Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА АНАЛИЗА ДАННЫХ И ТЕХНОЛОГИЙ ПРОГРАММИРОВАНИЯ

Направление: 09.03.03 – Прикладная информатика

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

СИСТЕМА ЗАПИСИ НА ПРИЁМ В МЕДИЦИНСКОЕ УЧРЕЖДЕНИЕ

Работа завершена:
Студент 4 курса
группы 09-951
«____»_____ 2023г. _____/Колесников Д.А.

Работа допущена к защите:
Научный руководитель
старший преподаватель
«____»_____ 2023г. _____/Еникеев И.А.

Заведующий кафедрой анализа данных
и технологий программирования
«____»_____ 2023г. _____/Бандеров В.В.

Казань — 2023

Содержание

Глоссарий

РКИБ - ГАУЗ «Республиканская клиническая инфекционная больница имени Агафонова»

Нормализация - Процесс организации данных в базе данных, нацеленный на повышение её гибкости и защищённости

API - Описание способов взаимодействия одной компьютерной программы с другими [2]

CRUD - акроним, обозначающий четыре базовые функции, используемые при работе с базами данных: создание (create), чтение (read), модификация (update), удаление (delete)

Сущность (Entity) - объект, представляющий запись в базе данных

URL - Строка, используемая для идентификации ресурса в Интернете

Конечная точка - url, на который приложения отправляют запросы для взаимодействия с веб-сервисами

Введение

Медицинской отрасли не хватает информатизации. Это устоявшаяся сфера, однако она не соответствует современности: никто не хочет стоять в очередях, вручную заполнять документы, потерять выданный рецепт. В 2018 году Правительство РФ создало национальный проект «Здравоохранение», направленный на улучшение медицины. Одна из задач проекта - перенос Минздравом в электронный формат части услуг: выдача рецептов, запись к врачу, подача заявления на полис, хранение медицинских документов [1].

На деле перевели услуги в электронный формат не везде. В России 30 000 медицинских учреждений и нужно много времени для внедрения системы в каждое из них. При этом нужно учитывать особенности каждого учреждения - то, что подойдёт больнице в Москве, может не подойти поликлинике из маленького города.

Одно из учреждений так и не перешедших в электронный формат - Республиканская клиническая инфекционная больница имени Агафонова (далее РКИБ). Сейчас в неё активно внедряются цифровые технологии, например - оплата услуг. Но некоторые элементы остаются прежними, в частности, запись на приём. С ней то нам и предстоит разобраться.

Цель: Информатизировать запись на приём в РКИБ

Задачи:

1. Изучить предметную область
2. Проанализировать существующие аналоги
3. Составить техническое задание
4. Спроектировать части будущей системы
5. Реализовать спроектированные части системы

Объект исследования: Информатизация в системе здравоохранения

Предмет исследования: Запись на приём в медицинское учреждение

Структура: В первой части работы анализируется предметная область и определяется необходимый функционал. Во второй главе будущая система проектируется, в третьей - реализуется. Заключительная, четвертая часть - введение готовой системы в эксплуатацию.

1. Предметная область

1.1. Основные сведения

РКИБ специализируется на помощи с инфекционными патологиями. Сюда приходят лечиться люди с хроническими заболеваниями и те, кто хочет исследоваться на наличие болезней. Другие поликлиники направляют пациентов в РКИБ, потому что здесь предоставляется большое количество услуг: рентген, диагностика и лечение разных болезней, проведение лабораторных исследований, содержание больных в стационаре.

В этой больнице нет живых очередей. Каждый пациент записывается на какое-то время и принимается только по записи. Нет ситуаций, когда приходят ”Только спросить”или когда требуется неотложная помощь. Этим РКИБ отличается от большинства других поликлиник.

Врачи сами задают время своего приёма. Они составляют расписание на 2 недели вперёд, но оно может и измениться, например, если врач заболел. Расписание передаётся в регистратуру, где пациентов и записывают на приём.

1.2. Существующие решения

1.2.1. Текущее решение проблемы

На момент начала работы в РКИБ нет никакой информатизированной системы записи на приём. На каждого врача и, иногда, услугу заведена отдельная тетрадь. Пример такой тетради приведён на Рисунке 1.

Пациент, приходя в больницу, иду в регистратуру. Там его вручную записывают в тетрадь на конкретное время к конкретному врачу, вписывают его данные в тетрадь. Далее в указанное время человек приходит к врачу. На первый взгляд схема проста и в ней сложно допустить ошибки, но если посмотреть внимательнее, то у неё много недостатков:

- **Отсутствие синхронизации.** Если пациент отменил запись, то изменение нужно сделать везде, где было упоминание о приёме.
- **Невозможность копирования.** Каждый врач должен несколько раз в день сверяться с записью к нему, потому что актуальная копия только в регистратуре. При составлении отчётов также требуется переносить сведения о приёме. То есть вручную переписывать сотни строк.

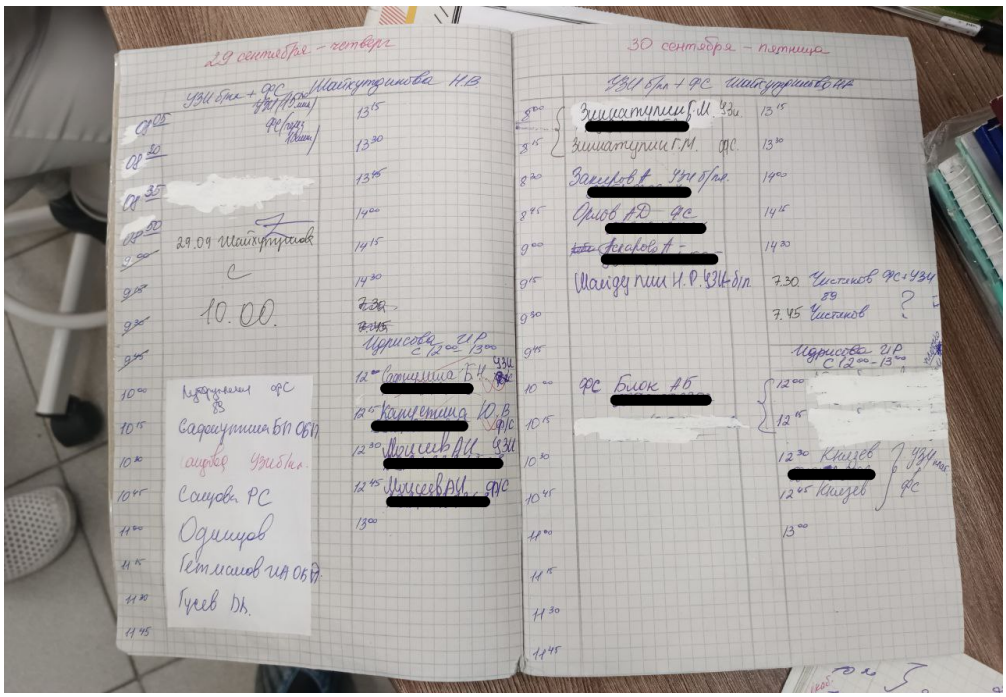

Рисунок 1. Пример заполненной тетради на УЗИ+ФС. Чёрным закрашены номера телефонов

- **Невозможность обработки.** Нужно постараться, чтобы найти данные о пациенте, который приходил месяц назад. Если тетрадь поменялась, то потребуется ещё и находить архивные записи. Восстановить таким образом графики врачей задача тоже не из простых.
- **Дублирование информации.** В РКИБ, по словам сотрудников, много регулярных пациентов. Если кто-то приходит каждый месяц, то в тетрадях он будет записан столько раз, сколько приходил. Редко когда у человека меняется, например, отчество или дата рождения. Эти данные записываются по нескольку раз, хотя это не имеет никакого смысла.
- **Невозможность качественного планирования расписания.** У врачей часто меняется график работы, а в тетради строки фиксированы и ограничены. При любом изменении в графике нужно вручную найти конфликтующие пункты и изменить их.
- **Хранение неактуальной информации.** Человек, поменявший место работы, навсегда в какой-то из тетрадей останется на прошлом.

Несмотря на недостатки, РКИБ использует этот способ. Всё из-за того, что он прост для понимания, доступен и к нему привыкли. Разработанная в результате система должна быть приближена к этому, чтобы под неё не нужно было долго переучиваться.

1.2.2. 1С:Медицина

«1С:Медицина» - решение от компании 1С. Продуманное, сделанное профессионалами, проверенное временем. Пример интерфейса на Рисунке 2. Это мог бы быть хороший вариант, но у него тоже есть недостатки:

- **Отсутствие интеграции с существующими системами.** Несмотря на то, что в РКИБ слабо информатизирована, что-то у неё уже есть. Переход на 1С означает отказ от практически всего разработано ранее, либо дополнительные затраты на интеграцию.
- **Избыточность.** Система нацелена на универсальность, поэтому в ней много фунцкций, не нужных этой больнице, их можно было бы упростить. Сложность системы приводит к сложности её внедрения и обслуживания.
- **Цена и поддержка.** Система стоит не малых денег, а поддержка будет требовать дополнительных вложений. При этом, если верить отзывам из открытых источников, поддержка и документация не всегда могут оперативно помочь.

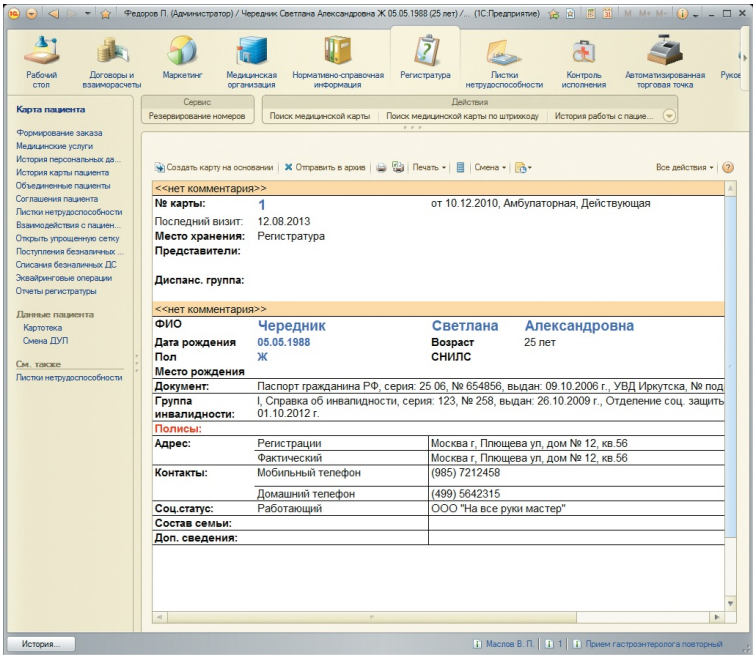

Рисунок 2. Интерфейс 1С:Медицина

1.2.3. Другая предлагавшаяся система

Разрабатываемая система - не первая, которую хотели внедрить в РКИБ. По словам сотрудников, им уже предлагали готовое решение. Проблема в

том, что система не была приспособлена к использованию в этой больнице. Она подходила обычной поликлинике, но РКИБ слишком сильно от них отличается. Здесь нет привычных участков. Обслуживаются не только жители ближайших районов, но и, во том числе, других регионов. Это помешало внедрить её в РКИБ.

Приспособленность к работе в условиях этой конкретной больницы - важное условие для запуска системы.

1.3. Техническое задание

В системе выделяются 4 роли:

- Пациент
- Доктор
- Регистратор
- Администратор

Каждая из них отличается требуемым функционалом. Диаграмма вариантов использования представленя на Рисунке 3. Далее рассмотрим их попо-дробнее.

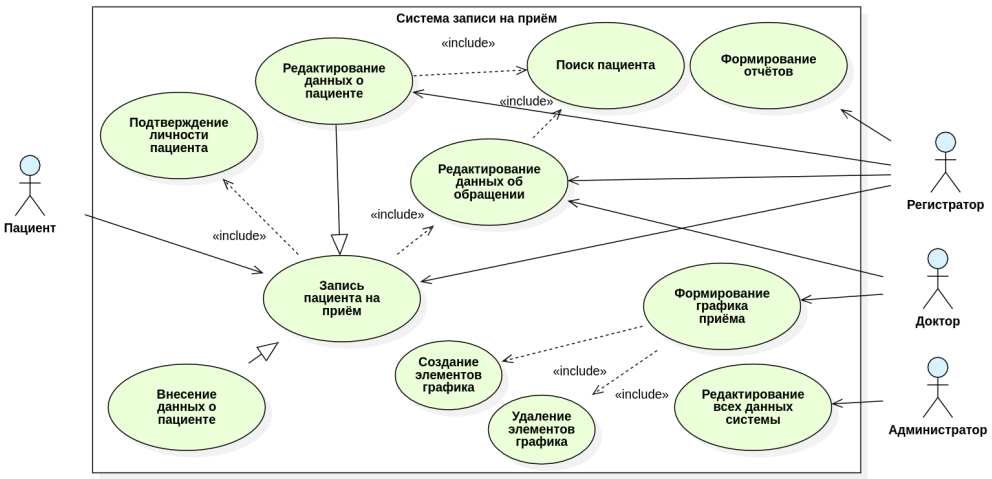

Рисунок 3. Диаграмма вариантов использования

1.3.1. Пациент

1.3.2. Доктор

1.3.3. Регистратор

1.3.4. Администратор

2. Проектирование

Разрабатываемая система - клиент-серверное приложение. Простейшая схема клиент-серверного приложения представлена на Рисунке 4.

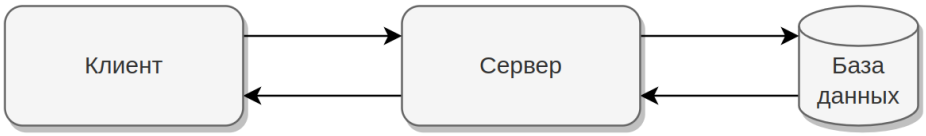

Рисунок 4. Схема клиент-серверного приложения

В этой главе займёмся проектированием указанных частей приложения, а в главе 3 реализуем спроектированные части.

2.1. База данных

Для реализации системы потребуется база данных. Её цель - упростить работу с данными для серверной логики. Появляется выбор - использовать реляционную или нереляционную базу данных.

Нереляционная база данных, с одной стороны, может повысить гибкость приложение, а с другой - усложняет работу с данными. В разрабатываемой системе выигрыш от использования нереляционной базы данных минимален, потому что даже в тетрадях всё записывалось в виде простых таблиц. Реляционная база данные в этом случае подойдёт больше - с ней гораздо проще работать и они лучше подойдут для, например, поиска, потому что список полей фиксирован.

Выделим список компонентов базы данных исходя из данных в пункте 1.2.1.

2.1.1. Основные компоненты

Для создания схемы данных нужно проанализировать задачу: Система должна позволять связывать заявки людей на приём и время приёма. Получается, что основных компонентов должно быть 2: **обращение** и **расписание**. К этому же можно прийти из пункта 1.2.1: Для каждого обращения в тетради используется отдельная строка. Для расписания врачей тоже использует-

ся одна строка на одно время приёма. Простейшая возможная база данных - простой перенос строк в поля таблиц, получим:

- **Обращение:** Содержит данные пациента: ФИО, занятость, адрес для идентификации пациента. Дополнительно содержит направление и диагноз.
- **Расписание:** Включает в себя ФИО врача, услугу, которую он оказывает, время и, опционально, данные об обратившемся человеке. Причём приём может быть сразу на 2 и более ячейки расписания. Связь «Много к одному».

2.1.2. Вспомогательные компоненты

Рассмотрим данные, которые можно было бы вынести в отдельные таблицы, для удобства работы с ними. Дополнительные компоненты можем разделить на 2 категории:

- Отражающие логику. Они напрямую взяты из тетрадей и понятны обычному человеку.
- Упрощающие работу с даными. Нужны только для использования внутри системы.

Отражающие логику:

- **Пациент:** Хранит информацию о пациенте в общем. Нужна для хранения общей информации о человеке, которая не обязательно должна быть у, например, врача. Так совсем не обязательно врачу указывать свой личный телефон, если он не является одновременно пациентом.
- **Сотрудник:** Нужна только для определения кто из людей какую роль занимает в учреждении.
- **Учреждение:** Если пациент пришёл по направлению, то это направление должен был кто-то выдать.
- **Услуга:** Медицинские учреждения предоставляют конкретный список услуг. Таблица фиксирует эти услуги.

Упрощающие работу с данными:

- **Человек:** Сущность, объединяющая пациента и сотрудника. В обеих таблицах есть ФИО. К тому же аккаунт должен быть и у тех, и у тех. Более

того, один и тот же человек может быть и пациентом, и сотрудником. Объединение нормализует данные.

- **Статус обращения:** Вспомогательная таблица. Нужна для определения того, что делать с обращением. Например, чтобы отменённое обращение не показывалось в расписании врача.
- **Статус элемента расписания:** Тоже вспомогательная таблица, но для других целей. Например, если врач заболел, то записанные к нему люди будут отдельно обработаны регистратором.

2.1.3. Логическая схема данных

Исходя из пунктов 2.1.1 и 2.1.2 получаем схему данных, представленную на Рисунке 5.

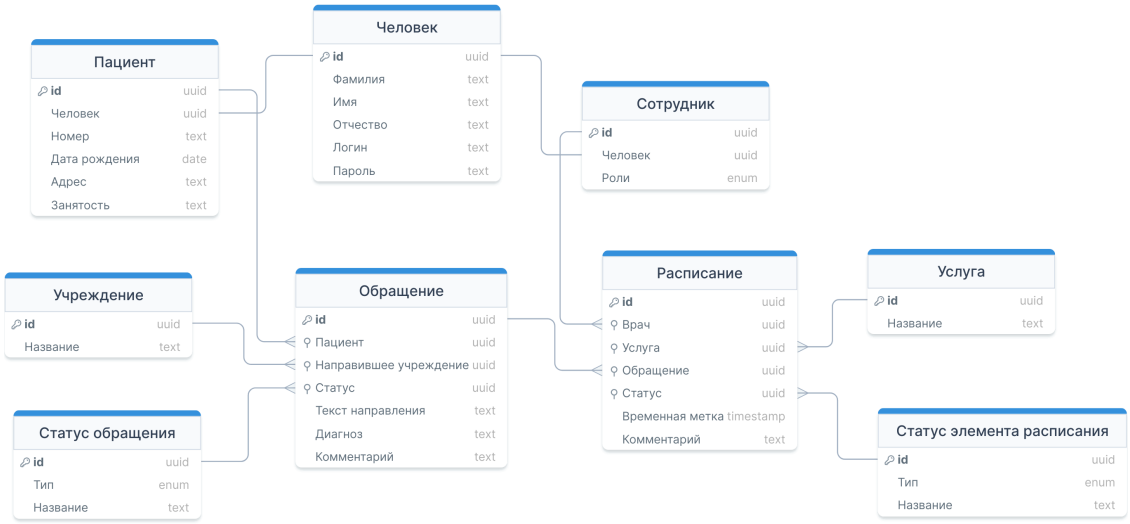

Рисунок 5. Логическая схема данных

2.2. Серверная часть

Клиентский код не должен напрямую обращаться к базе данных. Ему нужно промежуточное звено - сервер. Передавать данные клиент и сервер должны заранее определённым способом - API. Данные между клиентом и сервером передаются в формате, понятном обоим элементам. После преобразования данных в сущности, сервер работает напрямую с бд. Архитектура серверной части представлена на Рисунке 6

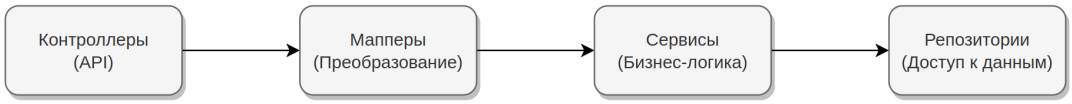

Рисунок 6. Архитектура серверной части

2.2.1. Выбор вида API

В зависимости от потребностей системы можно использовать разные виды API. Сравнение популярных видов API представлено в Таблице 2.

Название	Плюсы	Минусы
REST	- Простота - Универсальность - Поддержка CRUD - Не сохраняет состояние (легко масштабируется)	- Отсутствие строго стандарта - Низкая производительность при большом количестве запросов
SOAP	- Безопасность - Поддержка транзакций - Строгий стандарт	- Большая избыточность данных - Сложность использования и настройки
GraphQL	- Нет избыточных данных - Возможность сложных запросов - Сильная типизация	- Сложность разработки - Сложность масштабирования - Перегрузка из-за сложности запросов

gRPC	- Быстрое выполнение - Сильная типизация	- Сложность в настройке и использовании
JSON-RPC и XML-RPC	- Простота использования и разработки - Лёгкость интеграции	- Нет поддержки HTTP-методов - Затруднение масштабирования

Таблица 1. Популярные виды API

Для разработки был выбран REST. Его минусы в рамках проекта незначительны: в системе не будет большого количества запросов, а отсутствие строгого стандарта всего лишь делает его принципы рекомендациями. При этом REST удовлетворяет всем потребностям приложения - поддержка HTTP-методов, отсутствие сохранения состояния.

2.2.2. Авторизация

Название	Плюсы	Минусы
Basic Auth	- Хорошо работает с HTTPS - Не требует сессий - Прост в реализации	- Нет средств управления сессиями - Передача данных в незашифрованном виде
Digest Auth	- Безопасность из-за хеширования для передачи данных	- Сложность реализации - Нет управления сессиями
Token-based Auth (JWT, OAuth)	- Есть управление сессиями - Передача метаданных	- Сложная реализация и управление токенами - Проблемы при больших объёмах данных
Session-based Auth	- Управление сессиями - Сервер контролирует данные сессии	- Хранение и управление данными сессии - Плохо масштабируется

Таблица 2. Популярные виды API

Для данного проекта была выбрана Basic Auth, потому что он прост в

реализации. Его проблема безопасности решается использованием HTTPS и он хорошо подходит для проектов небольшого масштаба.

Заголовок авторизации - специальная строка, передающаяся вместе с запросом. Выглядеть строка должна так: «Authorization: Basic *токен*», где *токен* - Это зашифрованная с помощью Base64 строка вида «*логин*:*пароль*». При запуске приложения создаётся аккаунт по умолчанию, поэтому для всех методов, кроме установки пароля, потребуется авторизация.

2.2.3. Методы API

Каждый запрос на сервер в разрабатываемой системе характеризуется несколькими параметрами:

- URL
- Метод HTTP
- Заголовок авторизации
- Параметры в теле запроса, либо в URL

Зная эти данные мы можем дать нужный ответ. Теперь нужно определить - кто к каким конечным точкам должен иметь доступ. Для этого нужно составлять таблицы с указанием приведённых выше параметров. Поскольку полное расписывание API займёт очень много места, здесь будет приведён сокращённый вариант - без тел запроса. Подробный вариант можно найти в приложении.

Хорошей практикой считается разделение версий API и выделение его в URL. Поэтому перед каждой конечной точкой в реальных запросах к API дополнительно используется /api/v1, если не сказано иное. Спроектированные методы API представлены в Таблице 3

URL	HTTP метод	Краткое описание
/people/{id}	GET, PUT, DELETE	Получение, изменение, удаление общих данных о человеке
/people/me	PUT, GET	Получение, изменение данных о себе
/people	POST, GET	Получение списка всех пользователей
/patients/{id}	GET, PUT, DELETE	Получение, изменение, удаление пациента
/patients	POST, GET	Получение списка всех пациентов

/employees/{id}	GET, PUT, DELETE	Получение, изменение, удаление сотрудника
/employees	POST, GET	Получение списка всех сотрудников
/institutions/{id}	GET, PUT, DELETE	Получение, изменение, удаление учреждения
/institutions	POST, GET	Получение списка всех учреждений
/appointments/{id}	GET, PUT, DELETE	Получение, изменение, удаление обращения
/appointments	POST, GET	Получение списка всех обращений
/schedules/{id}	GET, PUT, DELETE	Получение, изменение, удаление элемента графика
/schedules	POST, GET	Получение списка всех элементов графика

Таблица 3. Методы API

Отдельно стоит выделить таблицы, к которым должен иметь доступ только администратор. Это такие таблицы, которые меняются очень редко и их изменение сильно влияет на всю систему. Например, изменение в таблице «Услуги». Перечень методов, доступных только Администратору представлен в Таблице 4.

URL	HTTP метод	Краткое описание
/procedures/{id}	GET, PUT, DELETE	Получение, изменение, удаление услуги
/procedures	POST, GET	Добавление услуги, получение всех услуг
/appointmentStatuses/{id}	GET, PUT, DELETE	Получение, изменение, удаление статуса обращения
/appointmentStatuses	POST, GET	Добавление статуса обращения, получение всех статусов обращений
/scheduleStatuses/{id}	GET, PUT, DELETE	Получение, изменение, удаление статуса элемента графика

/scheduleStatuses	POST, GET	Добавление статуса элемента графика, получение всех статусов элементов графика

Таблица 4. Методы API только для Администатора

2.3. Клиентская часть

3. Реализация

3.1. Выбор технологий

3.2. База данных

3.3. Серверная часть

3.4. Клиентская часть

Заключение

Список использованных источников

[1] Федеральный проект «Создание единого цифрового контура в здравоохранении на основе единой государственной информационной системы в сфере здравоохранения (ЕГИСЗ)» - 2019 - 9 августа [Электронный ресурс] - URL: https://minzdrav.gov.ru/poleznye-resursy/natsproektzdravoohranenie/tsifra/ (Дата обращения: 13.12.2022)

[2] Википедия. API - 2023 [Электронный ресурс] - URL: https://ru.wikipedia.org/wiki/API (Дата обращения: 20.01.2023)