4 РЕАЛИЗАЦИЯ

4.1 Выбор инструментальных средств, системного и дополнительного программного обеспечения

Приложение, разрабатывающееся в рамках дипломного проектирования, состоит из двух частей: бэкэнд и фронтэнд.

Для разработки бэкэнд части информационной системы был выбран язык C#. C# - это современный объектно-ориентированный и типобезопасный язык программирования. C# позволяет разработчикам создавать разные типы безопасных и надёжных приложений, выполняющихся в .NET. C# относится к широко известному семейству языков C, и покажется хорошо знакомым любому, кто работал с C, C++, Java или JavaScript.

C# - объектно-ориентированный, ориентированный на компоненты язык программирования. C# предоставляет языковые конструкции для непосредственной поддержки такой концепции работы. Благодаря этому C# подходит для создания и применения программных компонентов. С момента создания язык C# обогатился функциями для поддержки новых рабочих нагрузок и современными рекомендациями по разработке ПО. В основном C# - объектно-ориентированный язык. Вы определяете типы и их поведение.

Вот лишь несколько функций языка C#, которые позволяют создавать надёжные и устойчивые приложения. Сборка мусора автоматически освобождает память, занятую недостижимыми неиспользуемыми объектами. Типы, допускающие значение null, обеспечивают защиту от переменных, которые не ссылаются на выделенные объекты. Обработка исключений предоставляет структурированный и расширяемый подход к обнаружению

ошибок и восстановлению после них. Лямбда-выражения поддерживают приёмы функционального программирования. Синтаксис LINQ создаёт общий шаблон для работы с данными из любого источника. Поддержка языков для асинхронных операций предоставляет синтаксис для создания распределенных систем. В C# имеется Единая система типов. Все типы C#, включая типы-примитивы, такие, как int и double, наследуют от одного корневого типа object. Все типы используют общий набор операций, а значения любого типа можно хранить, передавать и обрабатывать схожим образом. Более того, C# поддерживает как определяемые пользователями ссылочные типы, так и типы значений. C# позволяет динамически выделять объекты и хранить упрощённые структуры в стеке. C# поддерживает универсальные методы и типы, обеспечивающие повышенную безопасность типов и производительность. C# предоставляет итераторы, которые позволяют разработчикам классов коллекций определять пользовательские варианты поведения для клиентского кода [7].

Язык C# практически универсален. Можно использовать его для создания любого ПО: продвинутых бизнес-приложений, видеоигр, функциональных веб-приложений, приложений для Windows, macOS, мобильных программ для iOS и Android.

C# популярен за счёт своей «простоты». Простоты для современных программистов и больших команд разработчиков, чтобы те могли в сжатые сроки создавать функциональные и производительные приложения. Этому способствуют нетипичные конструкции языка и специфичный синтаксис, помогающий максимально органично реализовать намеченные функции.

Популярность языка - ещё одно значимое преимущество. Большое количество поклонников C# способствуют его развитию. Также это благоприятно влияет на рост числа вакансий, связанных с разработкой на языке Microsoft. Программисты, хорошо знакомые с C#, востребованы в

индустрии, несмотря на их большое и постоянно увеличивающееся количество [8].

Также в языке присутствует обилие синтаксического сахара, который делает тяжёлую жизнь программиста капельку слаще. Вместо того чтобы писать 100500 строк кода, присутствует возможность использовать готовую конструкцию, а компилятор сделает всю остальную работу. Но некоторые такие конструкции являются не самыми оптимальными с точки зрения производительности. Но все это перекрывается за счёт удобочитаемости кода и высокой скоростью разработки [9].

Сама разработка бэкэнд части выполнена при помощи платформы ASP.NET Core.

ASP.NET Core - это веб-инфраструктура с открытым исходным кодом, оптимизированная для облачных вычислений, для разработки современных веб-приложений, которые можно разрабатывать и запускать на Windows, Linux и Mac. Он включает в себя инфраструктуру MVC, которая теперь объединяет функции MVC и веб-API в единую среду веб-программирования. Он был переработан с нуля, чтобы быть быстрым, гибким, современным и работать на разных платформах. В дальнейшем ASP.NET Core - это фреймворк, который можно использовать для веб-разработки с .NET. Также ASP.NET Core имеет схожие функции с MVC и веб-API [10].

Бэкэнд часть является веб-API. Платформа веб-API ASP.NET позволяет с легкостью создавать службы HTTP для широкого диапазона клиентов, включая браузеры и мобильные устройства. ASP.NET Web API - это идеальная платформа для сборки REST-приложений на базе .NET Framework [11].

Для написания приложения была выбрана среда программирования Microsoft Visual Studio.

Microsoft Visual Studio является средой программирования, разработанной компанией Microsoft. Эта среда позволяет создавать

кроссплатформенные проекты на различных языках программирования, таких как Visual Basic, Visual C#, Visual C++, Visual F# и другие. Также она позволяет создавать программы, использующие в своей работе платформу .NET, которая позволяет использовать большой набор сервисов, реализующихся в виде промежуточного, не зависящего от базовой архитектуры, кода. Основной целью создания платформы .NET является возможность реализации разработчиками специальных сервисно-ориентированных программ, работающих на любых платформах.

MS Visual Studio позволяет разработчику иметь доступ к огромной коллекции различных функций, которые позволяют вести разработки для любой версии операционной системы семейства Windows, для интернет-приложений и мобильных приложений. Также среда программирования открывает широкие возможности в области облачных технологий. Эта среда открывает разработчику широкие возможности для реализации самых разных проектов, реализуя высокую производительность и независимость от особенностей оборудования.

Microsoft Visual Studio позволяет осуществлять проектирование программ, используя любые по размеру команды. Эта среда разработки предоставляет инструменты планирования для возможности внедрения методов последовательной разработки, а также для гибкого планирования. Используя весь спектр возможностей, предоставляемых MS Visual Studio, можно реализовать максимально полную систему, наиболее удачно спроектировать любую архитектуру. Таким образом Microsoft Visual Studio представляет собой передовую среду разработки [12].

Фронтэед часть разработана при помощи JavaScript-библиотеки React при поддержке TypeScript.

React - библиотека, написанная на JavaScript, которая используется для работы с интерфейсами. В 2011 году её начали использовать для социальной сети Facebook, а уже в 2013 году библиотеку выложили в открытый доступ, и

энтузиасты со всего мира начали создавать инструменты для расширения её возможностей. Разработчики используют React, чтобы создавать интерфейсы, которые способны менять контент без перезагрузки страницы. Благодаря этому сайты или нативные приложения быстро отзываются на действия пользователей. Можно добавлять товары в корзину без перезагрузки страницы или заполнять формы без переадресации.

React считается самой популярной библиотекой в мире, написанной на JS. Её популярность обеспечивается тем, что с помощью компонента можно решать разные задачи. Особенно, если использовать дополнительные инструменты, которые интегрируются в проекты и открывают доступ к нестандартным возможностям [13].

React предпочтительнее использовать по следующим причинам:

- технология SPA (single-page application, по-русски: “разработка одностраничных приложений”): React поможет вам создать одностраничное приложение. С помощью ReactJS вы сможете изменять (управлять/манипулировать) контент всей страницы с минимальным кодом;

- декларативный подход: React использует декларативный дизайн со всем синтаксическим сахаром, что помогает написать поддерживаемый код высокого уровня. Вам просто нужно определить цель, и React будет обрабатывать инструкции JavaScript DOM с учётом ситуаций, в которых они используются;

- компонент-управляемый пользовательский интерфейс: React основан на компонентной концепции. Компоненты являются многоразовыми строительными блоками в пользовательском интерфейсе. С помощью ReactJS вы можете создать инкапсулированный компонент, который управляет своими данными, и избежать сценария влияния на другие состояния и действия компонентов в DOM-дереве. Это только одна из характеристик компонент-управляемого интерфейса. Он также помогает повторно использовать код, разделять ответственность и избегать повторения [14].

TypeScript (TS, TScript или «тайпскрипт») - это язык программирования для веб-разработки, основанный на JavaScript. Делает код понятнее и надёжнее, добавляет статическую типизацию (переменные привязаны к конкретным типам данных), а также может быть скомпилирован в JavaScript. TypeScript используют фронтенд- и бэкенд-разработчики.

TypeScript добавляет в язык строгую типизацию. Каждой переменной при создании присваивается определённый тип (type) — стандартный или созданный самим разработчиком. Создать тип можно в пределах возможностей языка: например, число от 1 до 31 для записи дня в месяц или массив из двух элементов для записи координат.

TypeScript помогает сократить время на выявление и устранение багов, которые иногда сложно найти в динамической среде JavaScript. С помощью TypeScript можно написать более понятный и читаемый код, который максимально описывает предметную область. Таким образом архитектура становится более выраженной.

Код, написанный на TypeScript, не выполнится напрямую в браузере. Поэтому TS - не самостоятельный язык, а именно языковая надстройка над JS.

Для его работы нужен дополнительный этап - транспиляция, когда программное обеспечение преобразует написанный на TypeScript код в «чистый» JavaScript.

JS не требует установки в систему: его по умолчанию поддерживает любой браузер. А вот TypeScript понадобится установить, потому что для транспиляции необходим модуль tsc [15].

Код для фронтэнд части приложения написан при помощи редактора кода Visual Studio Code.

Visual Studio Code - это «бесплатный редактор, который помогает программисту писать код, помогает в отладке и исправлении кода с помощью метода IntelliSense». В обычных условиях это облегчает пользователю

написание кода простым способом. Многие говорят, что это половина IDE и редактора, но решение остаётся за программистами. Visual Studio Code поддерживает несколько языков программирования, имеет кроссплатформенную поддержку, огромное количество расширений для упрощения создания кода, а также встроенную поддержку системы контроля версий Git.

Visual Studio Code поддерживает несколько языков программирования. Так что раньше программистам требовалась веб-поддержка: другой редактор для разных языков, но он имеет встроенную многоязычную поддержку. Это также означает, что он легко обнаруживает, если есть какая-либо ошибка или ссылка на другой язык, он сможет легко обнаружить её [16].

4.2 Описание реализации вариантов использования

Вариант использования - это связный блок функциональности, которую предоставляет классификатор (система, подсистема или класс). Этот блок описывает последовательность сообщений, которыми обменивается система и один или несколько внешних пользователей (актантов), а также действия, осуществляемые при этом системой.

Вариант использования служит для определения некой части поведения классификатора (которым можно также считать подсистему и даже всю систему целиком), без указания на его внутреннюю структуру. Каждый вариант использования описывает некую услугу, которую предоставляет своим пользователям классификатор. Иначе говоря, это некоторый способ использования классификатора, который виден со стороны. Вариант использования описывает всю последовательность сообщений, которую начинает пользователь (и модели - актант), в терминах взаимодействия между пользователем и классификатором, включая ответы классификатора. К взаимодействию относятся только коммуникации между

системой и актантами. Внутреннее поведение и реализация скрыты. Все множество вариантов использования какого-либо классификатора или системы разделяет и полностью описывает его поведение. Каждый вариант использования представляет собой некую разумную долю функциональности, которая доступна пользователям. Обратите внимание, что под термином пользователь следует понимать не только людей, но и компьютеры, а также прочие объекты. Актант представляет собой некую идеализацию намерений пользователя, а не самого этого пользователя. Один реальный пользователь может соответствовать нескольким актантам, а один актант может представлять одно и то же намерение сразу нескольких пользователей.

Вариант использования включает в себя описание основного поведения, осуществляемого в ответ на запрос пользователя, а также все возможные варианты этого поведения, например альтернативные последовательности, исключительное поведение и обработка ошибок. Для большего удобства варианты использования можно группировать в пакеты [17].

Диаграмма вариантов использования приложения пользователями представлена ниже (рисунок 4.1).

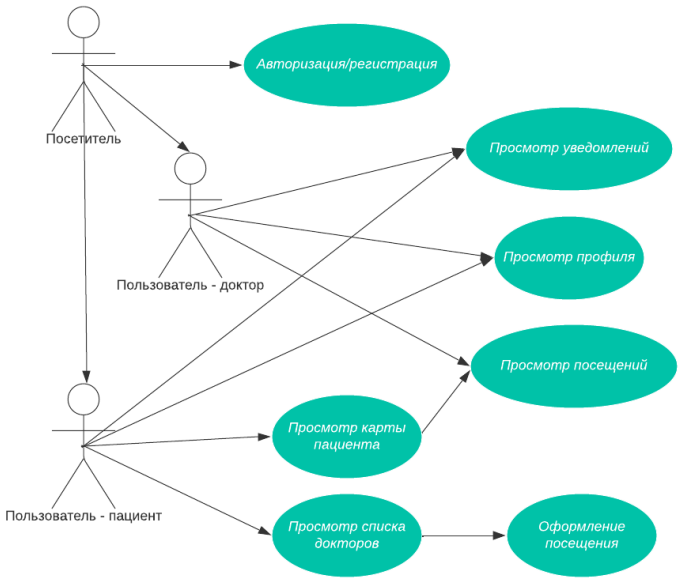


Рисунок 4.1 - Диаграмма вариантов использования

На схеме изображено 3 типа пользователей: анонимный посетитель, доктор, пациент. После авторизации или регистрации у анонимного пользователя появляется роль доктора или пациента. В зависимости от роли пользователям доступен различный функционал интерфейса, тем не менее, у всех типов пользователей имеются несколько одинаковых возможностей, например просмотр и редактирование своего профиля или просмотр уведомлений.

Из некоторых вариантов использования вытекают другие варианты, например из варианта просмотра докторов вытекает вариант оформления посещения для пациента.

В приложении реализована навигационная панель, таким образом пользователи с любой страницы всегда могут перейти на любую другую.

Для примера реализации использования рассмотрим некоторые функции приложения, для реализации которых необходимо применение некоторых нетривиальных алгоритмов:

1. «Быстрый» поиск докторов для пациента.

Доступ к данному поиску имеет пациент, так как доктор в нём не нуждается.

Входными данными в данном случае является строка с ключевыми словами, которые ввёл пациент в строку быстрого поиска.

Выходными данными является список докторов, полученный путём фильтрации списка всех докторов по ключевым словам.

Ключевые слова разделены символом пробела и в бекэнд части происходит разбиение строки на сами ключевые слова, после чего происходит фильтрация списка всех докторов по каждому ключевому слову. Листинг кода, отвечающего за фильтрацию приведён в приложении А. Диаграмма активности выполнения данного кейса приведена ниже (рисунок 4.2).

2. Подтверждение почты для нового пользователя.

После успешной регистрации для входа в систему пользователю необходимо подтвердить почту, путём перехода по ссылке в письме, посланном на почту в момент регистрации нового пользователя.

Входными данными является объект, представляющий собой нового пользователя для регистрации.

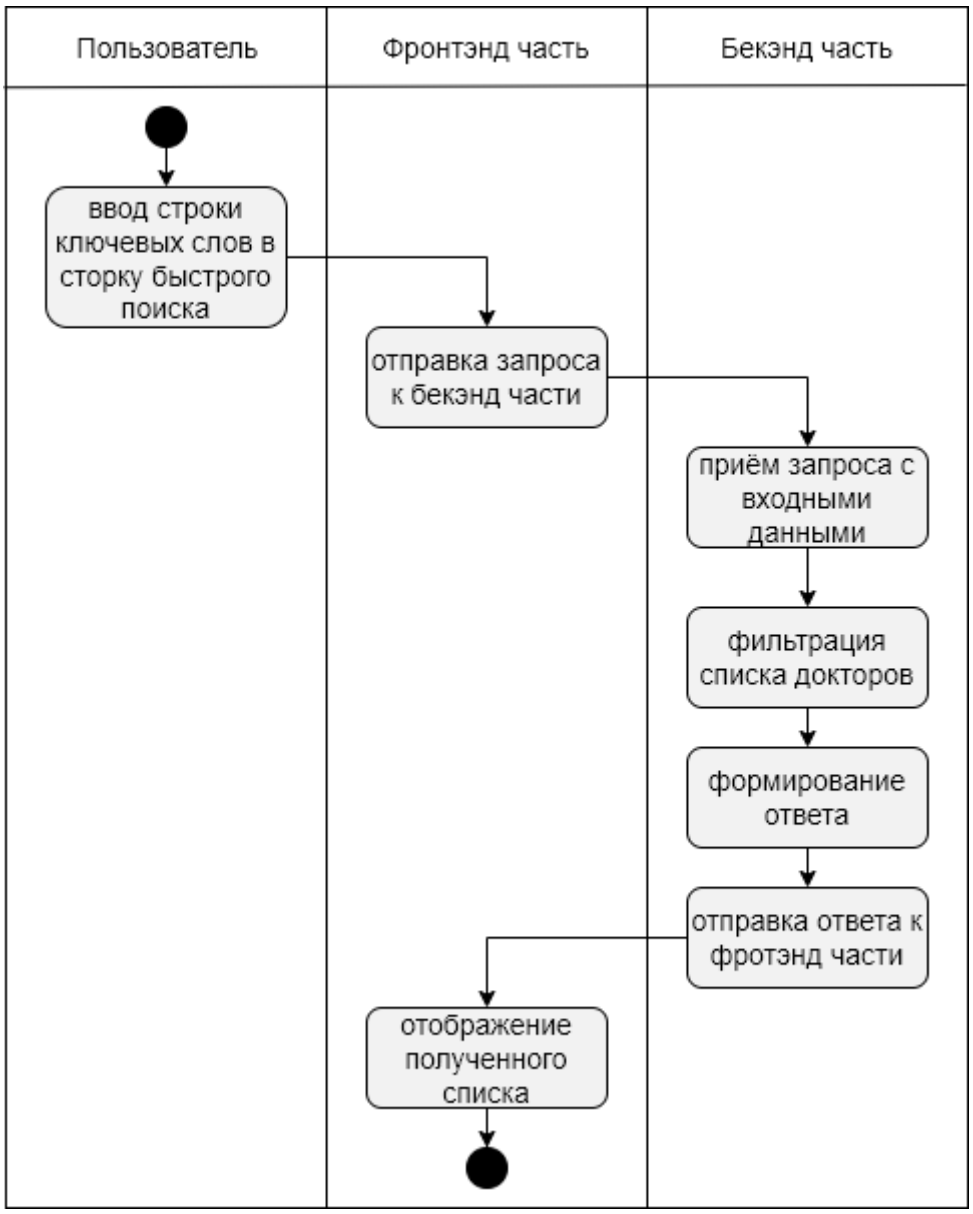


Рисунок 4.2 - Диаграмма активности выполнения кейса «быстрый» поиск докторов

Выходными данными является результат регистрации нового пользователя, если регистрация прошла неудачно, то результат будет содержать список ошибок, из-за которых это произошло.

При успешной регистрации пользователь переносится на страницу, которая содержит сообщение что необходимо подтвердить почту, перейдя по ссылке в письме, после чего пользователь может успешно войти в систему (рисунок 4.3).

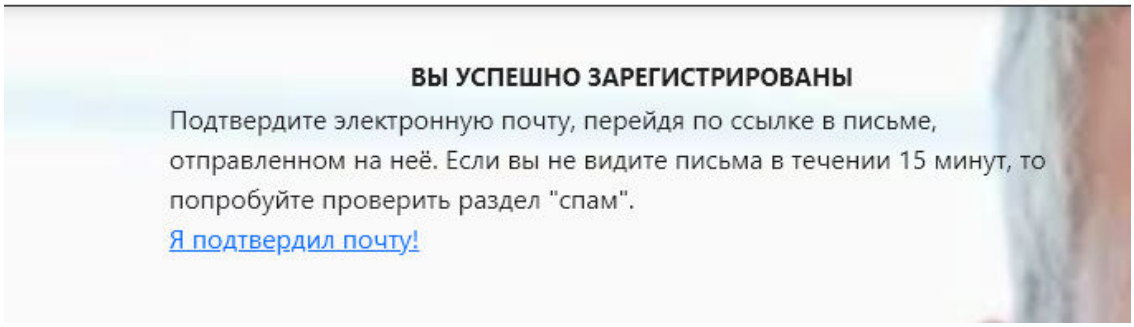


Рисунок 4.3 - Страница, сообщающая о необходимости подтверждения почты

Листинг кода, отвечающего за регистрацию нового пользователя и отправку сообщения для подтверждения почты приведён в приложении А. Диаграмма активности выполнения данного кейса приведена ниже (рисунок 4.4).

3. Получение доступного времени для посещения на дату для доктора.

При регистрации нового запроса на посещения пользователь может выбрать предпочтительную дату, после чего будет выведен список доступного на данную дату времени (рисунок 4.5).

Входными данными являются дата, выбранная пациентом, а также идентификатор доктора в базе данных.

Выходными данными является список доступного на данную дату времени.

Проверка на то, подходит ли определённое время в этот день проходит по многим параметрам, например, учитываются специфические даты

доктора(выходной, больничный, отпуск и т. д.), учитывается находится ли данная дата или время в прошлом, учитывается является ли данный день рабочим для доктора и т. д. Листинг кода, отвечающего за вычисление подходящего времени приведён в приложении А.

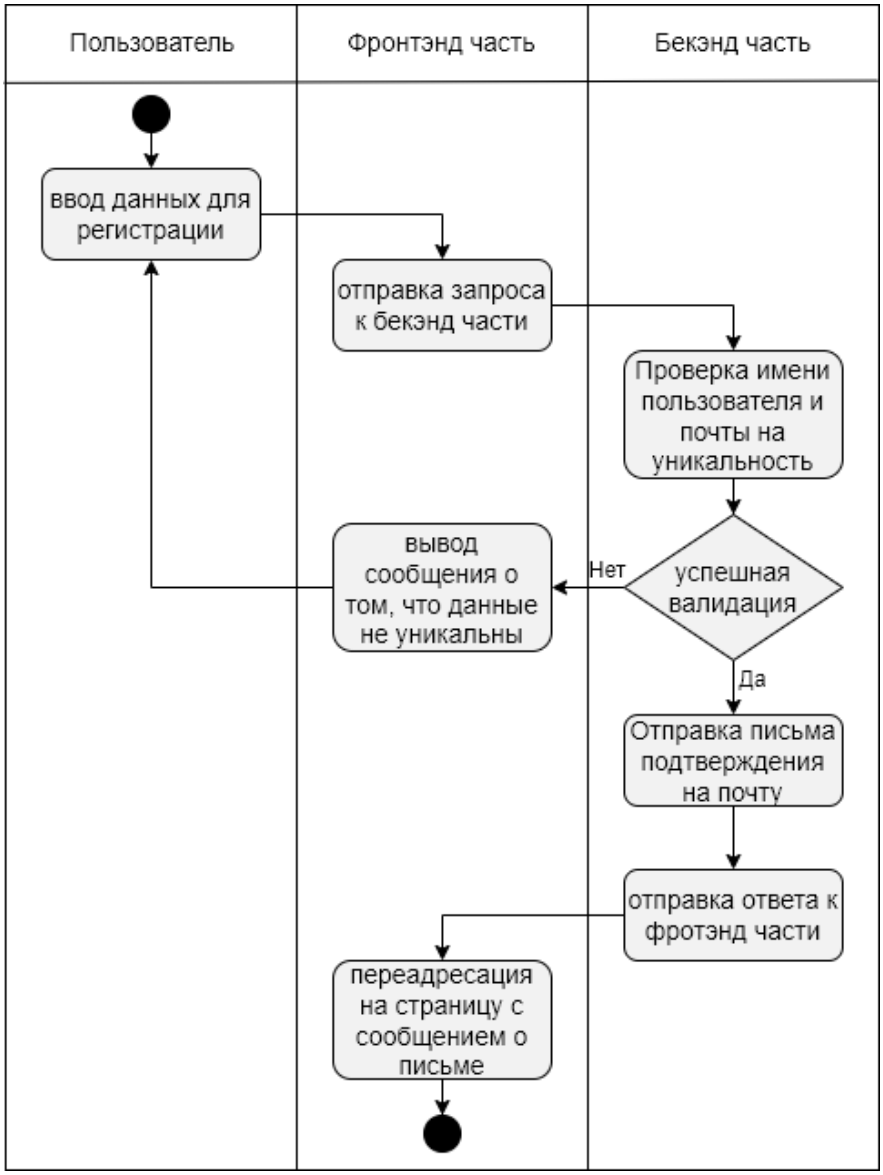


Рисунок 4.4 - Диаграмма активности выполнения кейса подтверждение почты для нового пользователя

Диаграмма активности выполнения данного кейса приведена ниже (рисунок 4.6).

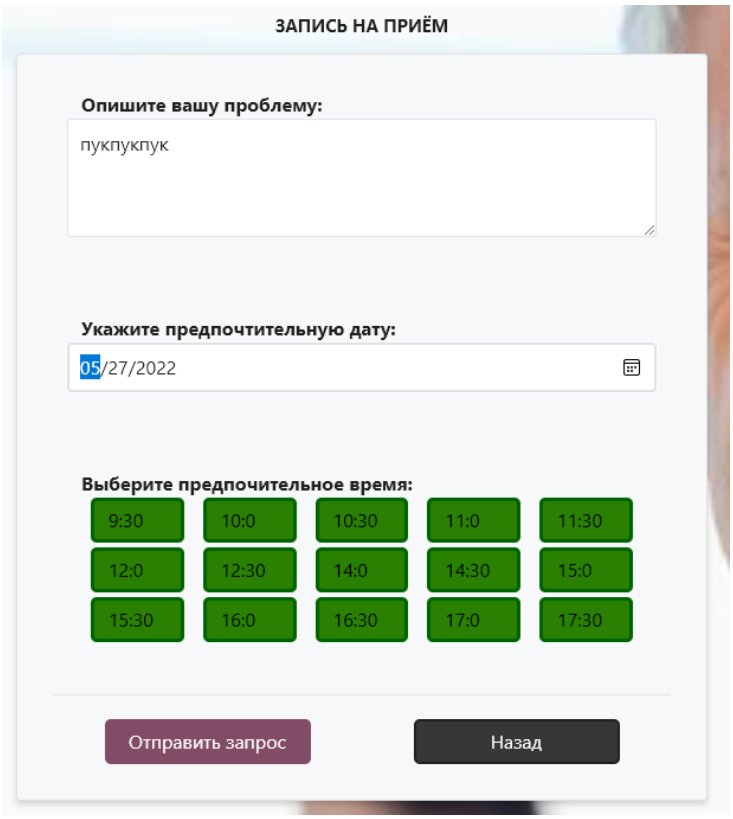


Рисунок 4.5 - Форма оформления посещения, содержащая список доступного на выбранную дату времени

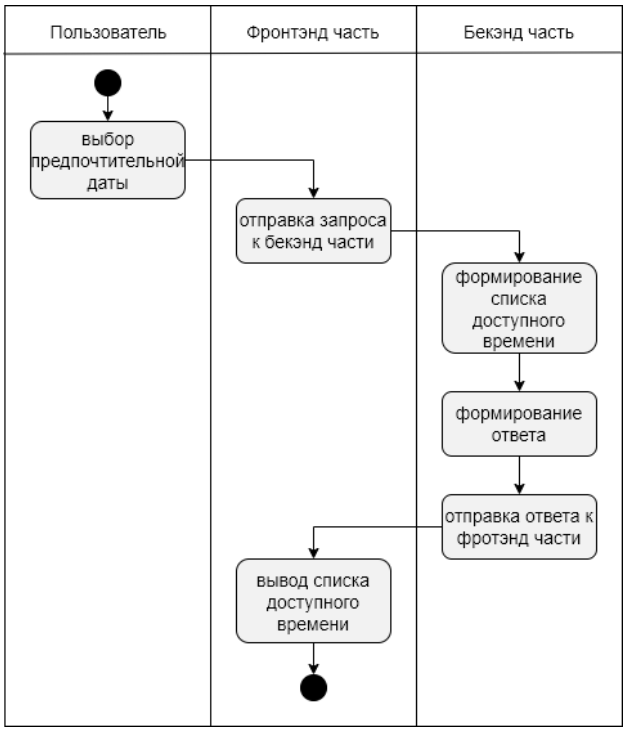


Рисунок 4.6 - Диаграмма активности для выполнения кейса получение доступного времени для посещения на дату для доктора

4.3 Модульное тестирование

Модульное тестирование было проведено для бекэнд части. Для написания методов контроллеров бекэнд приложения была использована библиотека MediatR.

MediatR - это небольшая простая библиотека, которая позволяет обрабатывать сообщения в памяти как команды, применяя декораторы или расширения функциональности. Использование шаблона медиатора помогает уменьшить количество связей и изолировать функциональность, связанную с запрашиваемой работой. При этом вы можете автоматически подключаться к обработчику, который выполняет эту работу — в данном случае к обработчику команд. Ещё одна причина для использования шаблона медиатора была раскрыта Джимми Богардом, создателем данной библиотеки: "Я думаю, здесь стоит упомянуть тестирование — вы получаете ясное и согласованное представление о поведении системы". Запрос поступает, ответ выдаётся — этот принцип оказался очень полезным при разработке согласованно работающих тестов [18].

Таким образом приложение содержит набор команд MediatR’a (рисунок 4.7), которые просто вызываются в методах контроллера, это позволяет не перегружать контроллеры и оставлять их код компактным и легко читаемым. Также это позволяет упростить процесс модульного тестирования, тестировать можно не отдельные методы репозиториев или методы бизнес-логики, а команды MediatR’a и таким образом покрыть тестами большую часть кода программы.

На основе всего вышесказанного были созданы модульные тесты, минимум по одному тесту на практически каждую команду MediatR’a.

Для создания тестов бекэнд части была использована библиотека xUnit. xUnit - это бесплатный инструмент тестирования с открытым исходным кодом для .NET, который разработчики используют для написания тестов для

своих приложений. По сути, это среда тестирования, которая предоставляет набор атрибутов и методов, которые мы можем использовать для написания тестового кода для наших приложений. Вот некоторые из этих атрибутов, которые можно использовать при работе с xUnit:

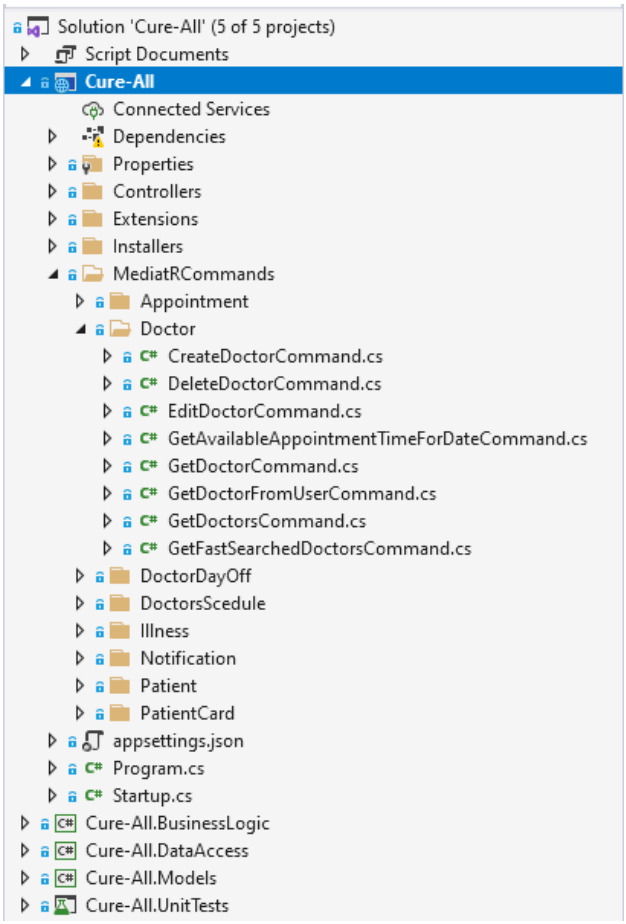


Рисунок 4.7 - Структура расположения команд MediatR’a в проекте

- [Fact] - атрибут указывает, что метод должен выполняться исполнителем теста;

- [Theory] - атрибут подразумевает, что мы собираемся отправить некоторые параметры в наш тестовый код. Таким образом, он похож на атрибут [Fact], потому что он утверждает, что метод должен выполняться исполнителем теста, но дополнительно подразумевает, что мы собираемся отправить параметры методу тестирования;

- [InlineData] - атрибут предоставляет те параметры, которые мы отправляем методу тестирования. Если мы используем атрибут [Theory], мы также должны использовать [InlineData] [19].

Файлы модульных тестов размещаются в отдельном проекте, Cure-All.UnitTests (рисунок 4.8).

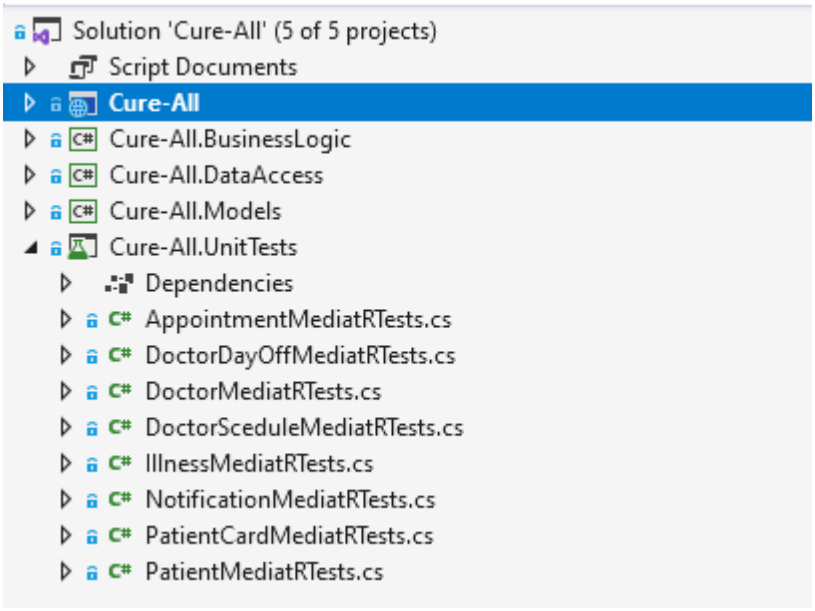


Рисунок 4.8 - Структура проекта, содержащего модульные тесты

Были созданы файлы для каждой из папок в директории MediatRCommands (рисунок 3.7), по одному файлу тестов для каждой папки. Названия файлов тестов формировались по правилу: название папки с командами MediatR'a + MediatRTests, что показывает что в данном файле происходит тестирование команд MediatR'a.

Так как тесты должны выполняться периодически, они не могут затрагивать данные из базы данных, таким образом, для тестирования нужно использовать «заглушку» для методов репозиториев, как будто они действительно возвращают некоторые данные. В качестве такой «заглушки» в проекте выступает фреймворк Moq.

Moq – это простой и легковесный изоляционный фреймворк (Isolation Framework), который построен на основе анонимных методов и деревьев

выражений. Для создания моков он использует кодогенерацию, поэтому позволяет «мокать» интерфейсы, виртуальные методы (и даже защищенные методы) и не позволяет «мокать» невиртуальные и статические методы [20].

Рассмотрим некоторые тесты для ознакомления с ними:

- GetDoctorsCommand_ShouldReturnAllDoctors - тест проверяет команду GetDoctorsCommand, которая должна вернуть всех докторов, которые были заранее указаны при помощи фреймворка Moq. Листинг кода теста приведён в приложении А;

- GetDoctorCommand_ShouldReturnNull - тест проверяет команду GetDoctorCommand, которая должно вернуть пустую ссылку на объект, так как в тесте намеренно указан неверный идентификатор доктора. Листинг кода теста приведён в приложении А;

- CreateDoctorCommand_ShouldCreateNewDoctor - тест проверяет команду CreateDoctorCommand, которая должна создать новый объект и успешно зарегистрировать его путём назначения идентификатора для него. Листинг кода теста приведён в приложении А.

В качестве аргумента в пользу обязательного покрытия кода тестами можно привести пример провала одного из тестов данного проекта, что позволило выявить ошибку.

Тест GetDoctorsCommand_ShouldReturnAllDoctors при первом запуске завершился провалом, причиной послужило то, что в команде MediatR’a, которая тестируется в данном тесте, при написании кода были указаны стандартные параметры фильтрации. Данная проблема была быстро решена после выявления путём очистки стандартных параметров фильтрации.

Результаты выполнения модульных тестов приведены ниже (рисунок 4.9).

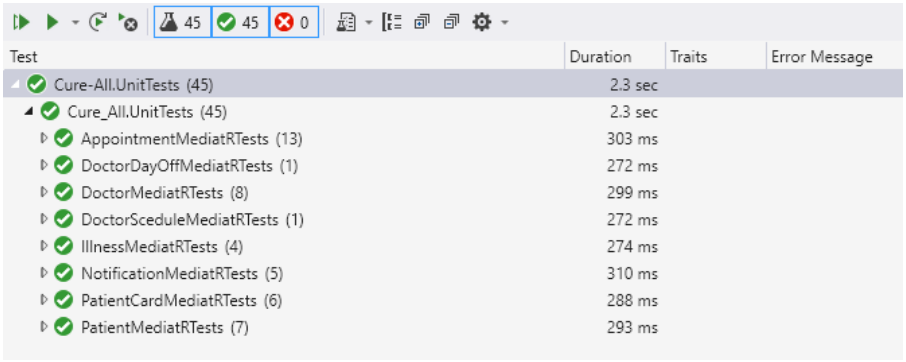


Рисунок 4.9 - Результаты выполнения модульных тестов

Таким образом, написав всего 45 тестов, тестами был покрыт практически весь код бекэнд части. Выполнение всех тестов в итоге завершилось успехом.

4.4 Функциональное тестирование

Функциональное тестирование было проведено над фронтэнд частью так как весь функционал приложения сосредоточен именно здесь. Исходя из того, что на текущий момент теряет актуальность, для функционального тестирования фронтэнд части был использован фреймворк Cypress.

Cypress - это open-source фреймфорк для E2E тестирования. Это также, как и Puppeteer относительно молодой инструмент, однако он вносит новые концепции и решения в способы осуществления автоматизации и тестирования. Ключевой особенностью, Cypress является то, что он исполняется внутри самого браузера. Это в том числе означает, что Cypress всегда отслеживает моменты вызова всякого рода событий в браузере и никогда не упустит любые манипуляции с элементами страницы, что намного уменьшает вероятность появления floating тестов.

Достоинства:

- встроенный набор инструментов для тестирования построенный на форке mocha, chai, sinon;

- встроенный механизм автоматического ожидания, это, собственно, означает, что при написании сценарием нет необходимости писать async/await функции как это делается в Puppeteer и Selenium. Cypress сам подождёт когда появится нужный элемент, подождёт когда закончится анимация, и подождёт когда очередной сетевой запрос завершится;

- time machine фича, которая позволяет в Cypress test runner откатываться на определённые шаги в последовательности выполнения теста;

- исчерпывающая документация с большим набором примеров;

- возможность написания в том числе и unit тестов [21].

Из-за невозможности предсказать состояние базы данных на момент запуска теста автоматически был протестирован лишь некоторый функционал. Таким образом были протестированы, например, появление сообщений об ошибке валидации при вводе неверных данных в формы входа в систему и регистрации нового пользователя, а также правильность перехода между страницами при помощи панели навигации и заголовка сайта. Из-за того, что запуск автоматизированных тестов не должен в итоге изменять данные в базе данных стало невозможным автоматически протестировать, например, регистрацию нового пользователя или оформление нового посещения к доктору.

Список созданных для Cypress тестов представлен ниже (рисунок 4.10).

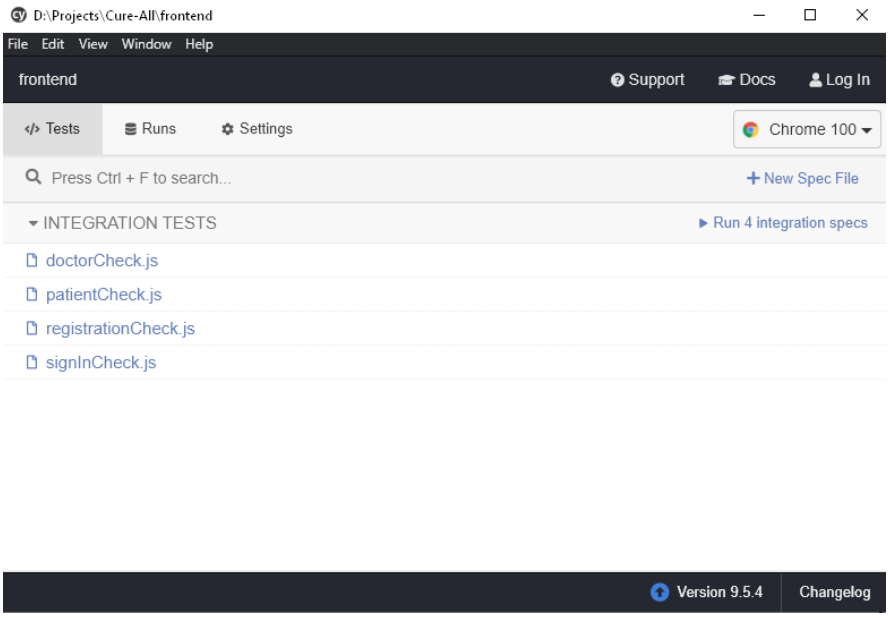
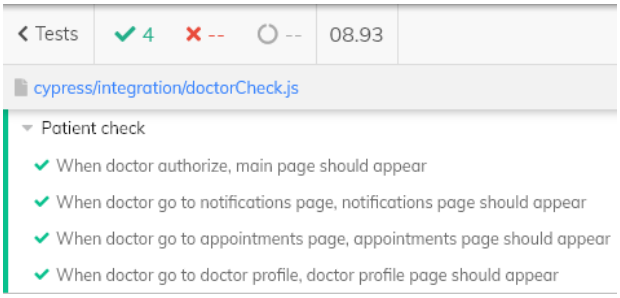
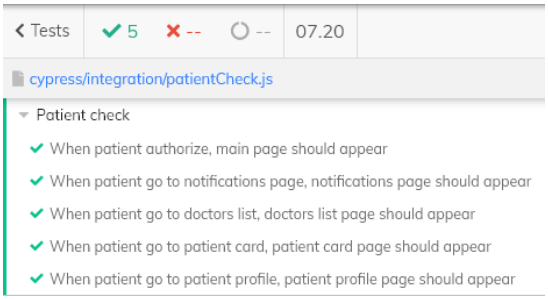


Рисунок 4.10 - Список автоматизированных функциональных тестов

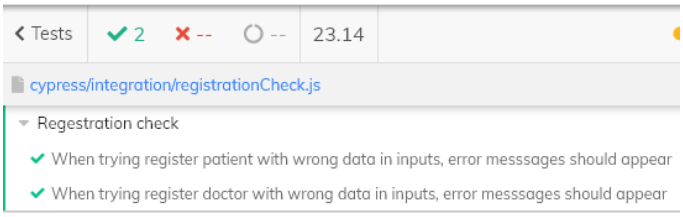
Результаты запуска автоматизированных тестов представлены ниже (рисунок 4.11 а-г).



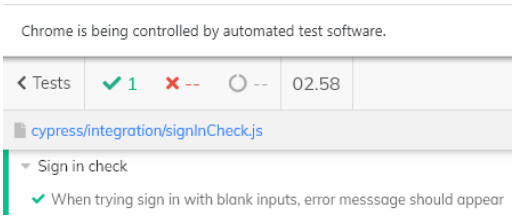
а



б



в



г

Рисунок 4.11 - Результаты автоматических тестов: а - результаты теста doctorCheck; б - результаты теста patientCheck; в - результаты теста registrationCheck; г - результаты теста signInCheck

Для ознакомления с кодом функциональных тестов рассмотрим некоторые тесты:

- When doctor go to notifications page, notifications page should appear - тест проверяет что при переходе на страницу со списком уведомлений доктор действительно видит страницу уведомлений. Листинг данного теста приведён в приложении А;

- When trying sign in with blank inputs, error messsage should appear - тест проверяет что при попытке входа в систему с неверными данными формы, появляются ошибки валидации. Листинг данного теста приведён в приложении А;

- When trying register doctor with wrong data in inputs, error messsages should appear - тест проверяет что при попытке регистрации нового доктора используя неверные данные будут появляться ошибки валидации. Листинг данного теста приведён в приложении А.

Остальной функционал приложения был протестирован вручную. Чек-лист тестирования приведён ниже (таблица 4.1).

Таблица 4.1 - Чек-лист тестирования ИС «Веб-сервис для записи к врачу»

Тестируемый модуль	**ИД кейса**	**Тестируемое требование**	**Результат**
1	**2**	**3**	**4**
Аутентификация	ТС-1	Регистрация нового доктора	Выполнено успешно
	ТС-2	Регистрация нового пациента	Выполнено успешно
	ТС-3	Вход в систему под различными пользователями	Выполнено успешно
	ТС-4	Редактирование профиля пользователя	Выполнено успешно

Окончание таблицы 4.1

1	2	3	4
	ТС-5	Удаление профиля пользователя	Выполнено успешно
Посещения	ТС-6	Изменение посещения доктором	Выполнено успешно
	ТС-6	Оформление нового посещения к доктору	Выполнено успешно
	ТС-8	Отклонение нового посещения доктором	Выполнено успешно
	ТС-9	Принятие нового посещения доктором	Выполнено успешно
Список докторов	ТС-10	Фильтрация по имени	Выполнено успешно
	ТС-11	Фильтрация по специализации	Выполнено успешно
	ТС-12	Фильтрация по городу	Выполнено успешно
	ТС-13	Фильтрация по стране	Выполнено успешно
	ТС-14	Фильтрация по опыту	Выполнено успешно
	ТС-15	Сортировка по опыту	Выполнено успешно

Ниже представлена группа тест-кейсов (таблица 4.2), содержащая по одному тест-кейсу из каждого модуля приведённого выше чек-листа (таблица 4.1).

Таблица 4.2 - Тест-кейсы тестирования ИС «Веб-сервис для записи к врачу»

ИД кейса	Название	№ шага	Действие	Данные	Ожидаемый результат
1	2	3	4	5	6
ТС-2	Регистрация нового пациента	1	Переход на страницу регистрации		
		2	Выбор кнопки «Пациент»		
		3	Ввод данных для регистрации	Имя - Пациент; Фамилия - Пациент; Имя пользователя - NewTestPatient; Эл. почта - newTestPatient@gmail.com; Дата рождения - 06.05.2022; Номер телефона - +375296105401; Почтовый код - 12345; Страна - Беларусь; Город - Витебск; Пароль - NewPassword123!; Подтверждение пароля - NewPassword123!.	

Продолжение таблицы 4.2

1	2	3	4	5	6
		4	Нажатие на кнопку подтверждения		Переход на страницу с сообщением о том, что необходимо подтвердить почту
		5	Переход по ссылке из письма на почте		Переход на страницу с успешным подтверждением почты
		6	Попытка входа в систему		Успешный переход на главную страницу
ТС-6	Оформление нового посещения к доктору	1	Вход в систему под любым пациентом		
		2	Переход к списку докторов		
		3	Выбор любого доктора из списка		
		4	Нажатие на кнопку «Записаться на приём»		

Окончание таблицы 4.2

1	2	3	4	5	6
		5	Заполнение формы для нового посещения	Проблема - Тест проблема; Предпочтительноя дата - 06.05.2022; Предпочтительное время - 12:00.	
		6	Нажатие на кнопку подтверждения		Успешное оформление посещения
ТС-11	Фильтрация по специализации	1	Вход в систему под любым пациентом		
		2	Переход к списку докторов		
		3	Выбор специализации для фильтрации списка	Специализация - Невролог	
		4	Нажатие на кнопку подтверждения		Получение списка докторов с выбранной специализацией

Таким образом тестами был покрыт практически весь функционал приложения. Все тесты в итоге завершились успешно.

4.5 Прочие виды тестирования ПС

4.5.1 Тестирование безопасности

Тестирование безопасности - это тип тестирования программного обеспечения, который выявляет уязвимости, угрозы, риски в программном приложении и предотвращает атаки от злоумышленников. Целью тестов безопасности является выявление всех возможных лазеек и слабых мест в программной системе, которые могут привести к потере информации, доходов, репутации со стороны сотрудников или посторонних лиц Организации. Целью тестирования безопасности является выявление угроз в системе и оценка её потенциальных уязвимостей, чтобы система не перестала функционировать или использовалась. Это также помогает в обнаружении всех возможных угроз безопасности в системе и помогает разработчикам в устранении этих проблем посредством кодирования [22].

За регистрацию и авторизацию пользователей в системе отвечает ASP.NET Identity, поэтому рассмотрим вопросы безопасности данной системы.

Платформа ASP.NET Identity полностью интегрирована в ASP.NET. Это означает, что вы можете использовать стандартные средства аутентификации/авторизации MVC вместе с Identity, такие как атрибут Authorize. Атрибут Authorize по умолчанию ограничивает доступ к методам действий для пользователей, не прошедших аутентификацию. Пользователь может зарегистрировать новый аккаунт, информация о котором будет, чаще всего, храниться в базе данных, в данном случае MS SQL Server. ASP.NET Identity поддерживает двухфакторную аутентификацию, когда пользователь для аутентификации должен ввести что-то дополнительное. Наиболее распространёнными примерами является ввод закрытого токена Secureid или код проверки подлинности, который может быть отправлен на email-адрес или по СМС [23].

По умолчанию в ASP.NET Identity невозможно зарегистрировать двух пользователей, использующих одинаковые имена пользователя или адреса почты, что предотвращает огромное количество ошибок, как, например, невозможность определить то, какой именно пользователь пытается войти в систему из-за того, что логин, являющийся именем пользователя или адресом электронной почты, присвоен нескольким пользователям.

Также для обеспечения дополнительной безопасности в приложении реализована система подтверждения почты, без подтверждённой почты пользователь не сможет войти в систему. Также реализована система смены пароля пользователя если он его забыл, письмо с ссылкой на страницу смены пароля приходит на почту пользователя.

4.5.2 Нагрузочное тестирование

Результаты нагрузочного тестирования приложения в большей степени зависят от того, насколько быстро осуществляются запросы к базе данных. В данном приложении в роли СУБД выступает Microsoft SQL Server, таким образом можно рассмотреть результаты мониторинга работы базы данных (рисунок 4.12).

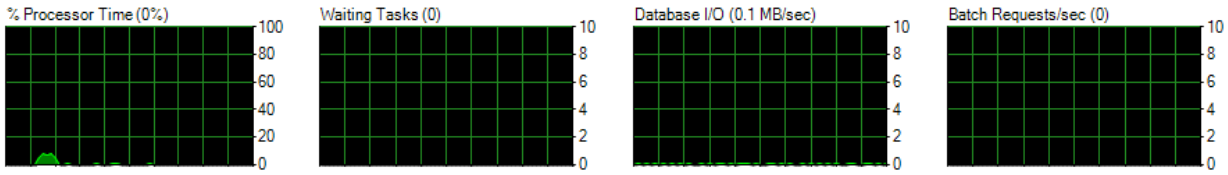


Рисунок 4.12 - Результаты мониторинга работы базы данных

На рисунке 4.12 представлены результаты работы одного пользователя с БД, бесплатная версия Microsoft SQL Server не позволяет смоделировать ситуацию работы с БД нескольких человек, но, можно предположить, что нагрузочные результаты будут расти примерно с той же скоростью с которой будет расти количество пользователей.

4.5.3 Объёмное тестирование

Для объёмного тестирования в базу данных были добавлены 100 докторов, результаты мониторинга работы базы данных при увеличенном количестве данных приведены ниже (рисунок 4.13).

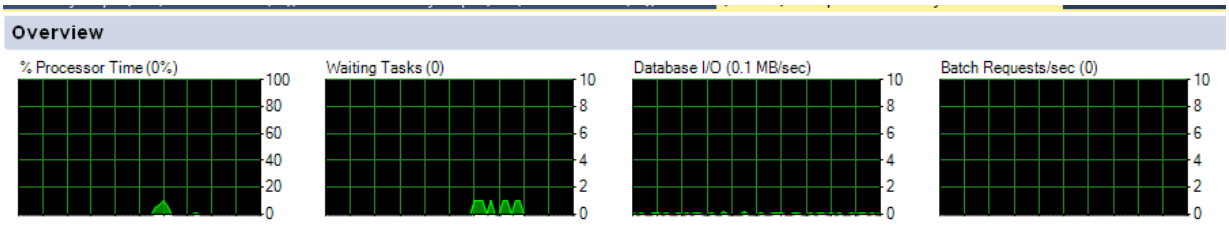


Рисунок 4.13 - Результаты мониторинга работы базы данных с увеличенным количеством данных

Таким образом, сравнивая результаты работы с базой данных с небольшим количеством данных (рисунок 4.12) с увеличенным количеством данных (рисунок 4.13), можно заметить, что появляются задачи в режиме ожидания, то есть, увеличивается время отклика на запрос.