Общеобразовательная автономная некоммерческая организация

«Школа «ЛЕТОВО»

Автоматизация сбора и обработки данных медицинских клинических исследований

**Отчет о работе над персональным проектом**

Выполнил ученик 9 класса

Коптев Дмитрий

Супервайзер

Гуровиц Владимир Михайлович

Количество слов

3442

Содержание

[Описание проблемы и обзор существующих методов решения 3](#_Toc67516075)

[Проблема 3](#_Toc67516076)

[Существующие способы решения 3](#_Toc67516077)

[Планирование 3](#_Toc67516078)

[Сроки и этапы работы над проектом 3](#_Toc67516079)

[Цель проекта 4](#_Toc67516080)

[Глобальный контекст 4](#_Toc67516081)

[Требования к продукту 4](#_Toc67516082)

[Бизнес-требования 4](#_Toc67516083)

[Основные роли пользователей 4](#_Toc67516084)

[Функциональные требования 4](#_Toc67516085)

[Сценарии использования (use cases) для роли «участник» 4](#_Toc67516086)

[Сценарии использования для роли «исследователь» 4](#_Toc67516087)

[Нефункциональные требования 5](#_Toc67516088)

[Интервьюирование целевой аудитории 5](#_Toc67516089)

[Архитектура продукта 6](#_Toc67516090)

[Диаграмма компонентов 6](#_Toc67516091)

[Язык программирования 7](#_Toc67516092)

[База данных 8](#_Toc67516093)

[Технология доступа к данным 8](#_Toc67516094)

[Веб-сервис 8](#_Toc67516095)

[API 9](#_Toc67516096)

[Безопасность 10](#_Toc67516097)

[Аутентификация 10](#_Toc67516098)

[Авторизация 11](#_Toc67516099)

[Логирование 12](#_Toc67516100)

[Веб-сайт 13](#_Toc67516101)

[Мобильное приложение 17](#_Toc67516102)

[Диаграмма классов 20](#_Toc67516103)

[Тестирование 21](#_Toc67516104)

[Юнит тесты 21](#_Toc67516105)

[Ручное тестирование 22](#_Toc67516106)

[Нагрузочное тестирование 23](#_Toc67516107)

[Результаты работы 23](#_Toc67516108)

[Приложения 24](#_Toc67516109)

[Выдержки из дневника работы над проектом 24](#_Toc67516110)

[Библиография 26](#_Toc67516111)

# Описание проблемы и обзор существующих методов решения

## Проблема

Во время проведения III-IV фазы клинических исследований препаратов и тестировании новых схем лечения необходимо регулярно осуществлять сбор, хранение и обработку медицинских показателей пациентов (ЧСС, давление, температура и т.д.) и данных о субъективном состоянии здоровья.

## Существующие способы решения

1. Сбор данных на бумажных носителях от пациента к врачу во время очных приемов, затем к исследователю.

*Основные недостатки*: трудоемкость, неэффективность сбора, отсутствие оперативной обратной связи, существует риск возникновения ошибки при переносе данных.

1. Использование распространенных сервисов составления опросов: Google Forms, SurveyMonkey, Survio, Яндекс Формы и т.п.

*Основные недостатки*: требуется подключение к интернету, слабые возможности систематизации и анализа данных, отсутствие единого стандартизированного формата сбора информации.

1. Использование программных комплексов ePRO (electronic patient-reported outcome - электронный результат, сообщаемый пациентом). Например, электронные дневники: Data Matrix, PROMIS, cPRO, Adaptest, дневник самонаблюдения на Госуслугах (использовался при тестировании вакцины Спутник V).

*Основные недостатки*: ориентированность на крупные и средние фармацевтические компании, высокая стоимость, отсутствие мобильного приложения пациента (кроме Data Matrix).

# Планирование

## Сроки и этапы работы над проектом

Сроки для каждого этапа распланированы в соответствии со сложностью

|  |  |
| --- | --- |
| Этапы: | Сроки: |
| Описание проблемы и обзор существующих методов решения | Выполнить до 12.11.2020 |
| Выбор используемых технологий | Выполнить до 24.11.2020 |
| Proof of concept – создание прототипа решения и оценка применимости выбранных технологий | Выполнить до 06.12.2020 |
| Формирование требований к продукту | Выполнить до 18.12.2020 |
| Разработка архитектуры продукта | Выполнить до 30.12.2020 |
| Реализация приложений | Выполнить до 26.01.2021 |
| Тестирование продукта | Выполнить до 24.02.2021 |
| Анализ результатов и выбор дальнейших направлений развития продукта | Выполнить до 24.03.2021 |

Цель проекта

Разработать программный комплекс позволяющий автоматизировать сбор и обработку данных клинических исследований

Глобальный контекст

Глобальный контекст проекта **- научно-технические инновации**, т.к. продукт проекта – распределенное приложение

# Требования к продукту

## Бизнес-требования

В отличие от уже существующих систем ePRO разрабатываемый продукт будет полностью бесплатен и ориентирован на широкую целевую аудиторию: индивидуальные исследователи (при составлении новых схем лечения), медицинские научные центры и НИИ, фармацевтические компании.

Также отличительными особенностями продукта будут являться: удобство использования за счёт наличия мобильного приложения пациента и веб-приложения исследователя и специализированные средства анализа данных.

В будущем планируется монетизация в виде предоставления премиальных функций, осуществления платной интеграции с существующими системами организаций, кастомизация функционала, реклама.

В первой версии (MVP – minimal viable product) продукта будет реализован базовый функционал без возможности настройки форм ввода исследователем.

## Основные роли пользователей

* Исследователь (Researcher)
* Участник исследования (Participant)

## Функциональные требования

### Сценарии использования (use cases) для роли «участник»

* Регистрация в приложении по номеру телефона
* Ввод фиксированного набора физиологических показателей в мобильном приложении: температура, ЧСС, давление, имеющиеся симптомы из списка (головная боль, головокружение, слабость, тошнота, кашель, насморк), субъективная оценка самочувствия по 10-бальной шкале

### Сценарии использования для роли «исследователь»

* Регистрация на сайте – выбор логина и пароля
* Возможность войти в личный кабинет на сайте путем ввода логина и пароля
* Возможность выйти из личного кабинета
* Регистрация пациента: ФИО, номер телефона, дата рождения, антропометрические показатели (вес, рост)
* Просмотр списка пациентов – ФИО, дата рождения, кол-во отправленных отчетов, дата последней активности
* Просмотр статистики физиологических показателей пациента
* Просмотр сводной статистики физиологических показателей пациентов
* Возможность скачать текущую статистику показателей пациентов в формате csv

## Нефункциональные требования

* Интуитивно понятный дизайн приложений
* Время отклика веб-приложения не более 1 секунды
* Логирование программных исключений
* Поддержка интеграции с внешними системами, за счёт предоставления API (интерфейса программного доступа) и его спецификации на базе Swagger (OpenAPI)

# Интервьюирование целевой аудитории

Интервьюируемым была предоставлена информация по разрабатываемому программному комплексу и предполагаемым сценарием использования с целью получения отзыва. Исходный вариант документа предполагал возможность использования разрабатываемого ПО в широкой врачебной практике, но по результатам отзывов проект был скорректирован в более узком направлении систем ePRO.

Врач 1-ой городской больницы г. Москва.

Приложение интересно во врачебной практике только в достаточно узкой области проведения исследований, где требуется собрать некоторую статистику на достаточно большом количестве пациентов. Для более широкого использования требуется его интеграция с существующими системами электронных карт пациентов. Также был бы интересен функционал уведомлений о времени приема назначенных лекарств. В случае с фармацевтическими компаниями, такая программа являлась бы удобным инструментом для сбора информации при проведении клинических исследований.

IT Service Delivery Manager Global Medical Affairs, Novartis, Swiss

Описанный продукт является комбинацией трех классов приложений:

* ePRO – для сбора информации, класс ПО использующийся пациентами (Participants) и исследователями (CRA - Clinical Research Associate)
* CTMS (Clinical Trial Management System) – системы обработки накопленной информации, используется исследователями, интегрируется с системами ePRO
* Hospital/Practice Management System – система управления клиникой, включающая в себя электронные карты пациентов, опционально системы телемедицины с интерфейсами взаимодействия врач-пациент.

Каждая из систем является сложным программным комплексом, слишком широкий выбор сценариев для реализации будет являться негативным фактором, я бы рекомендовал сконцентрироваться на реализации системы ePRO, тем более что в РФ активно продвигается курс на импортозамещение в том числе для информационных систем. В плане функционала, принципиальный момент — это обеспечение невозможности модификации собранной информации, а также возможность внешней проверки ее целостности, и конечно соответствие 152 ФЗ о персональных данных.

# Архитектура продукта

## Диаграмма компонентов



Программный комплекс построен по многоуровневой архитектуре. Вся бизнес-логика, аутентификация, и работа с базой данной находится в веб-сервисе, развернутом в публичном облаке. API предоставляемое этим сервисом должно быть доступно через интернет. Помимо веб-сервиса в облаке расположена база данных и сам веб-сайт. В соответствии с двумя пользовательскими ролями предполагается использование двух типов клиентов: мобильное приложение для участников испытаний и веб-сайт для исследователей. Оба приложения используют общую библиотеку доступа API Client, сгенерированную по спецификации.

## Язык программирования

При выборе языка программирования (ЯП) основным критериями являлись:

* Простота синтаксиса
* Возможность реализации всех компонент программного комплекса на выбранном ЯП
* Кроссплатформенность
* Широкий выбор технологий и фреймворков
* Распространённость ЯП и возможность получения помощи от сообщества программистов
* Наличие подробной документации, примеров и доступных книг

Для проекта был выбран язык C# на кроссплатформенном фреймворке .NET Core 3.1.

Синтаксис языка близок к C++, что облегчило его изучение.

Огромным плюсом являлось то, что все компоненты включая мобильное приложение получилось разработать, используя только C#.

## База данных

Для хранения и работы с данными используется реляционная база данных Microsoft SQL Server. Большинство облачных провайдеров предоставляет услуги хостинга MS SQL. Выбор решения от Microsoft обеспечит лучшую совместимость с остальными компонентами программного комплекса.

## Технология доступа к данным

Для удобства работы с БД из программного кода, обычно используются фреймворки ORM (object-relational mapping). Microsoft имеет свой фреймворк Entity Framework (EF) обеспечивающий встроенную интеграцию с MS SQL.

При использовании подхода EF Code First, сперва проектируются классы данных (модели), затем с помощью команды Add-Migration автоматически формируется код, создающий необходимую БД, таблицы, поля, и связи между таблицами.

Можно настроить программу таким образом что при ее запуске EF автоматически запустит миграцию если БД еще не создана.

При внесении изменений в классы модели создаются последующие миграции, представляющие из себя последовательный набор команд для SQL. При старте приложения будут запускаться только те миграции, версии которых еще не были применены на данном экземпляре БД.

В целом такой подход упрощает и ускоряет разработку.

Вся непосредственная работа с данными из C# представляет из себя обычные вызовы LINQ на основе коллекций класса контекста базы данных.

## Веб-сервис

Для реализации веб-сервиса, использовался подход RESTful Web API с использованием формата JSON для сериализации данных. Данный выбор обусловлен широким распространением данного подхода, клиентами такого REST сервиса могут быть приложения, написанные практически на любом языке, мобильные приложения и даже веб-сайты на JavaScript.

Отличительной особенностью REST запросов является использование HTTP(S) GET, POST, PUT, DELETE методов соответственно для получения информации, создания сущностей, редактирования сущностей и удаления сущностей (CRUD – Create Read Update Delete). Для методов обработки (специфичными операциями над сущностями) обычно используется POST методы.

Другим важным моментом является использование специфичных путей для запросов (routes) к методам API, общий принцип их конструирования: http://{domain.name}/{entity}/{action}/{id}

{entity} – например, это может быть Researcher, Participant, Response, Account

{action} – например, для Account, это TokenParticipant или TokenResearcher – соответствующие методам аутентификации участника и исследователя

{id} – необязательный параметр, обычно используется в PUT и DELETE методах

Непосредственно веб-сервис реализован на основе технологии ASP.NET Core. И может быть развернут как на Linux машинах, так и на Windows (с использованием IIS или как отдельный сервис).

## API

Для предоставления публичного доступа к API используется спецификация OpenAPI (Swagger).

Поддержка OpenAPI для веб-сервиса реализована с помощью nuget пакета NSwag. Необходимые обработчики NSwag подключены в Startup классе веб-сервиса, что позволяет его библиотекам перехватывать веб-запросы на определенные адреса http://{domain.name}/swagger и предоставлять веб-страницу с автоматически сгенерированной документацией на API (на основе анализа имеющихся классов контроллеров, их методов, параметров и возвращаемых значений). Любой внешний разработчик может воспользоваться такой документацией к этому API для интеграции со своими продуктами. Более того, swagger предоставляет возможности Try Out – вызывать описанные методы непосредственно из браузера, т.е. разработчик может пробовать вызовы методов с разными параметрами.

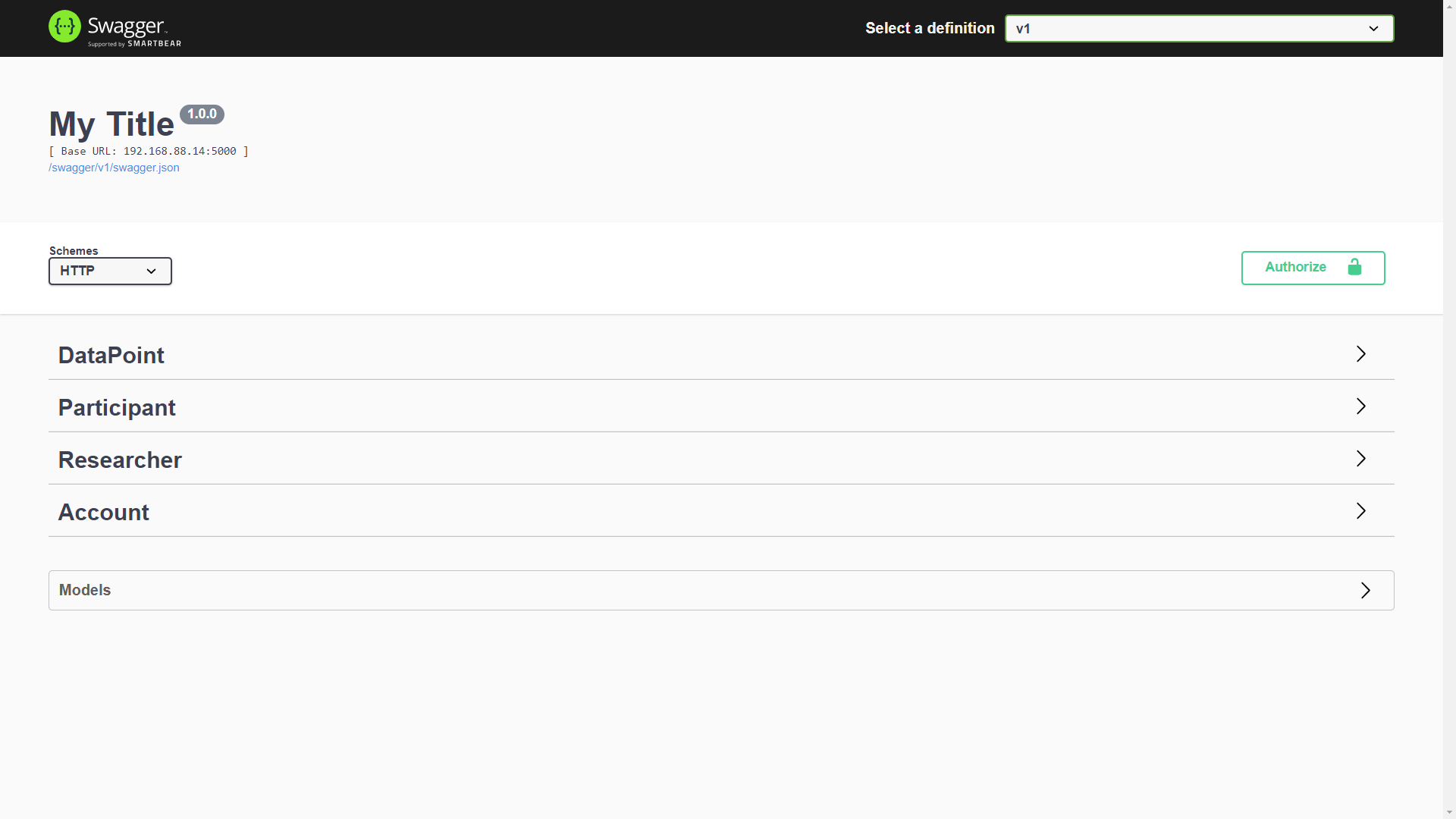
Само описание API это документ в формате json, его можно скачать, и используя его автоматически сгенерировать клиентский код для взаимодействия с сервисом под любой распространённый язык программирования: c#, java, php, golang, javascript, kotlin и т.д.

Для генерации кода клиента можно воспользоваться бесплатным онлайн приложением <https://app.swaggerhub.com/>, либо десктопным, например, NSwag Studio.

Клиентская библиотека для Medical.Client была сгенерирована с помощью NSwag Studio – предоставляющей большую гибкость в настройках чем онлайн приложение.

Особенностью Medical.Client является то, что он сгенерирован под платформу .net standard 2.0, т.к. библиотека используется как в веб-приложении, так и в мобильном приложении – которое не поддерживает .net core 3.1. По этой же причине невозможно было изначально вынести классы модели в отдельную сборку, т.к. некоторые ссылки на внешние модули необходимые для Entity Framework были недоступны для мобильного приложения.

Скриншот страницы swagger:



## Безопасность

Веб-сервис предоставляет публичное API и предназначен для одновременной работы многих пользователей разных ролей. Для обеспечения безопасности данных приложение должно обеспечивать два механизма: аутентификация и авторизация.

### Аутентификация

Аутентификация — это процедура проверки подлинности пользователя путём сравнения введённого им пароля (для указанного логина) с паролем, сохранённым в базе данных (либо обычно сравниваются хэши паролей).

Для реализации аутентификации в веб-сервисе используется nuget пакет Microsoft.AspNetCore.Authentication.JwtBearer. Сам механизм аутентификации основан на выдаче специально подписанной строки – токене.

JSON Web Token (JWT) — это JSON объект, который определен в открытом стандарте RFC 7519. Он считается одним из безопасных способов передачи информации между двумя участниками. Для его создания необходимо определить заголовок (header) с общей информацией по токену, например время его жизни, полезные данные (payload), такие как id пользователя, его имя и роль, и подписи (signature).

После успешной проверки подлинности пользователя на веб-сервисе генерируется токен (и соответственно сигнатура токена) основываясь на секретном ключе и payload информации. Токен хранится на клиенте (веб-сайте или мобильном приложении) и используется при необходимости авторизации какого-либо запроса. В реализации Bearer схемы токен передается от клиента к веб-сервису в заголовке HTTP запроса (Authorization: Bearer [token]). Таким образом в реализации вызываемых методов имеем возможность проверить корректность токена и идентифицировать пользователя, который вызвал эту операцию.

При попытке злоумышленником подменить данные в header или payload, токен станет некорректным, поскольку сигнатура не будет соответствовать изначальным значениям. А возможность сгенерировать новую сигнатуру у злоумышленника отсутствует, поскольку секретный ключ для формирования подписи находится на сервере.

Для упрощения аутентификации пользователей роли участник из мобильного приложения используется номер телефона как логин, а в качестве пароля автоматически генерируемый СМС код. На данном этапе проекта подключение СМС шлюза не планируется, аутентификация участников исследований происходит исключительно по номеру телефона при первом запуске мобильного приложения и далее сохраняется на устройстве. Токен участника имеет длительность жизни 1 год.

### Авторизация

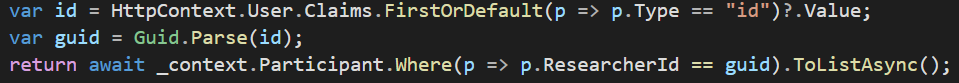
ASP.NET Core имеет встроенный механизм авторизации пользователей по ролям. Достаточно прописать атрибут [Authorization] перед методом для которого мы хотим добавить проверку прав доступа и перечислить названия ролей в конструкторе. Например, для роли исследователь, это выглядит так: [Authorization(“Researcher”)]

Таким образом методы доступные для роли Исследователь будет недоступны для вызова с использованием токена полученного при аутентификации пользователя роли Участник. В качестве ответа на такой запрос будет возвращаться HTTP код 403 Forbidden. Если токен не передается в заголовках запроса, или он не является валидным, то ASP.NET Core выдаст код 401 Unauthorized.

Все методы API защищены атрибутом Authorization за исключением двух методов аутентификации (для участников и исследователей) – для получения токена, и метода регистрации нового исследователя – для того, чтобы дать возможность исследователю самостоятельно регистрироваться используя веб-сайт.

Второй важный аспект авторизации относится к разделению данных между различными исследователями. Очевидно, что ни один исследователь не должен иметь доступ к данным участников другого исследователя. Для этих целей в каждом методе API исследователя используя claims (payload) авторизованного пользователя (из токена) мы получаем Id исследователя и фильтруем данные из БД по полученному Id. Т.е. исследователь видит и работает с данными участников, которые связаны с ним.

Пример получения участников исследователя, вошедшего в аккаунт:



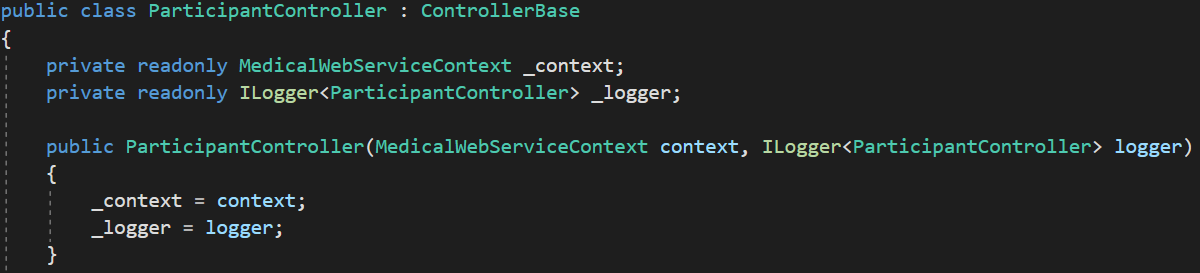
Из Claims получается id исследователя и из таблицы с участниками в базе данных выбираются только те, которые принадлежат данному исследователю.

## Логирование

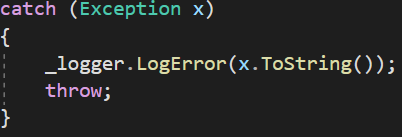
Для целей поиска и исправления ошибок выполнения программного кода и упрощения дальнейшей поддержки используется логирование основных операций, а также всех возникающих ошибок. При работе приложений его логи сохраняются на диске и в случае возникновения сбоя, логи можно скачать и проанализировать.

.NET Core поддерживает API ведения журналов, который работает с разными встроенными и сторонними поставщиками. В данном проекте используется встроенные возможности логирования. Для того чтобы добавить в контроллер возможности логирования достаточно в конструктор добавить специальный параметр, например, ILogger<ParticipantController> logger и сохранить его как поле \_logger в классе. Затем использовать из любого метода как для логирования информационных сообщений, так и для логирования возникающих ошибок выполнения (exceptions). При получении запроса к методу контроллера ASP.NET Core сам создаст экземпляр класса контроллера и автоматически передаст в конструктор ссылку на экземпляр встроенного логгера – это часть встроенного механизма Dependency Injection.

Пример подключения логирования в контроллере ParticipantController:



Пример логирования ошибки:



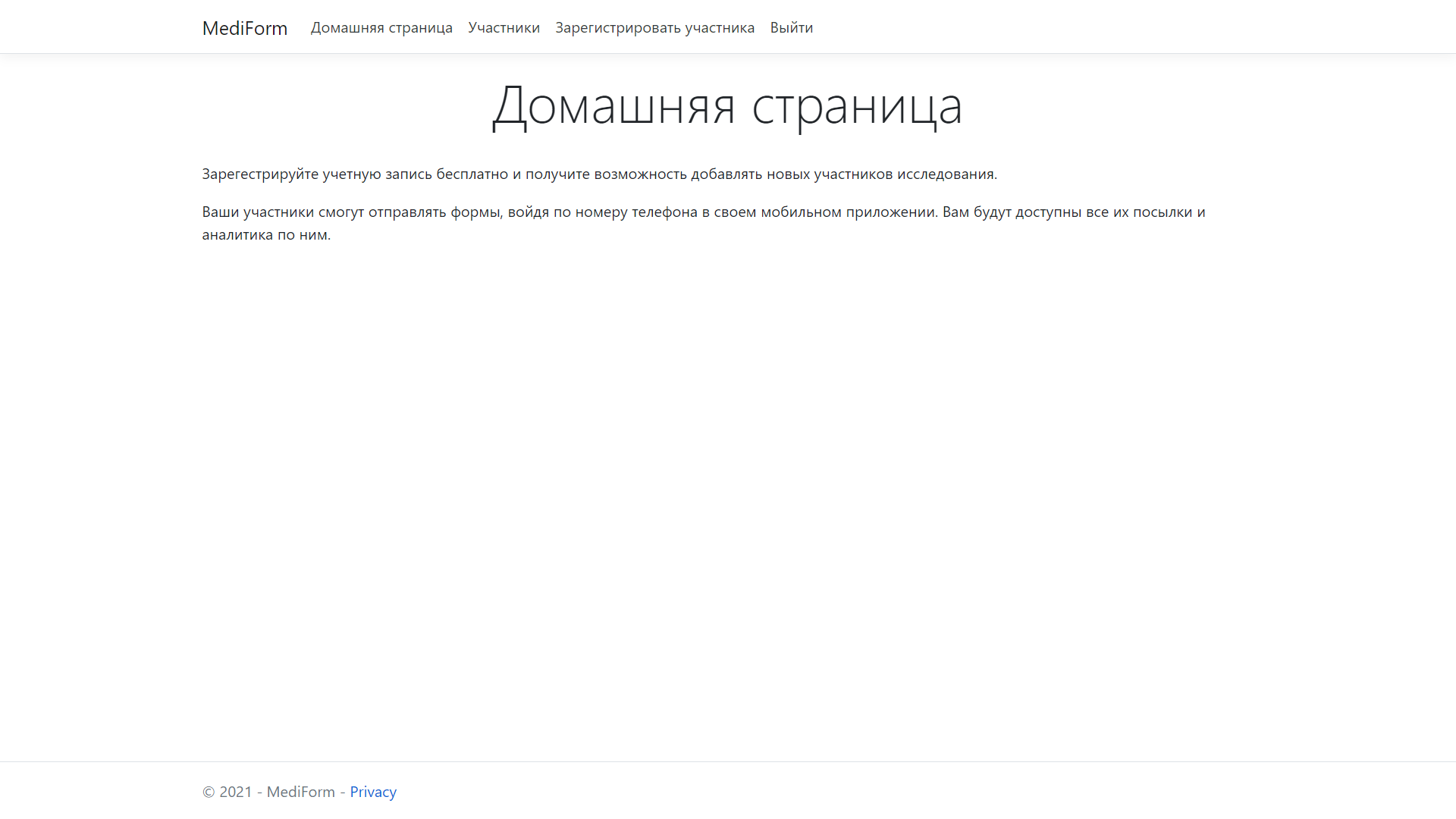
## Веб-сайт

Веб-сайт построен на той же платформе что и веб-сервис. Это ASP.NET Core MVC (концепция Model-View-Controller). В отличие от веб-сервиса, веб-сайт имеет набор файлов, отвечающих за отображение информации пользователю в браузере, это cshtml файлы, находящиеся в папке Views – гибрид html и встраиваемых инструкций на C#. Контроллеры аналогично находятся в папке Controllers, папка Models - для дополнительных классов данных, используемых веб-приложением.

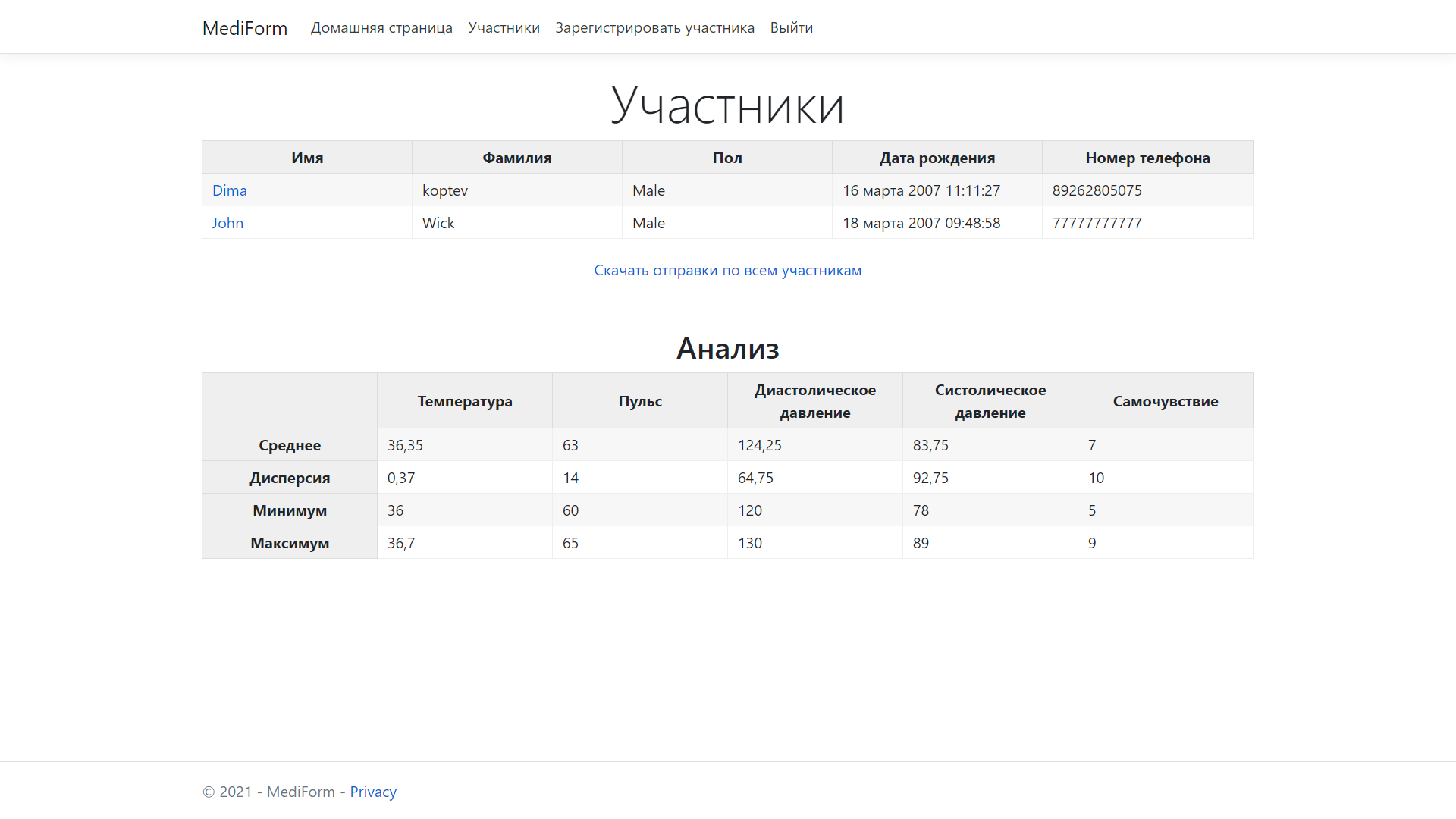
Веб-сайт отвечает исключительно за визуальную часть взаимодействия исследователя с программным комплексом, вся логика приложения находится в веб-сервисе, включая аутентификацию и авторизацию.

Страницы веб-сайта:

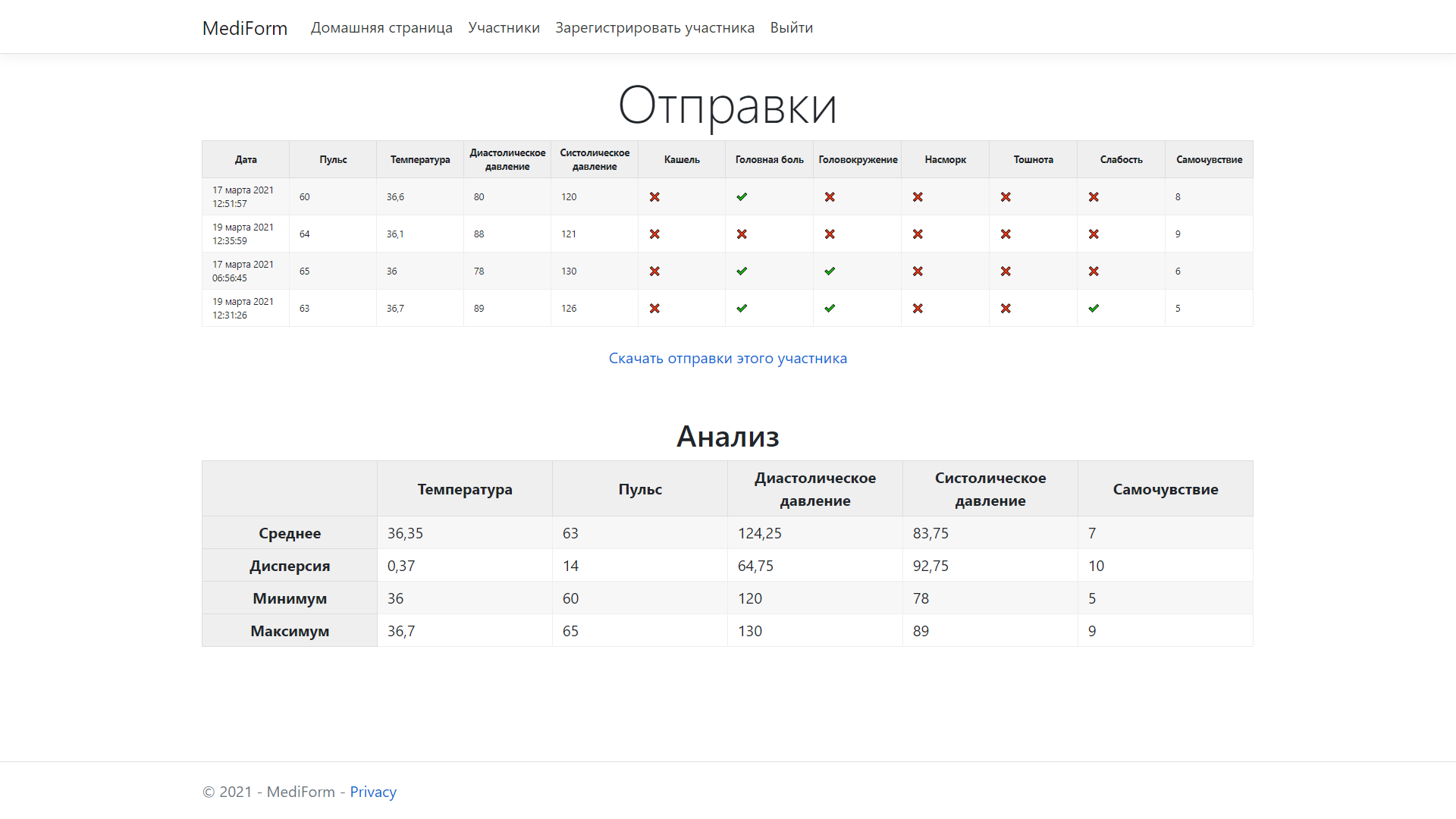
* **Домашняя страница**. При переходе на веб-сайт открывается первой. Домашняя страница содержит краткую инструкцию использования продукта



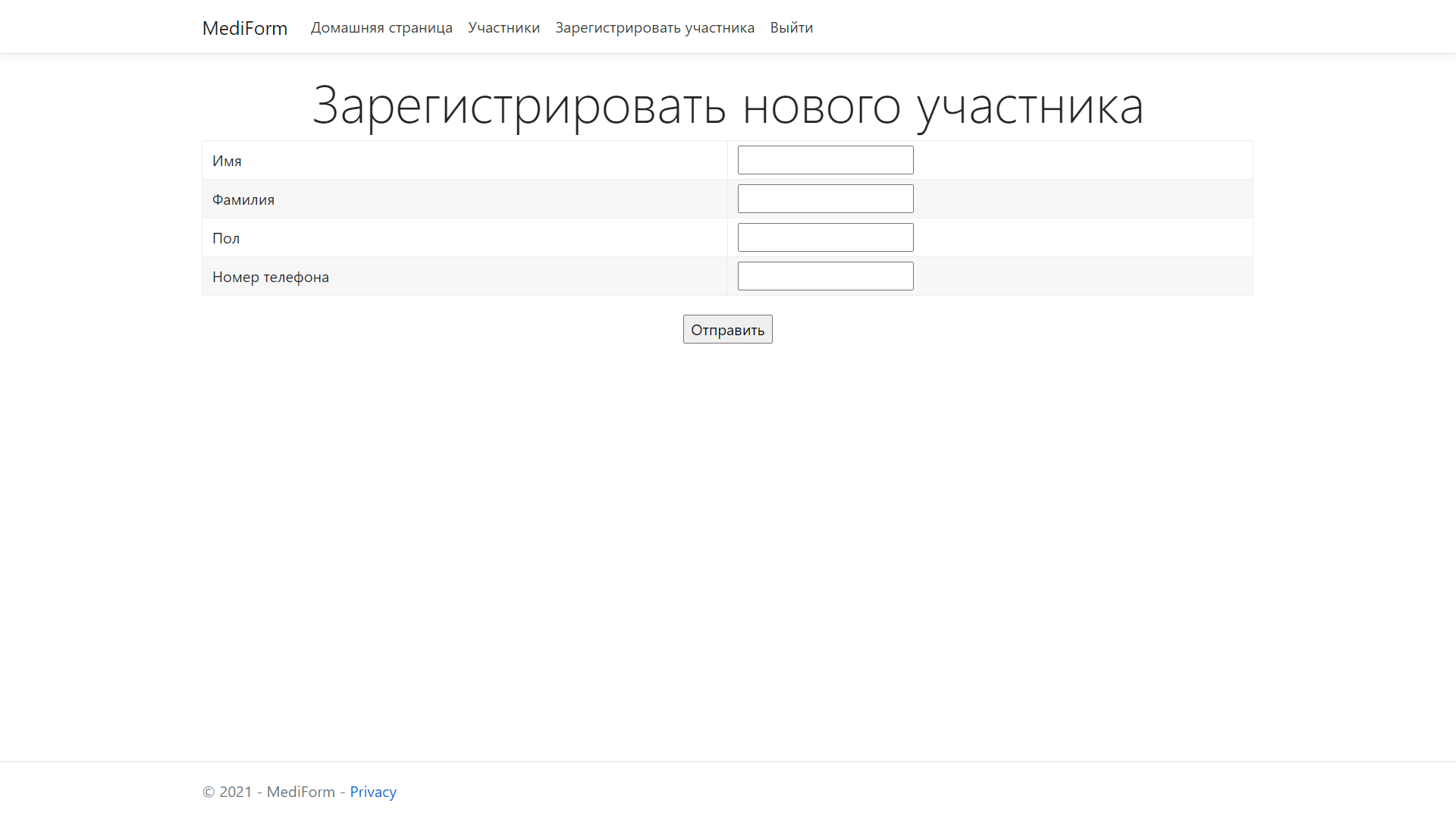
* **Страница с участниками**. Страница отображает всех участников пользователя, вошедшего в аккаунт. По клику на имя участника можно перейти на страницу с его отправленными формами. В нижней части страницы отображается таблица с данными анализа форм отправленных, всеми участниками исследователя. Имеется возможность при клике на ссылку скачать все отправки от участников вошедшего исследователя.



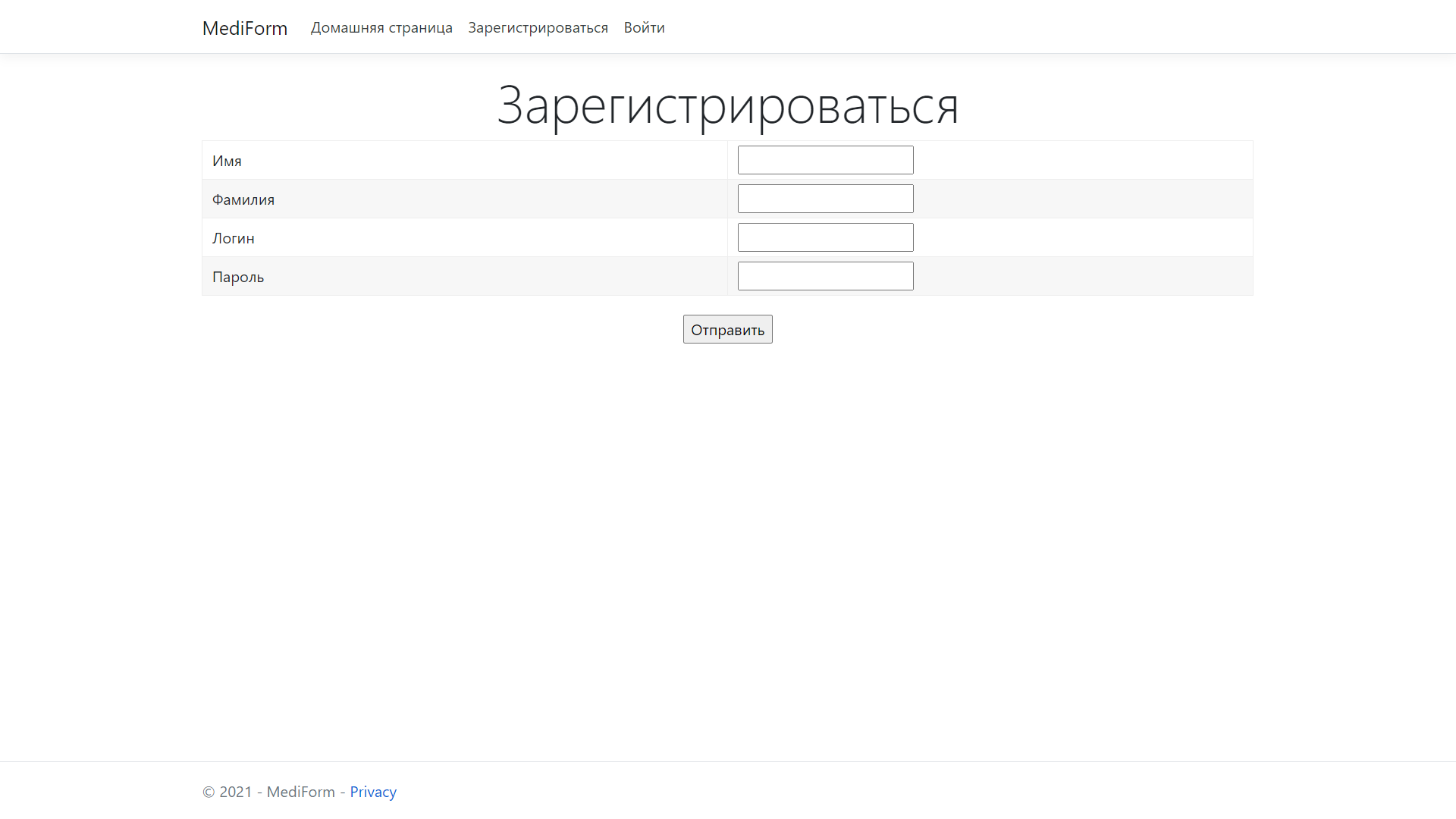
* **Страница просмотра отправленных форм.** При клике на имя участника исследователь переходит на страницу со всеми отправленными формами этого участника. В нижней части находятся аналитические данные по отправкам именно этого участника. Имеется возможность при клике на ссылку скачать данные с отправками этого участника.



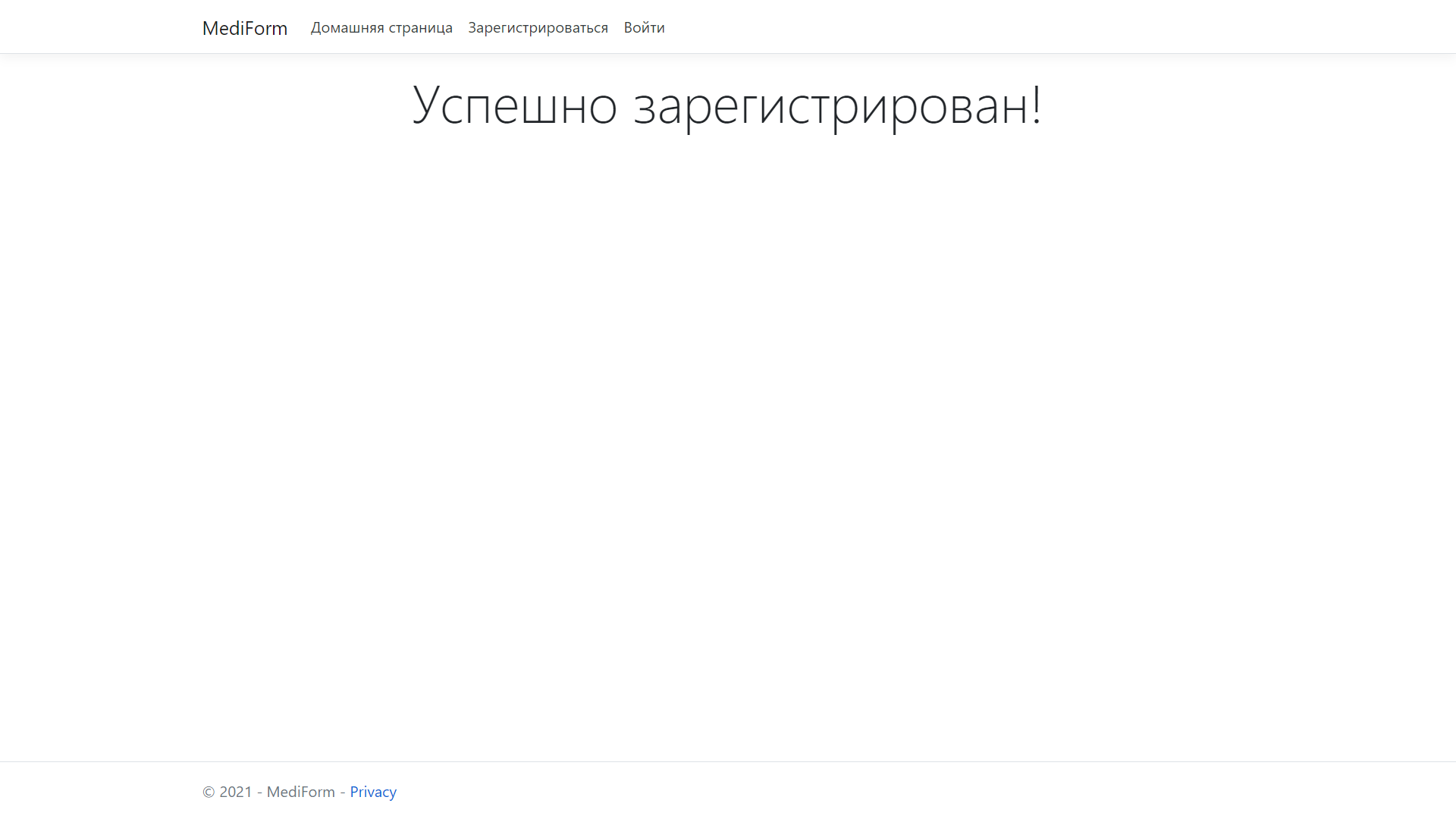
* **Страница регистрации участника**. На этой странице содержится форма для регистрации нового участника текущего исследователя.



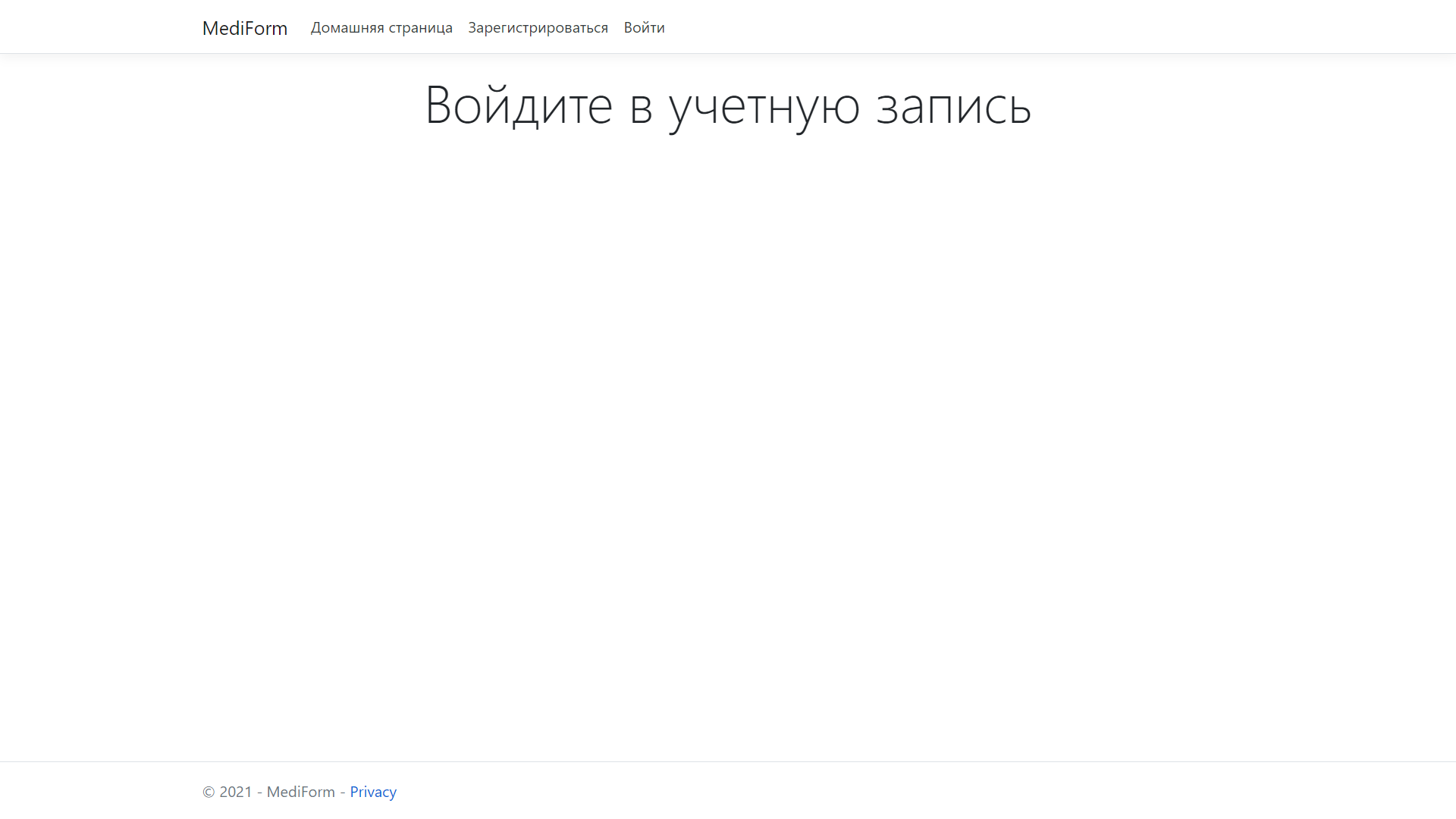
* **Страница регистрации исследователя.** Страница содержит форму регистрации нового исследователя.



После успешной регистрации нового участника, сайт переадресует пользователя на страницу, сообщающую об этом:



* При каком-либо действии не подразумевающем, что исследователь не вошел в аккаунт, например переходе на страницу с участниками, сайт переадресует на страницу с просьбой войти в аккаунт:



## Мобильное приложение

Для реализации мобильного приложения используется кросс платформенный фреймворк Xamarin Forms, позволяющий создавать быстро и относительно не сложно приложения для iOS и Android на одной общей кодовой базе на языке C#. Архитектура приложений на основе этого фреймворка основана на подходе MVVM.

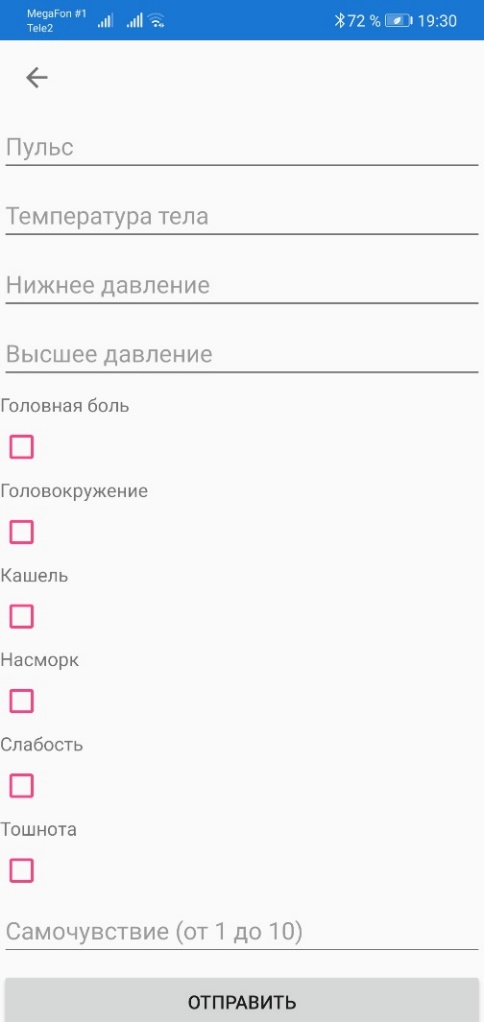
MVVM — это шаблон проектирования, используемый для разделения пользовательского интерфейса (view), данных (model) и логики приложения (view model). Xamarin имеет встроенную поддержку шаблона MVVM, включая привязку данных, что позволяет легко следовать этому шаблону проектирования, чтобы создавать код приложения, который лучше тестируется и легче расширяется, не требуя радикальных изменений.

Аналогично, как и с веб-сайтом, мобильное приложение отвечает только за непосредственное взаимодействие с участниками исследований. Это всего три экрана: экран аутентификации при первом запуске приложения, экран форма отправки данных и экран, сообщающий об успешной отправке формы. После успешной аутентификации при последующих запусках приложения пользователь сразу попадает на форму ввода данных исследования.

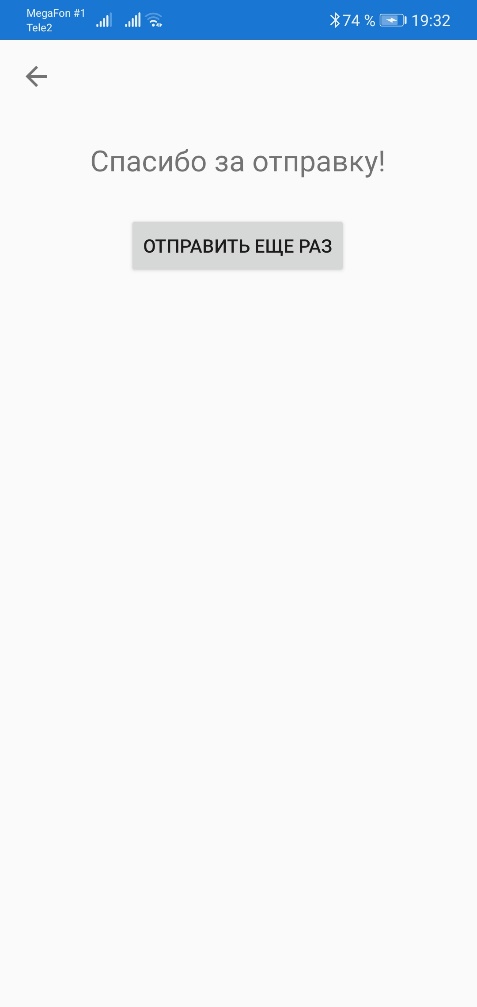
Экран аутентификации:



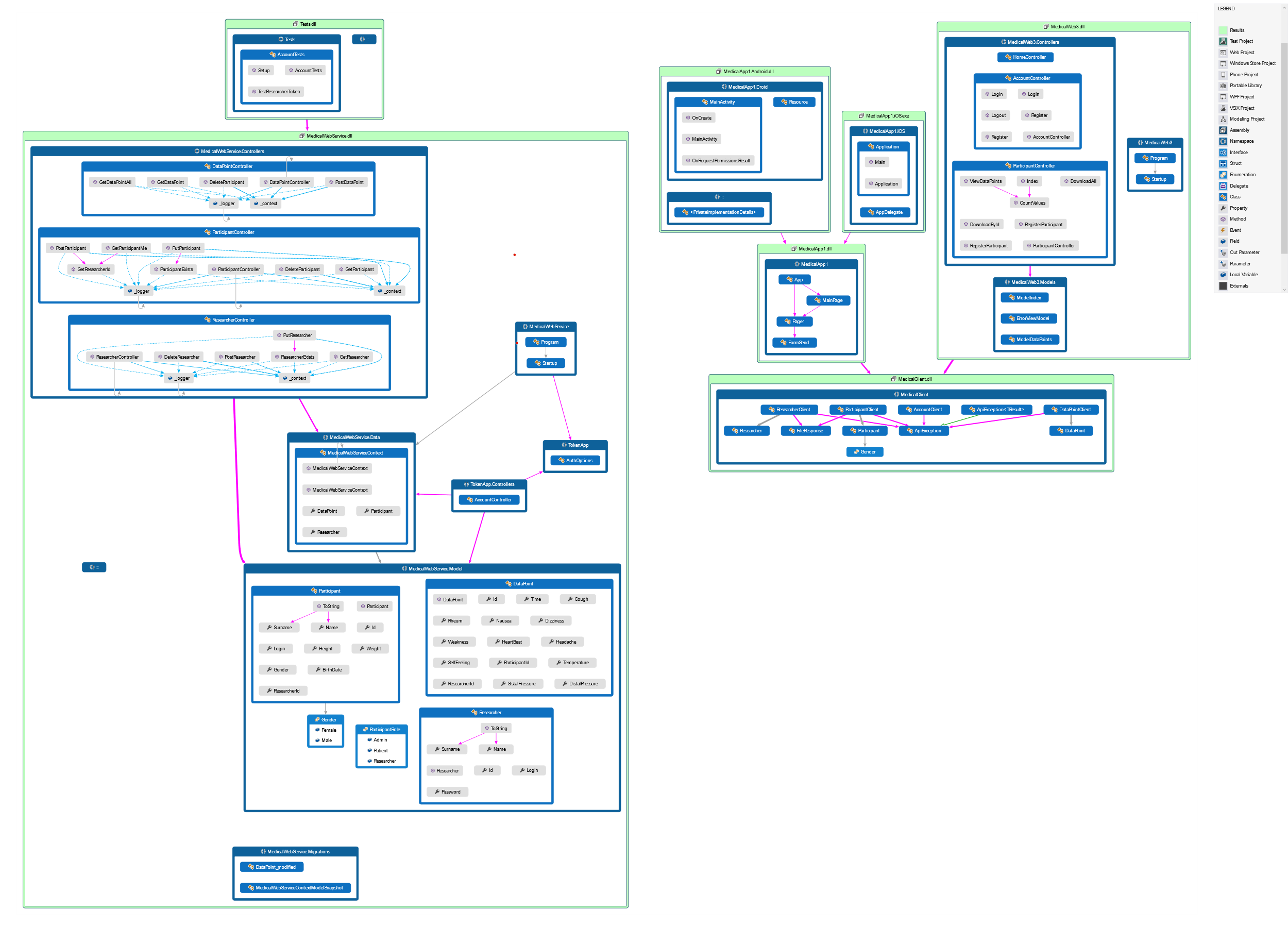
Экран отправки данных:



Экран, сообщающий об успешной отправке:



## Диаграмма классов



# Тестирование

Для тестирования программного комплекса использовалось несколько подходов: автоматическое модульное тестирование (юнит тесты), ручное в рамках интеграционного и функционального тестирования, и нагрузочное тестирование.

### Юнит тесты

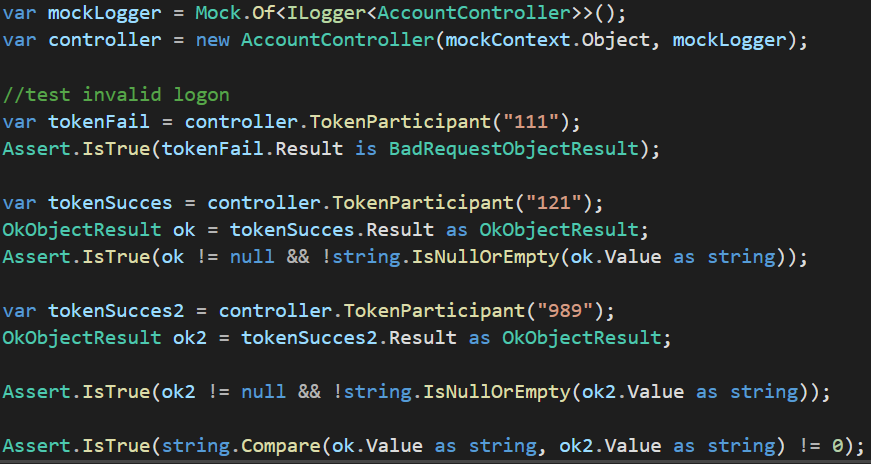
Модульное тестирование или юнит-тестирование (unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

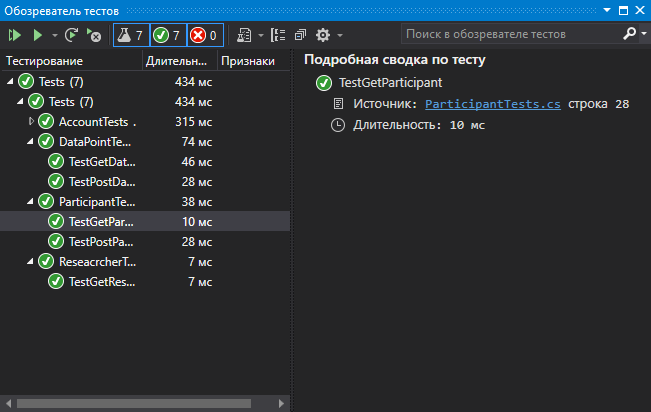
Для написания и запуска юнит тестов использовался nuget пакет NUnit, который интегрируется непосредственно в среду разработки Visual Studio. Так как ASP.NET Core проекты уже построены с использованием Dependency Injection шаблона, который обеспечивает слабую зависимость классов друг от друга, то написание юнит тестов значительно облегчается, отдельные модули могут тестироваться изолировано друг от друга. И даже если есть зависимость между ними, то для тестирования отдельного модуля создается Mock класс, эмулирующий работу компонента по его интерфейсу. Например, можно тестировать работу контроллеров веб-сервиса, заменив реальный объект работ с БД (DbContext) на его упрощенный аналог, который будет держать данные в памяти.

Основной принцип юнит тестирования — это проверка соответствия возвращаемого значения тестируемых функций или состояния объектов ожидаемым исходя из задаваемых параметров.

Пример юнит теста:



Результаты запусков:



### Ручное тестирование

Ручное тестирование программного обеспечения – это процесс проверки ПО, выполняемый тестировщиками вручную. Это значит, что для его проведения не используются какие-либо специальные автоматизированные средства.

Программное обеспечение проверяется инженерами по тестированию, которые берут на себя роль конечных пользователей, моделируют ситуации в соответствии с тестовыми сценариями и фиксируют результат. Задача ручного тестирования программного обеспечения — выявить любое поведение, отличающееся от ожидаемого пользователем.

Ручное тестирование может проводиться в рамках регрессионного (тестирование различных изменений), интеграционного (взаимодействие между сервисом и приложениями) и при системном функциональном тестировании - определить, насколько разработанное программное обеспечение соответствует функциональным требованиям, описанным в виде сценариев использования.

### Нагрузочное тестирование

Нагрузочное тестирование (англ. load testing) — подвид тестирования производительности, сбор показателей и определение производительности и времени отклика программно-технической системы на внешний запрос с целью установления соответствия требованиям, предъявляемым к данной системе (устройству).

Так как вся логика приложения находится в веб-сервисе нагрузочное тестирование проводим для основных операций этого модуля. Это регистрация новых данных от участников исследований и получение данных исследователем по своим участникам.

Для нагрузочного тестирования веб-сервиса используется та же библиотека NUnit. Основной принцип — это обеспечение повышенной нагрузки на сервис путем отправки множества параллельных запросов на тестируемую систему и отслеживания времени отклика – от отправки запроса до получения корректного ответа. Также при нагрузочном тестировании проверяется поведение системы при большом количестве записей в БД. Все временные показатели должны оставаться в рамках поставленных требований.

# Результаты работы

В результате проделанной работы получен работающий программный комплекс в соответствии с исходными требованиями к MVP. Проект и все его этапы выполнены в распланированные сроки. Я считаю, что достиг цели и доволен результатом работы. При выполнении проекта освоены: базовые технологии создания распределенных приложений, различные виды тестирования распределенных приложений – юнит-тестирование, нагрузочное тестирование и ручное тестирование, навыки разработки мобильных приложений на платформе Xamarin, навыки работы с такими фреймворками как .NET, ASP NET MVC, Entity Framework для разработки веб-сервиса и веб-приложения.

В процессе выполнение проекта были развиты следующие навыки учащегося IB: Knowledgeable, Thinking, Communicating, Reflective.

Планируется ознакомить с результатами работы целевую аудиторию: индивидуальных исследователей, представителей медицинских НИИ, фармацевтических компаний, потенциально заинтересованных в дальнейшем развитии продукта, с целью сбора отзывов, а также вариантов возможного сотрудничества.

Исходный код продукта можно просмотреть по ссылке: <https://github.com/ak-gr0up/PersonalProject>

# Приложения

## Выдержки из дневника работы над проектом

16.09.2020 — Начал выбирать тему для проекта

11.09.2020 — Посмотрел лекцию по персональному проекту в 9 классе

16.10.2020 — Определился с темой проекта

*Возможные версии*:

* Интерактивный учебник/сайт/YouTube канал для изучения тем по олимпиадной информатике/математике
* Автоматизация сбора и обработки данных медицинских клинических исследований

Выбрал вторую тему, т.к. она более интересная и актуальная. Тема интересна мне, т.к. я интересуюсь разработкой прикладного ПО. (мышление)

Продуктом проекта(предположительно) будет программный комплекс, состоящий из мобильного приложения на android и веб-приложения с веб-сервисом:

* Мобильное приложение необходимо для сбора данных с испытуемых
* Веб-сервис – для хранения и обработки данных испытуемых
* Веб-приложение – для представления обработанных данных и текущих результатов

18.10.2020 – Создание прототипа веб-сервиса на языке C#, .NET Core (реализация)

21.10.2020 – Освоил подключение спецификации Swagger к веб-сервису (реализация, исследование)

22.10.2020 – Разобрался как работать с базой данных SQL и Entity Framework (реализация, исследование)

28.10.2020 – Проведена первая встреча с супервайзером. Подправлен паспорт проекта (коммуникация)

6.11.2020 – Реализовал методы создания нового пользователя и получения их списка в веб-сервисе (реализация)

10.11.2020 – Протестировал технологии создания веб-сайта на основе фреймворков Angular, Vue.js, ASP.NET MVC (реализация, исследование)

3.11.2020 - Реализовал прототип веб-приложения с возможностью добавления и просмотра пользователей на ASP.NET MVC Core (реализация)

20.11.2020 - Протестировал технологии создания мобильных приложений на Kotlin, C# Xamarin.Forms (реализация)

27.12.2020 - Создал прототип мобильного приложения с возможностью добавления и просмотра пользователей на C# Xamarin.Forms (реализация)

08.01.2021 - Описание проблемы и обзор существующих решений (исследование, мышление)

09.01.2021 – Предварительная постановка требований к продукту (планирование, мышление)

12.01.2021 - Интервьюирование целевой аудитории. Опрошен врач 1-ой городской больницы г. Москвы и IT Service Delivery Manager Global Medical Affairs, Novartis, Swiss (коммуникация)

14.01.2021 - Корректирование требований к продукту на основе интервьюирования (исследование)

16.01.2021 - Добавлена возможность регистрации исследователей без повторяющихся логинов (реализация)

17.01.2021 - Добавлена возможность исследователям регистрировать своих пациентов (реализация)

19.01.2021 - Добавлена возможность выхода из аккаунта исследователя (реализация)

20.01.2021 – Проведена вторая встреча с супервайзером. Обсуждался отчет и дневник проекта. (коммуникация)

21.01.2021 - Изучил юнит-тестирование, добавил юнит-тесты для методов создания токена (реализация, исследование)

24.01.2021 - Создано мобильное приложение, исследователя могут входить и отправлять формы (реализация)

19.01.2021 - Добавлена возможность исследователю просматривать отправки своих участников (реализация)

20.01.2021 - Добавлена возможность скачивания отправок участников в виде csv файла (реализация)

23.01.2021 - При просмотре отправок добавлено отображение статистики как по всем отправкам, так и по отправкам конкретного участника (реализация)

25.01.2021 - Подправлен дизайн веб-сайта и мобильного приложения (реализация)

12.02.2021 - Добавлены юнит-тесты для всех контроллеров веб-сервиса (реализация)

18.02.2021 – Продукт протестирован (реализация)

24.02.2021 – 22.03.2021 Оформление отчета (рефлексия)

10.03.2021 – Проведена третья встреча с супервайзером. Обсудили черновой вариант отчета проекта. (коммуникация)

23.03.2021 – Отснято демонстрационное видео

# Библиография

ajcvickers. n.d. “Тестирование с помощью инфраструктуры макетирования — EF6.” Docs.microsoft.com. Accessed February 10, 2021. https://docs.microsoft.com/ru-ru/ef/ef6/fundamentals/testing/mocking.

“ASP.NET Core | Полное руководство.” n.d. Metanit.com. Accessed March 24, 2021. https://metanit.com/sharp/aspnet5/.

bricelam. n.d. “Обзор миграций — EF Core.” Docs.microsoft.com. Accessed March 24, 2021. https://docs.microsoft.com/ru-ru/ef/core/managing-schemas/migrations/?tabs=vs.

———. n.d. “Применение миграций — EF Core.” Docs.microsoft.com. Accessed November 24, 2020. https://docs.microsoft.com/ru-ru/ef/core/managing-schemas/migrations/applying?tabs=vs.

“C# - How to Add Claims in ASP.NET Identity.” n.d. Stack Overflow. Accessed October 09, 2020. https://stackoverflow.com/questions/20383955/how-to-add-claims-in-asp-net-identity.

“C# - How to Clear Cookie in Asp.net Mvc?” n.d. Stack Overflow. Accessed 26 November, 2020. https://stackoverflow.com/questions/42953181/how-to-clear-cookie-in-asp-net-mvc.

“C# - Mocking EF DbContext with Moq.” n.d. Stack Overflow. Accessed 17 Januray, 2021. https://stackoverflow.com/questions/25960192/mocking-ef-dbcontext-with-moq.

davidbritch. n.d. “Навигация в приложении Xamarin.Forms - Xamarin.” Docs.microsoft.com. Accessed November 23 24, 2020. https://docs.microsoft.com/ru-ru/xamarin/get-started/quickstarts/navigation?pivots=windows.

———. n.d. “Xamarin.Forms Tutorials - Xamarin.” Docs.microsoft.com. https://docs.microsoft.com/en-us/xamarin/get-started/tutorials/.

“DM RWE – EPRO.” n.d. Dm-Matrix.com. Accessed 29 September, 2020. https://dm-matrix.com/rus/resheniya/epro.

“Пишем MVP приложение на Kotlin под Android.” n.d. Habr.com. https://habr.com/ru/post/275255/.

“Работа с формами в ASP.NET MVC 5.” n.d. Metanit.com. Accessed October 05, 2020. https://metanit.com/sharp/mvc5/4.6.php.

wadepickett. n.d. “Документация по ASP.NET.” Docs.microsoft.com. Accessed November 07, 2020. https://docs.microsoft.com/ru-ru/aspnet/core/?view=aspnetcore-5.0.