

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1 ОБЗОР ЛИТЕРАТУРЫ	7
2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ	18
3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ	25
4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ	41
4.1 Расширение функциональности системных модулей	41
4.2 Выполнение HTTP-методов	42
4.3 Работа с базой данных посредством репозитория	45
4.4 Работа микросервисов	47
4.5 Работа DbContext и миграций	47
4.6 Работа SignalR	48
4.7 Работа вспомогательных классов	49
4.7.1 Вспомогательный класс AutoMapperProfiles	49
4.7.2 Вспомогательные классы Pagination	50
4.7.3 Обработка ошибок	51
5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ	52
5.1 Модульное тестирование	52
5.2 Функциональное тестирование	56
5.3 Интеграционное тестирование	57
6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ	59
6.1 Настройка сервера и развертывание приложения	59
6.2 Руководство по использованию ПО	61
7 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ИСПОЛЬЗОВАНИЯ НОВОСТНОГО ПОРТАЛА С ТЕМАТИКОЙ КОСМИЧЕСКОЙ НАПРАВЛЕННОСТИ НА МАССОВОМ РЫНКЕ	73
ЗАКЛЮЧЕНИЕ	80
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	81
ПРИЛОЖЕНИЕ А	82
ПРИЛОЖЕНИЕ Б	83
ПРИЛОЖЕНИЕ В	88
ПРИЛОЖЕНИЕ Г	89

ВВЕДЕНИЕ

Мы часто называем нашу расширяющуюся вселенную одним простым словом: космос. Но где начинается космос и, что более важно, что это такое?

Космос — это фантастическая «игровая площадка» для ученых — бесконечный источник знаний и обучения, который помогает ответить на некоторые из ключевых экзистенциальных вопросов о происхождении Земли и нашем месте во Вселенной.

Многие инновации в различных областях, от металлов и сплавов до биологии и медицины, являются результатом освоения космоса. Некоторые области применения — например, керамические покрытия на наших кухнях, системы очистки воздуха, детекторы дыма и устойчивое к царапинам стекло — уже стали частью нашей повседневной жизни.

Материалы, протестированные в космосе в уникальных условиях, которые трудно воспроизвести на Земле, могут помочь нам в разработке более прочных, легких и высокопроизводительных продуктов.

Понимание окружающей среды на других планетах, где однажды могут жить люди, и изучение наших собственных биологических систем и того, как материалы ведут себя, когда на них не действует гравитация, являются критически важными для нашего будущего.

Исследование космоса является движущей силой в наших усилиях по решению основных проблем, стоящих перед человечеством сегодня. Он учит нас ответственности перед Землей и ее благами. Именно поэтому обществу крайне необходимы новостные ресурсы, освещающие актуальную информацию о процессах, происходящих в космосе, и инновациях в космических технологиях.

Целью дипломного проекта является разработка новостного портала с тематикой космической направленности. Для выполнения поставленной задачи необходимо произвести обзор аналогов и, на основе анализа и корректно выстроенного плана разработки, создать конкурентноспособный продукт. Для создания работоспособного продукта потребуется уделить внимание вопросу разработки клиент-серверных приложений и выполнить проектирование проекта.

В соответствии с поставленной целью были определены следующие задачи:

1. Выбор платформы создания системы.
2. Разработка пользовательского интерфейса.
3. Разработка протокола взаимодействия клиентского интерфейса с серверной частью программного модуля.
4. Тестирование программного модуля.
5. Расчёт экономических затрат на создание проекта.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Обзор аналогов

В данном подразделе рассмотрим существующие аналоги реализуемого программного обеспечения.

1.1.1 SpaceNews

SpaceNews [1] (см. рисунок 1.1) – это источник новостей и аналитических материалов о компаниях, агентствах, технологиях и тенденциях, формирующих мировую космическую отрасль.

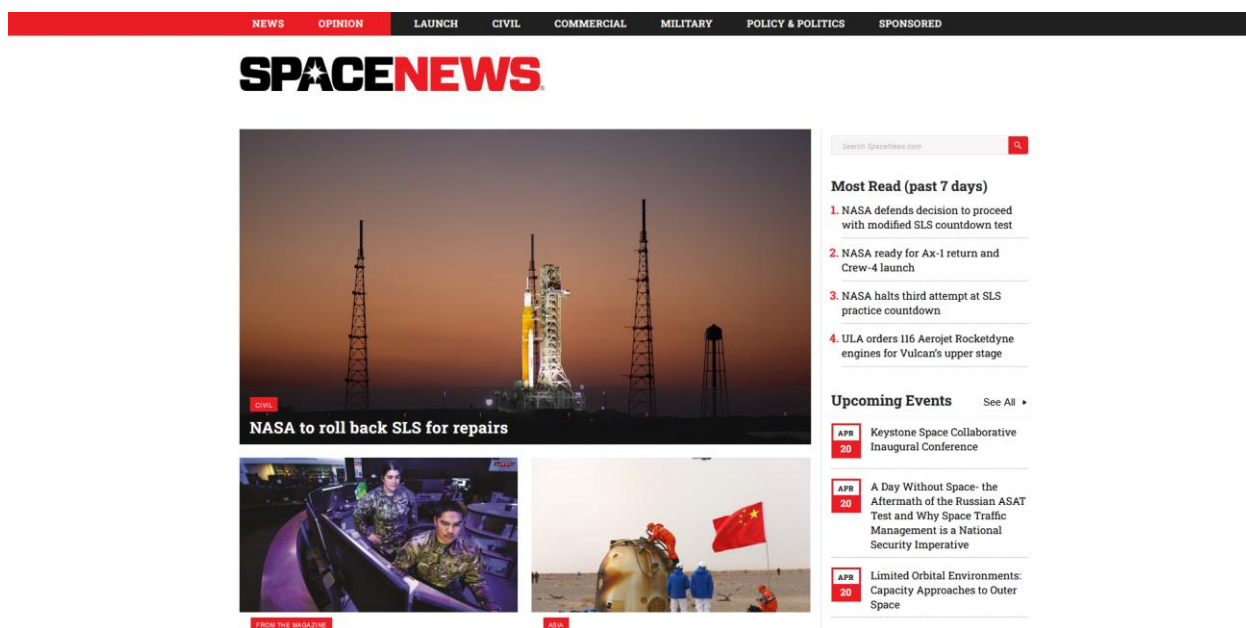


Рисунок 1.1 – Главная страница SpaceNews [1]

На протяжении более трех десятилетий SpaceNews остается единственным изданием, к которому первыми обращаются космические профессионалы во всем мире. Будь то последние тенденции в военных космических возможностях, разработках в области спутниковой связи или текущем состоянии бюджета, можно рассчитывать на SpaceNews. Этот сайт обеспечивает всестороннее освещение космической отрасли и критическую точку зрения, на которую пользователь может положиться, благодаря новостям, комментариям и анализу. Можно выделить явные достоинства и недостатки.

Достоинства:

- удобный и простой интерфейс, сайт легкий для восприятия;
- широкая функциональность;
- адаптация для мобильных устройств.

Недостатки:

- существует платная подписка;
- отсутствие русской локализации.

1.1.2 SpaceFlight Now

SpaceFlight Now [2] (см. рисунок 1.2) – веб-приложение компаньон, работающее на SpaceX API и Reddit. Предназначено для отслеживания информации о запусках SpaceX для космических энтузиастов.

Является онлайн-порталом космических новостей, на котором публикуются текущие новости, текущие миссии, предстоящие запуски и архивные статьи. Spaceflight Now обеспечивает непревзойденное ежедневное освещение космической программы. С момента своего запуска в декабре 1999 года он стал одним из самых надежных источников точных и последних новостей для космического сообщества. Существенным недостатком данного сайта является отсутствие оптимизации для использования в мобильной версии, что ухудшает удобство просмотра сайта.

Исходя из предложенных ранее фактов, резюмируем достоинства и недостатки данного ресурса.

Достоинства:

- не требует регистрации;
- условно бесплатный;
- простой интерфейс, сайт легкий для восприятия.

Недостатки:

- отсутствие адаптации для мобильных устройств;
- ограниченная функциональность.

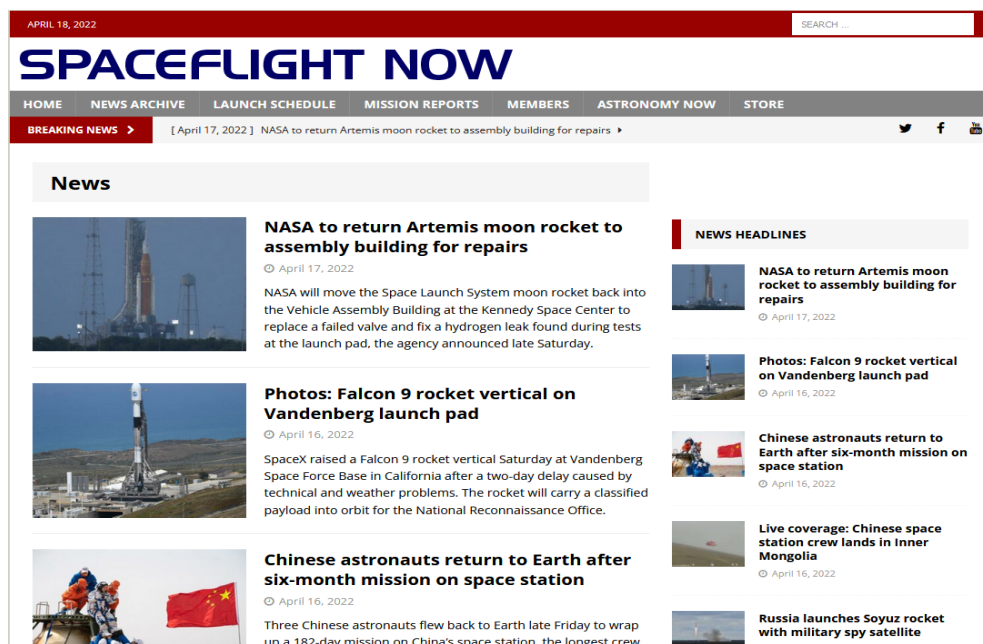


Рисунок 1.2 – Главная страница SpaceFlight Now [2]

1.2 Обзор технологий

Разрабатываемая в рамках данного дипломного проекта программная система является веб-приложением. Под веб-приложением понимается клиент-серверное приложение, в котором роль клиента играет браузер, а сервером является веб-сервер, который может храниться в облачном хранилище. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети. Благодаря такому подходу клиент может не зависеть от конкретной операционной системы. Из этого следует, что веб-приложения являются кроссплатформенными сервисами.

В качестве способа взаимодействия сайта и веб-приложения с сервером используется архитектура REST API.

API — это интерфейс прикладного программирования. Это набор правил, которые позволяют программам общаться друг с другом. Разработчик создает API на сервере и позволяет клиенту общаться с ним.

REST определяет, как выглядит API. Это набор правил, которым следуют разработчики при создании своего API. Одно из этих правил гласит, что вы должны иметь возможность получать фрагмент данных (называемый ресурсом) при ссылке на определенный URL-адрес.

Каждый URL-адрес называется запросом, а данные, отправленные вам, называются ответом.

Данные языки программирования чаще всего используются для разработки веб-сервера: C#, Ruby, Java, JavaScript, Python.

1.2.1 Архитектура клиент-сервер

Архитектура клиент-сервер — это вычислительная модель, в которой сервер размещает, доставляет и управляет большей частью ресурсов и услуг, потребляемых клиентом. В этом типе архитектуры один или несколько клиентских компьютеров подключены к центральному серверу по сети или через Интернет [3] (см. рисунок 1.3). Эта система разделяет вычислительные ресурсы. Архитектура клиент-сервер также известна как модель сетевых вычислений или сеть клиент-сервер, поскольку все запросы и услуги доставляются по сети.

Архитектура клиент-сервер — это архитектура компьютерной сети, в которой множество клиентов запрашивают и получают услуги от централизованного сервера.

Клиентские компьютеры предоставляют интерфейс, позволяющий пользователю компьютера запрашивать услуги сервера и отображать результаты, возвращаемые сервером.

Серверы ждут поступления запросов от клиентов и затем отвечают на них.

Многие клиенты могут одновременно получать доступ к информации сервера, и в то же время клиентский компьютер может выполнять другие задачи.



Рисунок 1.3 – Архитектура клиент-сервер [3]

Основные характеристики клиент-серверной архитектуры:

- горизонтальная масштабируемость и вертикальная масштабируемость;
- клиентское или серверное приложение напрямую взаимодействует с протоколом транспортного уровня для установления связи и отправки или получения информации;
- затем транспортный протокол использует протоколы нижнего уровня для отправки или получения отдельных сообщений. Таким образом, компьютеру необходим полный стек протоколов для запуска клиента или сервера;
- клиентские и серверные машины нуждаются в разном количестве аппаратных и программных ресурсов;
- клиентские и серверные машины могут принадлежать разным поставщикам;
- один компьютер серверного класса может одновременно предлагать несколько услуг; для каждой службы требуется отдельная серверная программа.

Преимущества:

- общие ресурсы среди разных платформ;
- совместная работа с данными;
- возможности обработки данных, несмотря на местоположение;
- простота обслуживания;
- улучшенный обмен данными;
- интеграция услуг;
- безопасность.

Недостатки:

- перегруженные серверы: при частых одновременных клиентских

запросах серверы сильно перегружаются;

влияние централизованной архитектуры: поскольку она централизована, в случае сбоя критического сервера запросы клиентов не выполняются.

1.2.2 Шаблон проектирования REST API

REST API [4] (также известный как RESTful API) — это интерфейс прикладного программирования, который соответствует ограничениям архитектурного стиля REST и позволяет взаимодействовать с веб-службами RESTful. REST означает передачу репрезентативного состояния и был создан компьютерным ученым Роем Филдингом.

API — это набор определений и протоколов для создания и интеграции прикладного программного обеспечения. Иногда его называют договором между поставщиком информации и пользователем информации, устанавливающим контент, требуемый от потребителя, и контент, требуемый производителем.

Другими словами, API помогает пользователю передать то, что он хочет, этой системе, чтобы она могла понять и выполнить запрос.

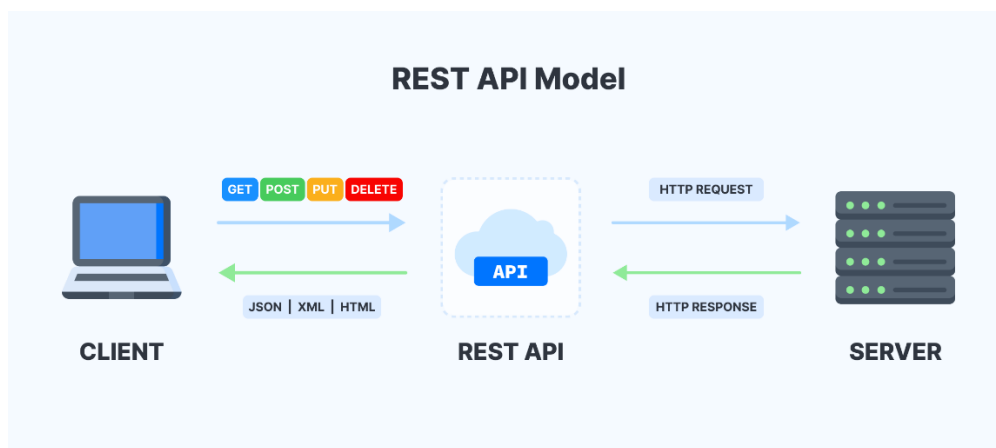


Рисунок 1.4 – Шаблон проектирования REST API

API выполняет роль посредника между пользователями или клиентами и ресурсами или веб-сервисами, которые они хотят получить. Это также способ для организации обмена ресурсами и информацией, который помогает сохранить при этом безопасность, контроль и аутентификацию, определяя, кто и к чему получает доступ.

Еще одно преимущество API заключается в том, что вам не нужно знать особенности кэширования - как извлекается ваш ресурс или откуда он берется.

REST — это набор архитектурных ограничений, а не протокол или стандарт. Когда клиентский запрос выполняется через RESTful API, он передает представление о состоянии ресурса запрашивающей стороне или конечной точке. Эта информация или представление доставляется в одном из

нескольких форматов через HTTP: JSON (обозначение объектов Javascript), HTML, XML, Python, PHP или обычный текст. JSON является наиболее популярным форматом файлов для использования, потому что, несмотря на свое название, он не зависит от языка, а также удобен для чтения как людьми, так и машинами.

Чтобы API считался RESTful, он должен соответствовать следующим критериям:

- архитектура клиент-сервер, которая состоит из клиентов, серверов и ресурсов, с запросами, управляемыми через HTTP;
- связь клиент-сервер;
- кэшируемые данные, которые оптимизируют взаимодействие клиент-сервер.

В отличие от этого, REST — это набор рекомендаций, которые можно реализовать по мере необходимости, что делает REST API более быстрыми и легкими, с повышенной масштабируемостью - идеально подходит для мобильных веб-приложений.

1.2.3 .NET

.NET [5] — это бесплатная кроссплатформенная платформа для разработчиков с открытым исходным кодом для создания различных типов приложений. В .NET используются несколько языков, редакторов и библиотек для создания приложений для Интернета, мобильных устройств, компьютеров, игр и Интернета вещей.

Независимо от того, работаете ли вы на C#, F# или Visual Basic, ваш код будет выполняться в любой совместимой ОС. Различные реализации .NET справятся с тяжелой работой за вас.

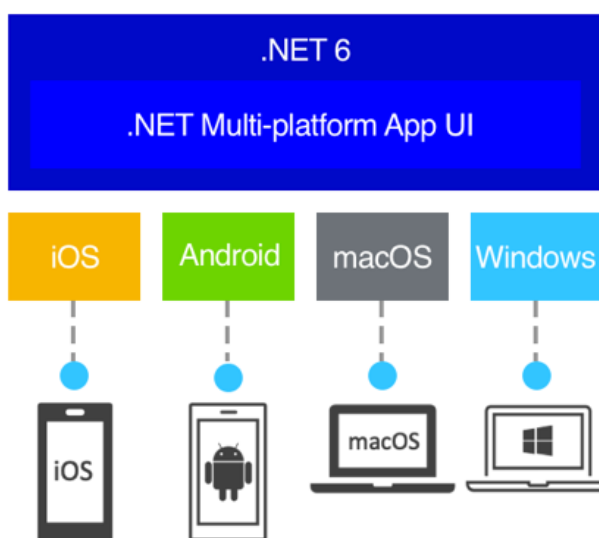


Рисунок 1.5 – Платформа .NET 6

.NET — это кроссплатформенная реализация .NET для веб-сайтов, серверов и консольных приложений в Windows, Linux и macOS.

.NET Framework поддерживает веб-сайты, службы, классические приложения и многое другое в Windows.

Xamarin/Mono — это реализация .NET для запуска приложений во всех основных мобильных операционных системах.

.NET Standard — это базовый набор API, общий для всех реализаций .NET. Каждая реализация также может предоставлять дополнительные API, специфичные для операционных систем, в которых она работает. Например, .NET Framework — это реализация .NET только для Windows, которая включает API для доступа к реестру Windows.

Чтобы расширить функциональность, Microsoft и другие компании поддерживают здоровую экосистему пакетов, построенную на .NET Standard. NuGet — это диспетчер пакетов, созданный специально для .NET и содержащий более 90 000 пакетов.

.NET имеет открытый исходный код и находится под управлением .NET Foundation. .NET Foundation — это независимая организация, способствующая открытой разработке и сотрудничеству в рамках экосистемы .NET.

1.2.4 Angular

Angular [6] - один из самых мощных интерфейсных фреймворков с открытым исходным кодом, построенный на TypeScript. Разработанный Google в 2010 году для создания динамичных и современных одностраничных приложений (SPA), он сначала начинался как среда JS, которая позже была преобразована в совершенно новую структуру TypeScript.

Angular — это современная структура и платформа MVVC. Он имеет огромное количество функций, включая двустороннюю привязку, внедрение зависимостей, RESTful API и обработку AJAX.



Рисунок 1.6 – Angular

Angular включает в себя:

- компонентную среду для создания масштабируемых веб-приложений;
- коллекцию хорошо интегрированных библиотек, которые охватывают

широкий спектр функций, включая маршрутизацию, управление формами, взаимодействие клиент-сервер и многое другое;

– набор инструментов разработчика, которые помогают разрабатывать, создавать, тестировать и обновлять код.

С Angular [7] можно масштабироваться от проектов с одним разработчиком до приложений корпоративного уровня.

Это всего лишь набор преимуществ, которые Angular предоставляет разработчикам. Angular - один из лучших вариантов для создания внешнего интерфейса, но он не единственный на рынке.

Хотя Angular полезен для создания небольших приложений, он предпочтительнее для больших приложений. Поддержка со стороны Google и его сообщества веб-разработчиков позволяет Angular быстро развиваться и быстро внедряться. Кроме того, его основной особенностью является использование языка Typescript, который совместим со всеми устройствами, операционными системами и браузерами.

1.2.5 C#

В качестве языка для написания веб-сервера данного проекта используется язык C# [8].

C# — это универсальный, современный и объектно-ориентированный язык программирования, произносимый как “C Sharp”. Он был разработан Microsoft во главе с Андерсом Хейлсбергом и его командой в рамках инициативы .NET и был одобрен Европейской ассоциацией производителей компьютеров (ЕСМА) и Международной организацией стандартов (ISO). C# является одним из языков для инфраструктуры общего языка. C# очень похож на Java синтаксически и прост для пользователей, которые знают C, C++ или Java.

Интересно, что Хейлсберг-выдающийся инженер Microsoft, который создал другие продукты и языки, включая Borland Turbo C++ и Borland Delphi. С C# они сосредоточились на том, чтобы взять то, что было правильно в существующих языках, и добавить улучшения, чтобы сделать что-то лучше.



Рисунок 1.7 – Язык программирования C#

C# - мощный и гибкий язык программирования. Как и все языки программирования, он может быть использован для создания различных приложений. Язык не накладывает ограничений на то, что можно реализовать. C# используется для таких разнообразных проектов, как динамические веб-сайты, инструменты разработки и компиляторы.

C# был создан как язык объектно-ориентированного программирования (ООП). Другие языки программирования включают объектно-ориентированные функции, но очень немногие из них полностью объектно-ориентированы.

1.2.6 TypeScript

Для разработки пользовательского интерфейса используется язык программирования TypeScript [9].

TypeScript был создан Microsoft и был выпущен в 2012 году после двух лет разработки. Он был создан для обеспечения дополнительной статической проверки типов, что было бы особенно полезно при разработке крупномасштабных приложений. Одна из причин, по которой Microsoft разработала TypeScript, заключалась в том, что их внутренние команды испытывали проблемы с масштабированием JavaScript для собственных проектов Microsoft.

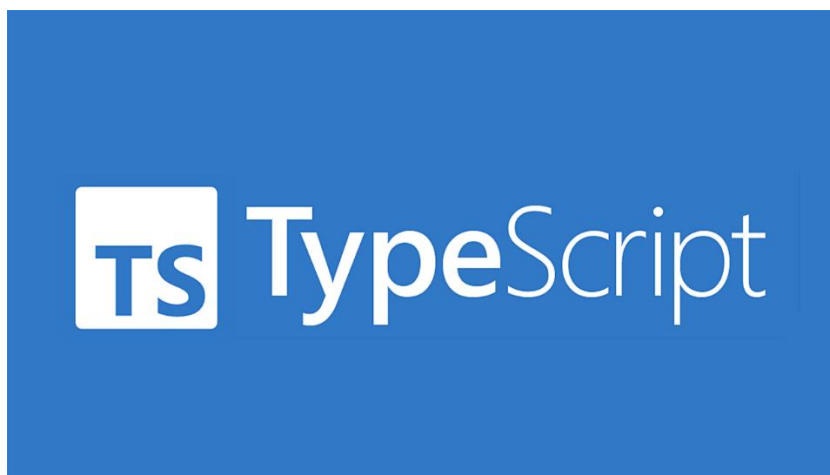


Рисунок 1.8 – Язык программирования TypeScript

TypeScript является открытым исходным кодом, и код написан на самом TypeScript. TypeScript является обратно совместимым с JavaScript и компилируется в последний. Будучи надмножеством, TypeScript обладает всеми функциями JavaScript, а также некоторыми дополнительными функциями.

JavaScript динамически типизируется. Поэтому программы, написанные на JavaScript, не знают тип данных переменной до тех пор, пока этой переменной не будет присвоено значение во время выполнения. Переменная

может быть переназначена или преобразована в значение другого типа без каких-либо проблем или предупреждений. Это может привести к ошибкам, которые часто упускаются из виду, особенно в больших приложениях.

TypeScript использует статическую типизацию. Переменным может быть присвоен тип, когда они объявлены. TypeScript проверяет типы во время компиляции и выдает ошибку, если переменной когда-либо присваивается значение другого типа.

TypeScript [10] поставляется с такими функциями, как улучшенные инструменты времени разработки, статический анализ кода, проверка типов во время компиляции и документация на уровне кода.

Все эти функции, которые поставляются с TypeScript, делают его идеальным языком программирования для создания крупномасштабных приложений JavaScript.

1.2.7 База данных

В качестве системы управления базами данных будет использоваться PostgreSQL.

PostgreSQL [11] — свободная объектно-реляционная система управления базами данных (СУБД). PostgreSQL является одной из наиболее популярных систем управления базами данных. Является мощной системой объектно-реляционных баз данных с открытым исходным кодом, активно разрабатывавшаяся более 30 лет и заслужившая прочную репутацию за надежность и производительность. Текущей версией является версия 14.2. Однако регулярно также выходят подверсии. PostgreSQL поддерживается для всех основных операционных систем – Windows, Linux, MacOS.



Рисунок 1.8 – Система управления БД PostgreSQL

Основными достоинствами PostgreSQL являются:

- надежность;
- производительность (основывается на использовании индексов, интеллектуальном планировщике запросов, тонкой системы блокировок,

системе управления буферами памяти и кэширования, превосходной масштабируемости при конкурентной работе);

- расширяемость (означает, что пользователь может настраивать систему путем определения новых функций, агрегатов, типов, языков, индексов и операторов);

- поддержка SQL;

- поддержка JSON;

- богатый набор типов данных.

1.3 Постановка задачи

Сегодня рынок программного обеспечения выставляет высокие требования ко всем разрабатываемым программным продуктам. Для современных программных средств важными требованиями являются масштабируемость, мультиплатформенность и переносимость. Использование вышеперечисленных технологий при разработке дипломного проекта позволяет сократить время на разработку, увеличить качество кода и за счет этого выполнить данные требования

Объектом исследования является процесс создания объектов и обработки данных, поступающих на сервер. Предметом исследования является создание программного комплекса, обеспечивающего полноценную работу веб-приложения.

Основными требованиями, которые были заложены в основу при разработке программного комплекса дипломного проекта стали: простота использования и расширяемость.

На основании вышесказанного, для разработки программного продукта были определены следующие задачи:

- разработка структуры БД;

- разработка серверной части;

- разработка клиентской части.

Веб-приложение будет представлять из себя сайт, который будет предоставлять следующие функции:

- регистрация пользователей в системе;

- авторизация пользователя в системе;

- создание и оформление новостной ленты;

- создание личного диалога с пользователем;

- добавление комментариев и личной информации пользователя.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Полностью изучив и проанализировав теоретическую часть разрабатываемой системы, получаем список требований необходимых для эффективного и стабильного функционирования разрабатываемого новостного портала. Исходя из полученных требований в целях обеспечения гибкой архитектуры разбиваем систему на функциональные блоки.

В разрабатываемом веб-приложении можно выделить следующие блоки:

- блок администрирования;
- блок взаимодействия с пользователями;
- блок работы сервера;
- блок классов взаимодействия с базой данных;
- блок пользовательского интерфейса;
- блок авторизации пользователя;
- блок реляционной базы данных;
- блок связи подсистем приложения.

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на чертеже ГУИР.400201.111 С1.

Рассмотрим каждый структурный блок подробнее.

2.1 Блок администрирования

Блок администрирования – это часть системы, отвечающая за управление всеми ресурсами сайта. В данном проекте за блок администрирования отвечает ASP.NET Core Identity. ASP.NET Core Identity предоставляет возможность назначать зарегистрированным пользователям роли, с помощью которых каждому отдельному пользователю предоставляются доступные для данной роли действия. Для роли admin в пользовательском интерфейсе открываются дополнительные опции для доступа ко всем ресурсам приложения, а также появляется возможность для проведения любых манипуляций с данными.



Рисунок 2.1 – ASP.NET Core Identity

2.2 Блок взаимодействия с пользователями

Блок взаимодействия с пользователями – это блок, основной задачей которого является отвечать на запросы пользователя. Для реализации данного блока используется библиотека SignalR.

ASP.NET SignalR - это библиотека для ASP.NET разработчиков. Данная библиотека упрощает добавление в приложения компонентов, работающих в реальном времени. Функциональность, работающая в реальном времени – это способность сервера отдать свежие данные подключенным клиентам немедленно, вместо того чтобы ждать пока клиенты запросят эти данные.



Рисунок 2.2 – ASP.NET SignalR

SignalR может быть использован для добавления в ASP.NET приложения любого вида веб-функциональности, работающей в реальном времени.

SignalR имеет простой API для вызовов удаленных процедур от сервера к клиенту (RPC server-to-client), которые вызывают Javascript функции в клиентских браузерах из кода .NET сервера. SignalR также имеет API для управления соединениями и группировкой соединений.

SignalR управляет соединениями автоматически, и отправляет сообщения всем подключенным клиентам одновременно, как в чате.

SignalR приложения могут масштабироваться на тысячах клиентах, используя Service Bus, SQL Server или Redis.

SignalR поставляется с открытым кодом, который доступен на GitHub.

2.3 Блок работы сервера

Блок веб-сервера отвечает за обрабатывание запросов, отправленных клиентом, в качестве серверной части будет использоваться ASP.NET.

ASP.NET — это веб-платформа с открытым исходным кодом для создания веб-приложений на платформе .NET. Она создана Microsoft, и в 2002 году была выпущена версия 1.0, позволяющая разработчикам создавать динамические веб-приложения, службы и сайты.

ASP.NET является преемником технологии ASP (Active Server Pages) и представляет собой значительное обновление с точки зрения гибкости и

мощности. Это расширение платформы .NET с дополнительными инструментами и библиотеками, специально предназначенными для создания вещей в Интернете, включая веб-приложения и веб-сайты.



Рисунок 2.3 – Веб-платформа ASP.NET

ASP.NET основывается на Common Language Runtime: разработчики могут писать код для ASP.NET, используя практически любые языки программирования, некоторые из которых входят в комплект .NET Framework (C#, Visual Basic.NET и JScript .NET), а другие могут быть установлены дополнительно (IronRuby, IronPython, PHP, Perl, Smalltalk, Haskell и др.).

Некоторые особенности ASP.NET:

Компилируемый код выполняется быстрее, а большинство ошибок отлавливается ещё на стадии разработки.

Расширяемый набор элементов управления и библиотек классов, ускоряющий разработку.

Возможность кэширования всей страницы, её частей или данных, используемых на странице.

Возможность разделения визуальной части и бизнес-логики по разным файлам, есть возможность выделять часто используемые шаблоны пользовательских элементов управления, таких как меню сайта, наличие master-страниц для задания шаблонов оформления, поддержка AJAX (расширение ASP.NET AJAX).

Расширяемые модели событий, обработки запросов и серверных элементов управления.

Поддержка CRUD-операций при работе с таблицами через GridView.

ASP.NET включает в себя базовые библиотеки из платформы .NET, а также библиотеки для распространенных веб-шаблонов. Одной из таких библиотек является Model View Controller (MVC), которая позволяет использовать шаблон проектирования MVC для разработки веб-приложений и

сайтов. Шаблон MVC позволяет создать веб-приложение, состоящее из трех ролей — бизнес-уровня, уровня отображения и контроля ввода.

2.4 Блок классов взаимодействия с базой данных

Взаимодействие с базой данных производится через классы `DataContext` и `Migrations`.

`Migrations` — это удобный способ изменить базу данных структурированным и организованным образом. Можно редактировать фрагменты SQL вручную. Миграции поддерживаются всеми бэкендами, если они запрограммированы на поддержку изменения схемы.

Класс `Migration` обрабатывает детали выполнения миграции за пользователя - перебор исходных записей, создание целевых объектов и отслеживание взаимосвязей между ними.

Класс `DataContext` обрабатывает подключение к базе данных. Он также обрабатывает запросы, обновления, вставки в базу данных, отслеживает идентичность, отслеживает изменения, обрабатывает их, обеспечивает целостность транзакций и даже создание базы данных. Класс `DataContext` транслирует запросы сущностных классов в операторы SQL, которые выполняются на подключенной базе данных.

`DataContext` является источником всех сущностей, отображаемых через соединение с базой данных. Он отслеживает изменения, внесенные вами во все извлеченные сущности, и поддерживает «кэш удостоверений», который гарантирует, что сущности, извлеченные более одного раза, представлены с использованием одного и того же экземпляра объекта.

2.5 Блок пользовательского интерфейса

Блок пользовательского интерфейса является клиентской частью веб-приложения. Для реализации данного блока используется фреймворк `Angular`.

`Angular` — это не просто фреймворк, а скорее платформа, которая позволяет разработчикам создавать приложения.

Фреймворк `Angular` содержит множество библиотек, некоторые из которых являются основными.

`Angular` активно поддерживается и имеет большое сообщество и экосистему. Можно найти множество материалов по этому фреймворку, а также множество полезных сторонних инструментов.



Рисунок 2.4 – Экосистема Angular

Angular предоставляет не только инструменты, но и шаблоны проектирования для создания проекта в удобном для сопровождения виде. Angular построен на TypeScript, который, в свою очередь, опирается на JS ES6.

Angular стремился устранить тесную связь между различными компонентами приложения. Внедрение происходит в стиле NodeJS.

Angular предназначен для тщательного тестирования и поддерживает как модульное, так и сквозное тестирование с помощью таких инструментов, как Jasmine и Protractor.

2.6 Блок авторизации пользователя

Блок авторизации пользователя является частью системы, которая отвечает за проверку существования пользователя и в случае его существования в системе, генерирует авторизационный токен для этого пользователя. Для данных целей будет использоваться Microsoft Identity

Платформа Microsoft Identity состоит из нескольких компонентов:

1. Служба аутентификации, совместимая со стандартами OAuth 2.0 и OpenID Connect, позволяющая разработчикам аутентифицировать несколько типов удостоверений, в том числе:

- личная учетная запись Microsoft, такая как Skype, Xbox и Outlook.com;
- рабочие или учебные учетные записи, подготовленные через Azure AD;
- социальные или локальные учетные записи с помощью Azure AD B2C.

2. Библиотеки с открытым исходным кодом: библиотеки проверки подлинности Microsoft (MSAL) и поддержка других библиотек, соответствующих стандартам.

3. Портал управления приложениями: возможность регистрации и настройки на портале Azure, а также другие возможности управления Azure.

4. API конфигурации приложений и PowerShell: программная настройка приложений с помощью API Microsoft Graph и PowerShell, позволяющая автоматизировать задачи DevOps.

5. Контент для разработчиков: техническая документация, включая краткие руководства, учебные пособия, практические руководства и примеры кода.

Для разработчиков платформа Microsoft Identity предлагает интеграцию современных инноваций в области идентификации и безопасности, таких как проверка подлинности без пароля, поэтапная проверка подлинности и условный доступ.

С помощью платформы Microsoft Identity можно написать код один раз и получить доступ к любому пользователю. Можно создать приложение один раз, и оно будет работать на многих платформах, или создать приложение, которое будет функционировать как клиент, а также ресурсное приложение (API).

2.7 Блок реляционной базы данных

Блок базы данных включает данные, используемые веб-приложением. При реализации используется реляционная база данных PostgreSQL.

PostgreSQL поддерживает транзакции со свойствами ACID. Это означает, что транзакции должны поддерживать четыре атрибута:

Атомарность - транзакции считаются завершенными единицами; транзакция может либо полностью завершиться успешно, либо полностью провалиться - в случае неудачи состояние базы данных остается неизменным.

Непротиворечивость - база данных между транзакциями может существовать только в допустимом состоянии; все данные, записываемые в базу данных, должны соответствовать существующим ограничениям, триггерам, каскадам и связанным с ними комбинациям.

Изоляция - функция контроля параллелизма — гарантирует, что данные не будут повреждены незаконными или параллельными транзакциями, поскольку транзакции обрабатываются так, как если бы они происходили последовательно.

Долговечность - гарантирует, что транзакция остается зафиксированной даже в случае сбоя системы - обычно завершенные транзакции записываются, например, в журнал упреждающей записи.

Основными достоинствами PostgreSQL являются:

- надежность;
- производительность (основывается на использовании индексов, интеллектуальном планировщике запросов, тонкой системы блокировок, системе управления буферами памяти и кэширования, превосходной масштабируемости при конкурентной работе);

- расширяемость (означает, что пользователь может настраивать систему путем определения новых функций, агрегатов, типов, языков, индексов и операторов);
- поддержка SQL;
- поддержка JSON;
- богатый набор типов данных;
- простота использования.



Рисунок 2.5 – Система управления БД PostgreSQL

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описывается функционирование и структура разрабатываемого программного средства.

Взаимоотношения между классами разрабатываемого программного обеспечения приведены на диаграмме классов ГУИР.400201.111 РР.1.

3.1 Описание модели данных

3.1.1 Таблица Messages

Данная таблица служит для хранения данных о сообщениях в диалогах. Поля таблицы Messages:

- id – первичный ключ;
- SenderId – идентификатор отправителя;
- SenderUsername – имя отправителя;
- RecipientId – идентификатор получателя;
- RecipientUsername – имя получателя;
- Context – текст;
- DateRead – дата прочтения;
- MessageSent – сообщение отправлено;
- SenderDeleted – отправитель удален;
- RecipientDeleted – пользователь удален.

Колонки id, SenderId, RecipientId имеют тип «integer». Колонки SenderUsername, RecipientUsername, Context имеют тип «text». Колонки DateRead, MessageSent имеют тип timestamp и заполняется автоматически. Колонки SenderDeleted, RecipientDeleted имеют тип «boolean».

3.1.2 Таблица Groups

Данная таблица служит для хранения информации о группах пользователей.

Поля таблицы Groups:

- Name – название группы.

Колонка Name имеет тип «text».

3.1.3 Таблица Connections

Данная таблица хранит информацию о подключившихся пользователях.

Поля таблицы Connections:

- ConnectionId – идентификатор подключения;
- Username – имя пользователя;
- GroupName – название группы.

Колонки ConnectionId, Username, GroupName имеют тип «text».

3.1.4 ТаблицаAspNetUsers

Данная таблица служит для хранения данных о сообщениях в диалогах.

Поля таблицы AspNetUsers:

- Id – первичный ключ;
- UserName – имя;
- NormalizedUserName – нормализованное имя пользователя;
- Email – почта;
- NormalizedEmail – нормализованная почта;
- EmailConfirmed – подтвержденная почта;
- PasswordHash – хэш пароля;
- SecurityStamp – печать безопасности;
- ConcurrencyStamp – печать параллельности;
- PhoneNumber – Номер телефона;
- PhoneNumberConfirmed – Подтвержденный номер телефона;
- TwoFactorEnabled – двухфакторный режим;
- LockoutEnd – блокировка отключена;
- LockoutEnabled – блокировка включена;
- AccessFailedCount – Счетчик неудачных попыток доступа;
- KnownAs – известность;
- Created – создатель;
- LastActive – последний раз активен;
- Gender – пол;
- Introduction – описание;
- Interests – интересы;
- City – город;
- Country – страна.

Колонки Id, AccessFailedCount имеют тип «integer». Колонки PasswordHash, SecurityStamp, ConcurrencyStamp, PhoneNumber, KnownAs, Gender, Introduction, LookingFor, Interests, City, Country имеют тип «text». Колонки LockoutEnd, DateOfBirth, Created, LastActive имеют тип timestamp и заполняется автоматически. Колонки EmailConfirmed, PhoneNumberConfirmed, TwoFactorEnabled, LockoutEnabled имеют тип «boolean». Колонки

UserName, NormalizedEmail, Email, NormalizedEmail имеют тип «varchar».

3.1.5 Таблица **AspNetUserTokens**

Данная таблица хранит информацию о токенах пользователя.

Поля таблицы AspNetUserTokens:

- UserId – идентификатор пользователя;
- LoginProvider – логин пользователя;
- Name – имя;
- Value – значение.

Колонка UserId имеет тип «integer». Колонки LoginProvider, Name, Value имеют тип «text».

3.1.6 Таблица **AspNetUserRoles**

Данная таблица хранит информацию о ролях пользователя.

Поля таблицы AspNetUserRoles:

- UserId – идентификатор пользователя;
- RoleId – логин пользователя.

Колонки UserId, RoleId имеют тип «integer».

3.1.7 Таблица **AspNetUserLogins**

Данная таблица хранит информацию о ролях пользователя.

Поля таблицы AspNetUserLogins:

- UserId – идентификатор пользователя;
- ProviderKey – идентификатор отправителя;
- ProviderDisplayName – отображаемое имя отправителя;
- LoginProvider – логин отправителя.

Колонки ProviderKey, ProviderDisplayName, LoginProvider имеют тип «text». Колонка UserId имеет тип «integer».

3.1.8 Таблица **AspNetUserClaims**

Данная таблица хранит информацию о претензиях пользователя.

Поля таблицы AspNetUserClaims:

- Id – первичный ключ;
- UserId – идентификатор пользователя;
- ClaimType – тип претензии;
- ClaimValue – обоснование претензии.

Колонки Id, UserId имеет тип «integer». Колонки ClaimType, ClaimValue имеют тип «text».

3.1.9 ТаблицаAspNetRoles

Данная таблица хранит информацию о ролях.

Поля таблицы AspNetRoles:

- Id – первичный ключ;
- Name – имя пользователя;
- NormalizedName – настоящее имя;
- ConcurrencyStamp – печать параллельности.

Колонка Id имеет тип «integer». Колонки Name, NormalizedName имеют тип «varchar». Колонка ConcurrencyStamp имеет тип «text».

3.1.10 ТаблицаAspNetRolesClaims

Данная таблица является связующей для таблиц AspNetRoles и AspNetUserClaims.

Поля таблицы AspNetRolesClaims:

- Id – первичный ключ;
- RoleId – идентификатор роли;
- ClaimType – тип претензии;
- ClaimValue – обоснование претензии.

Колонки Id, RoleId имеет тип «integer». Колонки ClaimType, ClaimValue имеют тип «text».

3.1.11 ТаблицаEFMigrationsHistory

Данная таблица является связующей для таблиц AspNetRoles и AspNetUserClaims.

Поля таблицы EFMigrationsHistory:

- MigrationId – идентификатор миграции;
- ProductVersion – версия.

Колонки MigrationId, ProductVersion имеет тип «varchar».

3.1.12 ТаблицаPhotos

Данная таблица служит для хранения фотографий.

Поля таблицы Photos:

- Id – первичный ключ;
- Url – адрес фотографии;

- `IsMain` – отвечает за отображение;
- `PublicId` – публичный ключ;
- `AppUserId` – ключ пользователя в системе.

Поле `Id` будет автоматически заполнено. У различных колонок могут быть разные типы данных. У колонок `Url`, `PublicId` тип данных «text». Данный тип используется для текстовых данных переменной длины без поддержки `Unicode`, такой тип не может служить для хранения двоичных данных. Колонки `Id`, `AppUserId` имеют тип «integer». Колонка `IsMain` имеет тип «boolean».

3.1.13 Таблица `Launches`

Данная таблица служит для хранения данных о запусках.

Поля таблицы `Launches`:

- `Id` – исходный идентификатор запуска;
- `PublicId` – публичный идентификатор запуска.
- `Name` – название запуска;
- `Description` – описание запуска.

Колонка `Id` имеет тип «integer». Колонки `PublicId`, `Name`, `Description` имеют тип «text».

3.2 Описание структуры и взаимодействия между классами

Рассмотрим классы разрабатываемого приложения.

3.2.1 Класс `BaseApiController`

Данный класс-контроллер является главным контроллером приложения, от которого наследуются все остальные контроллеры приложения. `BaseApiController` в свою очередь наследуется от `ControllerBase`, это базовый модуль, который предоставляет большое количество методов для работы с контроллерами.

3.2.2 Класс `AccountController`

Данный класс-контроллер предназначен для регистрации пользователя. Методы класса:

- `Register` – публичный метод экземпляра класса, используется для полной регистрации пользователя в приложении;
- `Login` – публичный метод экземпляра класса, используется для входа пользователя в систему;

– `UserExists` – публичный метод экземпляра класса, используется для проверки существующих пользователей. При попытке создать пользователя с имеющимся уже в системе логином, не даст создать новую запись и выведет предупреждение.

3.2.3 Класс `AdminController`

Данный класс-контроллер предназначен для описания возможностей главного пользователя.

Методы класса:

– `GetUserWithRoles` – публичный метод экземпляра класса, используется для получения списка пользователей с присвоенными им ролями;

– `EditRoles` – публичный метод экземпляра класса, используется для изменения роли пользователя.

3.2.4 Класс `BuggyController`

Данный класс-контроллер предназначен для обработки ошибок.

Методы класса:

– `GetNotFound` – публичный метод экземпляра класса, используется для вывода ошибки поиска. Если информация не найдена по заданному запросу, выводится ошибка;

– `GetServerError` – публичный метод экземпляра класса, используется для вывода ошибки сервера;

– `GetBadRequest` – публичный метод экземпляра класса, используется для вывода ошибки запроса. При некорректном запросе выводит ошибку.

3.2.5 Класс `FallbackController`

Данный класс-контроллер используется для связи с Angular.

Методы класса:

– `Index` – публичный метод экземпляра класса, используется для получения всех позиций, находящихся в приложении и отображения их пользователю.

3.2.6 Класс `LaunchController`

Данный класс-контроллер используется для обработки запросов с запусками.

Методы класса:

- CreateLaunch – публичный метод экземпляра класса, предназначен для создания запуска;
- GetLaunches – публичный метод экземпляра класса, предназначенный для получения списка запусков;
- GetLaunch – публичный метод экземпляра класса, предназначенный для получения запуска;
- UpdateLaunch – публичный метод экземпляра класса, предназначенный для обновления запуска;
- DeleteLaunch – публичный метод экземпляра класса, предназначенный для удаления запуска.

3.2.7 Класс MessageController

Данный класс-контроллер используется для обработки запросов, связанных с сообщениями.

Методы класса:

- GetMessagesForUser – публичный метод экземпляра класса, используется для получения всех сообщений, отправленных пользователем;
- CreateMessage – публичный метод экземпляра класса, используется для создания сообщения;
- DeleteMessage – публичный метод экземпляра класса, используется для удаления сообщения в одностороннем порядке.

3.2.8 Класс UserController

Данный класс-контроллер используется для получения информации о пользователях.

Методы класса:

- GetUsers – публичный метод экземпляра класса, используется для получения списка пользователей в системе;
- GetUser – публичный метод экземпляра класса, используется для получения пользователя по UserName;
- UpdateUser – публичный метод экземпляра класса, используется для изменения и обновления информации в профиле пользователя;
- AddPhoto – публичный метод экземпляра класса, используется для добавления фотографии в профиль пользователя;
- SetMainPhoto – публичный метод экземпляра класса, используется для установки выбранной фотографии главной в профиле;
- DeletePhoto – публичный метод экземпляра класса, используется для удаления выбранной фотографии из профиля.

3.2.9 Класс DataContext

Данный класс используется для построения моделей на основе базы данных.

Методы класса:

- OnModelCreating – публичный метод экземпляра класса, используется для создания отношений между таблицами базы данных.

3.2.10 Класс LaunchRepository

Данный класс используется для получения информации о лайках пользователя.

Методы класса:

- Create – публичный метод экземпляра класса, предназначен для создания запуска;
- GetLaunches – публичный метод экземпляра класса, предназначенный для получения списка запусков;
- GetLaunch – публичный метод экземпляра класса, предназначенный для получения запуска;
- Update – публичный метод экземпляра класса, предназначенный для обновления запуска;
- Delete – публичный метод экземпляра класса, предназначенный для удаления запуска.

3.2.11 Класс MessageRepository

Данный класс используется для получения информации о сообщениях пользователя.

Методы класса:

- AddGroup – публичный метод экземпляра класса, используется для создания новой группы пользователей. На данном этапе реализации под группой понимается, диалог из двух пользователей. Но данную функцию можно расширять;
- AddMessage – публичный метод экземпляра класса, используется для добавления сообщения;
- DeleteMessage – публичный метод экземпляра класса, используется для удаления сообщения;
- GetConnections – публичный метод экземпляра класса, используется для получения соединения;
- GetGroupForConnections – публичный метод экземпляра класса, используется для получения соединения группы;

- GetMessage – публичный метод экземпляра класса, используется для получения сообщения;
- GetMessageGroup – публичный метод экземпляра класса, используется для получения сообщения группы;
- GetMessageForUser – публичный метод экземпляра класса, используется для получения сообщения определенного пользователя;
- GetMessageThread – публичный метод экземпляра класса, используется для получения списка сообщений;
- RemoveConnection – публичный метод экземпляра класса, используется для удаления соединения;

3.2.12 Класс Seed

Данный класс используется для загрузки данных в базу данных при первом запуске.

Методы класса:

- SeedUsers – публичный метод экземпляра класса, используется для добавления начальных записей о пользователе в базу данных.

3.2.13 Класс UnitOfWork

Данный класс используется для описания надстроек над репозиториями.

Методы класса:

- Complete – публичный метод экземпляра класса, используется для сохранения изменения внесенных в базу данных;
- HasChanges – публичный метод экземпляра класса, используется для проверки базы данных на изменения.

3.2.14 Класс UserRepository

Данный класс используется для получения данных о пользователе.

Методы класса:

- GetMemberAsync – публичный метод экземпляра класса, используется для получения определенного пользователя;
- GetMembersAsync – публичный метод экземпляра класса, используется для получения списка пользователей;
- GetUsersByUsernameAsync – публичный метод экземпляра класса, используется для получения пользователя по имени;
- GetUserByIdAsync – публичный метод экземпляра класса, используется для получения пользователя по идентификатору;
- GetUserGender – публичный метод экземпляра класса, используется для получения пользователя по полу;

- `GetUsersAsync` – публичный метод экземпляра класса, используется для получения фото пользователей;
- `Update` – публичный метод экземпляра класса, используется для обновления данных о пользователе.

3.2.15 Класс `CreateMessageDto`

Данный класс используется для отправки сообщений между пользователями.

3.2.16 Класс `LaunchDto`

Данный класс используется для отображения запусков.

3.2.17 Класс `LoginDto`

Данный класс используется для авторизации пользователей.

3.2.18 Класс `MemberDto`

Данный класс используется для получения информации о пользователе.

3.2.19 Класс `MemberUpdateDto`

Данный класс используется для обновления информации о пользователе.

3.2.20 Класс `MessageDto`

Данный класс используется для хранения информации о диалогах пользователей.

3.2.21 Класс `PhotoDto`

Данный класс используется для хранения информации о фотографиях пользователей.

3.2.22 Класс `RegisterDto`

Данный класс используется для авторизации пользователей.

3.2.23 Класс UserDto

Данный класс используется для получения информации о пользователях.

3.2.24 Класс AppRole

Данный класс, реализующий связь с базой данных, используется для хранения информации о роли пользователя.

3.2.25 Класс AppUser

Данный класс, реализующий связь с базой данных, используется для хранения информации о пользователях.

3.2.26 Класс Connection

Данный класс, реализующий связь с базой данных, используется для создания соединения в реальном времени между пользователями.

3.2.27 Класс Group

Данный класс, реализующий связь с базой данных, используется для объединения пользователей в группы.

3.2.28 Класс Message

Данный класс, реализующий связь с базой данных, используется для хранения информации о диалогах пользователей.

3.2.29 Класс Photo

Данный класс, реализующий связь с базой данных, используется для хранения информации о фотографиях пользователей.

3.2.30 Класс UserLike

Данный класс, реализующий связь с базой данных, используется для хранения информации о понравившихся пользователях.

3.2.31 Класс ApiException

Данный класс используется для обработки ошибок.

3.2.32 Класс `ApplicationServiceExstensions`

Данный класс используется для конфигурирования сущностей приложения.

Метод класса:

- `AddApplicationServices` – публичный метод экземпляра класса, используется для подключения сервисов.

3.2.33 Класс `ClaimsPrincipleExtensions`

Данный класс используется для получения информации о текущем пользователе.

Методы класса:

- `GetUsername` – публичный метод экземпляра класса, используется для получения логина текущего пользователя;
- `GetUserId` – публичный метод экземпляра класса, используется для получения идентификатора текущего пользователя.

3.2.34 Класс `DateTimeExtensions`

Данный класс используется для расширения функциональности типа данных `DateTime`.

Метод класса:

- `CalculateAge` – публичный метод экземпляра класса, используется для расчета возраста пользователя.

3.2.35 Класс `HttpExtensions`

Данный класс используется для расширения функциональности протокола HTTP.

Метод класса:

- `AddPagingHeader` – публичный метод экземпляра класса, используется для добавления заголовков в запрос.

3.2.36 Класс `IdentityServiceExtensions`

Данный класс реализован для использования системного администрирования пользователей.

Метод класса:

- `AddIdentityServices` – публичный метод экземпляра класса, используется для добавления правил верификации.

3.2.37 Класс QueryableExtensions

Данный класс используется для расширения функциональности системного интерфейса.

Метод класса:

– MarkUnreadAsRead – публичный метод экземпляра класса, используется для изменения статуса сообщения.

3.2.38 Класс AutoMapperProfiles

Данный класс используется для автоматического приведения типов.

3.2.39 Класс CloudinarySettings

Данный класс используется для хранения настроек сервиса Cloudinary.

3.2.40 Класс LogUserActivity

Данный класс используется для отображения активного статуса пользователя.

Метод класса:

– OnActionExecutionAsync – публичный метод экземпляра класса, используется для изменения статуса пользователя.

3.2.41 Класс MessageParams

Данный класс используется для пагинации сообщений.

3.2.42 Класс PagedList

Данный класс используется для хранения информации в постраничном формате.

Метод класса:

– CreateAsync – публичный метод экземпляра класса, используется для создания постраничной коллекции данных.

3.2.43 Класс PaginationHeader

Данный класс используется для хранения информации о конфигурации постраничного формата.

3.2.44 Класс `PaginationParams`

Данный класс используется для запроса на получение информации в постраничном формате.

3.2.45 Класс `UserParams`

Данный класс используется для запроса на получение пользователей в постраничном формате.

3.2.46 Интерфейс `ILikesRepository`

Данный интерфейс используется для описания функциональности работы с лайками.

3.2.47 Интерфейс `IMessageRepository`

Данный интерфейс используется для описания функциональности работы с сообщениями.

3.2.48 Интерфейс `IPhotoService`

Данный интерфейс используется для описания функциональности работы с фотографиями.

3.2.49 Интерфейс `ITokenService`

Данный интерфейс используется для описания функциональности работы с токенами.

3.2.50 Интерфейс `IUnitOfWork`

Данный интерфейс используется для описания функциональности работы с репозиториями.

3.2.51 Интерфейс `IUserRepository`

Данный интерфейс используется для описания функциональности работы с пользователями.

3.2.52 Класс **ExceptionHandlerMiddleware**

Данный класс используется для предоставления информации об ошибках.

Метод класса:

- `InvokeAsync` – публичный метод экземпляра класса, используется для вывода информации об ошибке.

3.2.53 Класс **PhotoService**

Данный класс используется для работы с фотографиями.

Методы класса:

- `AddPhotoAsync` – публичный метод экземпляра класса, используется для добавления фотографий;
- `DeletePhotoAsync` – публичный метод экземпляра класса, используется для удаления фотографий.

3.2.54 Класс **TokenService**

Данный класс используется для работы с токенами.

Метод класса:

- `CreateToken` – публичный метод экземпляра класса, используется для создания токена.

3.2.55 Класс **MessageHub**

Данный класс используется для обработки сообщений.

Методы класса:

- `OnConnectedAsync` – публичный метод экземпляра класса, используется для создания соединения;
- `OnDisconnectedAsync` – публичный метод экземпляра класса, используется для разрыва соединения;
- `SendMessage` – публичный метод экземпляра класса, используется для отправки сообщения;
- `AddToGroup` – публичный метод экземпляра класса, используется для добавления пользователя в группу;
- `RemoveFromMessageGroup` – публичный метод экземпляра класса, используется для удаления пользователя из группы;
- `GetGroupName` – публичный метод экземпляра класса, используется для получения уникального имени группы.

3.2.56 Класс **PresenceHub**

Данный класс используется для отслеживания состояния пользователя.
Методы класса:

- `OnConnectedAsync` – публичный метод экземпляра класса, используется для создания соединения;
- `OnDisconnectedAsync` – публичный метод экземпляра класса, используется для разрыва соединения.

3.2.57 Класс **PresenceTracker**

Данный класс используется для уведомления пользователя об активном состоянии других пользователей.

Методы класса:

- `UserConnected` – публичный метод экземпляра класса, используется для создания пользовательского соединения;
- `UserDisconnected` – публичный метод экземпляра класса, используется для разрыва пользовательского соединения;
- `GetOnlineUsers` – публичный метод экземпляра класса, используется для получения списка онлайн пользователей;
- `GetConnectionsForUser` – публичный метод экземпляра класса, используется для получения списка использующихся соединений.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Расширение функциональности системных модулей

При разработке программного продукта часто используются уже готовые системные решения. Но иногда функциональности этих решений недостаточно. С этой целью расширения этой функциональности были разработаны дополнительные модули. Модули расширений представлены тремя классами: `ClaimsPrincipalExtensions`, `HttpExtensions`, `IdentityServiceExtensions`. Эти три класса расширяют функциональность существующего системного типа данных. Каждый класс расширений является статическим. Соответственно, и все его методы тоже будут статическими, и в качестве входного параметра будут принимать ссылку на существующий системный тип данных.

Первое из представленных расширений используется для получения информации о текущем пользователе при помощи `Claims`. Он реализует методы получения логина и идентификатора активного пользователя системы.

```
public static string GetUsername(this ClaimsPrincipal user)
{
    return user.FindFirst(ClaimTypes.Name)?.Value;
}

public static int GetUserId(this ClaimsPrincipal user)
{
    return
        int.Parse(user.FindFirst(ClaimTypes.NameIdentifier)?.Value);
}
```

Следующее расширение – `HttpExtensions`. Данный модуль используется для расширения функциональности протокола HTTP. Он содержит в себе метод, используемый для добавления заголовков пагинации в запрос.

```
var paginationHeader = new PaginationHeader(currentPage,
    itemsPerPage, totalItems, totalPages);

var options = new JsonSerializerOptions
{
    PropertyNamingPolicy = JsonNamingPolicy.CamelCase
};

response.Headers.Add("Pagination",
    JsonSerializer.Serialize(paginationHeader, options));
response.Headers.Add("Access-Control-Expose-Headers",
    "Pagination");
```

Следующее расширение – IdentityServiceExtensions. Данное расширение реализован с целью системного администрирования пользователей. Данный модуль отвечает за создания JWT токена авторизации пользователя и присвоения ему роли.

```
options.Events = new JwtBearerEvents
{
    OnMessageReceived = context =>
    {
        var accessToken =
context.Request.Query["access_token"];

        var path = context.HttpContext.Request.Path;

        if (!string.IsNullOrEmpty(accessToken) &&
path.StartsWithSegments("/hubs"))
        {
            context.Token = accessToken;
        }

        return Task.CompletedTask;
    }
};
```

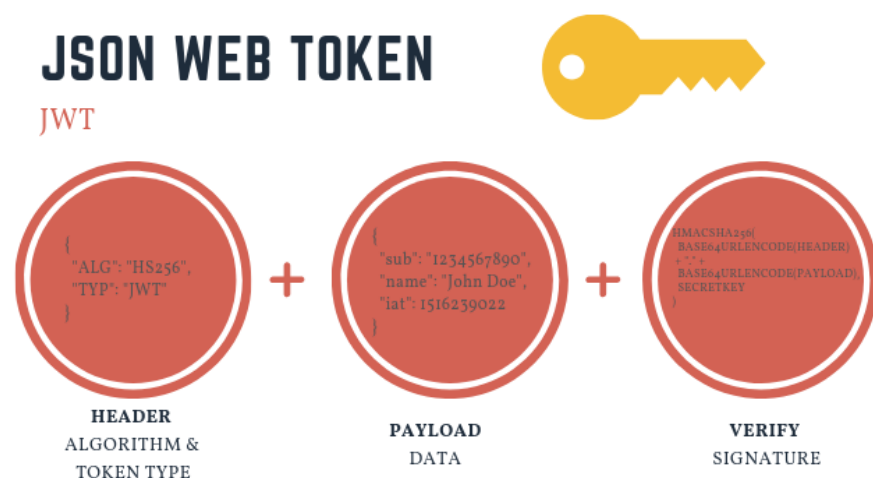


Рисунок 4.1 – Структура JWT токена

4.2 Выполнение HTTP-методов

HTTP-методы в приложении обрабатываются классами Controllers. К основным классам этой группы относятся: AccountController, MessagesController, UsersController. Эти три класса наследуются от абстрактного класса ControllerBase, который реализует основные методы для обработки запросов.

Первый из представленных классов предназначен для авторизации пользователя в системе. Он реализует методы Register и Login для обработки HTTP-запросов.



Рисунок 4.2 – Структура HTTP запроса

Метод Register используется для первоначальной регистрации пользователя в системе.

```
[HttpPost("register")]
public async Task<ActionResult<UserDto>>
Register(RegisterDto registerDto)
{
    if (await UserExists(registerDto.Username)) return
BadRequest("Username is taken");

    var user = _mapper.Map<AppUser>(registerDto);

    user.UserName = registerDto.Username.ToLower();

    var result = await _userManager.CreateAsync(user,
registerDto.Password);

    if (!result.Succeeded) return BadRequest(result.Errors);

    var roleResult = await _userManager.AddToRoleAsync(user,
"Member");

    if (!roleResult.Succeeded) return
BadRequest(roleResult.Errors);

    return new UserDto
```

```

        {
            Username = user.UserName,
            Token = await _tokenService.CreateToken(user),
            KnownAs = user.KnownAs,
            Gender = user.Gender
        };
    }
}

```

Метод Login используется для входа пользователя в систему.

```

[HttpPost("login")]
public async Task<ActionResult<UserDto>> Login(LoginDto loginDto)
{
    var user = await _userManager.Users
        .Include(p => p.Photos)
        .SingleOrDefaultAsync(x => x.UserName ==
loginDto.Username.ToLower());

    if (user == null) return Unauthorized("Invalid
username");

    var result = await
_signInManager.CheckPasswordSignInAsync(user,
loginDto.Password, false);

    if (!result.Succeeded) return Unauthorized();

    return new UserDto
    {
        Username = user.UserName,
        Token = await _tokenService.CreateToken(user),
        PhotoUrl = user.Photos.FirstOrDefault(x =>
x.IsMain)?.Url,
        KnownAs = user.KnownAs,
        Gender = user.Gender
    };
}

```

Следующий контроллер – MessagesController. Данный класс-контроллер используется для обработки запросов, относящихся к сообщениям. Он содержит в себе методы, используемые для создания, получения и удаления сообщений пользователя.

```

[HttpGet]
public async Task<ActionResult<IEnumerable<MessageDto>>>
GetMessagesFromUser(
    [FromQuery] MessageParams messageParams)
{

```



```

        messageParams.Username = User.GetUsername();

        var messages = await
            _unitOfWork.MessageRepository.GetMessagesForUser(messageParams);

        Response.AddPaginationHeader(messages.CurrentPage,
            messages.PageSize, messages.TotalCount,
            messages.TotalPages);

        return Ok(messages);
    }

```

Следующий класс-контроллер – `UserController`. Данный класс реализован с обработки запросов получения и изменения информации о пользователях.

```

[HttpPut]
public async Task<ActionResult> UpdateUser(MemberUpdateDto
memberUpdateDto)
{
    var user = await
        _unitOfWork.UserRepository.GetUserByUsernameAsync(User.GetUs
        ername());

    _mapper.Map(memberUpdateDto, user);

    _unitOfWork.UserRepository.Update(user);

    if (await _unitOfWork.Complete()) return NoContent();

    return BadRequest("Failed to update user");
}

```

4.3 Работа с базой данных посредством репозиторий

Для выполнения задач, связанных с обработкой данных, были разработаны классы репозиторий. Данные классы являются имплементацией `Microsoft Entity Framework` и предназначены для выполнения CRUD-операций над информацией в базе данных.

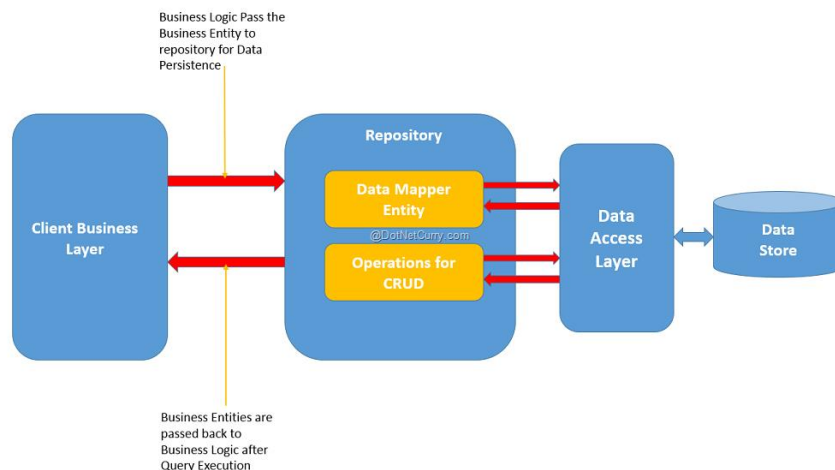


Рисунок 4.3 – Паттерн репозитория

Одним из представителей репозитория является класс `MessageRepository`. Примером одного из его основных методов является `GetMessagesForUser`.

В первую очередь создается переменная запроса списка сообщений пользователя к базе данных посредством класса `DataContext`.

```

var query = _context.Messages
    .OrderByDescending(m => m.MessageSent)
    .ProjectTo<MessageDto>(_mapper.ConfigurationProvider)
    .AsQueryable();

```

Затем применяется выборка по сообщениям на основе параметров поиска.

```

query = messageParams.Container switch
{
    "Inbox" => query.Where(u => u.RecipientUsername ==
messageParams.Username && u.RecipientDeleted == false),
    "Outbox" => query.Where(u => u.SenderUsername ==
messageParams.Username && u.SenderDeleted == false),
    _ => query.Where(u => u.RecipientUsername ==
messageParams.Username && u.DateRead == null &&
u.RecipientDeleted == false)
};

```

После этого получаем пагинационный список сообщений текущего пользователя.

```

return await PagedList<MessageDto>.CreateAsync(query,
messageParams.PageNumber, messageParams.PageSize);

```

4.4 Работа микросервисов

При разработке микросервисов применён подход с использованием паттерна `Scoped`. Это позволит уменьшить количество создаваемых объектов и тем самым сократить использование ресурсов.

Каждый микросервис реализует свой интерфейс. Интерфейсы позволяют повысить гибкость приложения в данном случае.

К основным сервисам данного приложения относятся: `PhotoService`, `TokenService`.

Первый из представленных классов предназначен для работы с фотографиями. Он реализует методы `AddPhotoAsync` и `DeletePhotoAsync` для обработки HTTP-запросов, связанных с изменениями фотографий.

Второй из представленных классов предназначен для работы с JWT токенами. Он реализует метод `CreateToken` для создания токена при помощи HTTP-запроса.

4.5 Работа `DataContext` и миграций

Для взаимодействия с базой данных репозитории используют `DataContext` и классы миграций. Данные классы также, как и микросервисы разработаны с использованием паттерна `Scoped` и реализуют соответствующие интерфейсы.

В классе `DataContext` содержится информация о коллекциях базы данных и их отношениях между собой. Основным методом данного класса является `OnModelCreating`. Он предназначен для описания взаимодействия сущностей базы данных.

```
protected override void OnModelCreating(ModelBuilder
builder)
{
    base.OnModelCreating(builder);

    builder.Entity<Message>()
        .HasOne(u => u.Recipient)
        .WithMany(m => m.MessagesReceived)
        .OnDelete(DeleteBehavior.Restrict);
}
```

Используемые в данном проекте миграции предназначены для непосредственной работы с базой данных посредством SQL-запросов и LINQ.

```
migrationBuilder.CreateTable(
    name: "Groups",
    columns: table => new
```

```

    {
        Name = table.Column<string>(type: "text", nullable:
false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Groups", x => x.Name);
    });

```

4.6 Работа SignalR

SignalR Core представляет библиотеку от компании Microsoft, которая предназначена для создания приложений, работающих в режиме реального времени. SignalR использует двунаправленную связь для обмена сообщениями между клиентом и сервером, благодаря чему сервер может отправлять в режиме реального времени всем клиентам некоторые данные.

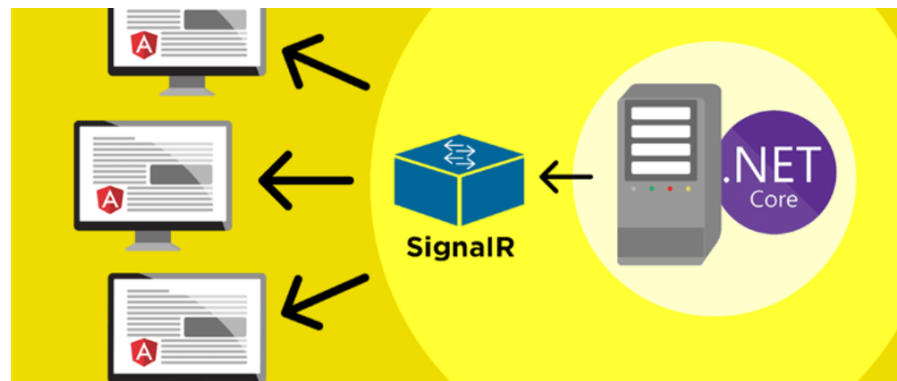


Рисунок 4.4 – Библиотека SignalR

Исходя из возможностей клиента и сервера инфраструктура SignalR выбирает наилучший механизм для взаимодействия. В частности, наиболее оптимальным является WebSockets, соответственно если и клиент, и сервер позволяют использовать этот механизм, то взаимодействие идет через WebSockets. Однако если WebSockets не поддерживается, то применяется Server-Side Events. И если SSE не поддерживается, то применяется Long Polling.

В данном проекте SignalR реализован при помощи группы классов типа Hub. Одним из них является MessageHub, который используется для обмена сообщениями между пользователями системы. В классе реализованы все основные методы обработки сообщений. Одним из которых является SendMessage, предназначенный для отправки сообщений.

```

var group = await
_unitOfWork.MessageRepository.GetMessageGroup(groupName);

```

```

if (group.Connections.Any(x => x.Username ==
recipient.UserName))
{
    message.DateRead = DateTime.UtcNow;
}
else
{
    var connections = await
_tracker.GetConnectionsForUser(recipient.UserName);
    if (connections != null)
    {
        await
_presenceHub.Clients.Clients(connections).SendAsync("NewMess
ageReceived", new
        {
            username = sender.UserName,
            knownAs = sender.KnownAs
        });
    }
}

_unitOfWork.MessageRepository.AddMessage(message);

if (await _unitOfWork.Complete())
{
    await Clients.Group(groupName).SendAsync("NewMessage",
_mapper.Map<MessageDto>(message));
}

```

4.7 Работа вспомогательных классов

4.7.1 Вспомогательный класс AutoMapperProfiles

В данном проекте при отправке данных между клиентом и сервером используются специализированные классы DTO (Data Transfer Object). Для создания объектов DTO необходимо преобразовать информацию из базы данных для ее дальнейшего отображения.

Основным методом создания соответствия между объектами является CreateMap.

```

public AutoMapperProfiles()
{
    CreateMap<AppUser, MemberDto>()
        .ForMember(dest => dest.PhotoUrl,
            opt => opt.MapFrom(
                src => src.Photos.FirstOrDefault(x =>
x.IsMain).Url))
        .ForMember(dest => dest.Age,
            opt => opt.MapFrom(
                src => src.DateOfBirth.CalculateAge()));
}

```

```

CreateMap<Photo, PhotoDto>();

CreateMap<MemberUpdateDto, AppUser>();

CreateMap<RegisterDto, AppUser>();

CreateMap<Message, MessageDto>()
    .ForMember(dest => dest.SenderPhotoUrl,
        opt => opt.MapFrom(
            src => src.Sender.Photos.FirstOrDefault(x =>
x.IsMain).Url))
    .ForMember(dest => dest.RecipientPhotoUrl,
        opt => opt.MapFrom(
            src => src.Recipient.Photos.FirstOrDefault(x
=> x.IsMain).Url));
}

```

4.7.2 Вспомогательные классы **Pagination**

Для получения информации от сервера в порционном виде используются классы пагинаций: `PaginationHeader` и `PagedList`.

Класс `PagedList` предназначен для получения и хранения информации в порционном виде.

```

public PagedList(IEnumerable<T> items, int count, int
pageNumber, int pageSize)
{
    CurrentPage = pageNumber;
    TotalPages = (int) Math.Ceiling(count / (double)
pageSize);
    PageSize = pageSize;
    TotalCount = count;
    AddRange(items);
}

public static async Task<PagedList<T>>
CreateAsync(IQueryable<T> source, int pageNumber,
    int pageSize)
{
    var count = await source.CountAsync();
    var items = await source.Skip((pageNumber - 1) *
pageSize).Take(pageSize).ToListAsync();
    return new PagedList<T>(items, count, pageNumber,
pageSize);
}

```

Класс `PaginationHeader` предназначен для хранения информации о конфигурации построчного формата.

```

public PaginationHeader(int currentPage, int itemsPerPage,
int totalItems, int totalPages)
{
    CurrentPage = currentPage;
    ItemsPerPage = itemsPerPage;
    TotalItems = totalItems;
    TotalPages = totalPages;
}

```

4.7.3 Обработка ошибок

Все ошибки в данном приложении обрабатываются при помощи классов `ExceptionHandlerMiddleware` и `ApiException`.

Класс `ExceptionHandlerMiddleware` предназначен для предоставления информации об ошибках. Основным методом данного класса является — `InvokeAsync`. В данном методе вызывается следующий делегат жизненного цикла приложения.

```

try
{
    await _next(context);
}

```

При возникновении исключительной ситуации выполняется логирование и обработка ошибки.

```

_logger.LogError(ex, ex.Message);
context.Response.ContentType = "application/json";
context.Response.StatusCode = (int)
    HttpStatusCode.InternalServerError;

var response = _env.IsDevelopment()
    ? new ApiException(context.Response.StatusCode,
        ex.Message, ex.StackTrace?.ToString())
    : new ApiException(context.Response.StatusCode,
        "Internal Server Error");

var options = new JsonSerializerOptions
{PropertyNamingPolicy = JsonNamingPolicy.CamelCase};

var json = JsonSerializer.Serialize(response, options);

await context.Response.WriteAsync(json);

```

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

В процессе разработки приложения были разработаны модульные и интеграционные тесты, проведено тестирование приложения в различных сценариях использования. На клиентской части было проведено также ручное тестирование.

При тестировании применялся фреймворк xUnit. xUnit.net — это бесплатный, ориентированный на разработчиков инструмент тестирования с открытым исходным кодом для .NET Framework. xUnit.net — это новейшая технология модульного тестирования C#, F#, VB.NET и других языков .NET. xUnit.net работает с ReSharper, CodeRush, TestDriven.NET и Xamarin. Он является частью .NET Foundation и действует в соответствии с их кодексом поведения. Он распространяется под лицензией Apache 2 (лицензия, одобренная OSI).

Для тестирования в случаях взаимодействия с базой данных и другими сервисами применялся фреймворк Moq. Moq (произносится как «Mock-you» или просто «Mock») — единственная библиотека имитации для .NET, разработанная с нуля, чтобы в полной мере использовать выражения LINQ и лямбда-выражения, что делает ее наиболее производительной, типобезопасной и доступной мокационной библиотекой, удобной для рефакторинга. И она поддерживает фиктивные интерфейсы, а также классы. Ее API чрезвычайно прост и понятен, и не требует каких-либо предварительных знаний или опыта работы с концепциями имитации. Moq разработан как очень практичный, ненавязчивый и простой способ быстрой настройки зависимостей для тестов. Его дизайн API помогает даже начинающим пользователям избежать наиболее распространенных ошибок разработчиков.

5.1 Модульное тестирование

Модульное тестирование — это метод тестирования программного обеспечения, при котором тестируются модули — отдельные компоненты программного обеспечения. Разработчики пишут модульные тесты для своего кода, чтобы убедиться, что код работает правильно. Это помогает обнаруживать ошибки и защищаться от них в будущем.

Иногда разработчики сначала пишут модульные тесты, а затем пишут код. Этот подход также известен как разработка через тестирование (TDD). В TDD требования превращаются в конкретные тестовые случаи, затем программное обеспечение совершенствуется, чтобы пройти новые тесты. При таком подходе не добавляется код, соответствие которого не доказано. Модульное тестирование похоже на то, что оно позволяет разработчикам изменять код, не влияя на функциональность других модулей или продукта в целом.

Модульное тестирование проводилось для каждого контроллера, репозитория и микросервиса. Все классы успешно прошли тесты, связанные с их функционалом (см. рисунки 5.1 – 5.3). Отдельно стоит выделить тесты валидаторов, которые обеспечивают основу правильного функционирования контроллеров и достоверность приходящей в систему информации.

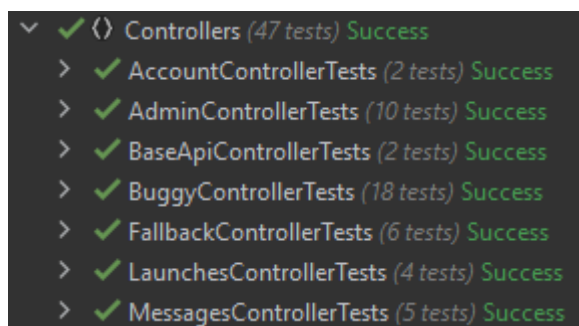


Рисунок 5.1 – Результаты тестирования контроллеров

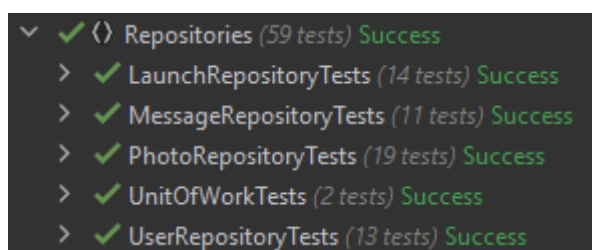


Рисунок 5.2 – Результаты тестирования репозиторий

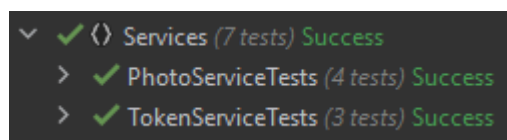


Рисунок 5.3 – Результаты тестирования микросервисов

Результаты тестирования валидаторов представлены в таблице 5.1.

Таблица 5.1 – Результаты тестирования валидаторов

Тестируемый параметр	Содержание теста	Ожидаемый результат	Тест пройден
1	2	3	4
Id	Верные данные	Отображение информации об успехе теста	Да
Id	Параметр представлен пустой строкой	Отображение информации об ошибке	Да
Id	Параметр представлен строкой пробелов	Отображение информации об ошибке	Да

Продолжение таблицы 5.1

1	2	3	4
Id	Параметр представлен куском кода (скриптом)	Отображение информации об ошибке	Да
Логин пользователя	Верные данных	Отображение информации об успехе теста	Да
Логин пользователя	Параметр представлен пустой строкой	Отображение информации об ошибке	Да
Логин пользователя	Параметр представлен строкой пробелов	Отображение информации об ошибке	Да
Логин пользователя	Параметр представлен куском кода (скриптом)	Отображение информации об ошибке	Да
Адрес электронной почты	Верные данных	Отображение информации об успехе теста	Да
Адрес электронной почты	Параметр представлен строкой, содержащей синтаксические ошибки	Отображение информации об ошибке	Да
Адрес электронной почты	Параметр представлен пустой строкой	Отображение информации об ошибке	Да
Адрес электронной почты	Параметр представлен строкой пробелов	Отображение информации об ошибке	Да
Адрес электронной почты	Параметр представлен куском кода (скриптом)	Отображение информации об ошибке	Да
Имя пользователя	Параметр содержит синтаксическую ошибку	Отображение информации об ошибке	Да
Имя пользователя	Параметр представлен пустой строкой	Отображение информации об ошибке	Да
Имя пользователя	Верные данных	Отображение информации об успехе теста	Да
Имя пользователя	Параметр представлен строкой пробелов	Отображение информации об ошибке	Да
Имя пользователя	Параметр представлен куском кода (скриптом)	Отображение информации об ошибке	Да
Описание пользователя	Верные данных	Отображение информации об успехе теста	Да
Описание пользователя	Параметр представлен пустой строкой	Отображение информации об ошибке	Да
Описание пользователя	Параметр представлен строкой пробелов	Отображение информации об ошибке	Да
Описание пользователя	Параметр представлен куском кода (скриптом)	Отображение информации об ошибке	Да
Пароль	Верные данных	Отображение информации об успехе теста	Да

Продолжение таблицы 5.1

1	2	3	4
Пароль	Параметр представлен строкой, превышающей граничное значение длины	Отображение информации об ошибке	Да
Пароль	Параметр представлен строкой, длина которой меньше минимальной	Отображение информации об ошибке	Да
Пароль	Параметр представлен пустой строкой	Отображение информации об ошибке	Да
Пароль	Параметр представлен строкой пробелов	Отображение информации об ошибке	Да
Пароль	Параметр представлен куском кода (скриптом)	Отображение информации об ошибке	Да
Город	Верные данных	Отображение информации об успехе теста	Да
Город	Параметр представлен пустой строкой	Отображение информации об ошибке	Да
Город	Параметр представлен строкой пробелов	Отображение информации об ошибке	Да
Город	Параметр представлен куском кода (скриптом)	Отображение информации об ошибке	Да
Страна	Верные данных	Отображение информации об успехе теста	Да
Страна	Параметр представлен пустой строкой	Отображение информации об ошибке	Да
Страна	Параметр представлен строкой пробелов	Отображение информации об ошибке	Да
Страна	Параметр представлен куском кода (скриптом)	Отображение информации об ошибке	Да
Интересы пользователя	Верные данных	Отображение информации об успехе теста	Да
Интересы пользователя	Параметр представлен пустой строкой	Отображение информации об ошибке	Да
Интересы пользователя	Параметр представлен строкой пробелов	Отображение информации об ошибке	Да
Интересы пользователя	Параметр представлен куском кода (скриптом)	Отображение информации об ошибке	Да
Комментарий	Верные данных	Отображение информации об успехе теста	Да
Комментарий	Параметр представлен пустой строкой	Отображение информации об ошибке	Да
Комментарий	Параметр представлен строкой пробелов	Отображение информации об ошибке	Да
Комментарий	Параметр представлен куском кода (скриптом)	Отображение информации об ошибке	Да

Продолжение таблицы 5.1

1	2	3	4
Повторный пароль при регистрации	Верные данных	Отображение информации об успехе теста	Да
Повторный пароль при регистрации	Пароль не совпадает с введённым ранее паролем	Отображение информации об ошибке	Да

5.2 Функциональное тестирование

Функциональное тестирование — это тип тестирования программного обеспечения, которое проверяет программную систему на соответствие функциональным требованиям/спецификациям. Целью функциональных тестов является тестирование каждой функции программного приложения путем предоставления соответствующих входных данных и проверки выходных данных на соответствие функциональным требованиям.

Функциональное тестирование в основном включает в себя тестирование черного ящика и не касается исходного кода приложения. Это тестирование проверяет пользовательский интерфейс, API, базу данных, безопасность, связь между клиентом и сервером и другие функции тестируемого приложения. Тестирование может проводиться как вручную, так и с помощью автоматизации.

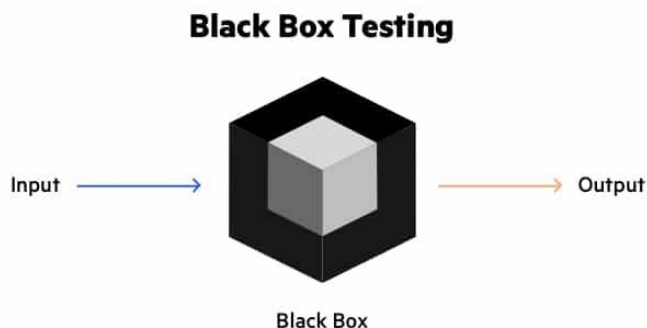


Рисунок 5.4 – Тестирование черного ящика

Основной целью функционального тестирования является проверка функциональных возможностей программной системы. Этот вид тестирования в основном концентрируется на:

- основных функциях: тестирование основных функций приложения;
- базовом удобстве использования: включает в себя базовое тестирование удобства использования системы;
- доступности: проверяет доступность системы для пользователя;
- условиях ошибки: использование методов тестирования для проверки условий ошибки.

5.3 Интеграционное тестирование

Интеграционное тестирование — это тип тестирования, при котором программные модули логически интегрируются и тестируются как группа. Типичный программный проект состоит из нескольких программных модулей, написанных разными программистами. Целью этого уровня тестирования является выявление дефектов взаимодействия между этими программными модулями при их интеграции.

Интеграционное тестирование жизненно важно в современных средах разработки ИТ и программного обеспечения, особенно когда требования динамичны, а сроки сжаты. Даже когда каждый модуль приложения проходит модульное тестирование, некоторые ошибки все равно могут существовать. Чтобы выявить эти ошибки и убедиться, что модули хорошо работают вместе после интеграции, интеграционное тестирование необходимо осуществлять обязательно.

Types of integration testing

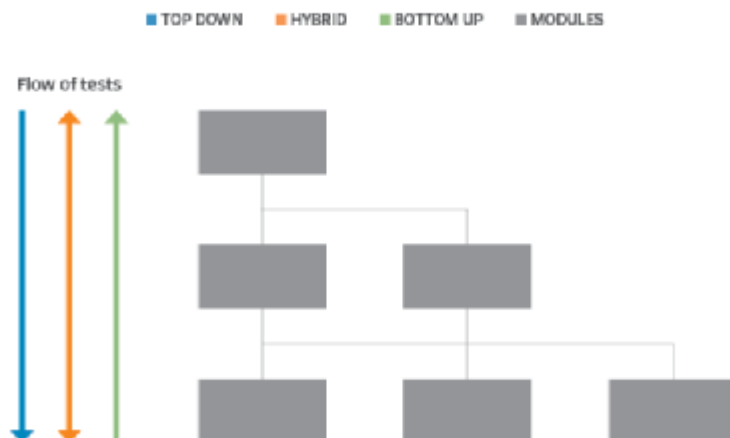


Рисунок 5.5 – Типы интеграционного тестирования

Когда разные разработчики работают над разными модулями, каждый привносит свое понимание и логику в процесс разработки. Это может вызвать функциональные проблемы или проблемы с удобством использования при объединении модулей. Интеграционное тестирование может помочь убедиться, что интегрированные модули функционируют должным образом как единое целое и соответствуют заявленным требованиям. Это также может гарантировать отсутствие ошибок между различными интерфейсами разных модулей.

Во многих сценариях приложений реального времени требования могут меняться. Эти новые требования могут не каждый раз подвергаться модульному тестированию, что может привести к упущенным дефектам или

отсутствующим функциям продукта. Интеграционное тестирование может заполнить эти пробелы, чтобы обеспечить включение новых требований в окончательное приложение.

Некоторые модули, которые взаимодействуют со сторонними программными интерфейсами приложений (API), необходимо протестировать, чтобы убедиться, что они работают правильно. Этого нельзя сделать во время модульного тестирования, поэтому требуется интеграционное тестирование.

Интеграционное тестирование также помогает устранить такие проблемы, как неадекватная обработка исключений, генерация ответов API, форматирование данных, ошибочные внешние аппаратные интерфейсы, неправильные интерфейсы сторонних служб и перехват ошибок.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

6.1 Настройка сервера и развертывание приложения

Для любого веб-приложения самым важным аспектом является его гибкая и отказоустойчивая работа, обрабатывающая достаточно большое количество пользовательских запросов, а также хранящая относительно большие объемы данных. Необходимы сервер, обладающий достаточными вычислительными ресурсами, а также инструменты, позволяющие обновлять приложение, осуществлять поддержку и исправление проблем в приемлемые сроки.

Для разработки данного приложения использовался сервер Kestrel Web Server. Kestrel — это сервер с открытым исходным кодом, управляемый событиями с асинхронным вводом-выводом, используемый для размещения приложений ASP.NET на любой платформе. Это прослушивающий сервер и интерфейс командной строки. Этот тип сервера упрощает процесс создания, запуска и тестирования веб-приложений, поскольку он работает со всеми версиями ASP.NET и поддерживает все формы приложений ASP.NET.

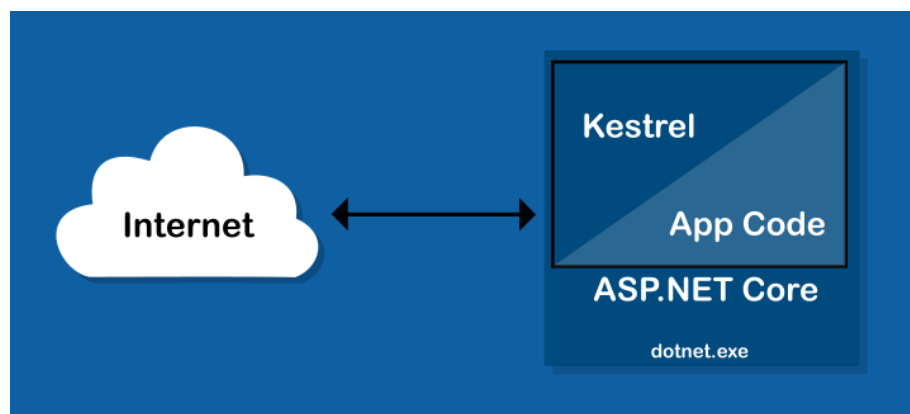


Рисунок 6.1 – Kestrel Web Server

Для автоматизации сборки проекта использован фреймворк Microsoft Build Engine – инструмент для компиляции проекта, сборки его в SO или DLL-файлы, генерации документации и разрешения зависимостей внешних библиотек приложения.

Microsoft Build Engine представляет собой платформу для сборки приложений. Компонент MSBuild обеспечивает для файла проекта схему XML, определяющую способы, используемые платформой сборки для обработки и сборки приложений. Вызывая msbuild.exe для файла проекта или решения, можно контролировать и создавать продукты в средах без установленного экземпляра Visual Studio.

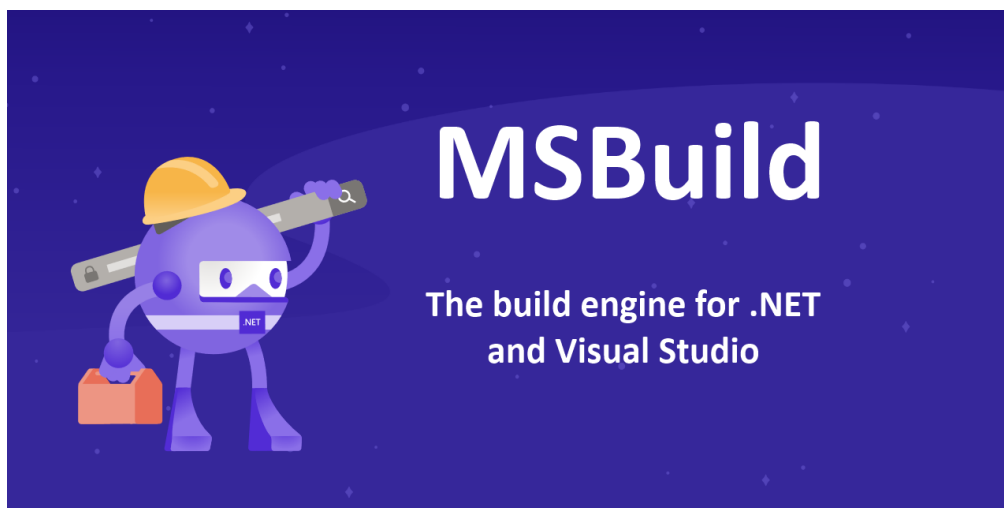


Рисунок 6.2 – Инструмент для компиляции проекта MSBuild

Так как проект разработан на языке C#, такие серверы приложений, как Kestrel Web Server, существуют на всех основных операционных системах, фреймворк MSBuild не зависит от операционной системы. Таким образом нет разницы, на какой операционной системе работает сервер, на котором будет развернуто приложение.

Первым шагом в сборке и развертывании приложения на сервере необходимо скачать и установить .NET SDK (Software Development Kit). Это выполняется при помощи следующей команды:

```
dotnet install Sdk $(SdkVersion), где SdkVersion – версия устанавливаемой SDK.
```

После этого необходимо скачать и установить пакетный менеджер NuGet. Это действие выполняется при помощи следующей команды:

```
dotnet install Nuget $(NugetVersion), где NugetVersion – версия устанавливаемого пакетного менеджера NuGet.
```

Следующим действием необходимо при помощи пакетного менеджера NuGet скачать и установить требуемые для корректной работы приложения пакеты. Это выполняется при помощи следующей команды:

```
dotnet restore $(projectFile) --configfile ./src/nuget.config), где projectFile – путь к решению развертываемого приложения.
```


Последующим производится сборка проекта. Это происходит при помощи следующей команды:

`dotnet build $(projectFile) --self-contained`, где `projectFile` – путь к решению разворачиваемого приложения.

После этого необходимо протестировать наше веб-приложение. Это действие выполняется при помощи следующей команды:

`dotnet test $(testProjectFiles)`, где `testProjectFiles` – пути к файлам тестирования.

Следующим действием необходимо опубликовать приложение на сервере. Это действие выполняется при помощи следующей команды:

`dotnet publish $(projectFile)`, где `projectFile` – путь к решению разворачиваемого приложения.

Последним действием необходимо собрать и опубликовать клиентскую часть разрабатываемого приложения. Это происходит при помощи следующей команды:

`ng build $(projectFile)`, где `projectFile` – путь к решению клиентской части приложения.

Для запуска на клиентской стороне существуют следующие минимальные системные требования для версии браузеров:

- Mozilla Firefox 54+;
- Opera 47+;
- Internet Explorer 10+;
- Microsoft Edge 14+;
- Safari 13+;
- Google Chrome 60+;

6.2 Руководство по использованию ПО

6.2.1 Авторизация и регистрация пользователей

Впервые перейдя по ссылке на страницу приложения, пользователь попадает на главную страницу приложения, которая содержит список последних актуальных новостей из мира космоса.

Если пользователь желает посмотреть детальную информацию о проходящем событии, ему необходимо нажать на кнопку «Learn more».

Для авторизации в приложении, пользователю необходимо в верхнем правом углу нажать на кнопку меню, и в открывшейся вкладке нажать на кнопку «Sign in».

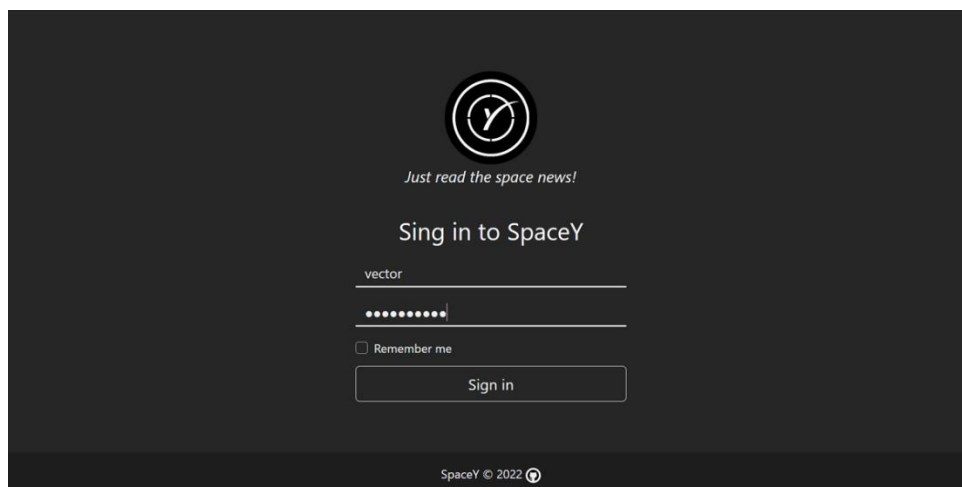


Рисунок 6.3 – Страница авторизации пользователя

Если у пользователя в данный момент не существует учётной записи, ему необходимо зарегистрироваться. Переход на страницу регистрации осуществляется при нажатии на кнопку «Sign up» внизу вкладки меню.

Для регистрации пользователю необходимо ввести следующие данные:

- логин;
- адрес электронной почты;
- пароль и подтверждение пароля.

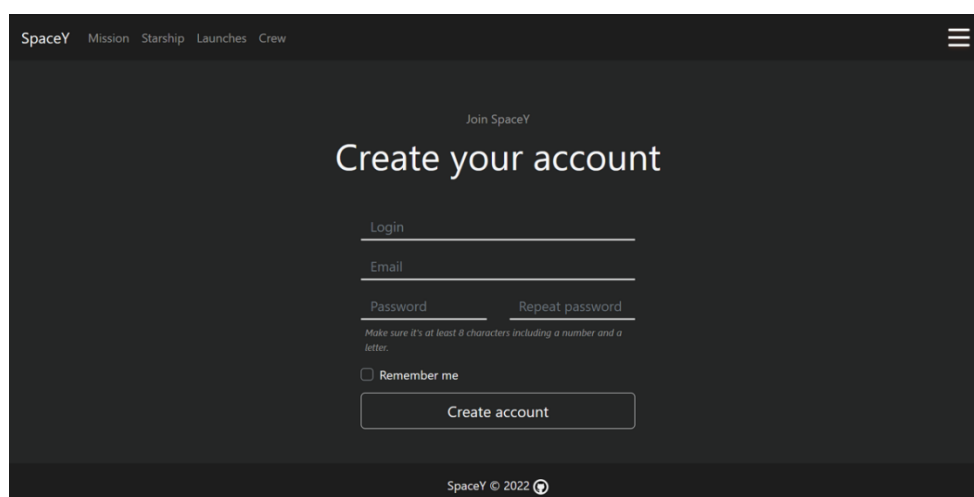


Рисунок 6.4 – Страница регистрации пользователя

После нажатия на кнопку «Create account» создается JWT-токен, хранящийся в локальном хранилище браузера, и пользователь переходит на главную страницу приложения.



Рисунок 6.5.1 – Главная страница приложения

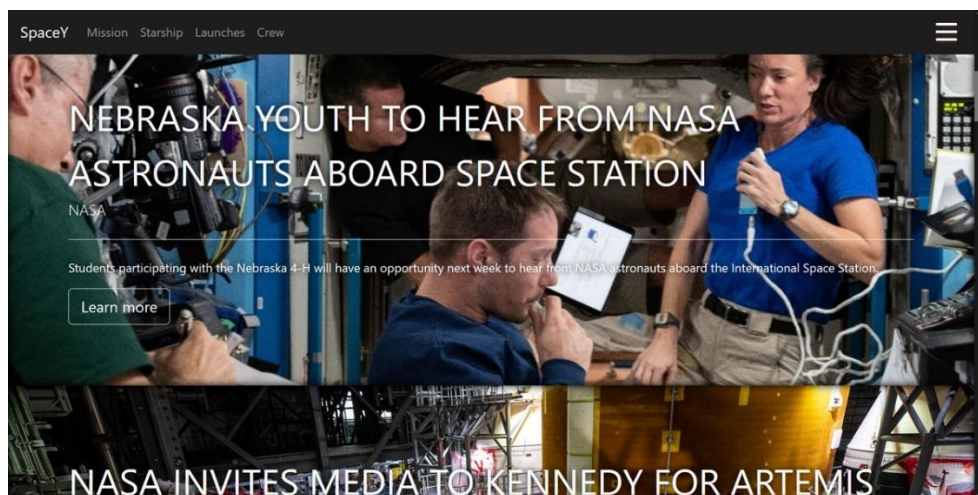


Рисунок 6.5.2 – Главная страница приложения

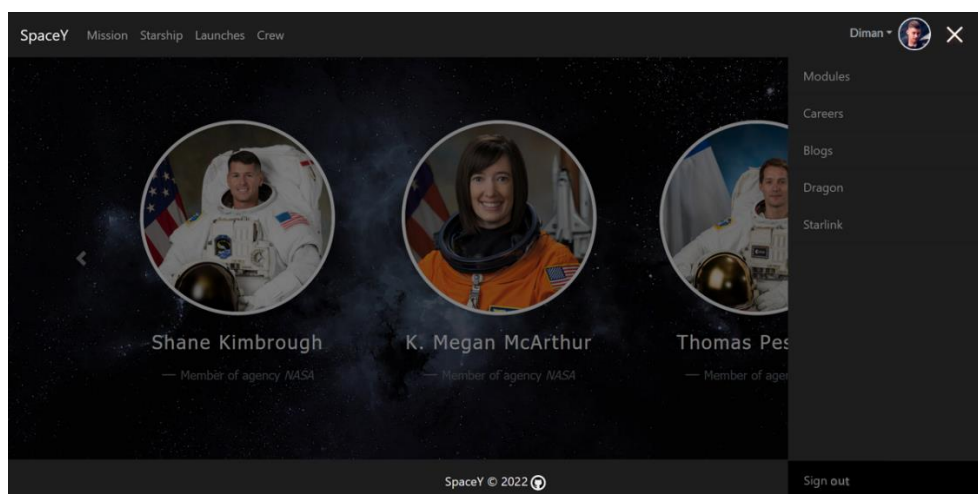


Рисунок 6.5.3 – Меню

6.2.2 Личный кабинет пользователя

Для перехода в личный кабинет пользователю необходимо в открытой вкладке меню нажать на свое имя рядом с иконкой профиля. После этого появляется выпадающий список, состоящий из пунктов:

- «Profile»;
- «Messages»;
- «Favorite launches».

Выбрав пункт «Profile» откроется личный кабинет пользователя.

В личном кабинете каждого пользователя отображается следующая информация:

- имя;
- фамилия;
- адрес электронной почты;
- фотография;
- номер телефона.

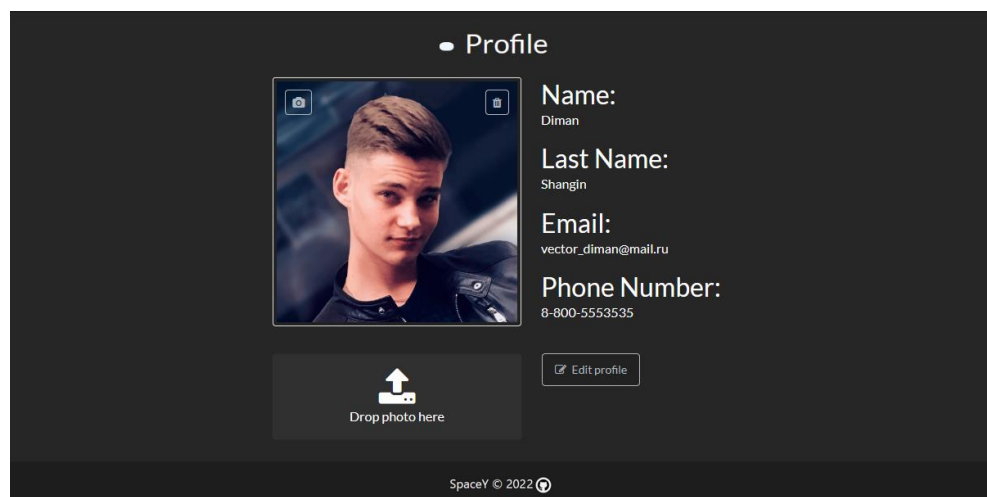


Рисунок 6.6 – Личный кабинет пользователя

При первоначальном входе, в окне фотографии отображается предустановленная фотография сервиса. Доступны три основные функции работы с фотографией пользователя:

- иконка «add»;
- иконка «delete»;
- «Drop photo here».

Функция «add» предоставляет пользователю возможность выбрать свою фотографию с устройства. При нажатии на иконку «add» в левом верхнем углу открывается окно устройства, в котором пользователь выбирает фотографию, после чего, нажатием кнопки «открыть», добавляет ее на страницу приложения.

Функция «delete» – удаление фотографии. У фотографии в личном кабинете пользователя в правом верхнем углу находится иконка «delete», при нажатии на которую, происходит удаление фотографии.

Функция «Drop photos here» предоставляет пользователю возможность перетаскивания его фотографии в указанное поле на странице редактирования фотографии, тем самым добавив ее в свой профиль.

При нажатии на кнопку «Edit profile» пользователь может поменять следующие параметры:

- имя;
- фамилию;
- адрес электронной почты;
- номер телефона.

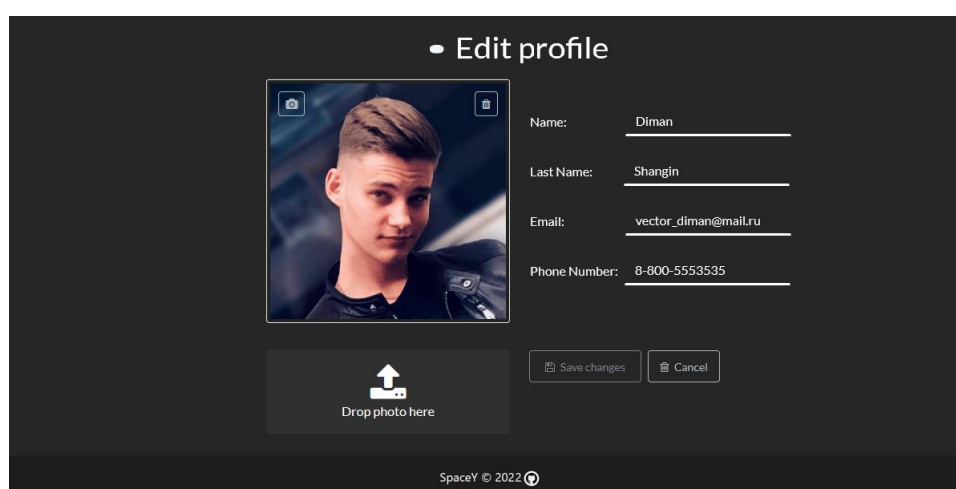


Рисунок 6.7 – Изменение личных данных пользователя

6.2.3 Страницы «Mission» и «Starship»

Для перехода на страницу «Mission» следует выбрать вкладку «Mission» вверху страницы приложения. Данная страница приложения содержит список последних миссий компании SpaceX.

Если пользователь желает посмотреть детальную информацию о проходящем событии, ему необходимо нажать на кнопку «Learn more».

Для перехода на страницу «Starship» следует выбрать вкладку «Starship» вверху страницы приложения. Данная страница приложения содержит список ракет и звездолетов компании SpaceX.

Если пользователь желает посмотреть детальную информацию о проходящем событии, ему необходимо нажать на кнопку «Learn more».

6.2.4 Страница «Messages»

Для перехода на страницу «Messages» следует в выпадающем списке

выбрать вкладку «Messages». В открывшемся окне отображаются три основные вкладки:

- «Unread»;
- «Inbox»;
- «Outbox».

При переходе на вкладку «Unread» открывается страница с непрочитанными сообщениями от всех пользователей. На данной странице отображается следующая информация:

- «Message» – отображает текст сообщения;
- «From/To» – отображает иконку и имя отправителя;
- «Sent/Received» – отображает дату отправления сообщения;
- иконка «Delete» – предоставляет возможность удалить определенное выбранное сообщение.

При переходе на вкладку «Inbox» открывается страница с уже прочитанными сообщениями от всех пользователей. Отображение данной страницы аналогично странице «Unread».

При переходе на вкладку «Outbox» открывается страница с отправленными пользователем сообщениями другим пользователям. Отображение данной страницы аналогично странице «Unread».

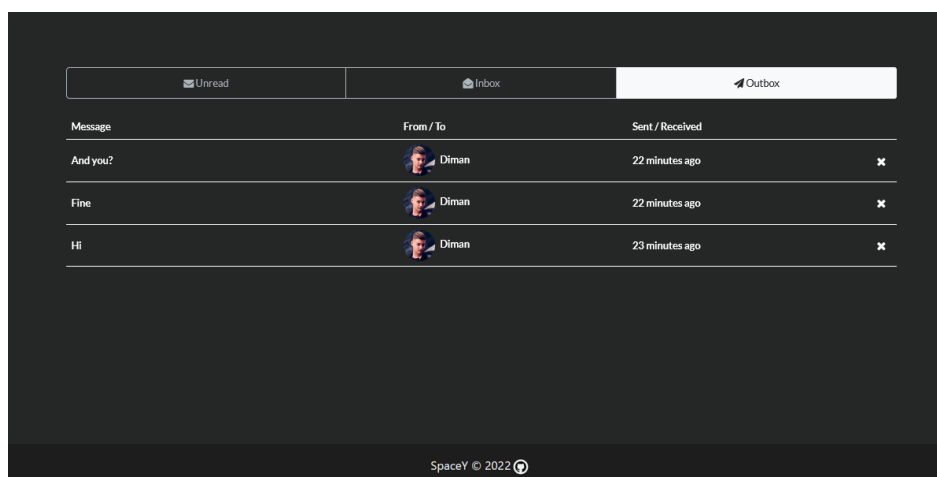


Рисунок 6.8 – Список сообщений пользователей

При нажатии на пользователя из предложенного списка открывается чат с этим пользователем.

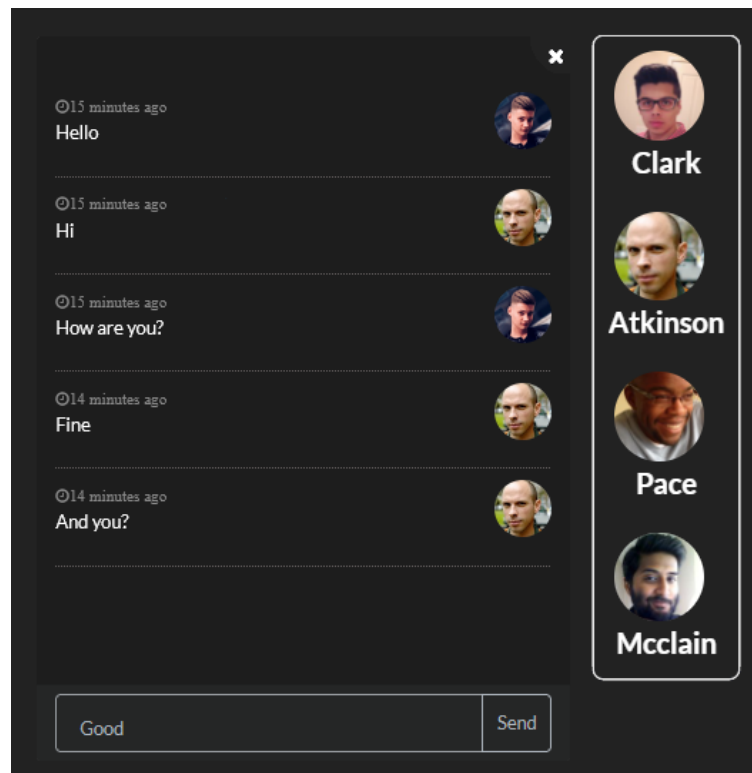


Рисунок 6.9 – Чат с пользователем

6.2.5 Страница «Launches»

Для перехода на страницу «Launches» следует выбрать вкладку «Launches» в верху страницы приложения. Данная страница приложения содержит список всех запусков компании SpaceX.

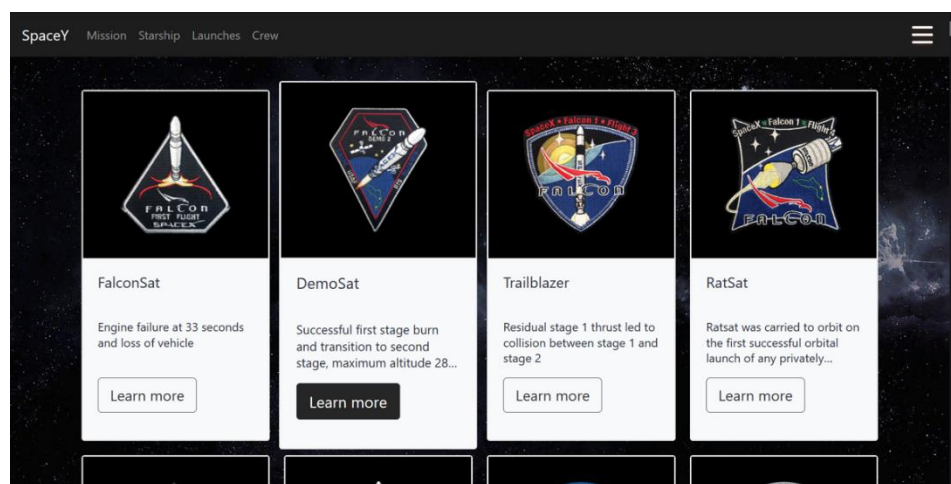


Рисунок 6.9 – Список запусков

Если пользователь желает посмотреть детальную информацию о конкретном запуске, ему необходимо нажать на кнопку «Learn more».

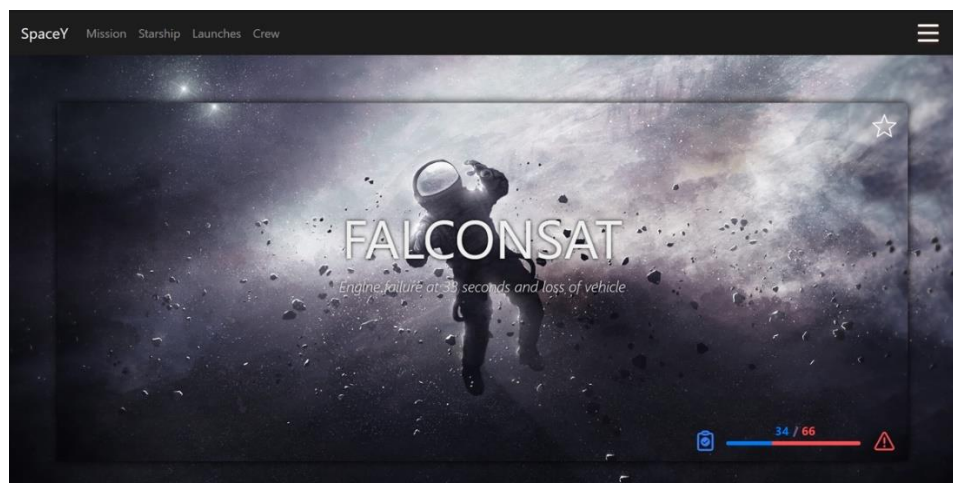


Рисунок 6.10.1 – Страница запуска

Пользователю предоставляется возможность просмотреть подробную информацию о ракете, с помощью которой был произведен текущий запуск. Эта информация подразделяется на несколько пунктов:

- «Overview»;
- «First stage»;
- «Second stage»;
- «Engines».

При переходе на соответствующую вкладку пользователю предоставляется описание одного из параметров ракеты.

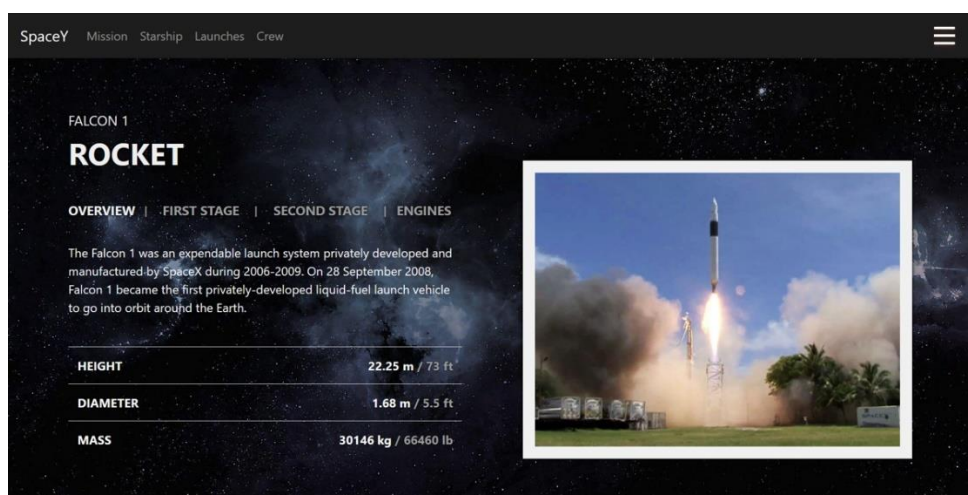


Рисунок 6.10.2 – Информация о ракете

Пользователю предоставляется возможность просмотреть подробную информацию о полезной нагрузке, которую переносит текущая ракета. Эта информация подразделяется на несколько пунктов:

- «Mass»;
- «Orbit»;

- «Customers»;
- «Reference system»;
- «Regime».

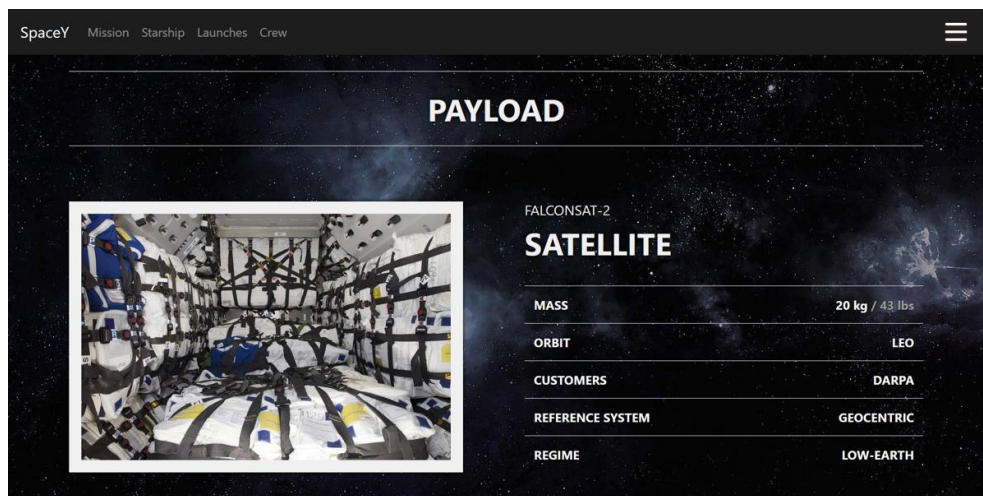


Рисунок 6.10.3 – Информация о полезной нагрузке

Пользователю предоставляется возможность просмотреть подробную информацию о месте запуска ракеты. Эта информация подразделяется на несколько пунктов:

- «Overview»;
- «Region»;
- «Timezone»;
- «Latitude»;
- «Longitude»;
- «Launch attempts»;
- «Launch successes».

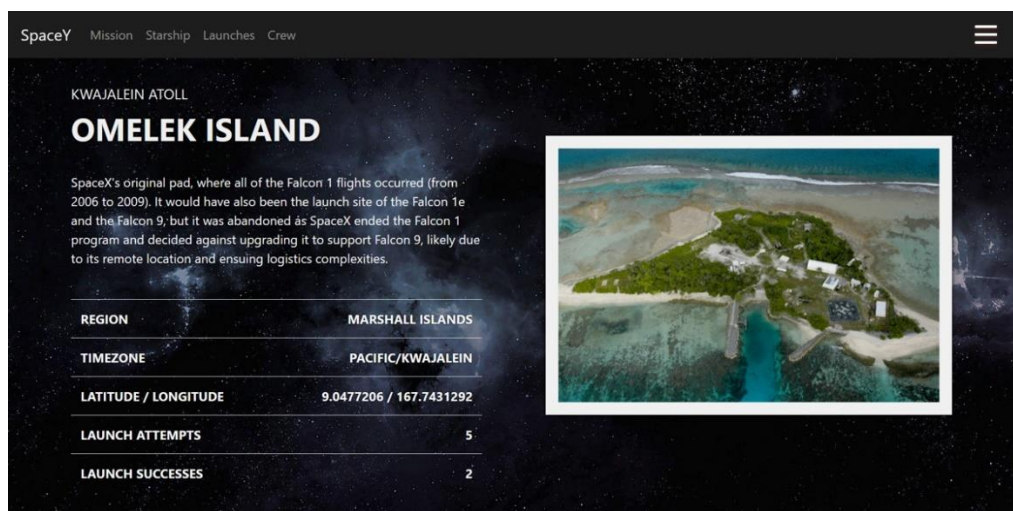


Рисунок 6.10.4 – Информация о месте запуска

Пользователю предоставляется возможность написать комментарий в соответствующем поле, после чего нажать кнопку «Comment» для его добавления. После создания комментария, он отобразится в блоке комментариев в самом верху.

В комментарии предоставляется следующая информация:

- имя и фамилия пользователя, оставившего комментарий;
- время публикации комментария;
- текст комментария.

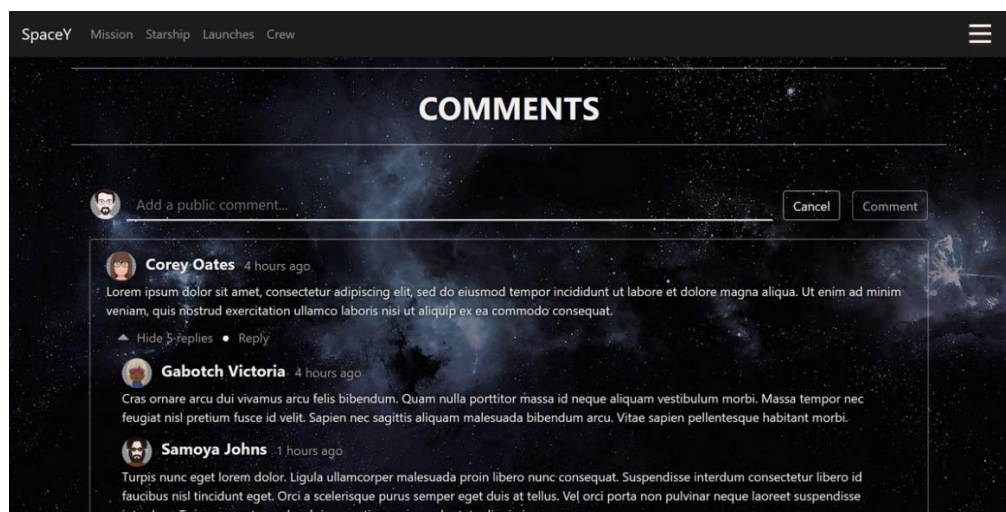


Рисунок 6.10.5 – Комментарии

6.2.6 Страница «Crew»

Для перехода на страницу «Crew» следует выбрать вкладку «Crew» в верху страницы приложения. Данная страница приложения содержит список экипажа космических миссий.

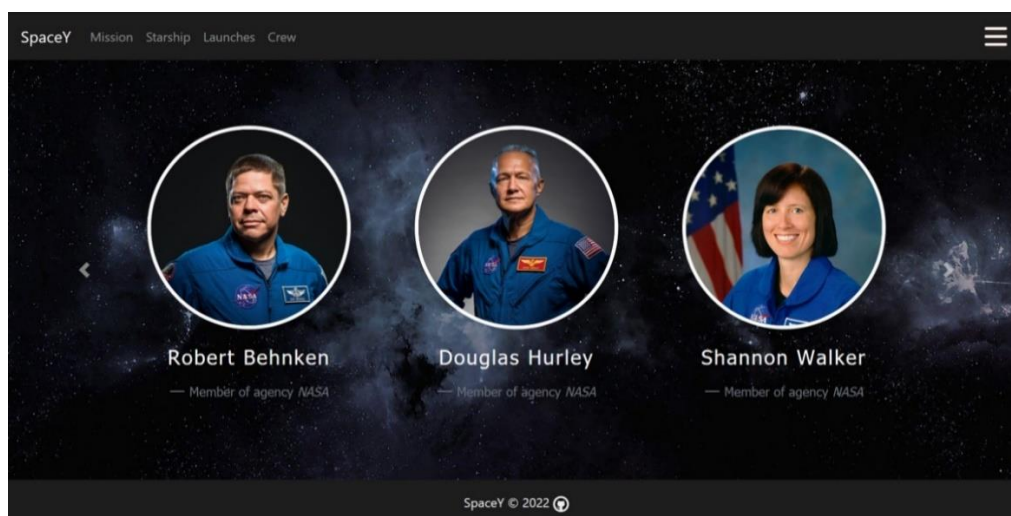


Рисунок 6.11 – Экипаж

6.2.7 Дополнительные возможности администратора

У администратора приложения есть дополнительные возможности управления сайтом.

При входе в аккаунт администратора в меню появляется дополнительная вкладка «Admin» в выезжающем списке.

Для перехода на страницу «Admin» следует выбрать вкладку «Admin» вверху страницы приложения. В открывшемся отображается страница редактирования ролей пользователей, в которой представлены следующие поля:

- «Username» – отвечает за отображение имени пользователя;
- «Active roles» – отвечает за отображение активных ролей пользователя.

Справа от каждой записи имеется кнопка «Edit Roles». При нажатии на данную кнопку открывается всплывающее окно «Edit roles for admin». В окне отображаются чек-боксы для выбора ролей пользователя. Всего предусмотрено три вида ролей:

- «Client» – включает основные функции приложения;
- «Admin» – включает функции роли «Client», а также функцию изменения ролей пользователей в вкладке «User management».

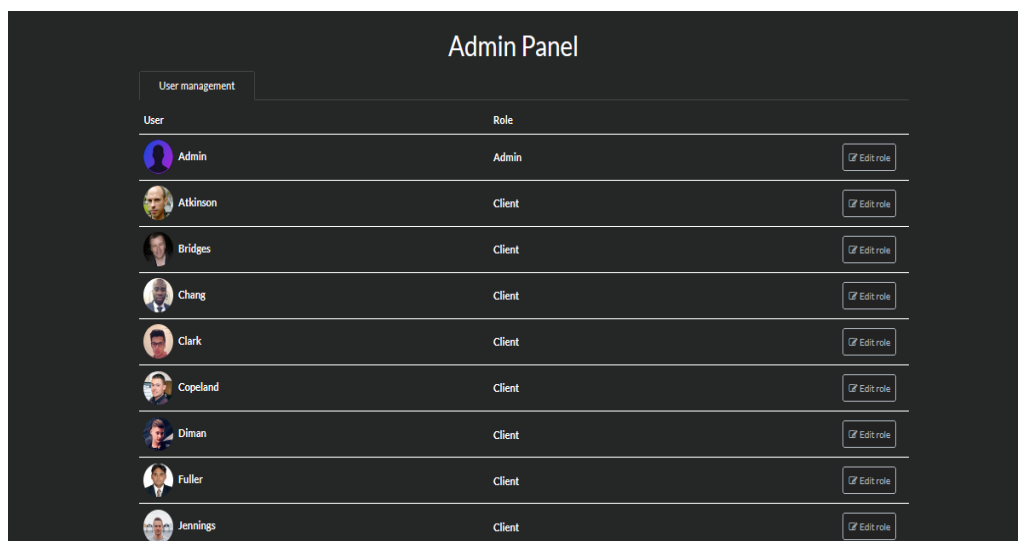


Рисунок 6.12 – Страница администрирования

6.2.8 Обратная связь с разработчиком приложения

Для получения дополнительной информации о приложении у самого разработчика пользователям предоставляется возможность обратной связи. Для этого на футере каждой из страниц приложения размещена иконка GitHub. При нажатии на эту иконку произойдет переход в репозиторий приложения, где содержится полный код приложения, описание доступных возможностей

для каждой из ролей по соответствующим разделам приложения, а также диаграмма базы данных для большего понимания системы хранения пользовательской информации.

7 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ИСПОЛЬЗОВАНИЯ НОВОСТНОГО ПОРТАЛА С ТЕМАТИКОЙ КОСМИЧЕСКОЙ НАПРАВЛЕННОСТИ НА МАССОВОМ РЫНКЕ

7.1 Краткая характеристика новостного портала с тематикой космической направленности

Разрабатываемое приложение предназначено для информирования общественности о событиях, которые происходят вокруг них. Основной его целью является предоставление веб-сервиса рядовым пользователям сети интернет для получения актуальной информации о событиях, происходящих в мировой космической отрасли. Программный модуль предоставляет следующий функционал:

- Авторизация пользователя в системе.
- Создание и оформление новостной ленты.
- Создание личного диалога с пользователем, интересующимся космическими тенденциями.
- Добавление комментариев к постам и личной информации пользователя.

Программный продукт устанавливается в облачном сервере. Разработчик получит экономический эффект от реализации подписок на данное программное обеспечение.

Данный программный модуль имеет узкую область применения. Он предназначен для интеграции в несколько основных сфер жизнедеятельности человека: образование, досуг, трудовая деятельность. Благодаря гибкости взаимодействия приложений с помощью протокола HTTP данный новостной портал может быть использован в качестве основного источника информации на космическую тематику в любом браузере с доступом к сети интернет: мобильное устройство, десктоп, веб-приложение и другие.

Разрабатываемый программный продукт адресован рядовым пользователям сети интернет, интересующимися новостями на космическую тематику. Люди предпочитают читать новости, которые наиболее важны для них самих, а также для жизни их семей, коллег и сообществ. Новостной портал поможет читателям сформировать основные предпочтения, когда они [читатели] сталкиваются с растущим объемом контента онлайн и оффлайн в медийной среде.

В современном мире спрос на новостные порталы серьёзно вырос. Они сообщают новости об экономическом положении страны, спорте, играх, развлечениях, торговле и коммерции. Изучение новостей становится

необходимостью и уже является частью современной жизни. Эта необходимость расширяет кругозор рядового гражданина и обогащает его знания. Чтение новостных порталов делает человека хорошо информированным. В целом, новости — это та часть коммуникации, которая держит общество в курсе меняющихся событий, проблем и характеров внешнего мира. Хотя это может быть интересно или даже забавно, главная ценность новостей заключается в том, что они помогают расширить возможности информированных граждан.

Например, компания Pocket Ventures LLC выпустила сайт SpaceNews.com, являющийся одним из самым надежным и всеобъемлющим источником новостей и аналитических материалов о компаниях, агентствах, технологиях и тенденциях, формирующих мировую космическую отрасль, и насчитывающим более 500 000 космических профессионалов и энтузиастов.

На основе маркетинговых исследований рынка данного продукта стоимость подписки составит 60 белорусских рублей в месяц. Эта стоимость подписки обусловлена тем, что пользователь, приобретший подписку, становится частью огромного комьюнити и получает уникальную возможность делиться своим опытом и мнением о событиях с другими членами этого комьюнити.

Продвижение программного продукта планируется реализовать через контекстную рекламу. Сайт программного продукта предоставит полное описание сервиса и возможность приобретения подписки.

Таким образом, организация-разработчик получит экономический эффект в виде прироста чистой прибыли, полученной от реализации программного продукта.

7.2 Расчёт инвестиций в разработку новостного портала с тематикой космической направленности для его использования на рынке

Для разработки новостного портала потребуется команда из двух человек: инженер-программист и старший инженер-программист. Расчёт затрат на основную заработную плату команды разработчиков (Z_o) приведено в таблице 7.1.

Таблица 7.1 – Затраты на основную заработную плату команды разработчиков

Категория исполнителя	Часовой оклад, р.	Трудоёмкость работ, ч.	Итого, р.
Инженер-программист	9	168	1512
Старший инженер- программист	14	168	2352
Итого			3864

Продолжение таблицы 7.1

Премия и иные стимулирующие выплаты (30%)	1159
Всего затрат на основную заработную плату разработчиков	5023

Дополнительная заработная плата исполнителей проекта определяется по формуле:

$$З_д = \frac{З_о \cdot Н_д}{100}, \quad (7.1)$$

где $Н_д$ – норматив дополнительной заработной платы, равный 10%.

После подстановки значений в формулу (7.1) дополнительная заработная плата составит:

$$З_д = \frac{5023 \cdot 10}{100} = 502,30 \text{ руб.}$$

Отчисления в фонд социальной защиты населения и на обязательное страхование ($Р_{соц}$) определяются в соответствии с действующими законодательными актами по формуле:

$$Р_{соц} = \frac{(З_о + З_д) \cdot Н_{сз}}{100}, \quad (7.2)$$

где $Н_{сз}$ – норматив отчислений в фонд социальной защиты населения и на обязательное страхование.

Размер отчислений в фонд социальной защиты населения и на обязательное страхование по формуле (7.2) составит:

$$Р_{соц} = \frac{(5023 + 502,3) \cdot 34,60}{100} = 1911,75 \text{ руб.}$$

Прочие расходы вычисляются по формуле:

$$Р_{пр} = \frac{З_о \cdot Н_{пр}}{100}, \quad (7.3)$$

где $Н_{пр}$ – норматив прочих расходов.

Принимая, что размер прочих расходов составит 30%, размер прочих расходов по формуле (7.3) составит:

$$Р_{пр} = \frac{5023 \cdot 30}{100} = 1506,90 \text{ руб}$$

Расходы на реализацию вычисляются по формуле:

$$P_p = \frac{Z_o \cdot H_p}{100}, \quad (7.4)$$

где H_p – норматив расходов на реализацию.

Принимая, что размер расходов на реализацию составит 3%, размер расходов на реализацию по формуле (7.4) составит:

$$P_p = \frac{5023 \cdot 3}{100} = 150,69 \text{ руб}$$

Общая сумма затрат на разработку и реализацию вычисляется по формуле:

$$Z_p = Z_o + Z_d + P_{\text{соц}} + P_{\text{пр}} + P_p. \quad (7.5)$$

Таким образом, общая сумма затрат на разработку и реализацию по формуле 7.5 составит:

$$Z_p = 5023 + 502,30 + 1911,75 + 1506,90 + 150,69 = 9094,64 \text{ руб}$$

7.3 Расчёт экономического эффекта от реализации новостного портала с тематикой космической направленности на рынке

Экономический эффект разработчика программного продукта заключается в получении прибыли от продажи подписок пользователям, использующим новостной портал.

На основе маркетинговых исследований рынка данного продукта стоимость подписки составит 60 белорусских рублей в месяц.

Прогнозируется, что каждый месяц будет предоставлено 20 подписок. Следовательно, в год будет предоставлено 240 подписок.

Поскольку компания разработчик является резидентом ПВТ, то прирост чистой прибыли, полученной разработчиком от реализации программного средства, определяется по формуле:

$$\Delta P_{\text{ч}}^p = C_{\text{отп}} \cdot N \cdot P_{\text{пр}}, \quad (7.6)$$

где $C_{\text{отп}}$ – отпускная цена подписки на программное средство, 60 руб.; N – количество подписок на программное обеспечение; $P_{\text{пр}}$ – рентабельность продаж новостного портала, 40%.

Таким образом, согласно формуле 7.6, прирост чистой прибыли составит:

$$\Delta P_{\text{ч}}^p = 60 \cdot 240 \cdot 0.4 = 5760 \text{ руб.}$$

7.4 Расчет показателей эффективности инвестиции в разработку новостного портала

Оценка экономической эффективности разработки и реализации программного продукта на рынке зависит от результата инвестиций в его разработку и полученного годового прироста чистой прибыли.

Приведение доходов и затрат к настоящему времени осуществляется посредством дисконтирования. Коэффициент дисконтирования вычисляется по формуле:

$$\alpha_t = \frac{1}{(1+d)^{t-t_p}}, \quad (7.7)$$

где d - норма дисконта, которая соответствует желаемому уровню рентабельности; t - порядковый номер года, доходы и затраты которого приводятся к расчётному году; t_p - расчётный год, к которому приводятся доходы и инвестиционные затраты ($t_p = 1$).

Норма дисконта равна 0.12(соответствует ставке рефинансирования). Тогда коэффициенты дисконтирования на следующие четыре года равны:

$$\alpha_1 = \frac{1}{(1+0,12)^0} = 1,$$

$$\alpha_2 = \frac{1}{(1+0,12)^1} = 0,89,$$

$$\alpha_3 = \frac{1}{(1+0,12)^2} = 0,80,$$

$$\alpha_4 = \frac{1}{(1+0,12)^3} = 0,71.$$

Так как разработка новостного портала не будет выполняться в годы, следующие за первым годом продаж, а также по причине отсутствия дополнительных затрат на реализацию в следующие годы обуславливается отсутствие затрат на разработку и реализацию программного модуля в последующие годы. Дополнительных затрат на поддержания актуальности наполнения и технического сопровождения также не будет по причине того, что для этих целей используется стороннее бесплатное API от компании SpaceX.

По причине того, что внедрение новостного портала начнется во второй половине года, чистая прибыль в первый год продаж будет равна половине от планируемой годовой прибыли.

Для расчета показателей экономической эффективности использования программного продукта следует рассчитать чистый дисконтированный доход, который рассчитывается по формуле:

$$\text{ЧДД} = \sum_{t=1}^n \Delta\Pi_{\text{ч}t} \cdot \alpha_t - \sum_{t=1}^n Z_t \cdot \alpha_t, \quad (7.8)$$

где $\Delta\Pi_{\text{ч}t}$ – прирост чистой прибыли в году t в результате реализации проекта, 5760 руб.; α_t – коэффициент дисконтирования года t ; Z_t – затраты в году t , 9094,64 руб.

Расчёт чистого дисконтированного дохода с нарастающим итогом по формуле 7.8 для первого года:

$$\text{ЧДД}_1 = 2880 \cdot 1 - 9094,64 \cdot 1 = -6214,64 \text{ руб.}$$

Для второго года:

$$\text{ЧДД}_2 = 2880 \cdot 1 + 5760 \cdot 0,89 - 9094,64 \cdot 1 = -1088,24 \text{ руб.}$$

Для третьего года:

$$\text{ЧДД}_3 = 2880 \cdot 1 + 5760 \cdot 0,89 + 5760 \cdot 0,8 - 9094,64 \cdot 1 = 3519,76 \text{ руб.}$$

Для четвёртого года:

$$\text{ЧДД}_4 = 2880 \cdot 1 + 5760 \cdot 0,89 + 5760 \cdot 0,8 + 5760 \cdot 0,71 - 9094,64 \cdot 1 = 7609,39 \text{ руб.}$$

В таблицу 7.2 внесены результаты вычислений.

Таблица 7.2 – Расчет эффективности инвестиционного проекта

Показатель	Расчетный период			
	2022	2023	2024	2025
РЕЗУЛЬТАТ				
Прирост чистой прибыли, руб.	2880	5760	5760	5760
Дисконтированный результат, руб.	2880	5126,4	4608	4089,6
ЗАТРАТЫ				
Инвестиции в реализацию программного решения, руб.	9094,64	-	-	-
Дисконтированные инвестиции, руб.	9094,64	-	-	-
Чистый дисконтированный доход нарастающим итогом, руб.	-6214,64	-1088,24	3519,76	7609,39
Коэффициент дисконтирования	1	0,89	0,8	0,71

Из таблицы 7.2 можно сделать вывод, что инвестиции в разработку новостного портала с тематикой космической направленности окупятся на третий год его реализации на рынке, чистый дисконтированный доход за четыре года реализации составит 3519,76 руб.

Индекс доходности инвестиций $\text{ИД}(PI)$ вычисляется по формуле:

$$PI = \frac{\sum_{t=1}^n \Delta\Pi_{чt} \cdot \alpha_t}{\sum_{t=1}^n 3_t \cdot \alpha_t} \quad (7.9)$$

Расчёт индекса доходности за четыре года по формуле 7.9:

$$PI = \frac{2880 \cdot 1 + 5760 \cdot 0,89 + 5760 \cdot 0,8 + 5760 \cdot 0,71}{9094,64 \cdot 1} = 1.84$$

Средняя норма прибыли $P_{и}$ (ARR) вычисляется по формуле:

$$P_{и} = \frac{\frac{1}{n} \sum_{t=1}^n \Delta\Pi_{чt}}{\sum_{t=1}^n 3_t} \quad (7.10)$$

Расчёт средней нормы прибыли за четыре года инвестиций по формуле 7.10:

$$P_{и} = \frac{\frac{1}{4} \cdot (2880 + 5760 + 5760 + 5760)}{9094,64} \cdot 100\% = 55\%$$

Простой срок окупаемости инвестиций без учёта фактора времени $T_{ок}(PP)$ вычисляется по формуле:

$$T_{ок} = \frac{\sum_{t=1}^n 3_t}{\frac{1}{n} \sum_{t=1}^n \Delta\Pi_{чt}} \quad (7.11)$$

Расчёт простого срока окупаемости инвестиций без учёта фактора времени за четыре года по формуле 7.11:

$$T_{ок} = \frac{9094,64}{\frac{1}{4} \cdot (2880 + 5760 + 5760 + 5760)} = 1,80$$

Таким образом, разработка новостного портала с тематикой космической направленности является эффективным вложением инвестиций, и его реализация на рынке является экономически целесообразной.

ЗАКЛЮЧЕНИЕ

За время работы над дипломным проектом был спроектирован и разработан новостной портал с тематикой космической направленности. Данное приложение предназначено для людей, которые интересуются новостями с тематикой космической направленности.

Разработка веб-серверной части данного проекта производилась на языке C#. Для описания пользовательского интерфейса использовался язык программирования TypeScript. Клиентская часть данного приложения создавалась в мощном интерфейсном фреймворке Angular. Хранение данных было реализовано при помощи реляционной базы данных PostgreSQL.

Разработанный новостной портал предоставляет пользователям платформу изучения и обсуждения новостей с тематикой космической направленности. Это веб-приложение предназначено для информирования общественности о событиях, которые происходят вокруг них. Основной его целью является предоставление веб-сервиса рядовым пользователям сети интернет для получения актуальной информации о событиях, происходящих в мировой космической отрасли. В проекте реализованы основные функции, которые характерны для данного типа приложений: авторизация пользователя в системе, создание и оформление новостной ленты, создания личного диалога и многое другое. В свою очередь администрация имеет возможность полного управления приложением в допустимых нормах.

Приложение является удобным и простым сервисом для знакомств. Функциональность проекта легка и понятна в использовании для любого пользователя. Данное приложение является конкурентоспособным.

Проведя расчет экономической эффективности, можно сделать вывод, что проектирование и разработка данного программного продукта являются целесообразными, принесут выгоду как компании-разработчику, так и покупателю программного продукта.

Данный проект был спроектирован и реализован по поставленным задачам. Созданная архитектура подразумевает дальнейшее развитие и усовершенствование данного приложения, а также расширение имеющейся функциональности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] SpaceNews [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://spacenews.com/segment/news/>. – Дата доступа: 24.03.2022.
- [2] SpaceFlight Now [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://spaceflightnow.com/category/news-archive/>. – Дата доступа: 27.03.2022.
- [3] Архитектура клиент-сервер [Электронный ресурс]. – Электронные данные. – Режим доступа: https://cio-wiki.org/wiki/Client_Server_Architecture/. – Дата доступа: 28.04.2022.
- [4] REST API [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.redhat.com/en/topics/api/what-is-a-rest-api>. – Дата доступа: 02.04.2022.
- [5] .NET [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet/>. – Дата доступа: 04.04.22.
- [6] Angular [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://angular.io/guide/what-is-angular>. – Дата доступа: 05.04.22.
- [7] Angular [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.hexacta.com/what-is-angular-and-why-should-we-consider-it/>. – Дата доступа: 05.04.22.
- [8] C# [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.developer.com/guides/what-is-c/>. – Дата доступа: 07.04.22.
- [9] TypeScript [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://medium.com/front-end-weekly/typescript-what-is-it-when-is-it-useful-c4c41b5c4ae7>. – Дата доступа: 08.04.2022.
- [10] TypeScript [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.freecodecamp.org/news/learn-typescript-basics/>. – Дата доступа: 08.04.2022.
- [11] PostgreSQL [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.postgresqltutorial.com/postgresql-getting-started/what-is-postgresql/>. – Дата доступа: 09.04.2022.

ПРИЛОЖЕНИЕ А
(обязательное)
Модель данных

ПРИЛОЖЕНИЕ Б

(обязательное)

Листинг кода

API/Controllers/AccountController.cs

```
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using API.Data;
using API.DTOS;
using API.Entities;
using API.Interfaces;
using AutoMapper;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace API.Controllers
{
    public class AccountController : BaseApiController
    {
        private readonly UserManager<AppUser> _userManager;
        private readonly SignInManager<AppUser> _signInManager;
        private readonly ITokenService _tokenService;
        private readonly IMapper _mapper;

        public AccountController(UserManager<AppUser> userManager, SignInManager<AppUser>
signInManager, ITokenService tokenService, IMapper mapper)
        {
            _userManager = userManager;
            _signInManager = signInManager;
            _tokenService = tokenService;
            _mapper = mapper;
        }

        [HttpPost("register")]
        public async Task<ActionResult<UserDto>> Register(RegisterDto registerDto)
        {
            if (await UserExists(registerDto.Username)) return BadRequest("Username is
taken");

            var user = _mapper.Map<AppUser>(registerDto);

            user.UserName = registerDto.Username.ToLower();

            var result = await _userManager.CreateAsync(user, registerDto.Password);

            if (!result.Succeeded) return BadRequest(result.Errors);

            var roleResult = await _userManager.AddToRoleAsync(user, "Member");

            if (!roleResult.Succeeded) return BadRequest(roleResult.Errors);

            return new UserDto
            {
                Username = user.UserName,
                Token = await _tokenService.CreateToken(user),
                KnownAs = user.KnownAs,
                Gender = user.Gender
            };
        }

        [HttpPost("login")]
        public async Task<ActionResult<UserDto>> Login(LoginDto loginDto)
        {
            var user = await _userManager.Users
                .Include(p => p.Photos)
                .SingleOrDefaultAsync(x => x.UserName == loginDto.Username.ToLower());

            if (user == null) return Unauthorized("Invalid username");
```

```

        var result = await _signInManager.CheckPasswordSignInAsync(user,
loginDto.Password, false);

        if (!result.Succeeded) return Unauthorized();

        return new UserDto
        {
            Username = user.UserName,
            Token = await _tokenService.CreateToken(user),
            PhotoUrl = user.Photos.FirstOrDefault(x => x.IsMain)?.Url,
            KnownAs = user.KnownAs,
            Gender = user.Gender
        };
    }

    private async Task<bool> UserExists(string username)
    {
        return await _userManager.Users.AnyAsync(x => x.UserName.ToLower() ==
username.ToLower());
    }
}

```

API/Controllers/AdminController.cs

```

using System.Linq;
using System.Threading.Tasks;
using API.Entities;
using API.Interfaces;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace API.Controllers
{
    public class AdminController : BaseApiController
    {
        private readonly UserManager<AppUser> _userManager;

        private readonly IUnitOfWork _unitOfWork;
        private readonly IPhotoService _photoService;
        public AdminController(UserManager<AppUser> userManager, IUnitOfWork unitOfWork,
IPhotoService photoService)
        {
            _photoService = photoService;
            _unitOfWork = unitOfWork;
            _userManager = userManager;
        }

        [Authorize(Policy = "RequireAdminRole")]
        [HttpGet("users-with-roles")]
        public async Task<ActionResult> GetUsersWithRoles()
        {
            var users = await _userManager.Users
                .Include(r => r.UserRoles)
                .ThenInclude(r => r.Role)
                .OrderBy(u => u.UserName)
                .Select(u => new
                {
                    u.Id,
                    Username = u.UserName,
                    Roles = u.UserRoles.Select(r => r.Role.Name).ToList()
                })
                .ToListAsync();

            return Ok(users);
        }

        [HttpPost("edit-roles/{username}")]
        public async Task<ActionResult> EditRoles(string username, [FromQuery] string
roles)
        {
            var selectedRoles = roles.Split(",").ToArray();

            var user = await _userManager.FindByNameAsync(username);

```



```

        if (user == null) return NotFound("Could not find user");

        var userRoles = await _userManager.GetRolesAsync(user);

        var result = await _userManager.AddToRolesAsync(user,
selectedRoles.Except(userRoles));

        if (!result.Succeeded) return BadRequest("Failed to add roles");

        result = await _userManager.RemoveFromRolesAsync(user,
userRoles.Except(selectedRoles));

        if (!result.Succeeded) return BadRequest("Failed to remove from roles");

        return Ok(await _userManager.GetRolesAsync(user));
    }
}
}

```

API/Controllers/BaseApiController.cs

```

using API.Helpers;
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers
{
    [ServiceFilter(typeof(LogUserActivity))]
    [ApiController]
    [Route("api/[controller]")]
    public class BaseApiController : ControllerBase
    {
    }
}

```

API/Controllers/BuggyController.cs

```

using API.Data;
using API.Entities;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers
{
    public class BuggyController: BaseApiController
    {
        private readonly DataContext _context;

        public BuggyController(DataContext context)
        {
            _context = context;
        }

        [Authorize]
        [HttpGet("auth")]
        public ActionResult<string> GetSecret()
        {
            return "secret text";
        }

        [HttpGet("not-found")]
        public ActionResult<AppUser> GetNotFound()
        {
            var thing = _context.Users.Find(-1);

            if (thing == null) return NotFound();

            return Ok(thing);
        }

        [HttpGet("server-error")]
        public ActionResult<string> GetServerError()
        {

```

```

        {
            var thing = _context.Users.Find(-1);

            var thingToReturn = thing.ToString();

            return thingToReturn;
        }

        [HttpGet("bad-request")]
        public ActionResult<string> GetBadRequest()
        {
            return BadRequest();
        }
    }
}

```

API/Controllers/FallbackController.cs

```

using System.IO;
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers
{
    public class FallbackController : Controller
    {
        public ActionResult Index()
        {
            return PhysicalFile(Path.Combine(Directory.GetCurrentDirectory(), "wwwroot",
            "index.html"), "text/HTML");
        }
    }
}

```

API/Controllers/LaunchController.cs

```

using System.Collections;
using System.Collections.Generic;
using System.Threading.Tasks;
using API.DTOs;
using API.Entities;
using API.Extensions;
using API.Helpers;
using API.Interfaces;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers
{
    [Authorize]
    public class LaunchesController : BaseApiController
    {
        private readonly IUnitOfWork _unitOfWork;

        public LaunchesController(IUnitOfWork unitOfWork)
        {
            _unitOfWork = unitOfWork;
        }

        [HttpPost("{username}")]
        public async Task<ActionResult> AddLaunch(string username)
        {
            var sourceUserId = User.GetUserId();
            var launchdUser = await _unitOfWork.UserRepository.GetUserByUsernameAsync(username);
            var sourceUser = await
            _unitOfWork.LaunchesRepository.GetUserWithLaunches(sourceUserId);

            if (launchdUser == null) return NotFound();

            if (sourceUser.UserName == username) return BadRequest("You cannot launch yourself");

            var userLaunch = await _unitOfWork.LaunchesRepository.GetUserLaunch(sourceUserId,
            launchdUser.Id);

            if (userLaunch != null) return BadRequest("You already launch this user");
        }
    }
}

```

```

        userLaunch = new UserLaunch
        {
            SourceUserId = sourceUserId,
            LaunchdUserId = launchdUser.Id
        };

        sourceUser.LaunchdUsers.Add(userLaunch);

        if (await _unitOfWork.Complete()) return Ok();

        return BadRequest("Failed to launch user");
    }

    [HttpGet]
    public async Task<ActionResult<PagedList<LaunchDto>>>
    GetUserLaunches([FromQuery]LaunchesParams launchesParams)
    {
        launchesParams.UserId = User.GetUserId();
        var users = await _unitOfWork.LaunchesRepository.GetUserLaunches(launchsParams);

        Response.AddPaginationHeader(users.CurrentPage, users.PageSize, users.TotalCount,
        users.TotalPages);

        return Ok(users);
    }
}

```

API/Controllers/MessagesController.cs

ПРИЛОЖЕНИЕ В
(обязательное)
Спецификация

ПРИЛОЖЕНИЕ Г
(обязательное)
Ведомость документов