").Collection(clients)
    result, err := collection.InsertOne(context.TODO(), client)
    if err != nil {
        log.Fatalf("Error inserting new client: %v", err)
    }
    fmt.Printf("Inserted client with id %v\n", result.InsertedID)
    return result.InsertedID.(bson.ObjectID)
}

func (r Repository) Test() {
    birthDate, err := time.Parse(time.RFC3339, "2003-10-01T15:30:00+03:00
")
    if err != nil {
        log.Fatalf("Error parsing date: %v", err)
    }

    clientID := r.InsertClient(Client{
        Person: Person{
            Name:       "Elizaveta Andreeva",
            Phone:      "7(999)999-9999",
            PictureURI: "https://cdn.example.com/default-picture",
            Classes:    []bson.ObjectID{},
            BirthDate:  birthDate,
            Gender:     GenderFemale,
            CreatedAt:  time.Now(),
```

```
            UpdatedAt:  time.Now(),
        },
        Password: "admin12345",
    })
}
```

Используется один запрос в одну коллекцию.

Создание занятия:

```
func (r Repository) MakeClass(studioID, trainerID bson.ObjectID, class Cl
ass) bson.ObjectID {
    db := r.mg.Database("yoga")
    ctx := context.TODO()

    collection := r.mg.Database("yoga").Collection(classes)
    result, err := collection.InsertOne(ctx, class)
    if err != nil {
        log.Fatalf("Error inserting new class: %v", err)
    }
    fmt.Printf("Inserted class with id %v\n", result.InsertedID)
    classID := result.InsertedID.(bson.ObjectID)

    _, err = db.Collection(studios).UpdateOne(
        ctx,
        bson.M{"_id": studioID},
        bson.M{"$addToSet": bson.M{"classes": classID}},
    )
    if err != nil {
        log.Fatalf("Error adding class to a studio: %v", err)
    }

    _, err = db.Collection(trainers).UpdateOne(
        ctx,
        bson.M{"_id": trainerID},
        bson.M{"$addToSet": bson.M{"classes": classID}},
    )
    if err != nil {
        log.Fatalf("Error adding class to a trainer: %v", err)
    }

    return classID
}

func (r Repository) Test() {
    classID := r.MakeClass(studioID, trainerID, Class{
        Name:      "Yoga for kids",
        Time:      tm,
        StudioID:  studioID,
        TrainerID: trainerID,
        Clients:   []bson.ObjectID{},
    })
}
```

Используется по одному запросу в три коллекции.

Запись на занятие:

```
func (r Repository) MakeAppointment(clientID, classID bson.ObjectID) {
    db := r.mg.Database("yoga")
    ctx := context.TODO()

    _, err := db.Collection(classes).UpdateOne(
        ctx,
        bson.M{"_id": classID},
        bson.M{"$addToSet": bson.M{"clients": clientID}},
    )
    if err != nil {
        log.Fatalf("Error adding client to a class: %v", err)
    }

    _, err = db.Collection(clients).UpdateOne(
        ctx,
        bson.M{"_id": clientID},
        bson.M{"$addToSet": bson.M{"classes": classID}},
    )
    if err != nil {
        log.Fatalf("Error adding class to a client: %v", err)
    }

    fmt.Printf("Created an appointment for client (%q) and class (%q)\n",
 clientID, classID)
}
```

Используется по одному запросу в две коллекции.

Удаление занятия:

```
func (r Repository) DeleteClass(classID bson.ObjectID) {
    db := r.mg.Database("yoga")
    ctx := context.TODO()

    var class Class
    if err := db.Collection(classes).FindOne(ctx, bson.M{"_id": classID})
.
        Decode(&class); err != nil {
        if errors.Is(err, mongo.ErrNoDocuments) {
            fmt.Printf("Class with id %q not found\n", classID)
            return
        }
        log.Fatalf("Error finding class to delete: %v", err)
    }

    if _, err := db.Collection(studios).UpdateOne(
        ctx,
        bson.M{"_id": class.StudioID},
        bson.M{"$pull": bson.M{"classes": classID}},
    ); err != nil {
        log.Fatalf("Error deleting class from a studio: %v", err)
    }

    if _, err := db.Collection(trainers).UpdateOne(
        ctx,
```

```
            bson.M{"_id": class.TrainerID},
            bson.M{"$pull": bson.M{"classes": classID}},
        ); err != nil {
            log.Fatalf("Error deleting class from a trainer: %v", err)
        }

        for _, client := range class.Clients {
            if _, err := db.Collection(clients).UpdateOne(
                ctx,
                bson.M{"_id": client},
                bson.M{"$pull": bson.M{"classes": classID}},
            ); err != nil {
                log.Fatalf("Error deleting class from a client: %v", err)
            }
        }

        if _, err := db.Collection(classes).DeleteOne(ctx, bson.M{"_id": clas
    sID}); err != nil {
            log.Fatalf("Error deleting class: %v", err)
        }

        fmt.Printf("Successfully deleted class (%q)\n", classID)
    }
```

Используется по одному запросу в коллекции "studios", "trainers" и "clients" и два запроса в коллекцию "classes".

Удаление записи:

```
    func (r Repository) DeleteAppointment(clientID, classID bson.ObjectID) {
        db := r.mg.Database("yoga")
        ctx := context.TODO()

        if _, err := db.Collection(classes).UpdateOne(
            ctx,
            bson.M{"_id": classID},
            bson.M{"$pull": bson.M{"clients": clientID}},
        ); err != nil {
            log.Fatalf("Error deleting client from a class: %v", err)
        }

        if _, err := db.Collection(clients).UpdateOne(
            ctx,
            bson.M{"_id": clientID},
            bson.M{"$pull": bson.M{"classes": classID}},
        ); err != nil {
            log.Fatalf("Error deleting class from a client: %v", err)
        }
    }
```

Используется по одному запросу в две коллекции.

Получение занятий пользователя:

```
    func (r Repository) GetClientClasses(clientID bson.ObjectID) []Class {
        ctx := context.TODO()
```

```
    classesCursor, err := r.mg.Database("yoga").Collection(classes).Find(
        ctx, bson.M{"clients": clientID})
    if err != nil {
        log.Fatalf("Error getting classes: %v", err)
    }
    defer func(classesCursor *mongo.Cursor, ctx context.Context) {
        err = classesCursor.Close(ctx)
        if err != nil {
            log.Fatalf("Error closing cursor: %v", err)
        }
    }(classesCursor, ctx)

    var classes []Class
    if err = classesCursor.All(ctx, &classes); err != nil {
        log.Fatalf("Error getting classes: %v", err)
    }

    if err = classesCursor.Err(); err != nil {
        log.Fatalf("Error getting classes: %v", err)
    }

    return classes
}
```

Используется один запрос в одну коллекцию.

Получение статистики по студии:

```
type Statistics struct {
    Interval Interval
    Count    int
}

type Interval struct {
    Begin time.Time
    End   time.Time
}

func (r Repository) GetStatistics(studioID bson.ObjectID, interval Interv
al) Statistics {
    db := r.mg.Database("yoga")
    ctx := context.TODO()

    classesCursor, err := db.Collection(classes).Find(
        ctx,
        bson.M{
            "studio_id": studioID,
            "time": bson.M{
                "$gte": interval.Begin,
                "$lte": interval.End,
            },
        },
    )
    if err != nil {
        log.Fatalf("Error finding classes: %v", err)
    }
```

```
    defer func(classesCursor *mongo.Cursor, ctx context.Context) {
        err = classesCursor.Close(ctx)
        if err != nil {
            log.Fatalf("Error closing classes: %v", err)
        }
    }(classesCursor, ctx)

    clientsSet := map[bson.ObjectID]struct{}{}

    for classesCursor.Next(ctx) {
        var class Class
        if err := classesCursor.Decode(&class); err != nil {
            log.Fatalf("Error decoding class: %v", err)
        }

        for _, client := range class.Clients {
            clientsSet[client] = struct{}{}
        }
    }

    if err = classesCursor.Err(); err != nil {
        log.Fatalf("Error during cursor iteration: %v", err)
    }

    return Statistics{
        Count:    len(clientsSet),
        Interval: interval,
    }
}
```

Используется один запрос в одну коллекцию.

Получение тренера по имени:

```
func (r Repository) GetTrainerByName(name string) Trainer {
    collection := r.mg.Database("yoga").Collection(trainers)

    var trainer Trainer
    if err := collection.FindOne(
        context.TODO(),
        bson.M{
            "name": bson.M{
                "$regex":   name, // to match substrings
                "$options": "i",  // ignore case
            },
        },
    ).Decode(&trainer); err != nil {
        if errors.Is(err, mongo.ErrNoDocuments) {
            fmt.Printf("Trainer with name %q not found\n", name)
            return Trainer{}
        }
        log.Fatalf("Error finding trainer: %v", err)
    }

    return trainer
}
```

Используется один запрос в одну коллекцию.

Получение пользователей без записей на занятия:

```
func (r Repository) FindClientsWithNoClasses() []Client {
    collection := r.mg.Database("yoga").Collection(clients)
    ctx := context.TODO()

    cursor, err := collection.Find(
        ctx,
        bson.M{"classes": bson.M{"$size": 0}},
    )
    if err != nil {
        log.Fatalf("Error finding classes: %v", err)
    }
    defer func(cursor *mongo.Cursor, ctx context.Context) {
        err = cursor.Close(ctx)
        if err != nil {
            log.Fatalf("Error closing cursor: %v", err)
        }
    }(cursor, context.TODO())

    var clients []Client
    if err = cursor.All(ctx, &clients); err != nil {
        log.Fatalf("Error getting clients: %v", err)
    }

    if err = cursor.Err(); err != nil {
        log.Fatalf("Error finding clients: %v", err)
    }

    return clients
}
```

Используется один запрос в одну коллекцию.

Получение самых загруженных тренеров:

```
func (r Repository) FindTrainersWithMaxClasses(limit int) []Trainer {
    collection := r.mg.Database("yoga").Collection(trainers)
    ctx := context.TODO()

    cursor, err := collection.Aggregate(
        ctx,
        []bson.M{
            {"$addFields": bson.M{"class_count": bson.M{"$size": "$classe
s"}}},
            {"$sort": bson.M{"classes_count": -1}},
            {"$limit": limit},
        },
    )
    if err != nil {
        log.Fatalf("Error finding classes: %v", err)
    }
    defer func(cursor *mongo.Cursor, ctx context.Context) {
        err = cursor.Close(ctx)
        if err != nil {
```

```
            log.Fatalf("Error closing cursor: %v", err)
        }
    }(cursor, ctx)

    var trainers []Trainer
    if err = cursor.All(ctx, &trainers); err != nil {
        log.Fatalf("Error getting trainers: %v", err)
    }

    if err = cursor.Err(); err != nil {
        log.Fatalf("Error finding trainers: %v", err)
    }

    return trainers
}
```

Используется один агрегированный запрос в одну коллекцию.

Наименее популярные студии за промежуток времени:

```
func (r Repository) FindStudiosWithMinClassesByTimeInterval(
    interval Interval, limit int,
) []Studio {
    db := r.mg.Database("yoga")
    ctx := context.TODO()

    classCursor, err := db.Collection(classes).Aggregate(
        ctx,
        []bson.M{
            {"$match": bson.M{
                "time": bson.M{"$gte": interval.Begin, "$lte": interval.E
nd},
            }},
            {"$group": bson.M{
                "_id":         "$studio_id",
                "class_count": bson.M{"$sum": 1},
            }},
            {"$sort": bson.M{"classes_count": 1}},
            {"$limit": limit},
        },
    )
    if err != nil {
        log.Fatalf("Error finding classes: %v", err)
    }
    defer func(cursor *mongo.Cursor, ctx context.Context) {
        err = cursor.Close(ctx)
        if err != nil {
            log.Fatalf("Error closing classCursor: %v", err)
        }
    }(classCursor, ctx)

    var results []struct {
        StudioID   bson.ObjectID `bson:"_id"`
        ClassCount int           `bson:"class_count"`
    }
    if err = classCursor.All(ctx, &results); err != nil {
```

```
            log.Fatalf("Error getting results: %v", err)
        }

        if err = classCursor.Err(); err != nil {
            log.Fatalf("Error finding results: %v", err)
        }

        var studioIDs []bson.ObjectID
        for _, result := range results {
            studioIDs = append(studioIDs, result.StudioID)
        }

        studioCursor, err := db.Collection(studios).Aggregate(
            ctx,
            bson.M{"_id": bson.M{"$in": studioIDs}},
        )
        if err != nil {
            log.Fatalf("Error finding studios: %v", err)
        }
        defer func(studioCursor *mongo.Cursor, ctx context.Context) {
            err = studioCursor.Close(ctx)
            if err != nil {
                log.Fatalf("Error closing studioCursor: %v", err)
            }
        }(studioCursor, ctx)

        var studios []Studio
        if err = studioCursor.All(ctx, &studios); err != nil {
            log.Fatalf("Error getting studios: %v", err)
        }

        if err = studioCursor.Err(); err != nil {
            log.Fatalf("Error finding studios: %v", err)
        }

        return studios
    }
```

Используется по одному запросу в две коллекции.

2. Реляционная модель

Графическое представление модели – Рисунок 8.

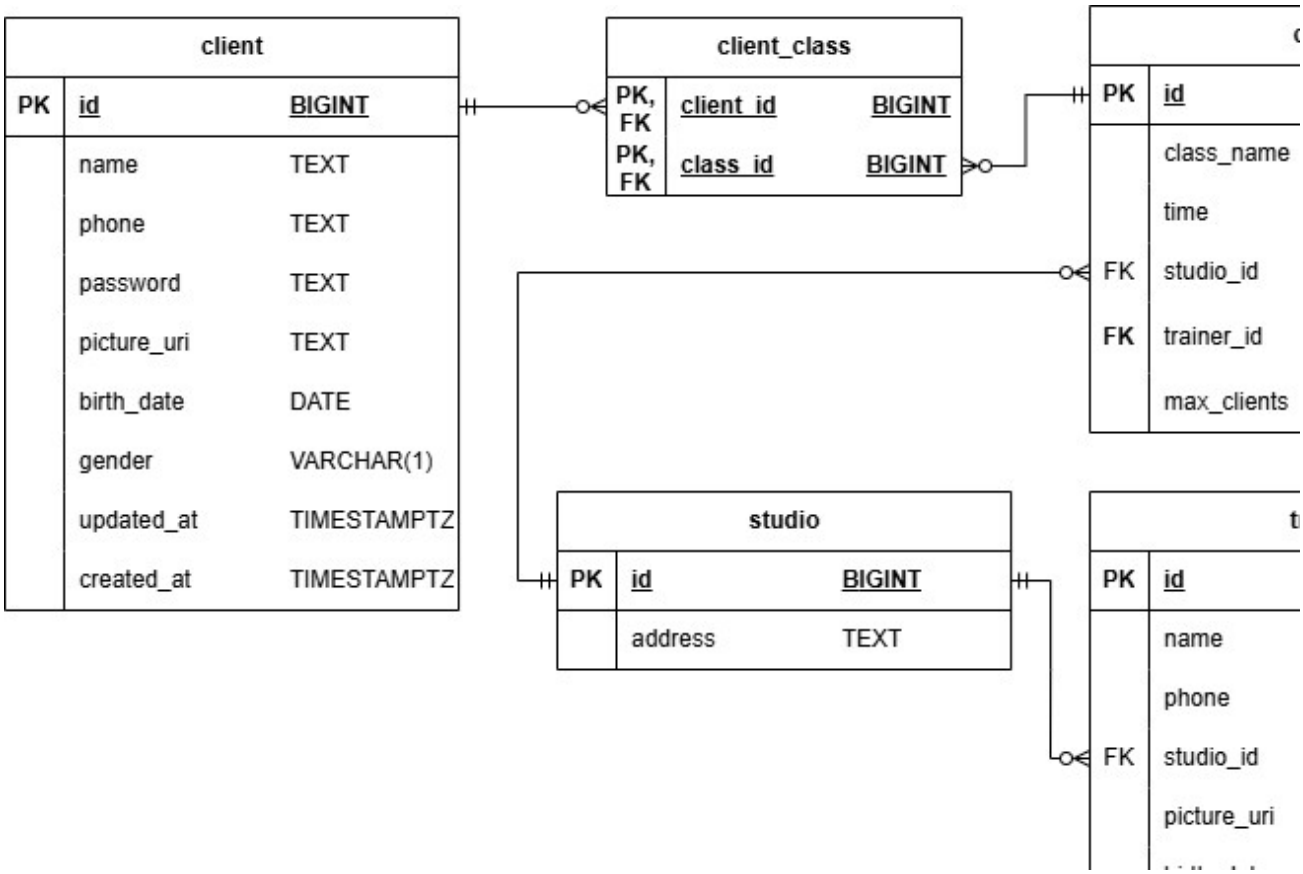

Рисунок 8 – ER диаграмма

Описание назначений коллекций, типов данных и сущностей

client - клиент студий йоги (~156 байт):

_id (BIGINT) - 8 байт) - уникальный идентификатор (id) клиента

name (TEXT - в среднем 50 байт) - имя клиента

phone (TEXT - 15 байт) - номер телефона клиента

password (TEXT - в среднем 12 байт) - пароль клиента для входа в свой профиль

picture_uri (TEXT - в среднем 50 байт) - ссылка на изображение клиента

birth_date (DATE - ~4 байт) - дата рождения клиента

gender (VARCHAR(1) - 1 байт) - пол клиента

updated_at (TIMESTAMPTZ - 8 байт) - время последнего обновления данных клиента

created_at (TIMESTAMPTZ - 8 байт) - время создания учётной записи клиента

trainer - тренер студии йоги (~152 байт):

_id (BIGINT - 8 байт) - уникальный идентификатор (id) тренера

name (TEXT - в среднем 50 байт) - имя тренера

phone (TEXT - 15 байт) - номер телефона тренера

studio_id (BIGINT - 8 байт) - идентификатор студии, в которой работает тренер

picture_uri (TEXT - в среднем 50 байт) - ссылка на изображение тренера

birth_date (DATE - ~4 байт) - дата рождения тренера

gender (VARCHAR(1) - 1 байт) - пол тренера

updated_at (TIMESTAMPTZ - 8 байт) - время последнего обновления данных тренера

created_at (TIMESTAMPTZ - 8 байт) - время создания учётной записи тренера

studio - студия йоги (~108 байт):

_id (BIGINT - 8 байт) - уникальный идентификатор (id) студии

address (TEXT - в среднем 100 байт) - адрес студии

class - занятие (~51 байт):

_id (BIGINT - 8 байт) - уникальный идентификатор (id) занятия

class_name (TEXT - в среднем 15 байт) - название занятия

time (TIMESTAMPTZ - 8 байт) - время начала занятия

max_clients (INT - 4 байта) - максимальное количество людей на занятии

studio_id (BIGINT - 8 байт) - идентификатор студии, в которой проводится занятие

trainer_id (BIGINT - 8 байт) - идентификатор тренера, который проводит занятие

client_class - отношение клиент-занятие (~16 байт):

client_id (BIGINT - 8 байт) - уникальный идентификатор (id) клиента

class_id (BIGINT - 8 байт) - уникальный идентификатор (id) занятия

Оценка объема информации, хранимой в модели

Пусть n - количество студий йоги. Каждый тренер проводит по ~3 занятия в день. В год студия работает ~350 дней. На каждое занятие записывается ~8 клиентов. Итого:

$$V(n) = \left(108 + \left(\frac{4320 \cdot 8}{96} \cdot 156\right) + (4 \cdot 152) + (4320 \cdot 51) + (8 \cdot 4320 \cdot 16)\right) n = 830\,156n$$

Избыточность данных

В class поле studio_id и trainer_id избыточны (16 байт). В trainer поле studio_id избыточно (8 байт). Отношение client_class избыточно (16 байт).

Итого:

$$V(n) = \left(108 + \left(\frac{4320 \cdot 8}{96} \cdot 156\right) + (4 \cdot 144) + (4320 \cdot 35)\right) n = 208\,044n$$

Избыточность как отношение объема модели к “чистому” объему:

$$\frac{V(n)}{V_{clean}(n)} = \frac{830\,156n}{208\,044n} \approx 3.990$$

Направление роста модели при увеличении количества объектов каждой сущности

При добавлении объекта любой коллекции никаких дополнительных объектов добавляться не будет.

Примеры данных

client

id	name	phone	password	picture_uri	birth_date	gender	update_at	created_at
1	Liza	+7(315) 123-1416	12341234	“https://cdn.example1.com”	2001-09-28	F	2021-04-05T05:00:00+02:00	2021-04-05T05:00:00+02:00
2	Misha	+7(314) 419-1591	43214321	“https://cdn.example2.com”	2002-07-23	M	2022-04-05T05:00:00+02:00	2021-04-05T05:00:00+02:00

trainer

id	name	phone	studio_id	birth_date	gender	update_at	created_at
1	Klim	+7(515)123-1416	1	2001-09-28	M	2021-04-05T05:00:00+02:00	2021-04-05T05:00:00+02:00
2	Peter	+7(716)419-1591	2	2002-07-23	M	2022-04-05T05:00:00+02:00	2021-04-05T05:00:00+02:00

class

id	class_name	time	studio_id	trainer_id	max_clients
1	Klim	2024-10-10T08:00:00+03:00	1	1	8
2	Peter	2024-10-10T12:00:00+03:00	2	2	8

studio

id	adress
1	Saint-Petersburg, str. Bronnaya 22
2	Saint-Petersburg, str. Studencheskaya 124

client_class

client_id	class_id
1	1
2	2

Примеры запросов

Авторизация:

```
SELECT COUNT(*) FROM Client
WHERE phone = "+7(999)999-9999" AND password = "111111"
```

Если запрос вернул не 1, значит невозможно авторизоваться.

Регистрация:

```
INSERT INTO Client(phone, name, password, updated_at, created_at)
    VALUES ("+7(999)999-9999", "Liza", "11111", @updated_at, @created_at)
```

Создание занятия:

```
INSERT INTO Class(class_name, time, studio_id, trainer_id)
     VALUES ("Yoga", "2024-11-01T12:30:00+03:00", 2, 1)
```

Удаление занятия:

```
DELETE FROM Class
WHERE id = 2
```

Удаление записи клиента:

```
DELETE FROM Client_Class
WHERE client_id = 2 AND class_id = 3
```

Изменение данных клиента:

```
UPDATE Client
SET photo_uri = "https://cdn.example3.com"
WHERE id = 1
```

Получение всех занятий клиента

```
SELECT c.*
FROM Client_class cc
        INNER JOIN Class c ON c.id = cc.class_id
WHERE cliend_id = 2
```

Поиск клиентов, которые не имеют записей в классы

```
SELECT c.*
FROM Client c
WHERE c.id NOT IN (
    SELECT DISTINCT cc.client_id
    FROM Client_class cc
)
```

Получение всех занятий тренера:

```
SELECT Class.class_name, Class.time, Class.max_clients, Trainer.name
FROM Trainer
    JOIN Class ON Trainer.id = Class.trainer_id
```

Поиск самых нагруженных тренеров (у которых больше всего классов):

```
SELECT tr.*
FROM (
    SELECT t.id, t.name, COUNT(t.id) AS classes_count
    FROM Trainer t
        JOIN Class c ON t.id = c.trainer_id
    GROUP BY t.id
) tr
ORDER BY tr.classes_count DESC
LIMIT 10
```

Получение статистики посещаемости студии:

```
SELECT SUM(COUNT(client_id)) FROM Client_Class
WHERE class_id IN (
    SELECT class_id FROM Class c
    WHERE studio_id = 2 AND c.time BETWEEN '2024-11-01T12:30:00+03:00' an
d '2024-12-01T12:30:00+03:00')
GROUP BY class_id
```

Получение студий, где меньше всего классов проводится на протяжении последнего месяца:

```
SELECT st.*
FROM (
    SELECT s.id, s.adress, COUNT(c.id) AS classes_count
    FROM Studio s
        JOIN Class c ON s.id = c.studio_id
    WHERE c.time >= DATE_TRUNC('month', CURRENT_TIMESTAMP - INTERVAL '1 m
onth'))
    GROUP BY s.id
) st
ORDER BY st.classes_count ASC
LIMIT 10
```

Импорт на примере Studio:

```
COPY Class(studio_id, address)
FROM 'example.csv'
DELIMITER ','
CSV HEADER
```

Экспорт на примере Studio:

```
COPY (SELECT * FROM Studio)
TO 'output.csv'
DELIMITER ','
CSV HEADER
```

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

Количество запросов для всех юзкейсов равно 1 и не зависит от числа объектов в БД.

Количество задействованных коллекций (если есть)

Авторизация - 1

Регистрация - 1

Создание занятия - 1

Удаление занятия - 1

Удаление записи клиента - 1

Изменение данных клиента - 1

Получение всех занятий клиента - 2

Получение всех занятий тренера - 2

Получение статистики посещаемости студии - 2

Импорт на примере Studio - 1

Экспорт на примере Studio - 1

Поиск клиентов, которые не имеют записей в классы - 2

Поиск самых нагруженных тренеров (у которых больше всего классов) - 2

Получение студий, где меньше всего классов проводится на протяжении последнего месяца - 2

Сравнение моделей

Удельный объем информации

Реляционная модель занимает меньший объем памяти чем нереляционная, так как в нереляционной модели данные дублируются для уменьшения сложности запросов к БД.

Вывод

Использование реляционной модели данных даёт преимущества и в объёме хранимых данных, и в операциях изменения, удаления и вставки записей.

Однако нереляционная модель выигрышно себя показывает на более частых сценариях использования: за счёт дублирования информации получение данных из базы с различного рода фильтрами происходит с использованием меньшего числа запросов и коллекций, — именно эти запросы являются наиболее частыми в нашем случае, поэтому и использование реляционной модели оправданно.

3. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

Разработанное приложение состоит из следующих модулей:

1. Frontend модуль:
   - Web-интерфейс на Vue.js
   - Отвечает за визуализацию данных и взаимодействие с пользователем
2. Backend модуль (Go):
   - API-сервер для обработки запросов
   - Бизнес-логика приложения
   - Управление авторизацией
   - Обработка импорта/экспорта данных
3. Модуль хранения данных:
   - MongoDB для данных (информация о пользователях, занятиях, студиях, тренерах)

Использованные технологии:

Frontend: Vue.js

Backend: Go

База данных: MongoDB

Контейнеризация: Docker

Полученные результаты – Рисунки 9-16.

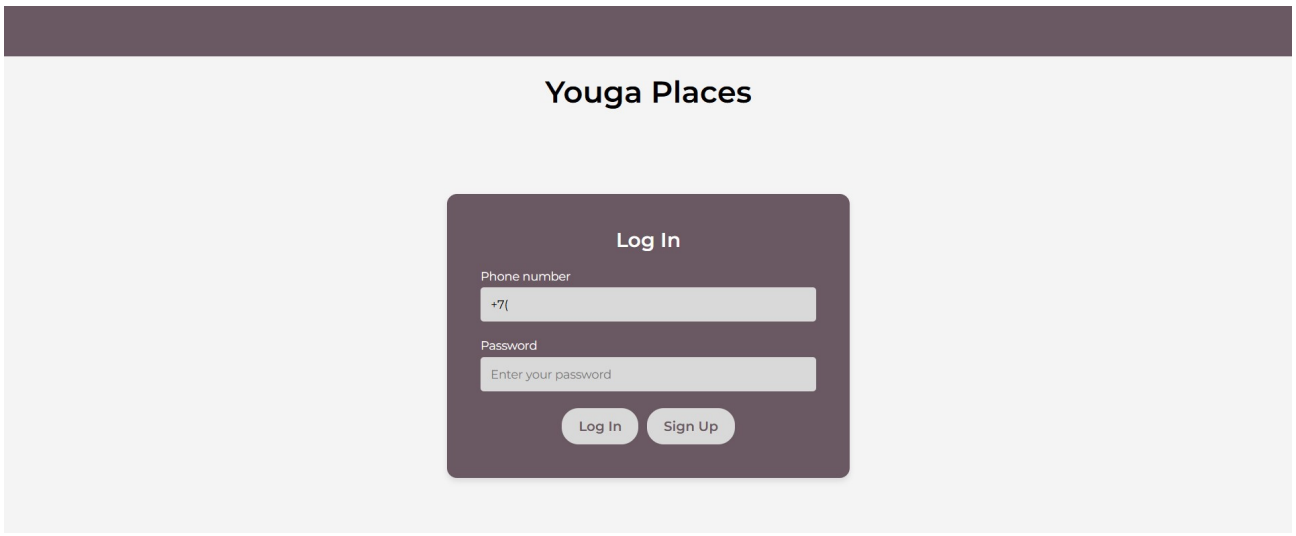

Рисунок 9 – Страница входа

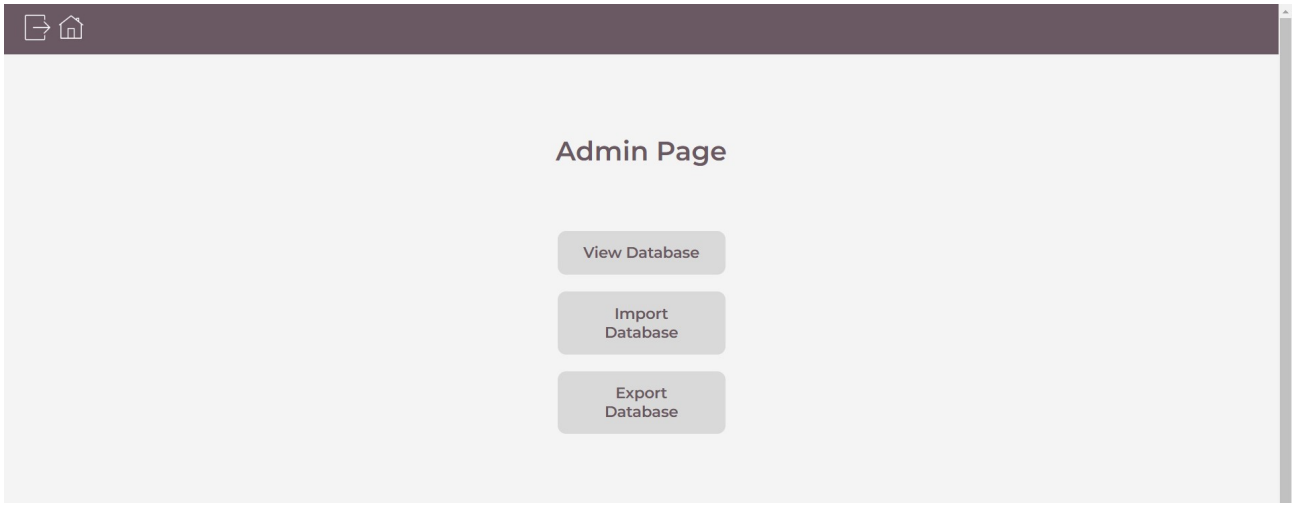

Рисунок 10 – Страница администратора

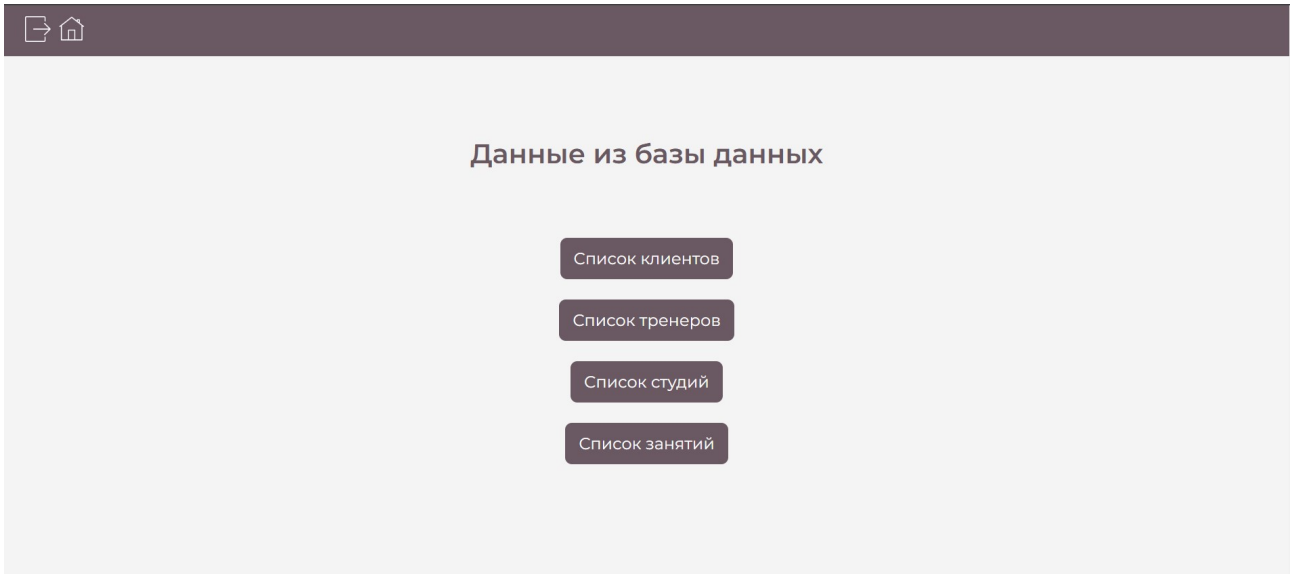

Рисунок 11 – Страница данных из БД

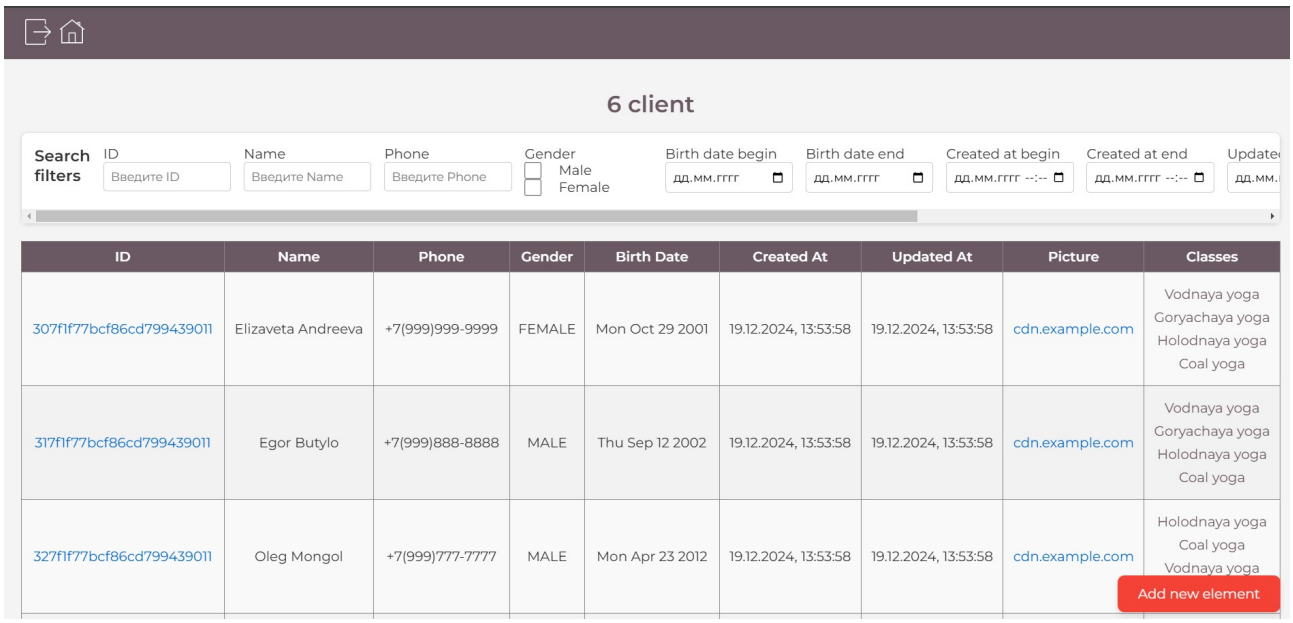

Рисунок 12 – Страница сущностей

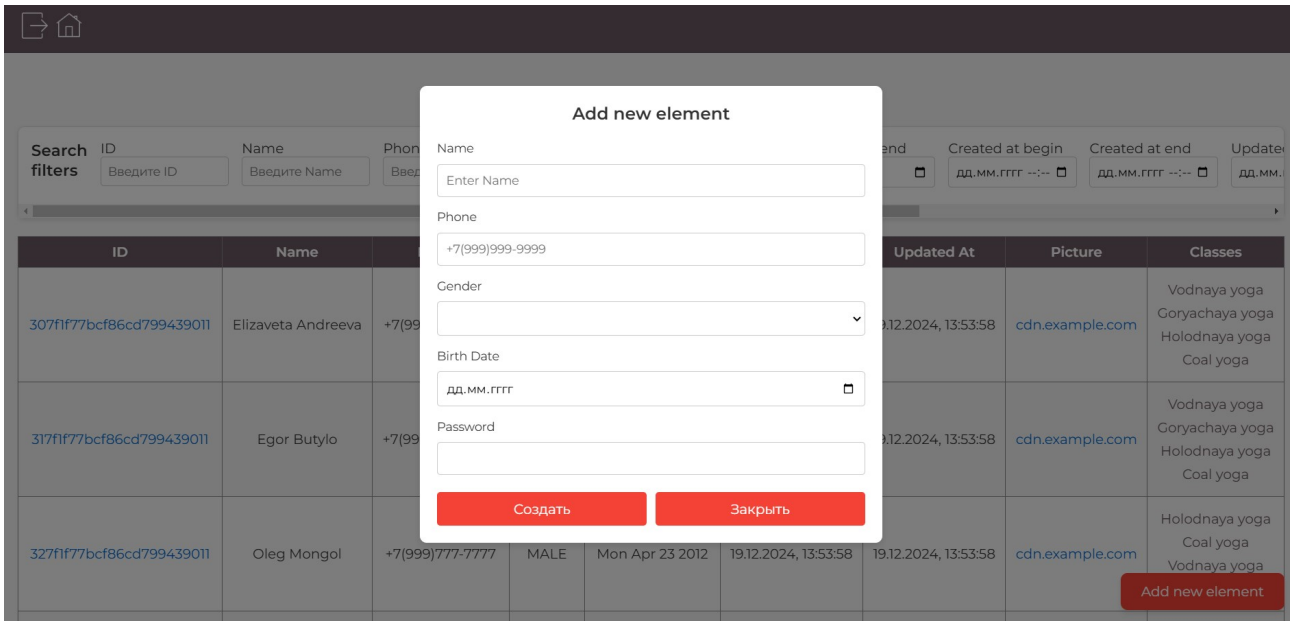

Рисунок 13 – Страница добавления сущности

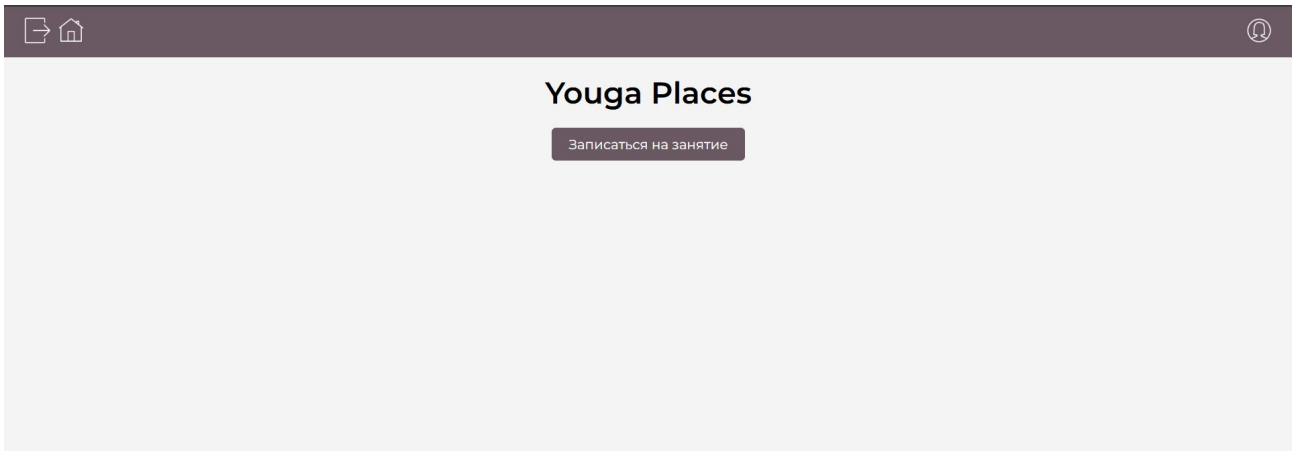

Рисунок 14 – Домашняя страница пользователя

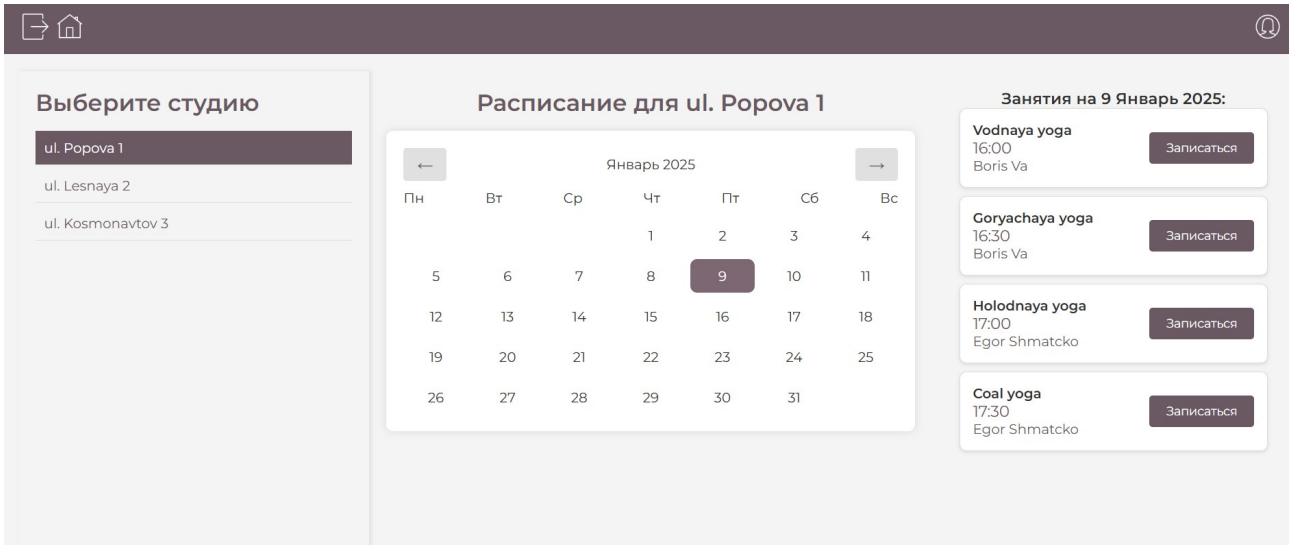

Рисунок 15 – Страница просмотра и записи на занятия

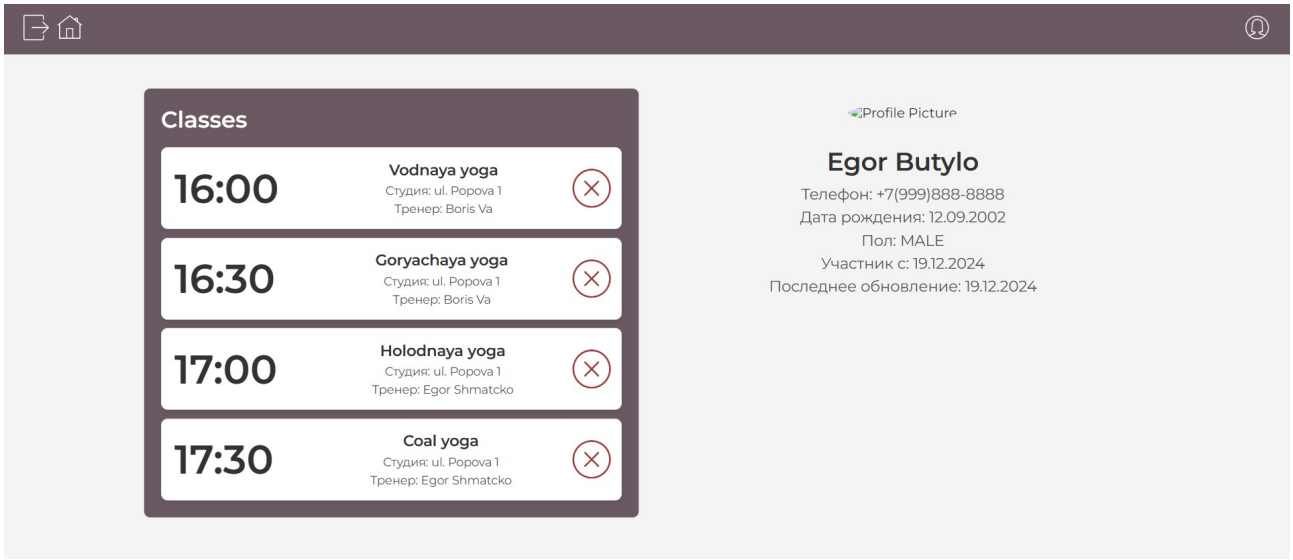

Рисунок 16 – Страница профиля пользователя

4. ВЫВОДЫ

Достигнутые результаты:

Разработанное веб-приложение успешно автоматизирует процесс записи на занятия в студиях йоги. Оно предоставляет клиентам удобный доступ к расписанию, возможность записи и отмены занятий. Администраторам обеспечен эффективный инструмент для управления данными. Приложение повышает удовлетворенность пользователей, улучшает планирование занятий и оптимизирует рабочие процессы студий.

Недостатки и пути улучшения:

- Возможность интеграции с платежными системами для автоматизации оплаты занятий.
- Улучшение системы уведомлений для своевременного оповещения пользователей об изменениях в расписании или отменах занятий.
- Оптимизация интерфейса для повышения удобства работы с приложением на мобильных устройствах.

Будущее развитие решения:

- Расширение функционала для поддержки нескольких студий с единым управлением.
- Добавление аналитических инструментов для прогнозирования популярности занятий.
- Интеграция с социальными сетями и мессенджерами для упрощения процесса регистрации и уведомления клиентов.
- Внедрение персонализированных рекомендаций по занятиям на основе пользовательских предпочтений и посещаемости.

5. ПРИЛОЖЕНИЕ

Запуск приложения:

1. Склонировать репозиторий:

   git clone https://github.com/moevm/nosql2h24-yoga-sch.git

2. Перейти в корневую папку проекта

3. Развернуть Docker:

   docker compose build

   docker compose up

4. В браузере открыть http://localhost:8080/

Функционал:

- Просмотр занятий студий йоги
- Запись и удаление записи на занятие
- Просмотр всех сущностей из БД

Требования

- Docker
- Ubuntu 22.04+
- Свободные порты 8443 и 8080

6. ЛИТЕРАТУРА

1. Проект сервис-агрегатор для поиска свободных мест на занятиях йоги. GitHub Repository – https://github.com/moevm/nosql2h24-yoga-sch
2. Go. Official page – https://go.dev/
3. MongoDB. Official page – https://www.mongodb.com/
4. Vue.js. Official page - https://vuejs.org/