**ГЛАВА 3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ СИСТЕМЫ "DRSapi"**

На основе спроектированной архитектуры и детализированных требований, изложенных в предыдущей главе, в данной главе описывается процесс программной реализации серверной части системы "Цифровая приёмная" ("DRSapi"). Рассматриваются ключевые аспекты разработки API контроллеров, механизма аутентификации и авторизации, реализации бизнес-логики управления очередями и системы уведомлений. Кроме того, описывается разработка прототипа тестового клиента в виде чат-бота для апробации функциональности API и проведения пользовательского тестирования, а также приводятся результаты различных видов тестирования системы.

**3.1 Реализация серверной части системы "DRSapi"**

Разработка серверной части системы "DRSapi" осуществлялась на платформе ASP.NET Core с использованием языка программирования C#. В качестве системы управления базами данных была выбрана PostgreSQL, а для взаимодействия с ней применялся ORM Entity Framework Core. Данный выбор обусловлен их производительностью, масштабируемостью, открытым исходным кодом и широкой поддержкой сообщества, что было обосновано в Главе 2. Данный раздел описывает реализацию основных компонентов бэкенда.

**3.1.1 Разработка API контроллеров для управления пользователями, встречами и очередями**

Ядром взаимодействия с клиентскими приложениями в системе "DRSapi" является REST API. Были разработаны следующие основные контроллеры, соответствующие ключевым сущностям и функциям системы:

**AuthController**: Ответственен за аутентификацию пользователей. Реализует эндпоинты для входа в систему (POST /auth/login), при котором генерируется JWT-токен, содержащий информацию о пользователе и его роли, и выхода (POST /auth/logout).

Пример спецификации эндпоинта для входа в систему представлен ниже

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 37. Фрагмент спецификации эндпоинта авторизации.

Фрагмент кода, демонстрирующий реализацию метода входа или генерации токена, приведен ниже:

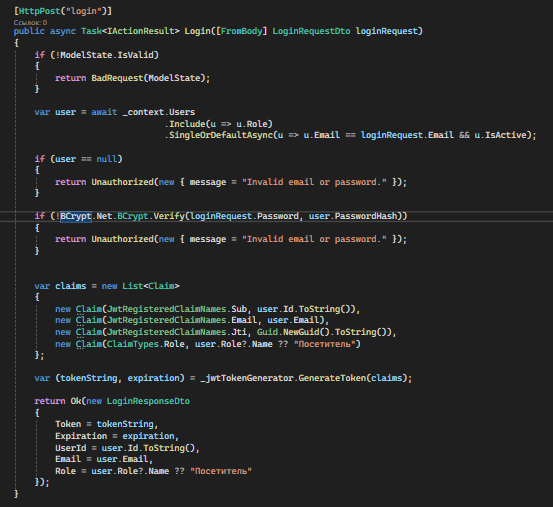
**

Рисунок 38. Листинг кода авторизации из AuthController.

**UsersController**: Предоставляет эндпоинты для управления информацией о пользователях.

Таблица 14. Эндпоинты UsersController

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Путь (Эндпоинт) | Описание | Доступ | Примечания |
| GET | /users/me | Получить информацию о своем профиле. | Аутентифицированный пользователь |  |
| PUT | /users/me | Обновить данные своего профиля (например, ФИО, телефон). | Пользователь (аутентифицированный) |  |
| GET | /users | Получение списка всех пользователей. | Администратор | Фильтрация, пагинация |
| GET | /users/{id} | Получение информации о конкретном пользователе. | Администратор, Секретарь (с проверкой прав) |  |
| POST | /users | Административное создание пользователей. | Администратор |  |
| PUT | /users/{id} | Административное обновление данных пользователя. | Администратор |  |
| DELETE | /users/{id} | Административное удаление пользователя. | Администратор |  |
| GET | /users/receivers | Предоставляет список пользователей с ролью "Принимающее лицо". | Любой пользователь (для выбора при создании встречи) | Для выбора при создании встречи |

В системе используются объекты, предназначенный для передачи данных между разными частями приложения или даже между разными приложениям, называемые DTO (Data Transfer Object). Ниже приведен пример DTO для создания пользователя.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 39. Файл CreateUserDTO.cs

**AppointmentsController**: Центральный контроллер для управления встречами. Логика контроллера учитывает ролевую модель и права доступа пользователей к операциям над встречами.

Таблица 15. Эндпоинты AppointmentsController

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | | Путь (Эндпоинт) | Описание | Доступ | Примечания |
| POST | /appointments | Создание новой заявки на встречу. | Аутентифицированный пользователь (предположительно) |  |
| GET | /appointments | Получение списка встреч. | Права зависят от роли пользователя | Фильтрация (статус, дата, участники), пагинация |
| GET | /appointments/{id} | Получение детальной информации о конкретной встрече. | Пользователь, имеющий доступ к встрече (зависит от роли) |  |
| PUT | /appointments/{id} | Обновление деталей существующей встречи. | Пользователь, имеющий право редактировать встречу (зависит от роли) | Например, время, место, тема |
| PATCH | /appointments/{id} | Изменение статуса встречи. | Пользователь, имеющий право изменять статус встречи (зависит от роли) | Например, подтверждение, отмена, завершение |
| DELETE | /appointments/{id} | Удаление встречи. | Администратор/Секретарь |  |

**QueueController**: Реализует функционал управления электронными очередями к принимающим лицам.

Таблица 16. Эндпоинты QueueController

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Путь (Эндпоинт) | Описание | Доступ | Примечания |
| GET | /receivers/{receiverId}/queue | Просмотр текущего состояния очереди для конкретного принимающего лица. | Принимающее лицо, Администратор/Секретарь (вероятно) |  |
| POST | /receivers/{receiverId}/queue/items | Добавление подтвержденной встречи в очередь. | Система (автоматически при подтверждении) / Администратор/Секретарь | Только подтвержденные встречи |
| PUT | /queue\_items/{itemId} | Изменение позиции элемента в очереди. | Администратор/Секретарь, возможно Принимающее лицо | Реализовано с логикой сдвига |
| PATCH | /queue\_items/{itemId} | Изменение статуса элемента в очереди. | Принимающее лицо (для "В процессе", "Завершен"), Администратор/Секретарь | Например, "В процессе", "Завершен" |
| DELETE | /queue\_items/{itemId} | Удаление элемента из очереди. | Администратор/Секретарь |  |

**AttachmentsController**: Отвечает за работу с вложениями к встречам. Реализована проверка прав доступа к операциям.

Таблица 17. Эндпоинты AttachmentsController

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Путь (Эндпоинт) | Описание | Доступ | Примечания |
| POST | /appointments/{appointmentId}/attachments | Загрузка файла и прикрепление его к встрече. | Только создатель встречи | Реализована проверка прав |
| GET | /attachments/{attachmentId}/file | Скачивание прикрепленного файла. | Участники встречи, Администраторы | Права доступа определяются |
| DELETE | /attachments/{attachmentId} | Удаление вложения. | Пользователь, загрузивший вложение |  |

**NotificationsController**: Управляет уведомлениями пользователей.

Таблица 18. Эндпоинты NotificationsController

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Путь (Эндпоинт) | Описание | Доступ | Примечания |
| GET | /notifications | Получение списка уведомлений для текущего аутентифицированного пользователя. | Текущий аутентифицированный пользователь | Пагинация, фильтрация |
| POST | /notifications/{id}/mark-as-read | Пометка уведомления как прочитанного. | Текущий аутентифицированный пользователь |  |
| POST | /notifications | Отправка уведомления вручную. | Администратор/Секретарь |  |

При разработке контроллеров особое внимание уделялось следованию принципам RESTful API, четкому именованию ресурсов и использованию соответствующих HTTP-методов для различных операций. Валидация входных данных осуществляется с использованием атрибутов Data Annotations и проверяется на уровне контроллера.

**3.1.2 Реализация механизма аутентификации и авторизации на основе JWT**

Для обеспечения безопасности доступа к API системы "DRSapi" был реализован механизм аутентификации на основе JSON Web Tokens (JWT). Процесс аутентификации был детально описан в Главе 2.

**Процесс аутентификации**: При успешном входе пользователя (через эндпоинт POST /auth/login) система генерирует JWT, содержащий идентификатор пользователя, его роль и другую необходимую информацию (claims), а также время жизни токена. Токен подписывается секретным ключом, хранящимся на сервере.

**Передача токена**: Клиентское приложение сохраняет полученный токен и передает его в заголовке Authorization при каждом запросе к защищенным эндпоинтам.

**Валидация токена**: Серверная часть при получении запроса валидирует токен: проверяет подпись, срок действия, издателя (Issuer) и потребителя (Audience). Настройка middleware для аутентификации JWT в Program.cs включает в себя определение этих параметров валидации токена.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 40. Фрагмент регистрации зависимостей в Program.cs

**Авторизация**: После успешной аутентификации доступ к различным эндпоинтам и операциям разграничивается на основе ролей пользователя, извлеченных из JWT. Для этого используются атрибуты [Authorize] и [Authorize(Roles = "...")] в контроллерах ASP.NET Core. Были определены следующие роли: "Посетитель", "Секретарь", "Принимающее лицо", "Администратор", каждой из которых соответствуют определенные права доступа. Например, доступ к административным функциям управления пользователями разрешен только пользователям с ролью "Администратор".

Изображение выглядит как снимок экрана, текст

Автоматически созданное описание  
 Рисунок 41. Фрагмент иллюстрирующий, ограничения по ролям

**3.1.3 Программная реализация бизнес-логики управления очередями и уведомлениями**

**Управление очередью**: Управление электронными очередями является одной из ключевых функций системы. Для инкапсуляции сложной логики работы с очередями был разработан сервисный слой QueueService. Он включает:

* 1. Механизм добавления подтвержденных встреч в очередь к конкретному принимающему лицу.
  2. Алгоритмы пересчета позиций при добавлении, удалении или изменении порядка элементов в очереди. Для обеспечения корректности операций со списком элементов очереди при одновременных изменениях используются транзакции базы данных.
  3. Логику изменения статусов элементов очереди (например, "Ожидание", "В процессе", "Завершен"), включая автоматическое обновление времени фактического начала и окончания приема. При завершении элемента очереди также обновляется статус соответствующей встречи.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 42. Фрагмент смены статуса в очереди.

**Система уведомлений**: Для обеспечения оперативного информирования пользователей об изменениях статусов встреч, их обновлении или появлении новых уведомлений в системе реализована интеграция с технологией SignalR.

* 1. Разработан NotificationHub, к которому подключаются аутентифицированные пользователи. Подключение пользователя ассоциируется с его уникальным идентификатором, что позволяет адресно доставлять уведомления. Настройка и регистрация хаба в Program.cs обеспечивают его доступность по определенному URL.
  2. При возникновении значимых событий (например, подтверждение встречи, изменение ее деталей принимающей стороной, изменение статуса в очереди) соответствующий контроллер или сервис инициирует отправку уведомления через IHubContext<NotificationHub> конкретному пользователю или группе пользователей.
  3. Уведомления также сохраняются в базе данных для последующего просмотра пользователем.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание  
Рисунок 43. Фрагмент отправки, сообщения пользователю

**3.2 Разработка тестового клиента (чат-бота) для апробации API**

Для практической апробации разработанного API, проведения первичного пользовательского тестирования и демонстрации возможностей API был создан прототип чат-бота. Чат-бот выступает в роли простого клиентского приложения, взаимодействующего с API "DRSapi" для выполнения основных пользовательских сценариев.

**3.2.1 Определение ключевых функций чат-бота для взаимодействия с API**

Функционал чат-бота был ограничен наиболее востребованными операциями для роли "Посетитель", доступными пользователю через текстовые команды или кнопки:

* Аутентификация пользователя в системе "DRSapi" (получение JWT-токена).
* Просмотр списка доступных принимающих лиц.
* Создание заявки на встречу к выбранному принимающему лицу с указанием желаемого времени и темы.
* Просмотр списка своих предстоящих встреч и их статусов.
* Отмена своей заявки на встречу.
* Получение уведомлений от системы "DRSapi" в чате

Для роли “Принимающее лицо” все реализованные функции для посетителя, а также:

* Просмотр и редактирование очереди приёма.
* Обработка и администрирование заявок.

**3.2.2 Описание процесса интеграции чат-бота с системой "DRSapi"**

Чат-бот был разработан с использованием языка программирования Python 3.12 и фреймворка python-telegram-bot для взаимодействия с Telegram Bot API. Интеграция с бэкенд-системой "DRSapi" осуществляется посредством HTTP-запросов к разработанному RESTful API. Для получения уведомлений в реальном времени используется технология WebSocket через клиентскую реализацию протокола SignalR.

Пример диалога пользователя с чат-ботом для записи на встречу показан на рисунке   
Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 44. Интерфейс пользователя чат-бота

Для аутентификации, при старте диалога или по специальной команде, чат-бот запрашивает у пользователя логин и пароль. Эти данные отправляются на эндпоинт POST /auth/login системы "DRSapi". Полученный JWT-токен сохраняется чат-ботом (например, в состоянии диалога или временном хранилище) и используется для последующих авторизованных запросов к API.

При выполнении других операций (например, создание встречи, просмотр руководителей) чат-бот формирует соответствующие HTTP-запросы (GET, POST) с JSON-телом (если необходимо) в соответствии со спецификацией API и отправляет их на соответствующий эндпоинт, добавляя JWT-токен в заголовок Authorization. Ответы от API парсятся чат-ботом и представляются пользователю в удобном для чат-интерфейса текстовом виде.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 45. Фрагмент обработки запроса на показ профиля ботом.

**3.3 Тестирование системы "DRSapi"**

Тестирование является неотъемлемой частью процесса разработки и направлено на проверку корректности работы системы, ее соответствия требованиям и выявление потенциальных ошибок.

**3.3.1 Модульное и интеграционное тестирование серверной части**

Для обеспечения качества кода серверной части проводилось модульное и интеграционное тестирование.

**Модульное тестирование (Unit-тесты)**: Было направлено на проверку отдельных логических компонентов системы в изоляции, в частности, методов сервисного слоя (например, QueueService) и некоторых сложных методов в контроллерах, инкапсулирующих специфическую логику. Для написания модульных тестов использовался фреймворк xUnit, а для создания объектов-заместителей (моков) зависимостей – библиотека Moq.  
Тесты покрывали следующие основные сценарии:

* **Работа с очередью (QueueService):** Корректность добавления элемента в очередь (в начало, конец, середину), правильность сдвига позиций других элементов, удаление элемента из очереди с последующим пересчетом позиций, логика изменения статусов элементов очереди.
* **Работа с уведомлениями:** Проверка логики пометки уведомления как прочитанного, включая проверку принадлежности уведомления пользователю и корректное изменение статуса IsRead.

**Интеграционное тестирование**: Использовалось для проверки взаимодействия между различными компонентами системы, включая API контроллеры и базу данных (в тестовом окружении). Для этого применялся пакет Microsoft.AspNetCore.Mvc.Testing, позволяющий отправлять HTTP-запросы к тестовому экземпляру API и проверять ответы. Были написаны тесты для основных контроллеров (AuthController, UsersController, AppointmentsController), проверяющие:

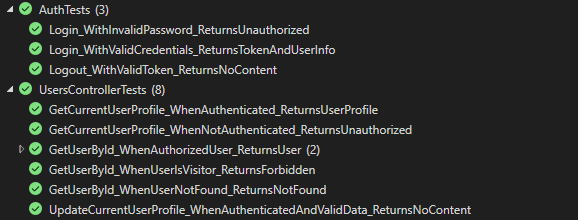
* + Успешное выполнение запросов при корректных данных.
  + Обработку некорректных входных данных и возврат соответствующих кодов ошибок.
  + Работу механизма аутентификации и авторизации (доступ к защищенным ресурсам с валидным токеном и отказ в доступе без токена или с неверными правами).
  + Корректность взаимодействия с базой данных использовалась InMemory-база данных.  
    

Рисунок 46. Фрагмент пройденных тестов.

**3.3.2 Проведение пользовательского тестирования с использованием чат-бота и анализ результатов**

Пользовательское тестирование проводилось с привлечением нескольких моих знакомых, выступивших в роли обычных пользователей, желающих записаться на прием, для оценки удобства использования API через разработанный прототип чат-бота и выявления возможных проблем с точки зрения конечного пользователя.

Методика тестирования включала выполнение пользователями заранее подготовленных сценариев, таких как:

Сценарии для роли "Посетитель"

1. Авторизация в системе через чат-бот.
2. Поиск и выбор принимающего лица.
3. Создание заявки на встречу на определенное время.
4. Просмотр статуса своей заявки.
5. Отмена встречи.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 47. Процесс создания записи.

Сценарии для роли "Принимающее лицо"

1. Авторизация в системе
2. Просмотр заявок на встречу
3. Одобрение (подтверждение) заявки
4. Отклонение заявки
5. Просмотр своей очереди
6. Управление активной очередью (ключевые действия):
   * Начало приёма посетителя
   * Завершение приёма посетителя:

В ходе тестирования собиралась обратная связь от пользователей посредством устного опроса и наблюдения за их взаимодействием с чат-ботом. Были выявлены следующие основные моменты:

Пользователи единодушно отметили, что Telegram-бот хорошо себя показал, как интуитивно понятный и удобный инструмент для записи на встречи. Все основные сценарии были выполнены успешно и без существенных затруднений. Процесс взаимодействия был оценен как быстрый и эффективный для конечного пользователя, желающего записаться.

Однако, в ходе обсуждения было отмечено, что текущий интерфейс чат-бота, ориентированный на записывающихся пользователей, несколько неудобен для задач администрирования очередей со стороны принимающих лиц. Для этой роли требуются другие инструменты и обзор информации.

Анализ результатов пользовательского тестирования позволил подтвердить работоспособность и удобство основного функционала для конечных пользователей. Одновременно была выявлена потребность в разработке или расширении функционала для административной роли, чтобы обеспечить полноценное управление встречами. Полученная обратная связь будет учтена при дальнейшем развитии системы, в частности, при проектировании интерфейса для руководителей.

**Выводы по главе 3**

В рамках данной главы была описана программная реализация ключевых компонентов серверной части системы "DRSapi", включая разработку REST API контроллеров, настройку механизма аутентификации и авторизации на основе JWT, а также реализацию бизнес-логики управления очередями и системы уведомлений с использованием SignalR. Для апробации API был разработан прототип чат-бота, через который проведено начальное пользовательское тестирование. Результаты модульного, интеграционного и пользовательского тестирования подтвердили работоспособность основного функционала системы и позволили выявить направления для возможных улучшений. Реализованный бэкенд представляет собой основу для дальнейшей разработки полнофункциональной системы электронной приемной.