**ГЛАВА 2. РАЗРАБОТКА ФУНКЦИОНАЛЬНОЙ И ТЕХНИЧЕСКОЙ АРХИТЕКТУРЫ СИСТЕМЫ**

В предыдущей главе был проведен анализ предметной области, выявлены проблемы существующего процесса организации приёма в ООО "Пансионат «Шексна»" и обоснована необходимость разработки автоматизированной информационной системы "Цифровая приемная" (DRSapi). На основе этого анализа и обзора существующих решений в данной главе будут сформулированы детальные требования к разрабатываемой системе, спроектирована ее архитектура и модель данных.

**2.1 Формализация функциональных и нефункциональных требований к системе "DRSapi"**

Определение требований является ключевым этапом проектирования любой информационной системы, так как именно они ложатся в основу дальнейшей разработки и определяют критерии успешности проекта. Требования к системе "DRSapi" были сформированы на основе анализа бизнес-процессов ООО "Пансионат «Шексна»", изучения потребностей потенциальных пользователей (посетителей, секретарей, принимающих лиц, администраторов) и с учетом необходимости интеграции с существующей корпоративной инфраструктурой.

**2.1.1 Функциональные требования**

Функциональные требования описывают, какие действия и операции должна выполнять система. Они сгруппированы по основным модулям и пользовательским ролям, детализируя сценарии использования Use Cases

1. **Общие функции системы:**

**Авторизация в системе:** Система должна предоставлять возможность аутентификации пользователей с использованием их учетных данных. Для сотрудников ООО "Пансионат «Шексна»" аутентификация должна производиться через интеграцию с Active Directory (AD)

Для начала рассмотрим общие функции для пользователей системы, они наследуются всеми ролями.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

Рисунок 2. Общая диаграмма Use Case

Далее детально рассмотрим каждый процесс.

**Авторизация в системе.** Система должна предоставлять возможность аутентификации пользователей с использованием их учетных данных. Для сотрудников ООО "Пансионат «Шексна»" аутентификация должна производиться через интеграцию с Active Directory (AD).

Изображение выглядит как текст, диаграмма, Параллельный, План

Автоматически созданное описание

Рисунок 3. Диаграмма последовательности “Авторизация в системе”

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание**Выход из системы:** Пользователь должен иметь возможность безопасно завершить сеанс работы с системой.

Рисунок 4. Диаграмма последовательности “Выход из системы”

Изображение выглядит как текст, снимок экрана, число, Параллельный

Автоматически созданное описание**Управление своим профилем:** Аутентифицированный пользователь должен иметь возможность просматривать и редактировать свои контактные данные (например, телефон). Смена пароля для пользователей AD должна происходить через стандартные механизмы AD; для локальных пользователей (если есть) – через систему.

Рисунок 5. Диаграмма последовательности “Управление своим профилем”

1. **Функции для роли "Посетитель":**

Далее рассмотрим посетителя

Изображение выглядит как текст, диаграмма, Шрифт, линия

Автоматически созданное описание

Рисунок 6. Диаграмма Use case для «Посетителя»

**Создание заявки на встречу:** Посетитель должен иметь возможность выбрать принимающее лицо, желаемую дату и время из доступных слотов, указать тему и, при необходимости, место встречи, а также прикрепить сопроводительные документы.

Изображение выглядит как текст, диаграмма, снимок экрана, Параллельный

Автоматически созданное описание

Рисунок 7. Диаграмма последовательности “Создание заявки на встречу”

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание**Просмотр своих встреч:** Посетитель должен видеть список своих созданных и/или предстоящих встреч с указанием их статуса

Рисунок 8. Диаграмма последовательности “Просмотр своих встреч”

**Редактирование своей заявки:** Посетитель должен иметь возможность редактировать детали своей заявки (например, время, тему, место) до тех пор, пока она находится в статусе "Запрошена".

Изображение выглядит как текст, снимок экрана, диаграмма, число

Автоматически созданное описание

Рисунок 9. Диаграмма последовательности “Редактирование своей заявки”

**Отмена своей заявки:** Посетитель должен иметь возможность отменить свою заявку на встречу, если это позволяет ее текущий статус

Изображение выглядит как текст, снимок экрана, диаграмма, число

Автоматически созданное описание

Рисунок 10. Диаграмма последовательности “Отмена своей заявки”

Изображение выглядит как текст, диаграмма, Параллельный, число

Автоматически созданное описание**Прикрепление документа к своей встрече:** Посетитель должен иметь возможность прикреплять файлы к создаваемой или существующей (если это разрешено) встрече.

Рисунок 11. Диаграмма последовательности “Прикрепление документа к своей встрече”

**Скачивание документа со своей встречи:** Посетитель должен иметь возможность скачивать файлы, прикрепленные к его встречам.

Изображение выглядит как текст, снимок экрана, диаграмма, число

Автоматически созданное описание

Рисунок 12. Диаграмма последовательности “Скачивание документа со своей встречи”

**Просмотр своей позиции в очереди:** Если встреча посетителя находится в активной электронной очереди, он должен иметь возможность видеть свою актуальную позицию и примерное время ожидания.

Изображение выглядит как текст, снимок экрана, диаграмма, число

Автоматически созданное описание

Рисунок 11. Диаграмма последовательности “Просмотр своей позиции в очереди”

1. **Функции для роли "Секретарь":**

Изображение выглядит как текст, чек, Шрифт, диаграмма

Автоматически созданное описание  
 Рисунок 12. Диаграмма Use case для «Секретаря»

**Управление графиком руководителя.** Секретарь должен иметь возможность просматривать, создавать, редактировать и отменять встречи для руководителя (или нескольких руководителей), к которому он прикреплен.

Изображение выглядит как текст, диаграмма, снимок экрана, Параллельный

Автоматически созданное описание

Рисунок 13. Диаграмма последовательности “Управление графиком руководителя”

**Просмотр всех встреч руководителя:** Секретарь должен иметь доступ к полному списку встреч своего руководителя с возможностью фильтрации и поиска.

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание

Рисунок 14. Диаграмма последовательности “Просмотр всех встреч руководителя”

**Подтверждение/отклонение заявки:** Секретарь должен иметь возможность подтверждать или отклонять заявки на встречи, поступившие его руководителю.

Изображение выглядит как текст, снимок экрана, диаграмма, число

Автоматически созданное описание

Рисунок 15. Диаграмма последовательности “Подтверждение/отклонение заявки на встречу руководителя”

**Изменение статуса встречи:** Секретарь должен иметь возможность изменять статус встреч руководителя (например, "Перенесена", "Завершена").

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание

Рисунок 16. Диаграмма последовательности “Изменение статуса встречи руководителя”

**Управление очередью приёма руководителя:** Секретарь должен иметь возможность формировать электронную очередь для приёма у руководителя, добавлять в нее подтвержденные встречи, изменять порядок элементов, фиксировать перерывы.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

Рисунок 17. Диаграмма последовательности “Управление очередью приёма руководителя”

**Просмотр очереди руководителя:** Секретарь должен видеть текущее состояние очереди своего руководителя.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

Рисунок 18. Диаграмма последовательности “Просмотр очереди руководителя”

**Изменение статуса элемента в очереди:** Секретарь должен иметь возможность изменять статус элемента в очереди (например, "В процессе", "Пропущено", "Завершен").

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

Рисунок 19. Диаграмма последовательности “Изменение статуса элемента в очереди руководителя”

**Прикрепление документа к встрече руководителя:** Секретарь должен иметь возможность прикреплять документы к встречам своего руководителя.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описаниеРисунок 20. Диаграмма последовательности “Прикрепление документа к встрече руководителя”

**Отправка уведомления вручную:** Секретарь должен иметь возможность отправлять не пред настроенные уведомления участникам встреч.

Изображение выглядит как текст, снимок экрана, диаграмма, число

Автоматически созданное описаниеРисунок 21. Диаграмма последовательности “Отправка уведомления вручную участникам встреч”

1. **Функции для роли "Принимающее лицо" (Руководитель):**  
   Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

   Автоматически созданное описание

Рисунок 22. Диаграмма Use case для «Принимающие лицо»

**Просмотр своего графика встреч:** Руководитель должен видеть свой актуальный график запланированных встреч.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описаниеРисунок 23. Диаграмма последовательности “Просмотр своего графика встреч”

**Подтверждение/отклонение своей встречи:** Руководитель должен иметь возможность самостоятельно подтверждать или отклонять заявки на встречи, адресованные ему.

Изображение выглядит как текст, снимок экрана, диаграмма, число

Автоматически созданное описаниеРисунок 24. Диаграмма последовательности “Подтверждение/отклонение своей встречи”

**Редактирование своей встречи:** Руководитель должен иметь возможность редактировать детали своих встреч (если это допустимо бизнес-процессом и статусом встречи).

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описаниеРисунок 25. Диаграмма последовательности “Редактирование своей встречи”

**Отмена своей встречи:** Руководитель должен иметь возможность отменять свои запланированные встречи.

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описаниеРисунок 26. Диаграмма последовательности “Отмена своей встречи”

**Управление своей очередью во время приёма:** Руководитель должен иметь возможность управлять активной очередью во время приёма (например, вызывать следующего, отмечать завершение приёма текущего посетителя, объявлять перерыв).

Изображение выглядит как текст, снимок экрана, Параллельный, число

Автоматически созданное описаниеРисунок 27. Диаграмма последовательности “Управление своей очередью во время приёма”

**Скачивание документа со своей встречи:** Руководитель должен иметь возможность скачивать вложения к своим встречам.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описаниеРисунок 28. Диаграмма последовательности “Скачивание документа со своей встречи”

1. **Функции для роли "Администратор системы":**

В конце рассмотрим администратора

Изображение выглядит как текст, диаграмма, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 28. Диаграмма Use case для «Администратора»

**Управление пользователями:** Администратор должен иметь возможность создавать, просматривать, редактировать (включая назначение ролей, активацию/деактивацию) и удалять учетные записи пользователей системы.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описаниеРисунок 30. Диаграмма последовательности “Управление пользователями”

**Управление ролями:** Администратор должен иметь возможность просматривать и, возможно, редактировать описания ролей (создание новых ролей может быть ограничено).

Изображение выглядит как текст, снимок экрана, диаграмма, число

Автоматически созданное описаниеРисунок 31. Диаграмма последовательности “Управление ролями”

**Управление системными справочниками:** Администратор должен иметь возможность управлять значениями системных справочников (статусы встреч, статусы очереди, типы уведомлений).

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описаниеРисунок 32. Диаграмма последовательности “Управление системными справочниками”

**Управление лицензиями:** Если система лицензируется, администратор должен иметь доступ к информации о лицензии и ее параметрах.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описаниеРисунок 34. Диаграмма последовательности “Управление лицензиями”

**Управление настройками интеграции:** Администратор должен иметь возможность настраивать параметры интеграции с Active Directory и календарными системами.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описаниеРисунок 33. Диаграмма последовательности “Управление настройками интеграции”

**Просмотр системных журналов:** Администратор должен иметь доступ к журналам событий системы для мониторинга и диагностики.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

Рисунок 35. Диаграмма последовательности “Просмотр системных журналов”

**6. Требования к интеграции**

Для обеспечения эффективного функционирования в ИТ-инфраструктуре организации, система "DRSapi" должна соответствовать следующим интеграционным требованиям.

**6.1. Интеграция с Active Directory [10]**

Необходима глубокая интеграция с корпоративной службой каталогов Active Directory, включающая:

* **Аутентификацию пользователей:** Обеспечение аутентификации сотрудников организации посредством их корпоративных учетных записей Active Directory. Приоритетной является реализация механизма единого входа (Single Sign-On, SSO).
* **Синхронизацию данных пользователей:** Реализация возможности импорта и/или периодической синхронизации атрибутов пользователей (ФИО, адрес электронной почты, контактный телефон, должность) из Active Directory в базу данных системы "DRSapi".
* **Сопоставление ролей:** Обеспечение механизма сопоставления групп безопасности Active Directory с ролями, определенными в системе "DRSapi", для автоматизированного управления правами доступа пользователей к функциям системы.

**6.2. Интеграция с календарными системами**

Требуется обеспечить взаимодействие с распространенными календарными системами, такими как Microsoft Exchange/Outlook Calendar и Google Calendar:

* **Проверка доступности:** Автоматизированная проверка доступности (занятости) принимающего сотрудника в его корпоративном календаре при создании или изменении заявки на встречу в системе "DRSapi".
* **Управление событиями в календарях:** Автоматическое создание, обновление и удаление событий в корпоративном календаре принимающего сотрудника при подтверждении, изменении или отмене встречи в "DRSapi". Опционально, аналогичные операции должны выполняться для календаря посетителя, являющегося сотрудником организации, при наличии его согласия.
* **Синхронизация изменений (опционально для прототипа):** Рассмотрение возможности реализации двусторонней синхронизации изменений, инициированных непосредственно в корпоративном календаре (например, перенос или отмена события), с системой "DRSapi". Данная функциональность является опциональной для прототипа ввиду ее повышенной сложности.

**7. Требования к системе уведомлений**

Система "DRSapi" должна обеспечивать своевременное информирование пользователей о ключевых событиях посредством автоматизированных уведомлений. Уведомления должны генерироваться при наступлении следующих событий:

* Создание новой заявки на встречу.
* Подтверждение встречи принимающей стороной.
* Отклонение заявки на встречу (с возможностью указания причины).
* Отмена ранее подтвержденной встречи (инициированная любой из сторон).
* Изменение параметров встречи (дата, время, место проведения).
* Изменение статуса обработки заявки или встречи.
* Напоминание о предстоящей встрече (например, за 24 часа и за 1 час до назначенного времени).
* Для посетителей: уведомление об изменении позиции в электронной очереди или о вызове на приём.

Доставка уведомлений должна осуществляться как минимум по электронной почте. Кроме того, требуется реализация уведомлений в режиме реального времени для веб-интерфейса с использованием технологии SignalR для оперативного отображения информации пользователям, работающим с системой.

**2.1.2. Нефункциональные требования**

Нефункциональные требования определяют качественные атрибуты системы "DRSapi", обеспечивающие ее эффективность, надежность и удобство эксплуатации.

1. **Производительность:**
   * Время отклика системы на ключевые пользовательские операции (например, создание заявки, просмотр расписания, поиск информации) не должно превышать 2-3 секунд при ожидаемой нагрузке.
   * Система должна обеспечивать стабильную работу при одновременной активности до 1000 пользователей без существенной деградации производительности.
2. **Надежность и доступность:**
   * Коэффициент доступности системы должен составлять не менее 99.5% в течение установленных регламентом рабочих часов.
   * Необходимо предусмотреть механизмы всестороннего логирования ошибок, сбоев и системных событий для обеспечения оперативной диагностики и восстановления работоспособности.
3. **Безопасность:**
   * Хранение и передача всех чувствительных данных, включая персональные данные пользователей, должны осуществляться в зашифрованном виде. Обязательным является использование протокола HTTPS для всех внешних коммуникаций.
   * Система должна соответствовать требованиям Федерального закона "О персональных данных" № 152-ФЗ [7].
   * Необходимо реализовать меры по противодействию распространенным веб-уязвимостям согласно рекомендациям OWASP Top Ten [9].
   * Доступ к функциям и данным системы должен строго регламентироваться на основе ролевой модели доступа (RBAC).
4. **Масштабируемость:**
   * Архитектура системы должна обеспечивать возможность как горизонтального (добавление новых экземпляров серверов), так и вертикального (увеличение вычислительных ресурсов существующих серверов) масштабирования для адаптации к росту числа пользователей, объемов данных и интенсивности нагрузки.
5. **Удобство использования (Usability):**
   * API системы должен быть спроектирован в соответствии с архитектурным стилем REST [11], обладать интуитивно понятной структурой и быть сопровожден исчерпывающей документацией (например, с использованием OpenAPI/Swagger) для разработчиков клиентских приложений, включая чат-бота.
   * При разработке пользовательского веб-интерфейса (даже если он не является частью данной ВКР) необходимо руководствоваться общепринятыми принципами эргономики, дизайна и юзабилити [5, 6] для обеспечения высокого уровня пользовательского опыта.
6. **Поддерживаемость и расширяемость:**
   * Исходный код системы должен быть структурированным, хорошо документированным (комментированным) и соответствовать общепринятым стандартам кодирования и принципам чистого кода (Clean Code) для облегчения его дальнейшей поддержки и модификации.
   * Архитектура должна быть модульной, позволяющей добавлять новый функционал и изменять существующие компоненты с минимальным взаимным влиянием и риском регрессии.
7. **Интегрируемость:**

API системы должен быть спроектирован с учетом обеспечения возможности простой интеграции с другими корпоративными информационными системами [12], помимо AD и календарных сервисов, путем предоставления стандартизированных интерфейсов взаимодействия.

Данные требования формируют базис для проектирования архитектуры, структуры базы данных и разработки программных компонентов системы "DRSapi".

**2.2.1 Описание сущностей предметной области и их атрибутов**

Предметная область системы «Цифровая приёмная» охватывает процессы организации и управления встречами между посетителями и руководством, включая планирование, управление очередью, уведомления и интеграцию с внешними сервисами. Для эффективного функционирования системы данные организованы в виде набора сущностей (таблиц), каждая из которых хранит определённый тип информации и обладает набором атрибутов.

Ниже представлено детальное описание каждой сущности с перечнем её атрибутов, их типами данных, ограничениями и назначением.

**1. Сущность users (Пользователи)**

Сущность users является центральной и хранит информацию обо всех участниках системы: посетителях, секретарях, принимающих лицах (руководителях) и администраторах.

Таблица 1. Структура сущности «Пользователи»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| id | UUID | PRIMARY KEY | Уникальный идентификатор пользователя. |
| full\_name | VARCHAR | NOT NULL | Полное имя пользователя. |
| email | VARCHAR | UNIQUE, NOT NULL | Уникальный адрес электронной почты пользователя. |
| phone | VARCHAR |  | Номер телефона пользователя. |
| role\_id | INT | FOREIGN KEY → roles.id | Идентификатор роли пользователя, ссылающийся на таблицу roles. |
| is\_active | BOOLEAN | DEFAULT TRUE | Флаг активности пользователя. |

**2. Сущность roles (Роли)**

Сущность roles представляет собой справочник, определяющий типы пользователей и их функции в системе.

Таблица 2. Структура сущности «Роли»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| id | INT | PRIMARY KEY | Уникальный идентификатор роли. |
| name | VARCHAR | UNIQUE, NOT NULL | Название роли (например, «Посетитель», «Секретарь», «Принимающее лицо», «Администратор»). |

**3. Сущность appointments (Встречи)**

Сущность appointments содержит информацию о запланированных встречах между посетителями и принимающими лицами.

Таблица 3. Структура сущности «Встречи»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| id | UUID | PRIMARY KEY | Уникальный идентификатор встречи. |
| visitor\_id | UUID | FOREIGN KEY → users.id | Идентификатор посетителя, участвующего во встрече. |
| receiver\_id | UUID | FOREIGN KEY → users.id | Идентификатор принимающего лица (руководителя), проводящего встречу. |
| start\_time | TIMESTAMP | NOT NULL | Время начала встречи. |
| end\_time | TIMESTAMP | NOT NULL | Время окончания встречи. |
| status\_id | INT | FOREIGN KEY → appointment\_statuses.id | Идентификатор текущего статуса встречи. |
| topic | VARCHAR |  | Тема встречи. |
| created\_by | UUID | FOREIGN KEY → users.id | Идентификатор пользователя, создавшего запись о встрече. |

**4. Сущность appointment**\_statuses **(Статусы встреч)**

Сущность appointment\_statuses является справочником, определяющим возможные состояния встречи.

Таблица 4. Структура сущности «Статусы встреч»

|  |  |  |  |
| --- | --- | --- | --- |
| **Поле** | **Тип данных** | **Ограничения** | **Описание** |
| id | INT | PRIMARY KEY | Уникальный идентификатор статуса встречи. |
| name | VARCHAR | UNIQUE, NOT NULL | Название статуса (например, «Запрошена», «Подтверждена», «Отменена», «Завершена»). |

**5. Сущность notifications (Уведомления)**

Сущность notifications хранит информацию о сообщениях, отправляемых системой пользователям по поводу встреч.

Таблица 5. Структура сущности «Уведомления»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| id | UUID | PRIMARY KEY | Уникальный идентификатор уведомления. |
| user\_id | UUID | FOREIGN KEY → users.id | Идентификатор пользователя, которому адресовано уведомление. |
| appointment\_id | UUID | FOREIGN KEY → appointments.id | Идентификатор встречи, к которой относится уведомление. |
| type\_id | INT | FOREIGN KEY → notification\_types.id | Идентификатор типа уведомления. |
| sent\_at | TIMESTAMP |  | Время отправки уведомления. |
| is\_read | BOOLEAN | DEFAULT FALSE | Флаг, указывающий, прочитано ли уведомление. |

**6. Сущность notification**\_types **(Типы уведомлений)**

Сущность notification\_types является справочником, определяющим различные виды уведомлений.

Таблица 6. Структура сущности «Типы уведомлений»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| id | INT | PRIMARY KEY | Уникальный идентификатор типа уведомления. |
| name | VARCHAR | UNIQUE, NOT NULL | Название типа уведомления (например, «Напоминание», «Подтверждение записи»). |

**7. Сущность queue\_items (Элементы очереди)**

Сущность queue\_items определяет порядок приёма посетителей у принимающего лица.

Таблица 7. Структура сущности «Элементы очереди»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| id | UUID | PRIMARY KEY | Уникальный идентификатор элемента очереди. |
| appointment\_id | UUID | FOREIGN KEY → appointments.id | Идентификатор встречи, которая находится в очереди. |
| position | INT |  | Позиция встречи в очереди. |
| status\_id | INT | FOREIGN KEY → queue\_statuses.id | Идентификатор текущего статуса элемента очереди. |

**8. Сущность queue\_statuses (Статусы очереди)**

Сущность queue\_statuses является справочником, определяющим возможные состояния элементов очереди.

Таблица 8. Структура сущности «Статусы очереди»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| id | INT | PRIMARY KEY | Уникальный идентификатор статуса очереди. |
| name | VARCHAR | UNIQUE, NOT NULL | Название статуса очереди (например, «Ожидание», «В процессе», «Перерыв», «Завершено»). |

**9. Сущность attachments (Вложения)**

Сущность attachments позволяет прикреплять документы к конкретным встречам.

Таблица 9. Структура сущности «Вложения»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| id | UUID | PRIMARY KEY | Уникальный идентификатор вложения. |
| appointment\_id | UUID | FOREIGN KEY → appointments.id | Идентификатор встречи, к которой относится вложение. |
| file\_name | VARCHAR | NOT NULL | Имя файла вложения. |
| file\_path | TEXT | NOT NULL | Путь к файлу вложения. |
| uploaded\_at | TIMESTAMP | NOT NULL | Время загрузки вложения. |
| uploaded\_by | UUID | FOREIGN KEY → users.id | Идентификатор пользователя, загрузившего вложение. |

**10. Сущность licenses (Лицензии)**

Сущность licenses управляет правами на использование системы, включая ограничения по количеству пользователей и срокам действия.

Таблица 10. Структура сущности «Лицензии»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| id | UUID | PRIMARY KEY | Уникальный идентификатор лицензии. |
| server\_id | VARCHAR | UNIQUE | Уникальный идентификатор сервера, к которому привязана лицензия. |
| max\_users | INT |  | Максимальное количество пользователей, разрешенное лицензией. |
| max\_receivers | INT |  | Максимальное количество принимающих лиц, разрешенное лицензией. |
| max\_panels | INT |  | Максимальное количество панелей (если применимо), разрешенное лицензией. |
| expires\_at | TIMESTAMP |  | Срок действия лицензии. |
| is\_active | BOOLEEN | DEFAULT TRUE | Флаг активности лицензии. |

**11. Сущность integration\_settings (Настройки интеграции)**

Сущность integration\_settings определяет, какие внешние сервисы интегрированы с системой.

Таблица 11. Структура сущности «Настройки интеграции»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| id | UUID | PRIMARY KEY | Уникальный идентификатор настроек интеграции. |
| license\_id | UUID | FOREIGN KEY → licenses.id | Идентификатор лицензии, к которой относятся настройки интеграции. |
| use\_ad | BOOLEAN |  | Флаг использования Active Directory. |
| use\_exchange | BOOLEAN |  | Флаг использования MS Exchange. |
| use\_google\_calendar | BOOLEAN |  | Флаг использования Google Calendar. |
| use\_apple\_calendar | BOOLEAN |  | Флаг использования Apple Calendar. |
| use\_chat\_bots | BOOLEAN |  | Флаг использования чат-ботов. |

**12. Сущность secretary\_receiver\_assignments (Назначения секретарь-руководитель)**

Эта сущность является связующей таблицей, явно определяющей, какой Секретарь уполномочен управлять расписанием какого Принимающего лица. Это необходимо, поскольку Секретарь управляет приёмами, связанными с закреплёнными принимающими.

Таблица12. Структура сущности «Назначения секретарь-руководитель»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип данных | Ограничения | Описание |
| id | UUID | PRIMARY KEY | Уникальный идентификатор записи о назначении. |
| secretary\_id | UUID | FOREIGN KEY → users.id, NOT NULL | Идентификатор Секретаря, ссылающийся на таблицу users. |
| receiver\_id | UUID | FOREIGN KEY → users.id, NOT NULL | Идентификатор Принимающего лица, ссылающийся на таблицу users. |
| assigned\_at | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP | Время, когда было сделано назначение. |
| is\_active | BOOLEAN | DEFAULT TRUE | Флаг активности назначения (позволяет деактивировать без удаления). |
| UNIQUE (secretary\_id, receiver\_id) |  |  | Гарантирует уникальность пары секретарь-руководитель. |

**2.2.2 Разработка логической ER-модели базы данных**

Логическая ER-модель (Entity-Relationship Model) системы «Цифровая приёмная» описывает взаимосвязи между сущностями предметной области, абстрагируясь от конкретной реализации базы данных. Она служит основой для проектирования реляционной схемы и обеспечивает целостность и согласованность данных.

**Основные взаимосвязи между сущностями**

Взаимосвязи между сущностями в системе «Цифровая приёмная» можно классифицировать по их кардинальности (один-к-одному, один-ко-многим, многие-ко-многим) и реализованы с использованием внешних ключей.

1. **Пользователи и Роли**

**Тип связи:** Один-ко-многим

**Описание:** Одна роль может быть назначена многим пользователям, но каждый пользователь имеет только одну роль.

**Реализация:** В таблице users присутствует внешний ключ role\_id, ссылающийся на первичный ключ id таблицы roles.

1. **Встречи и Пользователи**

**Тип связи:** Многие-ко-одному для каждой роли.

**Описание:** Каждая встреча связана с одним посетителем (visitor\_id), одним принимающим лицом (receiver\_id) и одним пользователем, создавшим встречу (created\_by). Один пользователь может быть участником или создателем множества встреч.

**Реализация:** Таблица appointments содержит три внешних ключа (visitor\_id, receiver\_id, created\_by), каждый из которых ссылается на первичный ключ id таблицы users.

1. **Встречи и Статусы встреч**

**Тип связи:** Многие-ко-одному.

**Описание:** Каждая встреча имеет один текущий статус, но один статус может быть присвоен множеству встреч.

**Реализация:** В таблице appointments присутствует внешний ключ status\_id, ссылающийся на первичный ключ id таблицы appointment\_statuses.

1. **Уведомления и Пользователи**

**Тип связи:** Многие-ко-одному.

**Описание:** Каждое уведомление адресовано одному пользователю, но один пользователь может получать множество уведомлений.

**Реализация:** В таблице notifications присутствует внешний ключ user\_id, ссылающийся на первичный ключ id таблицы users.

1. **Уведомления и Встречи**

**Тип связи:** Многие-ко-одному.

**Описание:** Каждое уведомление относится к одной конкретной встрече, но одна встреча может генерировать множество уведомлений.

**Реализация:** В таблице notifications присутствует внешний ключ appointment\_id, ссылающийся на первичный ключ id таблицы appointments.

1. **Уведомления и Типы уведомлений**

**Тип связи:** Многие-ко-одному.

**Описание:** Каждое уведомление имеет один тип, но один тип может быть использован для множества уведомлений.

**Реализация:** В таблице notifications присутствует внешний ключ type\_id, ссылающийся на первичный ключ id таблицы notification\_types.

1. **Элементы очереди и Встречи**

**Тип связи:** Многие-ко-одному.

**Описание:** Каждый элемент очереди соответствует одной встрече, но одна встреча может быть частью очереди.

**Реализация:** В таблице queue\_items присутствует внешний ключ appointment\_id, ссылающийся на первичный ключ id таблицы appointments.

1. **Элементы очереди и Статусы очереди**

**Тип связи:** Многие-ко-одному.

**Описание:** Каждый элемент очереди имеет один статус, но один статус может быть присвоен множеству элементов очереди.

**Реализация:** В таблице queue\_items присутствует внешний ключ status\_id, ссылающийся на первичный ключ id таблицы queue\_statuses.

1. **Вложения и Встречи**

**Тип связи:** Многие-ко-одному.

**Описание:** Каждое вложение относится к одной встрече, но одна встреча может иметь множество вложений.

**Реализация:** В таблице attachments присутствует внешний ключ appointment\_id, ссылающийся на первичный ключ id таблицы appointments.

1. **Вложения и Пользователи**

**Тип связи:** Многие-ко-одному.

**Описание:** Каждое вложение загружено одним пользователем, но один пользователь может загрузить множество вложений.

**Реализация:** В таблице attachments присутствует внешний ключ uploaded\_by, ссылающийся на первичный ключ id таблицы users.

1. **Настройки интеграции и Лицензии**

**Тип связи:** Многие-ко-одному.

**Описание:** Каждая запись настроек интеграции связана с одной лицензией, но одна лицензия может иметь множество настроек интеграции (хотя в данной модели предполагается одна запись настроек на лицензию, связь остается Many-to-One для гибкости).

**Реализация:** В таблице integration\_settings присутствует внешний ключ license\_id, ссылающийся на первичный ключ id таблицы licenses.

1. **Секретари и Принимающие лица**

**Тип связи:** Многие-ко-многим.

**Описание:** Один Секретарь может быть назначен для управления расписаниями нескольких Принимающих лиц, и одно Принимающее лицо может иметь нескольких Секретарей, управляющих его расписанием. Эта связь формализует делегирование административных полномочий, где Секретарь управляет приёмами, связанными с *закреплёнными* принимающими.

**Реализация:** Для реализации этой связи используется промежуточная таблица secretary\_receiver\_assignments. Эта таблица содержит два внешних ключа: secretary\_id (ссылающийся на users.id для пользователя с ролью «Секретарь») и receiver\_id (ссылающийся на users.id для пользователя с ролью «Принимающее лицо). Композитный уникальный ключ по (secretary\_id, receiver\_id) обеспечивает уникальность каждого назначения.

**Логическая ER-диаграмма**

Логическая ER-диаграмма визуализирует описанные сущности и их взаимосвязи. Она демонстрирует структуру базы данных, где прямоугольники представляют сущности, а линии — связи между ними, с обозначением кардинальности.

Изображение выглядит как текст, диаграмма, План, число

Автоматически созданное описание

Рисунок 36. Диаграмма связей

Эта логическая модель обеспечивает прочную основу для дальнейшего проектирования физической базы данных, гарантируя, что все ключевые бизнес-правила и взаимосвязи между данными будут корректно реализованы.

**2.3 Проектирование архитектуры API и выбор технологического стека**

В данном разделе представлен анализ и обоснование архитектурных решений, а также выбор технологического стека для реализации системы «Цифровая приёмная». Целью является создание надежной, масштабируемой, безопасной и легко поддерживаемой серверной инфраструктуры, способной эффективно обрабатывать запросы различных пользовательских ролей и обеспечивать взаимодействие в реальном времени. Принятые решения основаны на принципах модульности, производительности и безопасности, что является критически важным для корпоративной системы управления приёмами.

**2.3.1 Описание архитектуры REST API и взаимодействия компонентов**

Для построения программного интерфейса (API) системы «Цифровая приёмная» выбран архитектурный стиль Representational State Transfer (REST). REST является широко признанным стандартом для разработки сетевых приложений, который способствует созданию масштабируемых, простых и модульных систем за счет использования бессостоятельной клиент-серверной модели взаимодействия.

* **Принципы REST API:**

**Клиент-серверная архитектура (Client-Server Architecture):** RESTful API строго разделяет клиентские и серверные компоненты. Клиент отвечает за пользовательский интерфейс и взаимодействие с пользователем, в то время как сервер управляет хранением данных, бизнес-логикой и предоставлением API. Такое разделение позволяет разрабатывать, развертывать и масштабировать каждый компонент независимо. В контексте системы «Цифровая приёмная» это означает, что веб-интерфейс, мобильные приложения или киоски будут функционировать как клиенты, отправляя HTTP-запросы к бэкенд-API. Сервер, в свою очередь, будет обрабатывать эти запросы, взаимодействовать с базой данных и формировать ответы. Подобное четкое разграничение гарантирует, что изменения в пользовательском интерфейсе не потребуют модификаций в бэкенд-API, и наоборот, что способствует гибкой разработке и упрощенному сопровождению.

**Отсутствие состояния (Statelessness):** Каждый запрос, отправляемый клиентом серверу, должен содержать всю необходимую информацию для его обработки, без зависимости от предыдущих запросов или серверной сессии. Это означает, что сервер не сохраняет никакого контекста клиента между запросами. Для системы «Цифровая приёмная» это подразумевает, что каждый аутентифицированный запрос, будь то создание встречи или просмотр очереди, будет включать в себя токен аутентификации (например, JWT), позволяющий серверу обрабатывать его независимо. Подобный подход упрощает серверную логику, повышает надежность за счет самодостаточности каждого запроса и значительно улучшает масштабируемость, так как любой доступный экземпляр сервера может обработать любой запрос без необходимости поддержания сессионной привязки.

Применение бессостоятельного подхода в архитектуре API, особенно в сочетании с использованием самодостаточных токенов, таких как JWT, оказывает существенное влияние на масштабируемость и отказоустойчивость системы. Отсутствие необходимости хранить состояние клиента на сервере устраняет потребность в так называемых «липких сессиях» (sticky sessions) или сложной репликации сессий между серверными узлами. Это позволяет беспрепятственно добавлять или удалять серверные инстансы (горизонтальное масштабирование) без риска потери контекста пользователя, поскольку любой сервер способен обработать любой запрос от любого клиента. Более того, в случае выхода из строя одного из серверов, другие серверы могут немедленно принять его нагрузку, так как они не зависят от состояния, которое могло быть сохранено на отказавшем сервере. Это повышает общую отказоустойчивость системы. Для «Цифровой приёмной», которая должна обслуживать множество пользователей (посетителей, секретарей, руководителей) и потенциально испытывать высокую нагрузку, бессостоятельность API и применение JWT обеспечивают критически важную гибкость и надежность. Это позволяет системе легко масштабироваться для обработки пиковых нагрузок и минимизировать время простоя в случае сбоев, что является фундаментальным требованием для корпоративных приложений.

**Кэшируемость (Cacheability):** REST позволяет определять ответы как кэшируемые или некэшируемые, что дает возможность клиентам или промежуточным узлам (например, прокси-серверам) сохранять часто запрашиваемые данные. Это сокращает нагрузку на сервер и улучшает время отклика для повторяющихся запросов. В системе «Цифровая приёмная» статические данные, такие как списки статусов встреч или типы уведомлений, могут быть кэшированы клиентами. Это уменьшает количество избыточных запросов к серверу и базе данных, повышая воспринимаемую производительность для пользователей. Динамические данные, такие как текущий статус очереди или детали конкретных встреч, будут помечены как некэшируемые или иметь очень короткое время жизни кэша.

**Многослойная система (Layered System):** согласно этому принципу, клиент не должен знать, подключен ли он напрямую к конечному серверу или к промежуточному звену. Это позволяет вводить различные слои, такие как балансировщики нагрузки, прокси-серверы или шлюзы безопасности, между клиентом и сервером без влияния на клиент-серверное взаимодействие. Данный принцип поддерживает развертывание «Цифровой приёмной» в облачной среде с использованием таких слоев, как API-шлюзы, балансировщики нагрузки и межсетевые экраны. Он гарантирует, что система может развиваться (например, путем добавления новых микросервисов для специфических функций) без необходимости внесения изменений в клиентские приложения. Например, API-шлюз может выполнять валидацию JWT до того, как запросы достигнут основной логики приложения.

**Единообразный интерфейс (Uniform Interface):** Этот фундаментальный принцип диктует, что сервер передает информацию в стандартном формате (представлении) и что запросы идентифицируют ресурсы с помощью унифицированных идентификаторов ресурсов (URI). Он также подразумевает, что клиенты получают достаточно информации для взаимодействия с ресурсами (например, метаданные, самоописывающие сообщения, гиперссылки для динамического обнаружения — HATEOAS). В «Цифровой приёмной» все ресурсы (например, /users, /appointments, /notifications) будут доступны через предсказуемые URI с использованием стандартных HTTP-методов (GET, POST, PUT, DELETE). Ответы будут предоставляться в согласованном формате (например, JSON), включая коды состояния и соответствующие метаданные. Такая согласованность упрощает разработку клиентов и интеграцию, делая API интуитивно понятным и легким для использования различными клиентскими приложениями.

**Код по требованию (Code-On-Demand):** Этот принцип подразумевает, что сервер может временно расширять или настраивать функциональность клиента путем передачи программного кода клиенту. Однако данный принцип является необязательным и редко используется из-за потенциальных рисков безопасности и сложности реализации. В системе «Цифровая приёмная» этот принцип не будет активно применяться, поскольку основной акцент делается на передаче данных, а не исполняемого кода, что позволяет минимизировать риски безопасности и упростить архитектуру.

**Взаимодействие компонентов:** Архитектура системы «Цифровая приёмная» будет реализована с использованием трехуровневой модели: уровень представления (клиентские приложения), уровень бизнес-логики (API-бэкенд) и уровень данных (база данных).

**Потоки данных и коммуникации:**

**Клиентские Приложения (Веб, Мобильные, Киоски):** Пользователи взаимодействуют с системой через различные клиентские приложения. Эти клиенты отправляют HTTP-запросы (GET, POST, PUT, DELETE) к API-бэкенду для выполнения операций, таких как создание встречи или просмотр расписания.

**API-Бэкенд (ASP.NET Core):** выступает в качестве центрального узла, обрабатывающего все входящие запросы. Он содержит бизнес-логику, выполняет валидацию данных, аутентификацию и авторизацию (с использованием JWT). Бэкенд взаимодействует с базой данных через ORM (EF Core) и управляет коммуникациями в реальном времени посредством SignalR.

**База Данных (PostgreSQL):** хранит все данные системы, включая информацию о пользователях, встречах, ролях, уведомлениях, очередях и настройках интеграции. EF Core абстрагирует прямое взаимодействие с SQL, позволяя бэкенду работать с.NET объектами.

**Сервис Реального Времени (SignalR):** обеспечивает двустороннюю, постоянную связь между сервером и клиентами. Когда происходит событие, требующее немедленного оповещения (например, подтверждение встречи, изменение статуса очереди, напоминание), бэкенд использует SignalR для отправки push-уведомлений соответствующим клиентам.

**Внешние Сервисы (Active Directory, Календари, Чат-боты):** API-бэкенд может интегрироваться с внешними корпоративными сервисами для синхронизации данных пользователей (Active Directory) или событий календаря (MS Exchange, Google Calendar, Apple Calendar), а также для отправки уведомлений через чат-боты. Настройки этих интеграций хранятся в базе данных (integration\_settings).

Включение функциональности реального времени в архитектуру API имеет значительное влияние на дизайн системы и пользовательский опыт. Система «Цифровая приёмная» требует оперативного информирования всех сторон, включая автоматическое получение напоминаний и уведомлений, а также управление электронной очередью, которая автоматически пересчитывается при изменении порядка. Традиционный RESTful API, основанный на модели запрос-ответ, неэффективен для обеспечения мгновенных обновлений, поскольку клиентам пришлось бы постоянно опрашивать сервер на предмет изменений, что привело бы к избыточной нагрузке и задержкам в получении информации.

Для решения этой задачи в архитектуру включен SignalR. Это означает, что некоторые взаимодействия не будут чисто RESTful. Вместо этого, API будет использовать комбинацию REST для операций создания, чтения, обновления и удаления (CRUD) и SignalR для push-уведомлений и мгновенных обновлений состояния, таких как статус очереди или встречи. Это требует наличия отдельного слоя для SignalR Hubs, которые будут взаимодействовать с бизнес-логикой API. В результате, мгновенные уведомления и обновления очереди значительно улучшают пользовательский опыт. Посетители получают напоминания и подтверждения в реальном времени, секретари видят актуальное состояние очереди, а руководители могут оперативно управлять приёмом, объявляя перерывы, которые немедленно становятся видны ожидающим. Такая оперативность информации является критически важной для системы управления приёмами, где своевременность данных напрямую влияет на эффективность и удовлетворенность пользователей.

**2.3.2 Обоснование выбора средств реализации (ASP.NET Core, EF Core, PostgreSQL, SignalR, JWT)**

Выбор технологического стека для системы «Цифровая приёмная» был продиктован требованиями к производительности, масштабируемости, безопасности, кроссплатформенности и удобству разработки. Ниже представлено обоснование для каждой выбранной технологии.

**ASP.NET Core:** ASP.NET Core представляет собой кроссплатформенный, высокопроизводительный, открытый фреймворк, предназначенный для создания современных веб-приложений, сервисов и API. Он является результатом переработки ASP.NET 4.x, с архитектурными изменениями, направленными на повышение модульности и производительности.

Ключевые преимущества ASP.NET Core включают возможность разработки и развертывания на различных операционных системах, таких как Windows, macOS и Linux. Фреймворк демонстрирует высокую производительность, способную обрабатывать значительно больше запросов в секунду по сравнению с некоторыми другими платформами. Он предлагает унифицированный подход для создания как веб-интерфейсов, так и веб-API, что упрощает разработку. ASP.NET Core спроектирован с учетом удобства тестирования и включает встроенную инъекцию зависимостей (DI), что способствует созданию модульных и легко тестируемых приложений. Будучи проектом с открытым исходным кодом и активным сообществом, он гарантирует постоянное развитие и поддержку. Кроме того, фреймворк обладает облачно-ориентированной системой конфигурации, что делает его готовым к развертыванию в облачных средах.

Выбор ASP.NET Core для системы «Цифровая приёмная» обоснован несколькими факторами. Система может столкнуться с высокой нагрузкой, особенно в крупных организациях, и высокая производительность ASP.NET Core критически важна для обеспечения быстрого отклика и обработки большого количества одновременных запросов на запись, управление очередью и уведомления. Поскольку система включает как API для различных клиентов (веб, мобильные), так и потенциально административные веб-интерфейсы, унифицированный подход ASP.NET Core к созданию UI и API упрощает разработку и поддержку, позволяя использовать единую кодовую базу и инструменты. Использование ASP.NET Core также позволяет бесшовно интегрироваться с другими технологиями экосистемы.NET, такими как EF Core и SignalR, создавая согласованный и эффективный технологический стек.

Сочетание унифицированной разработки и кроссплатформенности в ASP.NET Core обеспечивает значительную синергию, ускоряющую вывод продукта на рынок. Унифицированный подход, при котором один фреймворк используется как для бэкенда API, так и для потенциального веб-интерфейса (например, для администратора или секретаря), позволяет разработчикам применять одни и те же языки (C#), библиотеки и инструменты. Это сокращает время на обучение, упрощает совместную работу в команде и снижает общую сложность проекта. Кроссплатформенность, то есть возможность развертывания на различных операционных системах (Windows, Linux), предоставляет гибкость при выборе инфраструктуры, включая облачные сервисы и контейнеры Docker, и уменьшает зависимость от конкретного поставщика. В условиях, когда требуется быстрое развертывание и адаптация к различным средам, а также поддержка разных типов клиентов (веб, мобильные), ASP.NET Core позволяет команде работать более эффективно. Это приводит к ускорению процессов разработки, тестирования и развертывания, что является критически важным для быстрого запуска системы и ее дальнейшего развития.

**EF Core (Entity Framework Core):** EF Core является легковесной, расширяемой, открытой и кроссплатформенной версией популярной технологии доступа к данным Entity Framework. Он функционирует как объектно-реляционный мэппер (O/RM), позволяя.NET разработчикам взаимодействовать с базой данных, используя.NET объекты, и значительно сокращая объем кода для доступа к данным, который обычно требуется писать вручную.

Среди ключевых преимуществ EF Core — упрощенный доступ к данным, который позволяет выполнять запросы к базе данных через LINQ и.NET объекты, существенно ускоряя процесс разработки. Встроенная поддержка параметризованных запросов обеспечивает защиту от распространенных уязвимостей, таких как SQL-инъекции. Портативность фреймворка гарантируется поддержкой множества различных систем управления базами данных (СУБД), включая PostgreSQL и SQL Server, что предоставляет гибкость при выборе или изменении базы данных. EF Core также включает инструменты для повышения производительности, такие как встроенные механизмы кэширования, ленивая загрузка (lazy loading) и отслеживание изменений (change tracking), которые помогают оптимизировать доступ к данным. Кроме того, функциональность миграций позволяет развивать схему базы данных по мере изменения модели приложения.

Выбор EF Core для системы «Цифровая приёмная» обусловлен несколькими факторами. Система имеет множество взаимосвязанных сущностей, таких как пользователи, встречи, уведомления, очереди, вложения, лицензии и настройки интеграции. EF Core значительно упрощает работу с этими сущностями, позволяя разработчикам концентрироваться на бизнес-логике, а не на написании низкоуровневого SQL-кода, что ускоряет разработку. Обработка записей на приём, персональных данных пользователей и настроек системы требует высокого уровня безопасности. EF Core обеспечивает надежную работу с данными, включая поддержку транзакций (ACID) и автоматическое предотвращение SQL-инъекций, что критически важно для системы, обрабатывающей конфиденциальные данные. Возможность использования миграций EF Core позволяет легко адаптировать схему базы данных к меняющимся требованиям системы «Цифровая приёмная» без необходимости ручного изменения SQL-скриптов.

Роль EF Core в поддержании сложной предметной области и обеспечении безопасности является центральной. Система «Цифровая приёмная» включает в себя множество ключевых сущностей с комплексными взаимосвязями. Ручное управление SQL-запросами для всех этих сущностей и их связей было бы чрезвычайно трудоемким и подверженным ошибкам. EF Core, как ORM, позволяет моделировать предметную область с помощью классов C#, что значительно упрощает манипулирование данными и поддержание целостности сложных связей. Более того, обработка записей на приём, персональных данных пользователей и системных настроек требует высокого уровня безопасности. Встроенная защита от SQL-инъекций в EF Core, обеспечиваемая параметризованными запросами, снижает риск одной из наиболее распространенных уязвимостей веб-приложений, обеспечивая более безопасное взаимодействие с базой данных без дополнительных усилий по валидации на уровне приложения. Таким образом, EF Core не только ускоряет разработку и упрощает управление сложной базой данных, но и повышает безопасность системы, автоматически обрабатывая многие аспекты защиты данных. Это позволяет разработчикам сосредоточиться на функциональности, будучи уверенными в базовой безопасности слоя доступа к данным.

**PostgreSQL:** — это мощная, открытая, объектно-реляционная система управления базами данных (СУБД), известная своей надежностью, функциональностью и производительностью.

Среди ключевых преимуществ PostgreSQL — полная ACID-совместимость, которая гарантирует атомарность, согласованность, изолированность и долговечность транзакций, обеспечивая тем самым высокую целостность данных. СУБД отличается высокой расширяемостью, поддерживая широкий спектр типов данных, включая JSON/JSONB, и позволяя добавлять пользовательские функции и операторы. PostgreSQL известен своей высокой производительностью, особенно при работе с большими объемами данных и сложными запросами. Механизм управления параллелизмом Multi-Version Concurrency Control (MVCC) позволяет нескольким транзакциям одновременно получать доступ к одним и тем же данным без конфликтов. Система поддерживает различные типы индексов для эффективного извлечения данных. Функции репликации и высокой доступности, включая синхронную и асинхронную репликацию, обеспечивают отказоустойчивость. PostgreSQL также предлагает продвинутые функции безопасности, такие как ролевой контроль доступа, шифрование данных и безопасность соединений. Будучи проектом с открытым исходным кодом, он не требует лицензионных отчислений, что делает его экономически эффективным решением.

Выбор PostgreSQL для системы «Цифровая приёмная» обусловлен несколькими факторами. Для системы управления приёмами, где точность расписаний, статусов встреч и данных пользователей критична, ACID-совместимость PostgreSQL обеспечивает высочайший уровень надежности данных. Способность PostgreSQL обрабатывать большие объемы данных и высокую конкуренцию делает его идеальным выбором для системы, которая может расти в масштабах организации. Поддержка JSON позволяет хранить неструктурированные данные (например, логи, дополнительные настройки) без изменения схемы, что обеспечивает гибкость для будущих расширений системы. Расширенные функции безопасности, такие как ролевой контроль доступа, соответствуют требованиям к защите конфиденциальных данных пользователей и расписаний.

PostgreSQL выступает в качестве надежного фундамента для обеспечения масштабируемости и гибкости данных в динамичной системе. «Цифровая приёмная» обрабатывает различные типы данных, включая структурированные (пользователи, встречи) и потенциально полуструктурированные (уведомления, настройки интеграции). Хотя большинство данных в системе хорошо структурированы, такие сущности, как «Уведомление» или «Настройки интеграции», могут со временем потребовать хранения более сложных или динамичных данных, например, различных параметров для разных типов уведомлений или интеграций. Поддержка JSON в PostgreSQL позволяет хранить такие данные без жесткой привязки к реляционной схеме, обеспечивая гибкость для будущих изменений и расширений. Кроме того, система должна быть способна обслуживать растущее число пользователей и встреч. Высокая производительность PostgreSQL, его возможности репликации и MVCC обеспечивают необходимую горизонтальную масштабируемость и эффективную обработку параллельных операций, что критично для поддержания отзывчивости системы при увеличении нагрузки. Таким образом, выбор PostgreSQL не только обеспечивает надежное хранение текущих структурированных данных, но и предоставляет платформу, готовую к эволюции, что позволяет системе адаптироваться к новым требованиям, эффективно управлять растущими объемами данных и поддерживать высокую доступность.

**SignalR:** — это библиотека ASP.NET Core, предназначенная для добавления функциональности веб-приложений в реальном времени. Она позволяет серверному коду отправлять асинхронные уведомления клиентскому коду в веб-браузере, мобильном или настольном приложении.

Ключевые преимущества SignalR включают обеспечение мгновенной двусторонней связи между клиентами и сервером. Библиотека автоматически управляет постоянными соединениями, используя WebSockets, Server-Sent Events или Long Polling в зависимости от возможностей клиента и сервера. SignalR предоставляет простой API для реализации функциональности реального времени без сложного кодирования. Он разработан для обработки большого количества пользователей и может масштабироваться с использованием бэкплейнов, таких как Redis, SQL Server или Azure Service Bus. Кроме того, SignalR включает встроенные функции безопасности, такие как аутентификация, авторизация и шифрование.

Выбор SignalR для системы «Цифровая приёмная» является критически важным. Система требует оперативного информирования пользователей о статусе встреч, напоминаниях, переносах или отменах. SignalR идеально подходит для отправки push-уведомлений, обеспечивая мгновенную доставку информации. Для секретарей и принимающих лиц критически важно видеть актуальное состояние очереди приёма и изменения в ней. SignalR позволяет мгновенно обновлять интерфейсы клиентов при изменении порядка или статуса встреч в очереди. Возможность получать живые обновления и уведомления значительно повышает удобство использования системы для всех ролей, делая взаимодействие более динамичным и эффективным.

SignalR является ключевым фактором повышения эффективности и удовлетворенности пользователей в системе «Цифровая приёмная». Система должна предоставлять оперативное информирование всех сторон, включая автоматическое получение напоминаний и уведомлений, а также управление электронной очередью с автоматическим пересчетом. Без SignalR, для получения актуальной информации о встречах и очереди, клиентам пришлось бы постоянно выполнять запросы к серверу (polling). Это создало бы задержки, увеличило бы нагрузку на сервер и ухудшило бы пользовательский опыт, так как информация не обновлялась бы мгновенно. SignalR решает эту проблему, устанавливая постоянное двустороннее соединение, что позволяет серверу мгновенно отправлять уведомления о новых встречах, изменениях статуса, напоминаниях и обновлениях очереди. Это приводит к следующим преимуществам: для посетителей мгновенные напоминания и подтверждения снижают количество пропущенных встреч и улучшают планирование; для секретарей и руководителей актуальная информация об очереди позволяет им более эффективно управлять приёмом, объявлять перерывы и оперативно реагировать на изменения, что оптимизирует рабочий процесс и снижает операционные издержки. В целом, повышается прозрачность и оперативность взаимодействия, что ведет к росту удовлетворенности всех пользователей системы.

**JWT (JSON Web Tokens):** JWT представляет собой компактный, безопасный для URL-адресов способ представления утверждений (claims) между двумя сторонами. Эти токены являются самодостаточными, поскольку содержат всю необходимую информацию для аутентификации и авторизации непосредственно в себе, и подписаны криптографическим ключом для обеспечения целостности.

Ключевые преимущества JWT включают масштабируемость: серверу не нужно хранить данные сессии, что делает JWT идеальным для распределенных и масштабируемых систем, включая микросервисные архитектуры. Компактный размер JWT уменьшает накладные расходы при передаче данных между клиентом и сервером. JWT широко поддерживаются различными языками программирования и платформами, обеспечивая кроссплатформенную совместимость. Повышенные функции безопасности включают криптографическую подпись, которая обеспечивает неизменность токена, и наличие срока действия (exp claim), что снижает риски компрометации. Кроме того, JWT хорошо подходят для сценариев единого входа (Single Sign-On).

Выбор JWT для системы «Цифровая приёмная» обусловлен необходимостью обеспечения безопасности API. JWT является стандартом для защиты RESTful API. В «Цифровой приёмной» он будет использоваться для аутентификации пользователей после входа в систему, выдавая токен, который клиент будет включать в последующие запросы. Сервер сможет валидировать этот токен без обращения к базе данных для каждой проверки. JWT может содержать утверждения о роли пользователя (например, «Посетитель», «Секретарь», «Принимающее лицо», «Администратор»). Это позволяет бэкенду быстро определять права доступа пользователя к определенным ресурсам или действиям. В соответствии с принципом бессостояния REST API, JWT устраняет необходимость в серверных сессиях, что упрощает масштабирование бэкенда и снижает нагрузку на сервер, так как не требуется хранить и управлять состоянием сессий для каждого пользователя. Поскольку «Цифровая приёмная» может иметь различные клиентские приложения (веб, мобильные), JWT обеспечивает унифицированный и безопасный механизм аутентификации для всех них.

JWT является краеугольным камнем для построения безопасной и масштабируемой ролевой модели в системе. Система «Цифровая приёмная» имеет гибкую ролевую модель доступа с четырьмя основными ролями, каждая из которых обладает набором разрешенных действий. Использование JWT обеспечивает бессостоятельную аутентификацию и может содержать утверждения о пользователе, включая его роль. Традиционные сессии требуют хранения состояния на сервере, что усложняет масштабирование. JWT, будучи самодостаточным, позволяет серверу проверять аутентификацию и авторизацию пользователя по каждому запросу, используя информацию, закодированную в токене (например, role\_id из таблицы users ), без необходимости обращения к базе данных. Это идеально согласуется с принципом бессостояния REST API. В JWT можно включать не только role\_id, но и другие утверждения, например, user\_id или receiver\_id. Это позволяет реализовать гранулярный контроль доступа: например, секретарь может управлять встречами только «связанными с закреплёнными принимающими», что требует проверки receiver\_id из токена или связанной с ним информации. Таким образом, применение JWT позволяет эффективно и безопасно реализовать сложную ролевую модель. Это гарантирует, что каждый пользователь имеет доступ только к тем функциям и данным, которые соответствуют его роли и правам, при этом поддерживая высокую производительность и масштабируемость системы, что критически важно для корпоративного приложения.

* 1. **Проектирование пользовательских сценариев и ролевой модели**

Данный раздел посвящен описанию взаимодействия пользователей с системой «Цифровая приёмная» на основе их определенных ролей и соответствующих сценариев использования. Эффективно спроектированная ролевая модель является фундаментальной основой для обеспечения безопасности, управляемости и удобства использования системы.

**Описание ролей пользователей:** Система «Цифровая приёмная» предусматривает четыре основные роли пользователей, каждая из которых обладает четко определенным набором прав и обязанностей, что обеспечивает эффективное и безопасное взаимодействие с функционалом системы. Эти роли явно определены в таблице roles базы данных и связаны с конкретными пользователями через поле role\_id в таблице users.

**Посетитель (Сотрудник):**

**Назначение:** Основной пользователь системы, целью которого является запись на приём к руководителю.

**Ключевые действия:**

1. Создание запроса на встречу: Выбор свободной даты и времени встречи из доступных слотов.
2. Редактирование или отмена встречи: Возможность изменить параметры назначенной встречи до её начала или полностью отменить её.
3. Получение уведомлений: Автоматическое получение напоминаний и уведомлений о переносе, отмене или подтверждении встречи.
4. Подтверждение встречи через уведомление: например, получение push-уведомления за час до встречи с возможностью подтвердить участие.
5. Прикрепление документов: Возможность загружать и прикреплять к встрече различные документы, такие как заявления или служебные записки.

**Секретарь:**

**Назначение:** Отвечает за администрирование расписания руководителя и управление очередью приёма.

**Ключевые действия:**

1. Создание встречи от имени сотрудника: если сотрудник обращается офлайн, например, через стойку или киоск.
2. Редактирование очереди приёма: Изменение порядка встреч, временных интервалов, а также отметка начала и окончания приёма.
3. Отмена или перенос встреч: Изменение параметров уже назначенных встреч.
4. Начало и завершение встречи: Управление фактическим процессом приёма.
5. Управление приёмами, связанными с закреплёнными принимающими лицами.

Функциональное описание роли «Секретарь» указывает на то, что секретарь «управляет приёмами, связанными с закреплёнными принимающими». В то же время, таблица users связывает пользователя с ролью через role\_id, но явной таблицы для связи «секретарь-руководитель» в предоставленных описаниях отсутствует. Без явной модели этой связи, система не сможет определить, какой секретарь имеет право управлять расписанием какого руководителя. Это может привести к нарушению принципа наименьших привилегий и потенциальным проблемам безопасности или несанкционированному доступу к расписаниям. Для обеспечения точного контроля доступа и соответствия описанной функциональности, необходимо ввести новую сущность или механизм, который явно связывает секретарей с одним или несколькими принимающими лицами (руководителями), чьим расписанием они управляют. Это может быть таблица secretary\_receiver\_assignments с полями secretary\_id (внешний ключ к users.id) и receiver\_id (внешний ключ к users.id). Включение такой таблицы позволит системе точно авторизовать действия секретаря, гарантируя, что он может управлять только расписаниями тех руководителей, за которыми он закреплен. Это повысит безопасность, управляемость и соответствие бизнес-процессам.

**Принимающее лицо (Руководитель):**

**Назначение:** Лицо, проводящее встречи и управляющее своим личным расписанием.

**Ключевые действия:**

1. Просмотр очереди встреч: Доступ к списку предстоящих приёмов.
2. Управление приёмом: Запуск встречи, завершение, перенос.
3. Объявление перерыва: Возможность временно приостановить приём и оповестить об этом ожидающих посетителей.

**Администратор системы:**

**Назначение:** Обеспечивает техническую поддержку, настройку системы и управление её инфраструктурой.

**Ключевые действия:**

1. Интеграция с Active Directory и календарями: Автоматическая синхронизация данных пользователей и событий.
2. Настройка прав пользователей и ролей: Определение, кто что может делать в системе.
3. Активация лицензии: Ввод ключа, ограничивающего количество пользователей, срок действия и другие параметры.
4. Настройка уведомлений: Определение времени отправки напоминаний, каналов уведомлений и шаблонов сообщений.
5. Управление настройками интеграции.

**Таблица 2: Описание ролей и их функций в системе «Цифровая приёмная»** Данная таблица предоставляет структурированный обзор ролей пользователей в системе «Цифровая приёмная» и их основных функций. Она служит быстрым справочником для понимания распределения обязанностей и прав доступа.

Таблица 13. Описание ролей и их функций в системе «Цифровая приёмная»

|  |  |  |
| --- | --- | --- |
| Роль | Назначение | Ключевые действия/Функции |
| Посетитель (Сотрудник) | Обычный пользователь, желающий записаться на приём к руководителю. | Создание запроса на встречу, редактирование/отмена встречи, получение уведомлений, подтверждение встречи через уведомление, прикрепление документов. |
| Секретарь | Отвечает за администрирование расписания руководителя и управление очередью приёма. | Создание встречи от имени сотрудника, редактирование очереди приёма, отмена/перенос встреч, начало/завершение встречи, управление приёмами закреплённых принимающих лиц. |
| Принимающее лицо (Руководитель) | Проводит встречи и управляет личным расписанием. | Просмотр очереди встреч, управление приёмом (запуск, завершение, перенос), объявление перерыва. |
| Администратор системы | Обеспечивает техническую поддержку, настройку системы и управление её инфраструктурой. | Интеграция с Active Directory и календарями, настройка прав пользователей и ролей, активация лицензии, настройка уведомлений, управление настройками интеграции. |

**Пользовательские сценарии:** Ниже представлены примеры ключевых пользовательских сценариев, демонстрирующих взаимодействие различных ролей с системой «Цифровая приёмная», а также последовательность действий и их влияние на состояние системы.

**Сценарий 1: Запись Посетителя на Приём**

* + 1. Посетитель (сотрудник) входит в систему через веб-интерфейс или мобильное приложение.
    2. Посетитель выбирает принимающее лицо из списка доступных и просматривает свободные слоты в его расписании.
    3. Посетитель выбирает подходящую дату и время, указывает тему встречи и при необходимости загружает и прикрепляет к встрече дополнительные вложения (например, заявление или справку).
    4. Система создает новую запись в таблице appointments со статусом «запрошена» и автоматически отправляет уведомление о создании встречи посетителю и принимающему лицу.
    5. Посетитель получает уведомление о подтверждении записи или напоминание за час до встречи.
    6. Посетитель может подтвердить или отменить встречу, взаимодействуя с уведомлением.

**Сценарий 2: Управление Очередью Секретарем**

* + 1. Секретарь входит в систему и переходит в раздел управления очередью для закрепленного за ним руководителя.
    2. Секретарь просматривает текущую очередь приёма (queue\_items) и актуальные статусы встреч (appointment\_statuses, queue\_statuses).
    3. Секретарь имеет возможность изменить порядок встреч в очереди, например, переместить встречу вперед или назад.
    4. Секретарь инициирует начало встречи (статус queue\_items.status\_id меняется на «в процессе»), и система автоматически отправляет уведомление следующему в очереди посетителю.
    5. Секретарь может перенести или отменить встречу, при этом система автоматически уведомляет всех участников о произошедших изменениях.
    6. Секретарь завершает встречу (статус queue\_items.status\_id меняется на «завершено»).

**Сценарий 3: Управление Приёмом Принимающим Лицом**

* + 1. Принимающее лицо (руководитель) входит в систему и просматривает свою очередь предстоящих встреч.
    2. Руководитель инициирует текущую встречу, и её статус оперативно обновляется в системе.
    3. Руководитель объявляет перерыв, и система оповещает ожидающих посетителей о временной приостановке приёма.
    4. Руководитель завершает текущую встречу.

**Сценарий 4: Администрирование Системы**

* + 1. Администратор системы входит в систему с полными правами доступа.
    2. Администратор настраивает параметры интеграции с внешними сервисами, такими как Active Directory или различные календари (integration\_settings).
    3. Администратор управляет правами пользователей и ролями, назначая или изменяя роли для существующих пользователей.
    4. Администратор активирует или обновляет лицензию системы (licenses), устанавливая ограничения по количеству пользователей или сроку действия.
    5. Администратор настраивает шаблоны и каналы для уведомлений, определяя время отправки и содержание сообщений.

**Реализация ролевой модели:** Реализация ролевой модели в системе «Цифровая приёмная» базируется на использовании двух основных таблиц в базе данных PostgreSQL: users и roles.

**Таблица roles:** Эта таблица служит справочником для определения всех возможных ролей в системе, таких как «Администратор», «Секретарь», «Посетитель» и «Принимающее лицо». Каждая роль имеет уникальный идентификатор (id) и название (name).

**Таблица users:** В этой таблице хранится информация обо всех пользователях системы. Поле role\_id в таблице users является внешним ключом, который ссылается на id в таблице roles. Это устанавливает прямую связь «один-ко-многим» между ролью и пользователями: одна роль может быть назначена многим пользователям, но каждый пользователь имеет только одну основную роль.

**Авторизация:** При каждом запросе к API, после успешной аутентификации пользователя (с использованием JWT, содержащего role\_id и другие утверждения), бэкенд выполняет проверку роли пользователя и его прав доступа к запрошенному ресурсу или действию. Этот механизм гарантирует, что только авторизованные пользователи могут выполнять определенные операции, соответствующие их назначенной роли.

Хотя таблица users имеет role\_id , что позволяет назначить пользователю одну роль, в реальных системах часто требуется более гранулярный контроль, чем просто «роль». Например, секретарь «управляет приёмами, связанными с закреплёнными принимающими». Простая связь «один пользователь — одна роль» не позволяет реализовать такие детализированные правила, как «секретарь X может управлять только расписанием руководителя Y». Для этого необходимы дополнительные атрибуты или связи, выходящие за рамки базовой ролевой модели.

Для решения этой задачи, помимо базовой role\_id, в системе могут быть реализованы:

**Привязка к сущностям:** Для секретаря это может быть таблица secretary\_receiver\_assignments, которая явно определяет, какие руководители закреплены за каким секретарем.

**Настройки прав на уровне пользователя/группы:** Хотя roles определяют общие права, для некоторых специфических случаев может потребоваться переопределение или добавление прав для отдельных пользователей или групп пользователей. Это может быть реализовано через дополнительную таблицу user\_permissions или group\_permissions.

**Контекстная авторизация:** Бизнес-логика API должна проверять не только роль, но и контекст операции. Например, пользователь может отменить только *свою* встречу, а секретарь — любую встречу закрепленного руководителя.

Таким образом, хотя базовая структура users и roles заложена, для полноценной реализации бизнес-требований потребуется более сложная логика авторизации, которая учитывает не только role\_id, но и специфические связи между пользователями и сущностями. Это обеспечит высокую гибкость и безопасность системы, позволяя точно контролировать доступ к данным и функциям.

**Выводы по главе 2**

В рамках второй главы, базируясь на результатах анализа предметной области и обосновании необходимости создания системы «Цифровая приемная» (DRSapi) из первой главы, была проведена комплексная работа по проектированию ее функциональной и технической архитектуры.

Были детально формализованы функциональные требования, сгруппированные по четырем пользовательским ролям и общим функциям, включая интеграцию с Active Directory и систему уведомлений на базе SignalR. Также были определены ключевые нефункциональные требования, охватывающие, надежность, безопасность масштабируемость, удобство использования и поддержки.

На основе этих требований была спроектирована модель данных: описаны 12 ключевых сущностей предметной области, их атрибуты, типы данных и ограничения. Разработана логическая ER-модель базы данных, наглядно демонстрирующая взаимосвязи и обеспечивающая целостность данных.

Была спроектирована архитектура API на основе стиля REST и обоснован выбор технологического стека, включающего ASP.NET Core, Entity Framework Core PostgreSQL, SignalR и JWT.

В завершение главы были разработаны пользовательские сценарии и детализирована ролевая модель. Описаны четыре основные пользовательские роли, их ключевые функции и права доступа, а также принципы реализации ролевой модели на основе таблиц users и roles и связующих сущностей для гранулярного контроля.

Таким образом, в рамках второй главы были заложены все необходимые теоретические и проектные основы – определены функциональные и нефункциональные характеристики системы, ее структура данных, архитектура взаимодействия компонентов и принципы работы пользователей – которые являются фундаментом для последующей практической реализации программных модулей системы «DRSapi».

1. **Nielsen, J.** Usability Engineering. – Morgan Kaufmann, 1993. – 362 p. (Или аналогичный авторитетный труд по юзабилити, например, **Круг, С.** Не заставляйте меня думать. Веб-юзабилити и здравый смысл. – СПб.: Символ-Плюс, (год издания).)
2. **Cooper, A., Reimann, R., Cronin, D., Noessel, C.** About Face: The Essentials of Interaction Design. – 4th ed. – Wiley, 2014. – 720 p. (Или другой авторитетный источник по дизайну интерфейсов и пользовательского опыта, например, **Норман, Д. А.** Дизайн привычных вещей. – М.: Вильямс, (год издания).)
3. **Федеральный закон "О персональных данных"** от 27.07.2006 N 152-ФЗ (ред. от [указать актуальную дату редакции, например, 06.02.2023]) // Собрание законодательства РФ. – 31.07.2006. – N 31 (1 ч.). – ст. 3451.
4. *[Источник не указан в предоставленном фрагменте под номером [8]]*
5. **OWASP Top Ten Project** // OWASP Foundation. – URL: [https://owasp.org/www-project-top-ten/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fowasp.org%2Fwww-project-top-ten%2F) (дата обращения: ДД.ММ.ГГГГ). (Следует указать актуальный год списка Top Ten, например, OWASP Top 10 2021).
6. **Active Directory Domain Services Overview** // Microsoft Learn. – URL: [https://learn.microsoft.com/ru-ru/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview](https://www.google.com/url?sa=E&q=https%3A%2F%2Flearn.microsoft.com%2Fru-ru%2Fwindows-server%2Fidentity%2Fad-ds%2Fget-started%2Fvirtual-dc%2Factive-directory-domain-services-overview) (дата обращения: ДД.ММ.ГГГГ). (Или более общая книга/статья, описывающая Active Directory, например, **Мюллер, Дж. П.** Microsoft Windows Server. Полное руководство. – (Издательство, год)).
7. **Fielding, R. T.** Architectural Styles and the Design of Network-based Software Architectures: Doctoral dissertation. – University of California, Irvine, 2000. – 180 p. (Это первоисточник по REST).
8. **Hohpe, G., Woolf, B.** Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. – Addison-Wesley Professional, 2003. – 736 p. (Или другой релевантный источник по шаблонам интеграции корпоративных приложений).

9. What Is the Future of ORM in .NET Development? // Built In. URL: [https://builtin.com/software-engineering-perspectives/orm-net-development](https://www.google.com/url?sa=E&q=https%3A%2F%2Fbuiltin.com%2Fsoftware-engineering-perspectives%2Form-net-development) (дата обращения: ЧЧ.ММ.ГГГГ).  
10. Overview of Entity Framework Core - EF Core // Microsoft Learn. URL: [https://learn.microsoft.com/en-us/ef/core/](https://www.google.com/url?sa=E&q=https%3A%2F%2Flearn.microsoft.com%2Fen-us%2Fef%2Fcore%2F) (дата обращения: ЧЧ.ММ.ГГГГ).  
11. What are the 6 characteristics of a REST API? // ScrapingBee. URL: [https://www.scrapingbee.com/blog/rest-api-characteristics/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fwww.scrapingbee.com%2Fblog%2Frest-api-characteristics%2F) (дата обращения: ЧЧ.ММ.ГГГГ).  
12. What is RESTful API? - RESTful API Explained // AWS. URL: [https://aws.amazon.com/what-is/restful-api/](https://www.google.com/url?sa=E&q=https%3A%2F%2Faws.amazon.com%2Fwhat-is%2Frestful-api%2F) (дата обращения: ЧЧ.ММ.ГГГГ).  
13. JWT Authentication: A Secure & Scalable Solution for Modern Applications // Authgear. URL: [https://www.authgear.com/post/jwt-authentication](https://www.google.com/url?sa=E&q=https%3A%2F%2Fwww.authgear.com%2Fpost%2Fjwt-authentication) (дата обращения: ЧЧ.ММ.ГГГГ).  
14. JWT API Authentication for a Microservices Architecture // miniOrange API Security. URL: [https://apisecurity.miniorange.com/jwt-api-authentication-for-microservices-architecture/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fapisecurity.miniorange.com%2Fjwt-api-authentication-for-microservices-architecture%2F) (дата обращения: ЧЧ.ММ.ГГГГ).  
15. Overview of ASP.NET Core // Microsoft Learn. URL: [https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core](https://www.google.com/url?sa=E&q=https%3A%2F%2Flearn.microsoft.com%2Fen-us%2Faspnet%2Fcore%2Fintroduction-to-aspnet-core) (дата обращения: ЧЧ.ММ.ГГГГ).  
16. Building Real-Time Communication Application with SignalR // Step2gen. URL: [https://www.step2gen.com/blog/real-time-communication-application-with-signalr](https://www.google.com/url?sa=E&q=https%3A%2F%2Fwww.step2gen.com%2Fblog%2Freal-time-communication-application-with-signalr) (дата обращения: ЧЧ.ММ.ГГГГ).  
17. Web APIs with ASP.NET Core: A Compilation of Process and Practices // Radixweb. URL: [https://radixweb.com/blog/web-apis-with-asp-net-core](https://www.google.com/url?sa=E&q=https%3A%2F%2Fradixweb.com%2Fblog%2Fweb-apis-with-asp-net-core) (дата обращения: ЧЧ.ММ.ГГГГ).  
18. What is SignalR library for real-time connectivity? // PubNub. URL: [https://www.pubnub.com/learn/glossary/what-is-signalr/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fwww.pubnub.com%2Flearn%2Fglossary%2Fwhat-is-signalr%2F) (дата обращения: ЧЧ.ММ.ГГГГ).  
19. What is PostgreSQL? Features and Benefits // Quest Software. URL: [https://www.quest.com/solutions/develop-manage-postgresql/what-is-postgresql.aspx](https://www.google.com/url?sa=E&q=https%3A%2F%2Fwww.quest.com%2Fsolutions%2Fdevelop-manage-postgresql%2Fwhat-is-postgresql.aspx) (дата обращения: ЧЧ.ММ.ГГГГ).