**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) - *опционально*:** 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.

* **Таблица 2.3.1: Примеры RESTful ресурсов API и их операций** Данная таблица демонстрирует, как принципы RESTful API будут применены к ключевым сущностям системы «Цифровая приёмная», предоставляя конкретные примеры URI, HTTP-методов и соответствующих операций. Это обеспечивает четкое понимание структуры API для разработчиков и интеграторов.

|  |  |  |
| --- | --- | --- |
| **Ресурс API** | **HTTP Метод** | **Описание операции** |
| /api/users | GET | Получение списка всех пользователей |
| /api/users/{id} | GET | Получение информации о конкретном пользователе |
| /api/users | POST | Создание нового пользователя |
| /api/users/{id} | PUT | Полное обновление информации о пользователе |
| /api/users/{id} | DELETE | Удаление пользователя |
| /api/appointments | GET | Получение списка всех встреч |
| /api/appointments/{id} | GET | Получение информации о конкретной встрече |
| /api/appointments | POST | Создание новой встречи |
| /api/appointments/{id} | PUT | Обновление информации о встрече |
| /api/appointments/{id}/cancel | POST | Отмена встречи |
| /api/notifications | GET | Получение списка уведомлений для текущего пользователя |
| /api/notifications/{id}/read | POST | Отметка уведомления как прочитанного |
| /api/queue | GET | Получение текущего состояния очереди для принимающего лица |
| /api/queue/{id}/start | POST | Начало приёма по конкретной встрече в очереди |
| /api/queue/{id}/complete | POST | Завершение приёма по конкретной встрече в очереди |
| /api/attachments | POST | Загрузка нового вложения к встрече |
| /api/attachments/{id} | GET | Скачивание вложения |
| /api/licenses | GET | Получение информации о текущей лицензии |
| /api/licenses/{id} | PUT | Обновление информации о лицензии |
| /api/integration-settings | GET | Получение текущих настроек интеграции |
| /api/integration-settings | PUT | Обновление настроек интеграции |

Export to Sheets

**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:** 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 бессостояние и 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.

|  |  |  |
| --- | --- | --- |
| **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. |

Export to Sheets

* **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 (через SignalR) 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 (через SignalR) 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.

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).