Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и рАдиоэлектроники

Факультет Компьютерных систем и сетей

Кафедра Информатики

|  |
| --- |
| *К защите допустить:* |
| Заведующий кафедрой информатики |
| \_\_\_\_\_\_\_\_\_\_\_\_\_ Н.А. Волорова |

Пояснительная записка

к дипломному проекту

на тему:

**ПОРТАЛ ДЛЯ ЛЮДЕЙ, ЗАНИМАЮЩИХСЯ СПОРТОМ**

БГУИР ДП 1-40 01 03 00 023 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент |  | А. М. Волчецкий |
| Руководитель |  | Е. Л. Богушевский |
| Консультанты: |  |  |
| *от кафедры информатики* |  | Е. Л. Богушевский |
| *по экономической части* |  | В. А. Палицын |
| Нормоконтролер |  | В. В. Шиманский |
|  |  |  |
| Рецензент |  | О. Ю. Евменова |

Минск 2017

**РЕФЕРАТ**

ПОРТАЛ ДЛЯ ЛЮДЕЙ, ЗАНИМАЮЩИХСЯ СПОРТОМ: дипломный проект / А. М. Волчецкий – Минск: БГУИР, 2017, - п.з. – 86 с., чертежей (плакатов) – 6 л. формата А1.

Ключевые слова: ВЕБ-ПРИЛОЖЕНИЕ, УПРАЖНЕНИЕ, ТРЕНИРОВКА, JAVASCRIPT, C#, БАЗА ДАННЫХ.

Объектом исследования и разработки является веб-приложение для контроля занятий спортом.

Целью проекта является разработка портала для сбора и обработки информации о результатах тренировок пользователей.

При разработке и внедрении приложения использовался такой стек технологий как JavaScript, Angular 2, HTML5, CSS3, ASP.NET CORE и C#.

В первом разделе данной работы выдвигаются общие требования к созданию приложения, а также определяются перспективы развития первой версии продукта.

Второй раздел содержит предметную область разработки и функциональные требования.

Третий раздел посвящен проектированию архитектуры приложения.

В четвертом разделе описано создание программного средства.

Пятый раздел содержит результаты тестирования приложения.

В шестом разделе приведено технико-экономическое обоснование эффективности разработки и внедрения программного продукта.

В разделе заключение содержатся краткие выводы по дипломному проекту.

Дипломный проект является завершенным, поставленная задача решена в полной мере, присутствует возможность дальнейшего развития приложения и увеличение его функциональности.

Министерство образования Республики Беларусь

Учреждение образования   
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет КСиС Кафедра Информатики

Специальность 1-40 01 03 Специализация 00

УТВЕРЖДАЮ

\_\_\_\_\_\_\_\_\_\_\_\_\_ Зав. кафедрой

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2017 г.

**ЗАДАНИЕ**

**по дипломному проекту (работе) студента**

Волчецкого Александра Михайловича

(фамилия, имя, отчество)

1 Тема проекта (работы): «Портал для людей, занимающихся спортом»

утверждена приказом по университету от 28.02.2017 г. № 391‑с

2 Срок сдачи студентом законченного проекта (работы): июня 2017 года

3 Исходные данные к проекту (работе): Тип операционной системы: Microsoft Windows; Язык программирования – C#, ASP.NET, MS SQL;

Назначение разработки:

Создание программного средства для людей, занимающихся спортом

4 Содержание пояснительной записки (перечень подлежащих разработке вопросов):

Перечень условных обозначений, символов и терминов

Введение

1. Анализ прототипов и формирование требований к программному средству

2. Обзор используемых технологий

3. Проектирование программного средства

4. Создание программного средства

5. Тестирование программного средства

6. Технико-экономическое обоснование разработки программного средства

Заключение

Список используемых источников

Приложение А Исходный текст программы

5 Перечень графического материала (с точным указанием обязательных чертежей):

ГУИР.400103.023 ПД1 Диаграмма классов (контроллеров). Чертеж. Формат А1. лист 1.

ГУИР.400103.023 ПД2 Диаграмма классов (моделей). Чертеж. Формат А1. лист 1.

ГУИР.400103.023 ПД3 Диаграмма компонентов (front-end). Чертеж. Формат А1. лист 1.

ГУИР.400103.023 ПД1 Диаграмма деятельности. Плакат. Формат А1. лист 1.

ГУИР.400103.023 ПЛ2 Диаграмма вариантов использования. Плакат. Формат А1. лист 1.

ГУИР.400103.023 ПЛ3 Диаграмма базы данных. Плакат. Формат А1. лист 1.

6 Содержание задания по технико-экономическому обоснованию:

Расчет экономической эффективности от внедрения «Портала для людей, занимающихся спортом»

Задание выдал: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ / В.А. Палицын /

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Наименование этапов дипломного проекта (работы) | Объем этапа, % | Срок выполнения этапов | Примечание |
| 1 | Анализ предметной области, разработка технического задания | 15 – 20 | 03.02–16.02 |  |
| 2 | Разработка функциональных требований, проектирование архитектуры программы | 20 – 15 | 18.02–07.03 |  |
| 3 | Разработка схемы программы, алгоритмов, схемы данных | 20 – 15 | 11.03–22.03 |  |
| 4 | Разработка программного средства | 15 – 20 | 25.03–26.04 |  |
| 5 | Тестирование и отладка | 10 | 29.04–10.05 |  |
| 6 | Оформление пояснительной записки и графического материала | 20 | 13.05–31.05 |  |

КАЛЕНДАРНЫЙ ПЛАН

Дата выдачи задания \_\_\_\_\_\_\_\_\_ Руководитель \_\_\_\_\_\_\_ / Е.Л. Богушевский /

Задание принял к исполнению / А. М. Волчецкий /

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 6

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ 8

1 АНАЛИЗ ПРОТОТИПОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ 9

1.1 Обзор существующих аналогов 9

1.1.1 fitocracy.com 9

1.1.2 fitnessblender.com 11

1.1.3 trainonline.com 13

1.1.4 fitness.com 14

1.1.5 sportswiki.ru 16

1.2 Вывод 18

1.3 Формирование требований 18

2 ОБЗОР ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ 19

2.1 Angular 19

2.2 Язык программирования TypeScript 19

2.3 Программная платформа Microsoft .NET 21

2.4 Язык программирования C# 24

2.5 Entity Framework Core 27

2.6 ASP.NET Core 28

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 30

3.1 Слой представления 34

3.2 Серверный слой 37

3.3 Слой доступа к данным 40

3.4 Слой сервисов 42

4 СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА 44

5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 59

5.1 Модульное тестирование 59

5.2 Интеграционное тестирование 60

6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО СРЕДСТВА 64

6.1 Расчет затрат и отпускной цены программного средства 64

6.2 Оценка экономической эффективности применения программного средства у пользователя 68

6.3 Заключение 73

ЗАКЛЮЧЕНИЕ 74

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 76

ПРИЛОЖЕНИЕ А 77

# ВВЕДЕНИЕ

Повсеместная пропаганда здорового образа жизни заставляет граждан стремиться к самосовершенствованию. Сегодня каждый, не зависимо от социального статуса и возраста, мечтает о стройной и красивой фигуре. В связи с относительной экономической стабилизацией, по сравнению с девяностыми годами, у граждан возрос интерес к физическому здоровью и активному отдыху, что, в свою очередь, объясняет высокий спрос на данный род услуг. При благоприятной экономической ситуации бурный рост фитнес индустрии неизбежен, поскольку каждый человек, может повысить качество своей жизни, занимаясь фитнесом. С каждым годом количество людей, занимающихся спортом, продолжает увеличиваться.

Информационных технологии также не стоят на месте. В связи с их постоянным развитием многие привычные программы уже сейчас перенесены или переносятся во всемирную сеть, что позволяет пользователям иметь к ним постоянный доступ с любого устройства, позволяющего подключиться к интернету. Интернет уже стал непосредственной частью нашей жизни. Сейчас у нас имеется возможность заказывать еду, покупать одежду и технику не выходя при этом из дома и экономя свое время и силы. С каждым годом влияние всемирной сети продолжает увеличиваться.

Учитывая во внимание эти два факта в Интернете просто не могло не появиться порталов, связанных со спортом. Используя их люди могут делиться своим опытом занятий спортом, составлять программы тренировок и питания. Все это пользователи могут делать в большинстве бесплатно, что позволит начать делать первые успехи в занятиях спортом самому, без помощи профессионального тренера. А значит это не потребует дополнительных денежных затрат. Для многих это является большим плюсом и помогает начать вести здоровый образ жизни.

В итоге бурное развитие Интернета является одним из факторов популярности здорового образа жизни сейчас. Ведь в настоящее время в сети имеется много информации, связанной с занятиями спортом. И людям нужно всего лишь изучить эту информацию, систематизировать и подстроить ее под себя для достижения прогресса в занятиях спортом.

Целью данного дипломного проекта является написание портала для людей, занимающихся спортом. В разделе «Анализ прототипов и формирование требований к программному средству» представлен результат анализа существующих программных средств для занятий спортом, их сравнительная характеристика, а также сформированы требования к программному средству. Раздел «Обзор используемых технологий» пояснительной записки содержит характеристику технологий, которые были использованы при разработке программного средства. Раздел «Проектирование программного средства» содержит подробное описание структуры и схемы работы программного средства. Информация о реализации портала находится в разделе «Создание программного средства». Также пояснительная записка содержит результаты тестирования программного средства в разделе «Тестирование программного средства» и технико-экономическое обоснование принятых решений в разделе «Технико-экономическое обоснование разработки программного средства».

Дипломный проект выполнен самостоятельно, проверен в системе «Атиплагиат». Процент оригинальности соответствует норме, установленной кафедрой информатики. Цитирования обозначены ссылками на публикации, указанные в «Списке использованных источников».

# ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ

В настоящей пояснительной записке применяются следующие определения и сокращения.

API – Application Programming Interface – интерфейс программирования приложений.

DI – Dependency injection – инъекция зависимости

DOM – Document Object Model – объектная модель документа.

HTTP – HyperText Transfer Protocol – протокол передачи гипертекста.

JSON – JavaScript Object Notation – текстовый формат обмена данными, основанный на языке JavaScript.

LINQ – Language Integrated Query – интегрированные в язык запросы.

LоС – Lines of Code – количество строк кода.

MSIL – Microsoft Intermediate Language – промежуточный язык для платформы Microsoft .NET.

MVC – Model-View-Controller – модель-представление-контроллер.

MVW – Model-View-Whatever – модель-вид-что угодно.

REST – Representation State Transfer – передача состояния представления.

SPA – Single Page Application – одностраничное приложение

SQL – Structured Query Language – язык структурированных запросов.

URI – Uniform Resource Identifier – унифицированный идентификатор ресурса.

URL – Uniform Resource Locator – единый указатель ресурсов.

WWW – World Wide Web – всемирная паутина

XML – eXtensible Markup Language – расширяемый язык разметки.

БД – база данных.

ООП – объектно-ориентированное программирование.

СУБД – система управления базами данных.

# АНАЛИЗ ПРОТОТИПОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ

## Обзор существующих аналогов

### fitocracy.com

Fitocracy предлагает двухуровневую систему, чтобы помочь вам стать более приспособленными. Он предоставляет интерактивные онлайн-инструменты, которые помогают отслеживать прогресс, а также разнообразные тренировочные планы, чтобы убедиться, что пользователь движется в правильном направлении. Его онлайн-ресурсы бесплатны. Но чтобы нанять тренера, придется заплатить. Однако цены являются доступными и предлагают очень хорошее соотношение цены и качества.

Базовое членство бесплатно. Это позволяет получить бесплатные тренировки, независимо от того, являетесь ли вы любителем или занимаетесь в течение многих лет. Чтобы вы были мотивированы, вы можете получать очки и отслеживать свой прогресс. Прогресс отслеживания здесь немного более сложный, чем на некоторых сайтах, так как есть возможность проверить статистику на интерактивных графиках, чтобы оставить пользователю мало сомнений в том, в каких областях он преуспевает, и тех, над которыми ему нужно работать.

На сайте Fitocracy.com есть позитивное сообщество, которое может помочь сосредоточиться на своих целях и мотивировать. Также есть возможность прочитать ежедневные статьи, в том числе советы, в информационном центре сайта.

Имеется возможность выбора из широкого спектра программ тренировок, каждая из которых ориентирована на различные аспекты фитнеса (см. рис. 1.1). Они могут быть направлены на потерю или набор веса, на увеличение силы или выносливости.

Возможно выбрать свою программу на основе тренеров, с которыми вы предпочитаете работать. Тренеры нанимаются, чтобы предоставить столько советов, сколько нужно пользователю. Они могут помочь установить цели, убедившись, что полезны для пользователя, но также реалистичны. Это отличный мотиватор, и пользователи не чувствует себя плохо. Тренеры могут также помочь разработать планы диеты, которые будут соответствовать конкретному графику тренировки. Если пользователь хочет похудеть, например, то, вероятно, захочет обезжиренные диеты. Принимая во внимание, что, если пользователь надеется наращивать мышцы, то продукты с высоким содержанием белка могут быть важны.

Приятно иметь профессионала по вызову и поскольку большинство планов стоят от 10 до 20 долларов в неделю. Частный наставник в спортзале стоил бы дороже.

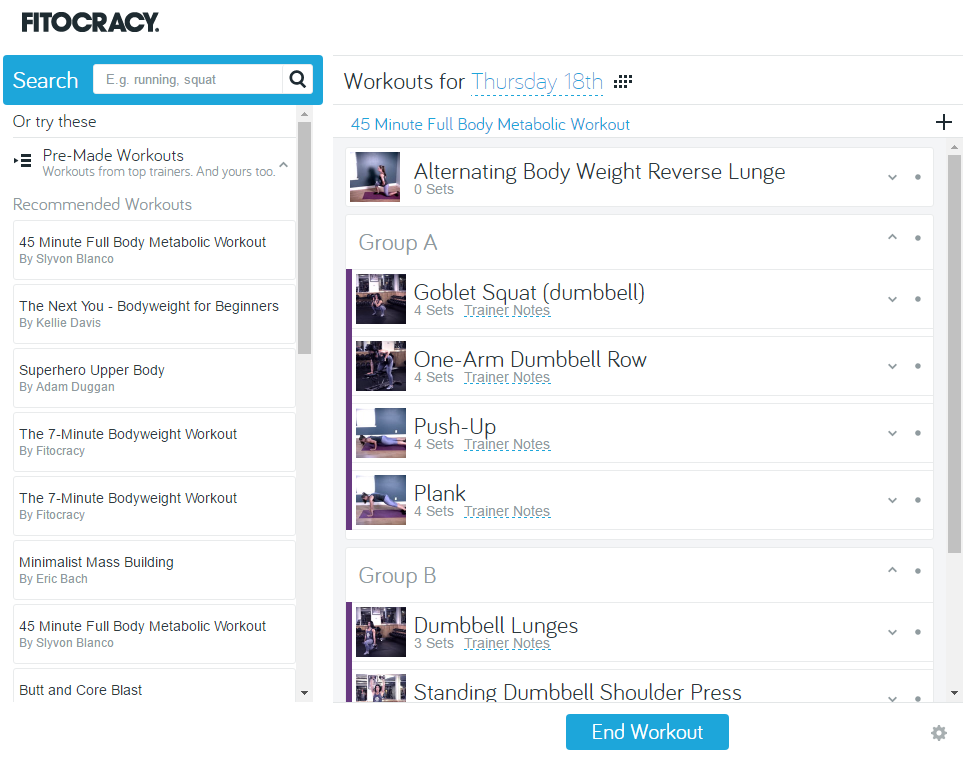


Рисунок 1.1 – fitocracy.com, программа тренировок

На сайте нет магазина, где можно купить книги, DVD-диски, приложения или оборудование. Бесплатные инструменты также немного ограничены, но стоит попробовать, чтобы посмотреть, что предлагается, и решить или бесплатные тренировки принесут пользу. Оттуда у пользователя есть возможность принять решение о том, хочет ли он заплатить за услуги тренера.

Плюсы:

* наличие бесплатных функций;
* невысокая стоимость услуг тренера;
* широкий набор упражнений и программ тренировок;
* удобный и современный дизайн

Минусы:

* небольшое количество планов питания
* краткое описание техники выполнения упражнений (новичок может не уловить все моменты и выполнять их неправильно)

### fitnessblender.com

Fitnessblender предлагает большой выбор бесплатных видеороликов и информации о фитнесе. Хотя некоторые из его программ стоят денег, они не являются дорогостоящими или обязательными. У пользователей сайта есть много полезной информации для изучения, и некоторые действительно удобные инструменты календаря для использования.

Зарегистрироваться очень просто, это можно сделать так через свою учетную запись Facebook или Google+. После регистрации можно начать работу над своим календарем (см. рис. 1.2). Это сердце учетной записи FitnessBlender, так как это позволяет пользователям планировать и организовывать свои тренировки. Однако, это не просто случай написания «тренировки 1 час» в определенный день, имеется возможность выбрать, какие упражнения включать, и сайт позволит легко найти подходящие видео.

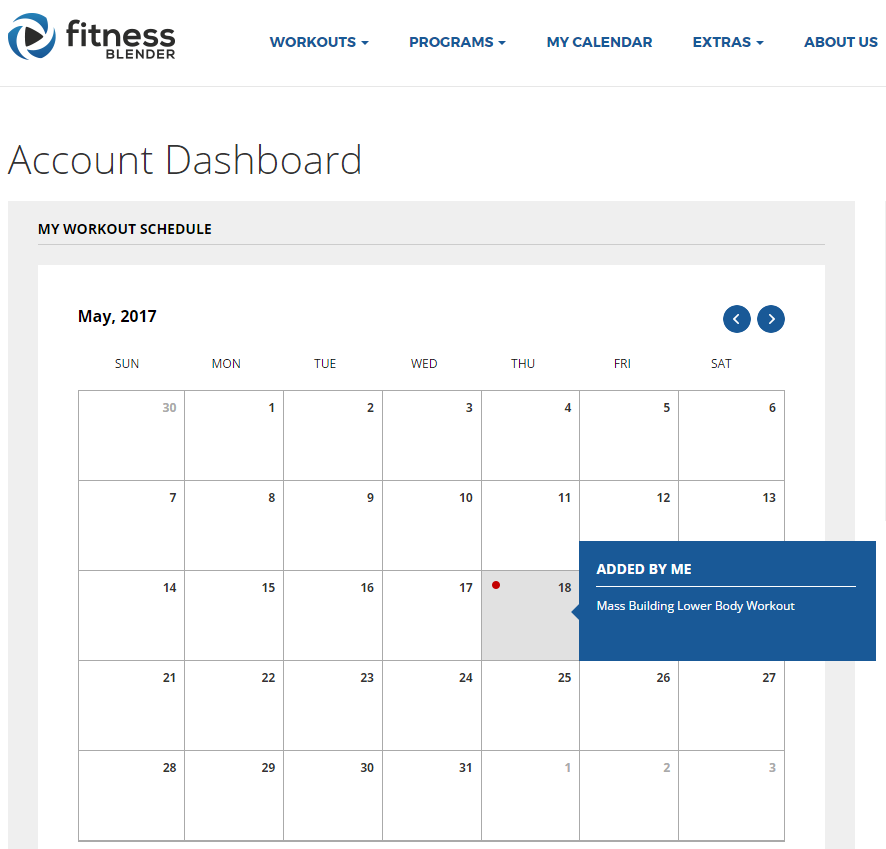


Рисунок 1.2 – fitnessblender.com, календарь

Итак, допустим, что пользователь хочет потерять немного веса вокруг своего живота. Как сайт может помочь в этом? Во-первых, можно искать тренировки, которые сосредоточены на этих ключевых областях. Функция поиска хорошо разработана, что позволяет находить ключевые слова, длину тренировки, ее сложность и многое другое. Затем пользователю показываются все видеоролики, которые соответствуют выбранным критериям.

Можно увидеть множество потенциальных упражнений одновременно. Наведение на одно из них покажет, как долго они будут длиться, сколько калорий сжигается и какое оборудование понадобится. Это позволяет легко определить упражнение, которое подходит пользователю.

Видео являются ясными и информативными, и их можно легко добавить в список избранных. Затем этот список легко получить из панели управления. Кроме того, можно добавить тренировку в свой календарь, просто нажав вкладку «добавить в календарь». Затем пользователь выбирает, когда хочет выполнить тренировку, и все это записывается. Это работает очень хорошо, и пользователь вряд ли столкнется с трудностями при организации календарей тренировок.

Пользователи могут давать отзывы по каждому видео, чтобы другие могли узнать, что другие люди думают об упражнении. То же самое можно сделать по всему сайту, чтобы люди могли понять, понравится ли им план диеты.

Некоторая информация на сайте не является бесплатной. Прежде всего, это, как правило, специализированные режимы тренировки для потери жира или диеты. Хотя это не слишком дорого, они часто входят в составные части. Даже если покупается целая серия, то это не выходит слишком дорого.

FitnessBlender.com, возможно, немного не хватает, когда речь заходит о бесплатных планах питания, так как это, кажется, стоит денег. Тем не менее, здесь так много свободной информации, что это станет действительно полезным ресурсом для широкого круга людей. Это поможет эффективно организовать тренировки, обеспечив достаточный выбор, чтобы заинтересовать людей.

Основная область, где сайт проявил себя нехорошо, - это поддержка клиентов. Это не удивительно, учитывая, что большинство его услуг предоставляются бесплатно, но на сайте есть форум, где пользователи могут обсуждать проблемы друг с другом. Так же есть блог, и сами видео очень хороши. Есть возможность поделиться своими тренировками в Facebook, Twitter и других социальных сетях.

Плюсы:

* подробное описание упражнений;
* невысокая стоимость программ тренировок;
* календарь тренировок;

Минусы:

* большинство планов питания платные

### trainonline.com

Когда дело доходит до бесплатных видео о фитнесе, TrainOnline.com будет трудно превзойти. На сайте есть сотни хорошо снятых видеороликов с экспертными советами о том, как их безопасно выполнять (см. рис. 1.3). Также можно следовать или разрабатывать собственный планы тренировок, добавляя упражнения в избранное. Пока на сайте не хватает информации о планах питания и нескольких дополнительных функций, что будет подталкивать к поиску более профессионального веб-сайта для занятий спортом, который предлагает столько всего бесплатно.

Есть возможность получить доступ к большому количеству контента сайта без регистрации, хотя, если пользователь хочет пообщаться с другими людьми на форумах, создать тренировки и отслеживать любимые упражнения, ему понадобится учетная запись. Членство на самом деле очень доступным по цене около 20 долларов в течение всего года.

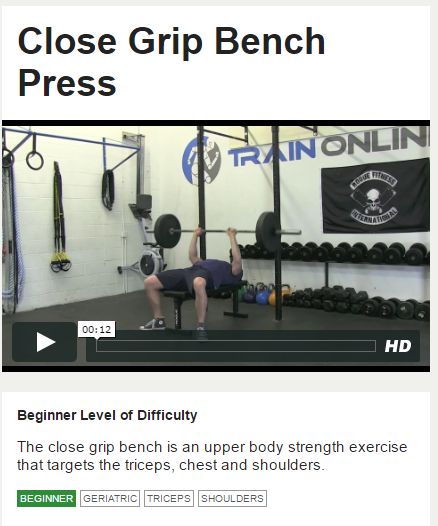


Рисунок 1.3 – trainonline.com, страница с упражнением

Личный кабинет пользователя является отправной точкой, позволяя ему контролировать свою учетную запись и следить за своими действиями. Когда пользователь обнаруживает упражнение, которое ему нравится, он просто просмотрит видео и затем жмет «добавить в избранное». Это упражнение будет сохранено, и позже он сможет добавить его в свои тренировки.

Если пользователь новичок, то он может просто следовать заранее разработанным тренировкам. Они описаны просто, и также можно увидеть уровень сложности каждого сеанса.

Более продвинутые участники найдут подходящие для них тренировки, но, возможно, пожелают создать больше собственных тренировок. К сожалению, на данный момент нет возможности загружать свои собственные видеоролики, поэтому у пользователей есть выбор упражнений, предоставленных командой сайта. Однако они открыты для загрузки новых видеороликов, основываясь на рекомендациях, и новые упражнения добавляются регулярно.

Раздел Fitness101 сайта TrainOnline.com будет очень полезен новичков в занятиях спортом. В нем содержится много информации о преимуществах здорового образа жизни, типах оборудования, которое может понадобиться, а также о том, как настроить расписание занятий. Однако, отсутствие информации о правильном питании может быть проблемой.

На этом сайте создано активное сообщество. Можно общаться в чате по различным темам, и форумы, по-видимому, находятся в хорошем состоянии. Есть также блог, чтобы следить за последними обновлениями из мира фитнеса.

Учитывая все обстоятельства, Train Online - отличный ресурс. Если у пользователя нет денег, то он все равно может просмотреть массу отличных видеороликов. Если он можете позволить себе очень небольшой членский взнос, то у пользователя появляется возможность использовать простые, но эффективные инструменты для создания тренировок. Желательно было бы увидеть функцию календаря и дополнительную информацию о питании, но, в целом, они делает хорошую работу по предоставлению упражнений и тренировок в Интернете.

Плюсы:

* большая бесплатная база упражнений с видео;
* наличие отдельного раздела для новичков;
* активное сообщество пользователей;

Минусы:

* нет планов питания
* отсутствие календаря

### fitness.com

С различными бесплатными видео упражнениями, статьями и фитнес- инструментами, Fitness.com может многое предложить своим пользователям. Похоже, что он удалил свои платные варианты, вместо этого сосредоточившись на предоставлении доступного фитнес-ресурса для всех.

Не все упражнения на сайте показываются с использованием видео; Некоторые из них загружаются поэтапно, но в описаниях должно быть достаточно информации для точного выполнения упражнений.

Есть возможность выполнить поиск, чтобы найти упражнения, основанные на части тела, которую пользователь хочет тренировать, и оборудовании, которое он может использовать. Там же можно исследовать длинный список видеороликов (многие из которых выглядят так, как будто их загружали участники, поскольку они находятся в домах и офисах) и выбрать те, которые подходят лично пользователю. Затем можно выбрать свои любимые упражнения и добавить их в план тренировки.

На сайте также есть много полезных статей для чтения, а также несколько форумов. Форумы были разделены на четыре секции (бодибилдинг, фитнес, потеря веса и спорт). Это хорошо, потому что они предлагают членам сообщества поддержку друг друга и обмен информацией.

Присутствуют на сайте и рецепты блюд правильного питания (см. рис. 1.4).

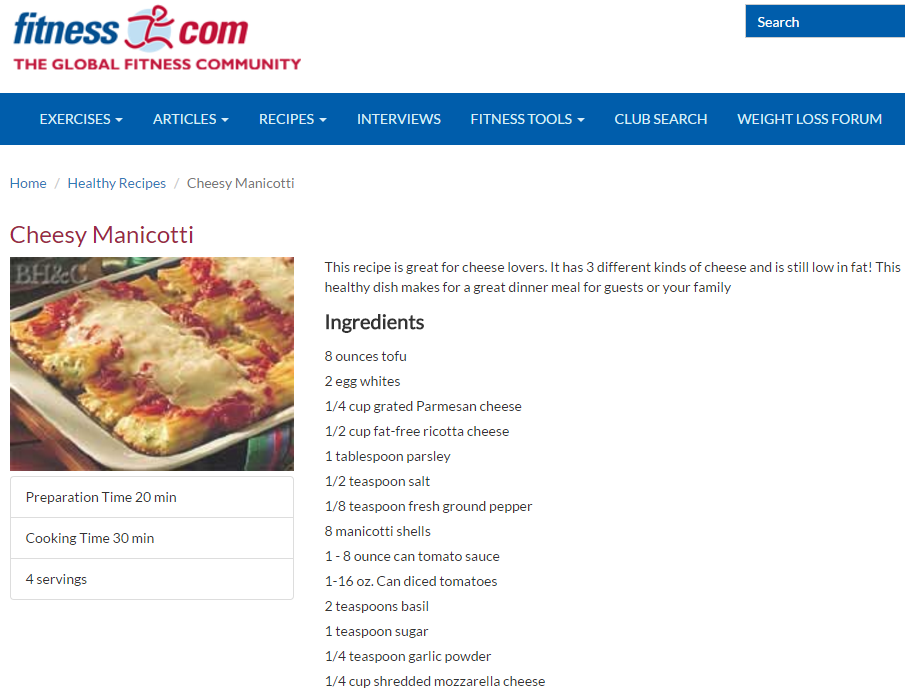


Рисунок 1.4 – fitness.com, рецепт блюда спортивного питания

Очень понравилось, что пользователи могут создавать свои собственные упражнения. Затем можно добавить ключевые слова (например, часть тела или часть оборудования), что означает, что другие пользователи могут найти эти упражнения. Затем они могут быть вставлены в программу тренировки.

Предыдущие воплощения Fitness.com имели платный вариант членства, который позволял настраивать свои тренировки и получать доступ к видео. Тем не менее, кажется, что это было заброшено.

Несмотря на то, что некоторые видеоролики кажутся немного любителями, понравилось, что участники могут участвовать в создании и адаптации упражнений, а не просто следовать тому, что говорят им профессионалы все время. Там также много достойной информации о спорте, который делает его сайтом, который стоит посетить, даже если вы решите использовать другой сайт.

Плюсы:

* сайт абсолютно бесплатен;
* возможность создать собственные упражнения;
* сайт абсолютно бесплатен;

Минусы:

* не у всех упражнений есть видео;

### sportswiki.ru

У российских сайтов по сравнению со своими зарубежными коллегами, по сути, преимущество только одно – они на русском. Популярных социальных сетей, посвященных здоровому и активному образу жизни у нас нет, а те, что имеются, находятся в самом начале своего развития и не поражают ни функциональностью, ни оформлением. Видео-порталы и полноценные трекеры просто отсутствуют, а основная доля общения происходит в группах популярных социальных сетей и сообществ, таких как ЖЖ, «Одноклассники», «Вконтакте», Facebook, Mail.ru и др. Из фитнес-сайтов у нас популярны информационные порталы. Однако есть все поводы полагать, что рано или поздно и у нас появятся трекеры и полноценные спортивные сообщества, в которых можно будет не только общаться, но и сравнивать результаты и стремиться к достижению целей.

Отдельно хотелось бы отметить сайт sportswiki.ru на котором собрана подробная информация о программах тренировок, упражнениях и программах питания.

Sportswiki – это спортивная энциклопедия на wiki-движке, эдакая «Википедия» про тренировки и питание (см. рис. 1.5). Ресурс позиционирует себя как «энциклопедия научного бодибилдинга», но, по сути, одним бодибилдингом дело не ограничивается – здесь есть множество статей о правильном питании, похудении и здоровье в целом.

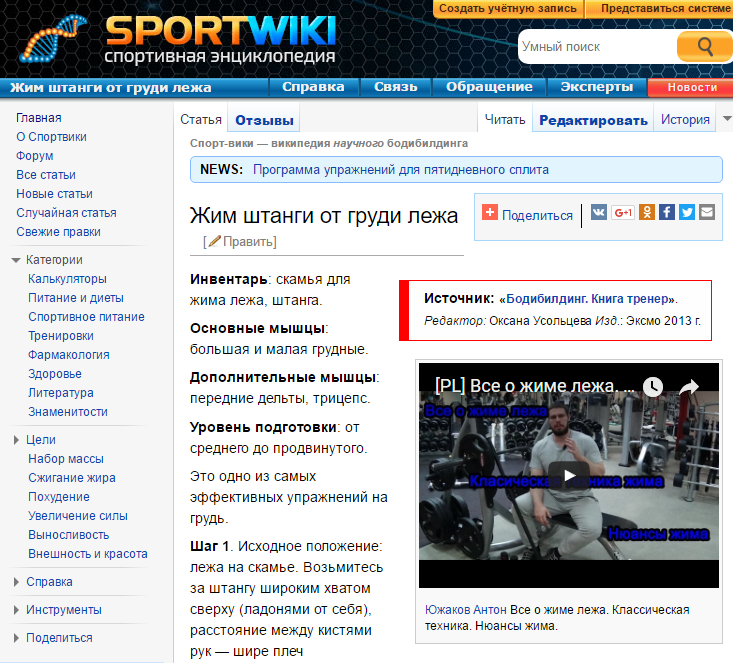


Рисунок 1.5 – sportswiki.ru, страница упражнения

SportsWiki – некоммерческий проект, однако имеет в своём штате квалифицированных специалистов, поэтому претендует на достоверность.

Плюсы:

* вся информация на сайте бесплатная;
* информации много;
* сайт на русском языке;

Минусы:

* нет возможности составлять программы тренировок, программы питания;
* сайт можно рассматривать только как онлайн-энциклопедию.

## Вывод

Рассмотрев существующие аналоги, можно сказать, что наиболее интересными и полезными являются англоязычными сайтами. Среди них есть достойные представители, которые с уверенностью могут называться многофункциональными порталами для людей, занимающихся спортом. Используя их, любой человек, даже новичок, сможет в скором времени начать добиваться успехов в занятиях спортом. На порталах можно создавать собственную программу тренировок, питания. Добавлять дни занятий в календарь. Наглядно видеть свой прогресс на графиках. У людей пропадает надобность вести бумажный дневник тренировок, ведь онлайн-дневник может оказаться наиболее полезен и информативен.

К сожалению, все вышесказанное является правдой в большинстве случаев только для зарубежных сайтов. Русскоязычные сайты в основном имеют или скудную функциональность, или устаревший дизайн, который может отпугивать пользователей. Это вынуждает людей пользоваться англоязычными ресурсами, что может вызывать затруднение для некоторых пользователей. Однако эту ситуацию некоторые люди могут рассматривать как возможность не только улучшить свое тело, но и знания иностранного языка.

## Формирование требований

В рамках дипломного проекта поставлена цель разработать веб-платформы с использованием технологии ASP.NET Core, предназначенной для людей, занимающихся спортом.

Минимальными требованиями к реализации программного продукта являются:

* реализовать удобную систему пользовательской функциональности, авторизацию, регистрацию, роли и группы пользователей, права доступа, управление доступом для администратора;
* обеспечить гибкость и расширяемость приложения;
* реализовать возможность составления собственной программы тренировок для пользователя

Для достижения поставленной цели необходимо решить следующие задачи:

изучить существующие функциональные аналоги порталов для людей, занимающихся спортом;

* на основании произведенного анализа выбрать и изучить необходимые технологии для реализации дипломного проекта;
* спроектировать гибкое архитектурное решение портала.

# ОБЗОР ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ

## Angular

Angular — [JavaScript](https://ru.wikipedia.org/wiki/JavaScript)-[фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) с [открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5). Предназначен для разработки [одностраничных приложений](https://ru.wikipedia.org/wiki/Single_Page_Application). Его цель — расширение браузерных приложений на основе [MVC](https://ru.wikipedia.org/wiki/Model-View-Controller) шаблона, а также упрощение тестирования и разработки.

Фреймворк работает с [HTML](https://ru.wikipedia.org/wiki/HTML), содержащим дополнительные пользовательские атрибуты, которые описываются директивами, и связывает ввод или вывод области страницы с моделью, представляющей собой обычные переменные JavaScript. Значения этих переменных задаются вручную или извлекаются из статических или динамических JSON-данных. [1]

Angular представляет фреймворк от компании Google для создания клиентских приложений. Прежде всего он нацелен на разработку SPA-решений (Single Page Application), то есть одностраничных приложений. В этом плане Angular является наследником другого фреймворка AngularJS. В то же время Angular это не новая версия AngularJS, а принципиальной новый фреймворк.

Angular 2 предоставляет такую функциональность, как двустороннее связывание, позволяющее динамически изменять данные в одном месте интерфейса при изменении данных модели в другом, шаблоны, маршрутизация и так далее.

Одной из ключевых особенностей Angular является то, что он использует в качестве языка программирования TypeScript. Поэтому перед началом работы рекомендуется ознакомиться с основами данного языка, про которые можно прочитать здесь.

Но мы не ограничены языком TypeScript. При желании можем писать приложения на Angular с помощью таких языков как Dart или JavaScript. Однако TypeScript все таки является основным языком для Angular.

## Язык программирования TypeScript

TypeScript — язык программирования, представленный Microsoft в 2012 году и позиционируемый как средство разработки веб-приложений, расширяющее возможности JavaScript.

Разработчиком языка TypeScript является Андерс Хейлсберг (англ. Anders Hejlsberg) создавший ранее Turbo Pascal, Delphi и C#.

Спецификации языка открыты и опубликованы в рамках соглашения Open Web Foundation Specification Agreement (OWFa 1.0).

TypeScript является обратно совместимым с JavaScript и компилируется в последний. Фактически, после компиляции программу на TypeScript можно выполнять в любом современном браузере или использовать совместно с серверной платформой Node.js. Код экспериментального компилятора, транслирующего TypeScript в JavaScript, распространяется под лицензией Apache. Его разработка ведётся в публичном репозитории через сервис GitHub.

TypeScript отличается от JavaScript возможностью явного статического назначения типов, поддержкой использования полноценных классов (как в традиционных объектно-ориентированных языках), а также поддержкой подключения модулей, что призвано повысить скорость разработки, облегчить читаемость, рефакторинг и повторное использования кода, помочь осуществлять поиск ошибок на этапе разработки и компиляции, и, возможно, скорость выполнения программ.

TypeScript возник из-за предполагаемых недостатков JavaScript в крупномасштабных приложениях как в Microsoft, так и у прочих пользователей JavaScript. Проблемы с разработкой сложных программ на JavaScript привели к необходимости облегчения разработки компонентов языка.

Разработчики TypeScript искали решение, которое не будет нарушать совместимость со стандартом и его кросс-платформенной поддержкой. Зная, что только стандарт ECMAScript предлагает поддержку в будущем для программирования на базе классов (Class-based programming), TypeScript был основан на этом предположении. Это привело к созданию компилятора JavaScript с набором синтаксических языковых расширений, увеличенным на основе предложения, которое трансформирует расширения в JavaScript. В этом смысле TypeScript является представлением того, что ожидать от ECMAScript 6. Уникальный аспект не в предложении, а в добавлении в TypeScript статической типизации, что позволяет статически анализировать язык, облегчая оснастки и IDE поддержку.

TypeScript является надстройкой над JavaScript. Таким образом, программа JavaScript также является правильной программой TypeScript, и программы TypeScript могут легко включать JavaScript. TypeScript компилирует ES3-совместимый JavaScript. По умолчанию компилируется ECMAScript 3, как преобладающей стандарт; также есть возможность создавать конструкции, используемые в ECMAScript 5.

С TypeScript можно использовать существующий JavaScript-код, включать популярные библиотеки JavaScript, и вызывать TypeScript-код, сгенерированный из других JavaScript. Объявление типов для этих библиотек поставляются вместе с исходным кодом. [2]

Однако, казалось бы, зачем нужен еще один язык программирования для клиентской стороны в среде Web, если со всей той же самой работой прекрасно справляется и традиционный JavaScript, который используется практически на каждом сайте, которым владеет множество разработчиков и поддержка которого в сообществе программистов довольно высока. Но TypeScript это не просто новый JavaScript.

Во-первых, следует отметить, что TypeScript - это строго типизированный и компилируемый язык, чем, возможно, будет ближе к программистам Java, C# и других строго типизированных языков. Хотя на выходе компилятор создает все тот же JavaScript, который затем исполняется браузером. Однако строгая типизация уменьшает количество потенциальных ошибок, которые могли бы возникнуть при разработке на JavaScript.

Во-вторых, TypeScript реализует многие концепции, которые свойственны объектно-ориентированным языкам, как, например, наследование, полиморфизм, инкапсуляция и модификаторы доступа и так далее.

В-третьих, потенциал TypeScript позволяет быстрее и проще писать большие сложные комплексные программы, соответственно их легче поддерживать, развивать, масштабировать и тестировать, чем на стандартном JavaScript.

В-четвертых, TypeScript развивается как opensource-проект и, как и многие проекты, хостится на гитхабе. Адрес репозитория - https://github.com/Microsoft/TypeScript. Кроме того, он является кроссплатформенным, а это значит, что для разработки мы можем испольвать как Windows, так и MasOS или Linux.

В то же время TypeScript является надмножеством JavaScript, а это значит, что любая программа на JS является программой на TypeScript. В TS можно использовать все те конструкции, которые применяются в JS - те же операторы, условные, циклические конструкции. Более того код на TS компилируется в javascript. В конечном счете, TS - это всего лишь инструмент, который призван облегчить разработку приложений.

## Программная платформа Microsoft .NET

.NET Framework — программная платформа, выпущенная компанией Microsoft в 2002 году. Основой платформы является общеязыковая среда исполнения Common Language Runtime (CLR), которая подходит для разных языков программирования. Функциональные возможности CLR доступны в любых языках программирования, использующих эту среду.

Считается, что платформа .NET Framework явилась ответом компании Microsoft на набравшую к тому времени большую популярность платформу Java компании Sun Microsystems (ныне принадлежит Oracle).

Хотя .NET является патентованной технологией корпорации Microsoft и официально рассчитана на работу под операционными системами семейства Microsoft Windows, существуют независимые проекты (прежде всего это Mono и Portable.NET), позволяющие запускать программы .NET на некоторых других операционных системах. В настоящее время .NET Framework получает развитие в виде .NET Core, изначально предполагающей кросcплатформенную разработку и эксплуатацию.[3]

При разработке .NET Framework учитывались следующие цели [4]:

- обеспечение согласованной объектно-ориентированной среды программирования для локального сохранения и выполнения объектного кода, для локального выполнения кода, распределенного в Интернете, либо для удаленного выполнения;

- обеспечение среды выполнения кода, минимизирующей конфликты при развертывании программного обеспечения и управлении версиями;

- обеспечение среды выполнения кода, гарантирующей безопасное выполнение кода, включая код, созданный неизвестным или не полностью доверенным сторонним изготовителем;

- обеспечение среды выполнения кода, исключающей проблемы с производительностью сред выполнения сценариев или интерпретируемого кода;

- обеспечение единых принципов работы разработчиков для разных типов приложений, таких как приложения Windows и веб-приложения;

- разработка взаимодействия на основе промышленных стандартов, которое обеспечит интеграцию кода платформы .NET с любым другим кодом.

Среди особенностей платформы .NET следует выделить наличие промежуточного языка выполнения (Common Intermediate Language, CIL). Все компиляторы, поддерживающие платформу .NET, должны транслировать код с языков высокого уровня платформы .NET на язык CIL.

По синтаксису и мнемонике язык CIL напоминает язык ассемблера. Его можно рассматривать как ассемблер виртуальной машины .NET. В то же время язык CIL содержит некоторые достаточно высокоуровневые конструкции, повышающие его уровень по сравнению с ассемблером для любой реально существующей машины, и писать код непосредственно на CIL легче, чем на ассемблере для реальных машин. Поэтому CIL можно рассматривать как своеобразный высокоуровневый ассемблер [5].

При запуске на выполнение приложения происходит JIT-компиляция (Just-In-Time) в машинный код, который затем выполняется. Основная цель использования JIT - достичь и превзойти производительность статической компиляции, сохраняя при этом преимущества динамической компиляции [6]:

- большинство тяжеловесных операций, таких как парсинг исходного кода и выполнение базовых оптимизаций, происходит во время компиляции (до развёртывания), в то время как компиляция в машинный код из байт-кода происходит быстрее, чем из исходного кода;

- байт-код более переносим (в отличие от машинного кода);

- среда может контролировать выполнение байт-кода после компиляции, поэтому приложение может быть запущено в песочнице.

- компиляторы из байт-кода в машинный код легче в реализации, так как большинство работы по оптимизации уже было проделано компилятором.

JIT, как правило, эффективней, чем интерпретация кода. К тому же в некоторых случаях JIT может показывать большую производительность по сравнению со статической компиляцией за счёт оптимизаций, возможных только во время исполнения [6]:

- Компиляция может осуществляться непосредственно для целевого процессора и операционной системы, на которой запущено приложение. Например, JIT может использовать векторные SSE2 расширения процессора, если он обнаружит их поддержку. Однако, до сих пор нет основных реализаций JIT, где этот подход бы использовался, ведь чтобы обеспечить подобный уровень оптимизации, сравнимый со статическими компиляторами, потребовалось бы либо поддерживать бинарный файл под каждую платформу, либо включать в одну библиотеку оптимизаторы под каждую платформу.

- Среда может собирать статистику о работающей программе и производить оптимизации с учётом этой информации. Некоторые статические компиляторы также могут принимать на вход информацию о предыдущих запусках приложения.

- Среда может делать глобальные оптимизации кода (например, встраивание библиотечных функций в код) без потери преимуществ динамической компиляции и без накладных расходов, присущих статическим компиляторам и линкерам.

Платформа Microsoft .NET состоит из общеязыковой среды разработки (Common Language Runtime, CLR) и общирного набора библиотек.

Общеязыковая среда разработки выполняет большое количество различных задач [4]:

- Управление памятью, выполнением потоков, выполнением кода, проверкой безопасности кода, компиляцией и другими системными службами. Эти средства являются внутренними для управляемого кода, который выполняется в среде CLR.

- Среда выполнения обеспечивает управление доступом для кода. Например, пользователи могут доверить исполняемому приложению, внедренному в веб-страницу, воспроизведение анимации на экране или звукозаписи, не позволяя ему при этом получить доступ к личным данным, файловой системе или сети. Таким образом, средства безопасности CLR предоставляют подлинному развернутому в Интернете программному обеспечению исключительно богатые функции.

- Обеспечение наджености кода, реализуя инфраструктуру строгой типизации и проверки кода, которую называют системой общих типов (CTS). Система общих типов обеспечивает самоописание всего управляемого кода. Различные языковые компиляторы корпорации Microsoft и независимых изготовителей создают управляемый код, удовлетворяющий системе общих типов. Это означает, что управляемый код может принимать другие управляемые типы и экземпляры, при этом обеспечивая правильность типов и строгую типизацию.

- Исключение многих часто возникающие проблемы с программным обеспечением. Например, среда выполнения автоматически управляет размещением объектов и ссылками на объекты, освобождая их, когда они больше не используются. Автоматическое управление памятью исключает две наиболее часто возникающие ошибки приложений: утечки памяти и недействительные ссылки на память.

- Среда выполнения также повышает продуктивность разработчиков. Например, программисты могут писать приложения на привычном языке разработки, при этом используя все преимущества среды выполнения, библиотеку классов и компоненты, написанные другими разработчиками на других языках. Это доступно любому производителю компиляторов, обращающихся к среде выполнения. Языковые компиляторы, предназначенные для платформы .NET Framework, делают средства .NET Framework доступными для существующего кода, написанного на соответствующих языках, существенно облегчая процесс переноса существующих приложений/

- Хотя среда выполнения разрабатывалась для будущего программного обеспечения, она также поддерживает сегодняшнее и вчерашнее программное обеспечение. Взаимодействие управляемого и неуправляемого кодов позволяет разработчикам использовать необходимые компоненты COM и библиотеки DLL.

- Среда выполнения разработана для повышения производительности. Хотя общеязыковая среда выполнения предоставляет многие стандартные службы времени выполнения, управляемый код никогда не интерпретируется. Средство компиляции по требованию (JIT) позволяет всему управляемому коду выполняться на машинном языке компьютера. Между тем диспетчер памяти устраняет возможность фрагментации памяти и увеличивает объем адресуемой памяти для дополнительного повышения производительности.

## Язык программирования C#

C# является универсальным, безопасным в отношении типов, объектно-ориентированным языком программирования. Цель C# заключается в обеспечении продуктивности работы программистов. Для этого в языке соблюдается баланс между простотой, выразительностью и надежностью. С самой первой версии главным архитектором языка C# был Андерс Хейлсберг (создатель Turbo Pascal и архитектор Delphi). Язык C# нейтрален в отношении платформ, но проектировался для эффективной работы с платформой Microsoft .NET Framework.

Особенностями языка C# c объектно-ориентированной точки зрения являются [7]:

- Унифицированная система типов. Фундаментальным строительным блоком в C# является инкапсулированная единица данных и функций, называющаяся типом. Язык имеет унифицированную систему типов, где все типы, независимо от того, представляют ли они бизнес-объекты или примитивные сущности вроде чисел, совместно используют один и тот же базовый набор функциональности.

- Классы и интерфейсы. В рамках традиционной объектно-ориентированной парадигмы единственной разновидностью типа является класс. В C# присутствуют типы многих других видов, один из которых представляет собой интерфейс. Интерфейс похож на класс за исключением того, что он только описывает члены. Реализация для этих членов поступает из типов, которые реализуют данный интерфейс.

- Свойства, методы и события. В чистой объектно-ориентированной парадигме все функции называются методами. В C# методы представляют собой только одну разновидность функций-членов, и различают также свойства и события (помимо прочих). Свойства – функции-члены, которые инкапсулируют фрагмент состояния объекта. События – функции-члены, которые упрощают выполнение действий при изменении состояния объекта.

С# является типобезопасным языком. Это значит, что экземпляры типов могут взаимодействовать только через определяемые ими протоколы, подобным образом обеспечивая внутреннюю согласованность каждого типа. Поддерживаются статическая типизация и типизация времени выполнения. Статическая типизация обеспечивает безопасность типов на этапе компиляции, что позволяет избежать значительной категории ошибок еще до запуска программы, а также повышает производительность труда программистов [7].

По историческим причинам C# применялся в основном для написания приложений, выполняющихся на платформах Windows. Однако на сегодняшний день есть возможность писать и под другие платформы, такие как Mac OS X, iOS, Android.

C# постоянно эволюционирует. Команда разработки языка стремится взять лучшее из существующих парадигм и приемов программирования для создания максимально удобного и продуктивного средства написания кода. Далее приведен краткий обзор версий языка и их особенностей.

Первая версия C# была в значительной схожа с Java 1.4, но имея свои особенности как свойства (выглядящие в коде как поля объекта, но на деле вызывающие при обращении к ним методы класса), индексаторы (подобные свойствам, но принимающие параметр как индекс массива), события, делегаты, циклы foreach, структуры, передаваемые по значению, автоматическое преобразование встроенных типов в объекты при необходимости (boxing), атрибуты, встроенные средства взаимодействия с неуправляемым кодом (DLL, COM) и прочее [8].

Кроме того, в C# решено было перенести некоторые возможности C++, отсутствовавшие в Java: беззнаковые типы, перегрузку операторов (с некоторыми ограничениями, в отличие от C++), передача параметров в метод по ссылке, методы с переменным числом параметров, оператор goto (с ограничениями). Также в C# оставили ограниченную возможность работы с указателями — в местах кода, специально обозначенных словом unsafe и при указании специальной опции компилятору [8].

Вторая версия языка предоставила следующие возможности [8]:

- Частичные типы (разделение реализации класса более чем на один файл).

- Обобщённые, или параметризованные типы (generics). В отличие от шаблонов C++, они поддерживают некоторые дополнительные возможности и работают на уровне виртуальной машины. Вместе с тем, параметрами обобщённого типа не могут быть выражения, они не могут быть полностью или частично специализированы, не поддерживают шаблонных параметров по умолчанию, от шаблонного параметра нельзя наследоваться.

- Новая форма итератора, позволяющая создавать сопрограммы с помощью ключевого слова yield.

- Анонимные методы, обеспечивающие функциональность замыкания.

- Обнуляемые типы-значения, представляющие собой те же самые типы-значения, способные принимать также значение null. Такие типы позволяют улучшить взаимодействие с базами данных через язык SQL.

- Поддержка 64-разрядных вычислений, что кроме всего прочего, позволяет увеличить адресное пространство и использовать 64-разрядные примитивные типы данных.

Третья версия языка привнесла следующие возможности [8]:

- ключевые слова select, from, where, позволяющие делать запросы из XML документов, коллекций;

- инициализация объекта вместе с его свойствами;

- деревья выражений: лямбда-выражения теперь могут представляться в виде структуры данных, доступной для обхода во время выполнения, тем самым позволяя транслировать строго типизированные C#-выражения в другие домены (например, выражения SQL);

- анонимные типы.

Новые возможности в четвертой версии [8]:

- возможность использования позднего связывания;

- именованные и опциональные параметры;

- контракты в коде;

- библиотека параллельных задач TPL (Task Parallel Library).

Новые возможности в пятой версии C# [8]:

- Шаблон TAP (Task-based Asynchronous Pattern). TAP использует один метод для представления инициализации и завершения асинхронной операции.

- Асинхронные методы (async и await) - как реализация шаблона TAP.

Шестая версия языка обладает следующими особенностями [8]:

- null-условные операции;

- функции сжатые до выражений (expression-bodied functions)

- инициализаторы свойств;

- инициализаторы индексов;

- интерполяция строк;

- фильтры исключений;

- импорт статических функций из класса.

Новые возможности седьмой версии языка:

- Сопоставление с шаблоном. Вводится понятие шаблона (pattern), который представляет собой синтаксическую конструкцию, позволяющую проверить соответствие переменной определенной форме и извлечь из нее информацию.

- Кортежи.

- Локальные функции. Теперь функцию, которая используется только в теле какого-либо метода можно объявить прямо в теле этого метода.

- Локальные переменные и возвращаемые значения по ссылке. Расширена функциональность ключевого слова ref. Теперь можно возвратить данные из метода или сохранить их в локальной переменной по ссылке.

- Расширение списка типов, возвращаемых асинхронными методами.

## Entity Framework Core

Entity Framework Core (EF Core) представляет собой объектно-ориентированную, легковесную и расширяемую технологию от компании Microsoft для доступа к данным. EF Core является ORM-инструментом (object-relational mapping - отображения данных на реальные объекты). То есть EF Core позволяет работать базами данных, но представляет собой более высокий уровень абстракции: EF Core позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работаем с объектами.

Entity Framework Core поддерживает множество различных систем баз данных. Таким образом, мы можем через EF Core работать с любой СУБД, если для нее имеется нужный провайдер.

По умолчанию на данный момент Microsoft предоставляет ряд встроенных провайдеров: для работы с MS SQL Server, для SQLite, для PostgreSQL. Также имеются провайдеры от сторонних поставщиков, например, для MySQL.

Также стоит отметить, что EF Core предоставляет универсальный API для работы с данными. И если, к примеру, мы решим сменить целевую СУБД, то основные изменения в проекте будут касаться прежде всего конфигурации и настройки подключения к соответствующим провайдерам. А код, который непосредственно работает с данных, получает данные, добавляет их в БД и т.д., останется прежним.

Entity Framework Core многое унаследовал от своих предшественников, в частности, Entity Framework 6. В тоже время надо понимать, что EF Core - это не новая версия по отношению к EF 6, а совершенно иная технология, хотя в целом принципы работы у них будут совпадать. Поэтому в рамках EF Core используется своя система версия. Текущая версия - 1.1 была выпущена в ноябре 2016 года. И технология продолжает развиваться. Что уже есть в EF Core, а что только планируется добавить, можно посмотреть в роудмапе на гитхабе.

Как технология доступа к данным Entity Framework Core может использоваться на различных платформах стека .NET. Это и стандартные платформы типа Windows Forms, консольные приложения, WPF, ASP.NET 4.6/4.5. Это и новые технологии как UWP и ASP.NET Core. При этом кроссплатформенная природа EF Core позволяет задействовать ее не только на ОС Windows, но и на Linux и Mac OS X.

Центральной концепцией Entity Framework является понятие сущности или entity. Сущность определяет набор данных, которые связаны с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их коллекциями. [9]

Любая сущность, как и любой объект из реального мира, обладает рядом свойств. Например, если сущность описывает человека, то мы можем выделить такие свойства, как имя, фамилия, рост, возраст. Свойства необязательно представляют простые данные типа int или string, но могут также представлять и более комплексные типы данных. И у каждой сущности может быть одно или несколько свойств, которые будут отличать эту сущность от других и будут уникально определять эту сущность. Подобные свойства называют ключами.

При этом сущности могут быть связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим, подобно тому, как в реальной базе данных происходит связь через внешние ключи.

Отличительной чертой Entity Framework Core, как технологии ORM, является использование запросов LINQ для выборки данных из БД. С помощью LINQ мы можем создавать различные запросы на выборку объектов, в том числе связанных различными ассоциативными связями. А Entity Framework при выполнении запроса транслирует выражения LINQ в выражения, понятные для конкретной СУБД (как правило, в выражения SQL).

## ASP.NET Core

Платформа ASP.NET Core представляет технологию от компании Microsoft, предназначенную для создания различного рода веб-приложений: от небольших веб-сайтов до крупных веб-порталов и веб-сервисов.

ASP.NET Core представляет собой продолжения развития платформы ASP.NET, но в то же время это не очередной релиз, а переосмысление всей платформы, ее революционное изменение.

ASP.NET Core построен на основе кросс-платформенной среды .NET Core, которая может быть развернута на основных популярных операционных системах: Windows, Mac OS X, Linux, что позволяет запускать веб-приложения не только на ОС Windows, но и на Linux и Mac OS. А для развертывания веб-приложения можно использовать традиционный IIS, либо кроссплатформенный веб-сервер Kestrel.

Хотя ASP.NET Core преимущественно нацелено на использование .NET Core, но фреймворк также может работать и с полной версией фреймворка .NET.

ASP.NET Core характеризуется расширяемостью. Фреймворк построен из набора относительно независимых компонентов. Теперь у пользователя есть возможность использовать встроенную реализацию этих компонентов, либо расширить их с помощью механизма наследования, либо вовсе создать и применять свои компоненты со своим функционалом.

Также было упрощено управление зависимостями и конфигурирование проекта. Фреймворк теперь имеет свой легковесный контейнер для внедрения зависимостей, и больше нет необходимости применять сторонние контейнеры, такие как Autofac, Ninject. Хотя при желании их также можно продолжать использовать.

При разработке в Visual Studio 2015/2017 проекты приложений имеют встроенную поддержку с такими популярными инструментами, как Bower, Grunt, Gulp, который позволяют управлять скриптами JavaScript и стилями CSS, автоматизировать и оптимизировать процесс веб-разработки.

Для обработки запросов теперь используется новый конвейер HTTP, который основан на компонентах Katana и спецификации OWIN. А его модульность позволяет легко добавить свои собственные компоненты. [10]

Если суммировать, то можно выделить следующие ключевые отличия ASP.NET Core от предыдущих версий ASP.NET:

- новый легковесный и модульный конвейер HTTP-запросов;

- возможность развертывать приложение как на IIS, так и в рамках своего собственного процесса;

- использование платформы .NET Core и ее функциональности;

- распространение пакетов платформы через NuGet;

- интегрированная поддержка для создания и использования пакетов NuGet;

- единый стек веб-разработки, сочетающий Web UI и Web API;

- конфигурация для упрощенного использования в облаке;

- встроенная поддержка для внедрения зависимостей;

- расширяемость;

- кроссплатформенность: возможность разработки и развертывания приложений ASP.NET на Windows, Mac и Linux;

- развитие как open source, открытость к изменениям.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Архитектура программного обеспечения – это структура программы или вычислительной системы, которая включает программные компоненты, видимые снаружи свойства этих компонентов, а также отношения между ними.

Процесс проектирования архитектуры программного обеспечения включает в себя сбор требований клиентов, их анализ и создание проекта для компонента программного обеспечения в соответствие с требованиями. Успешная разработка ПО должна обеспечивать баланс неизбежных компромиссов вследствие противоречащих требований; соответствовать принципам проектирования и рекомендованным методам, выработанным со временем; и дополнять современное оборудование, сети и системы управления. Надежная архитектура программного обеспечения требует значительного опыта в теоретических и практических вопросах, а также воображения, необходимого для преобразования бизнес-сценариев и требований, которые могут казаться неясными, в надежные и практичные рабочие проекты.

Архитектура программного обеспечения включает в себя определение структурированного решения, соответствующего всем техническим и рабочим требованиям, одновременно оптимизируя общие атрибуты качества, такие как производительность, безопасность и управляемость. Сюда входит серия решений, основанных на широком диапазоне факторов, и каждое из этих решений может значительно влиять на качество, производительность и общий успех программного обеспечения.

Современное программное обеспечение редко бывает автономным. Как минимум, в большинстве случаев оно будет взаимодействовать с источником данных, например, корпоративной базой данных, предоставляющим информацию, с которой работают пользователи программного обеспечения. Обычно современное программное обеспечение также должно взаимодействовать с другими службами и сетевыми функция для выполнения проверки подлинности, получения и публикации информации.

Архитектуру программного обеспечения можно рассматривать как сопоставление между целью компонента ПО и сведениями о реализации в коде. Правильное понимание архитектуры обеспечит оптимальный баланс требований и результатов. Программное обеспечение с хорошо продуманной архитектурой будет выполнять указанные задачи с параметрами исходных требований, одновременно обеспечивая максимально высокую производительность, безопасность, надежность и многие другие факторы.

На самом высоком уровне проект архитектуры должен предоставлять структуру системы, но скрывать детали реализации; охватывать все случаи применения и сценарии; пытаться учитывать требования всех заинтересованных лиц; и удовлетворять настолько, насколько возможно, всем функциональные требованиям и требованиям к качеству (см. рис. 3.1).

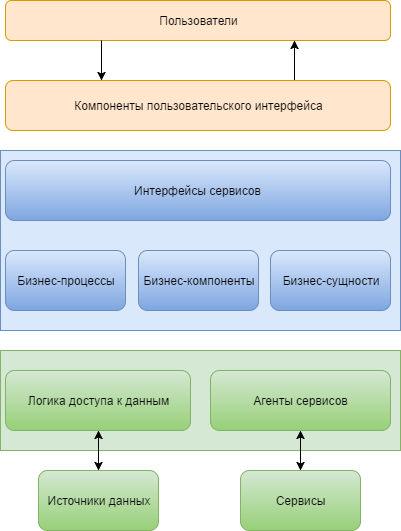


Рисунок 3.1 – Типичная архитектура web-приложения

При написании дипломного проекта использовался паттерн MVC (см. рис. 3.2).

По большому счету MVC паттерн обозначает, что MVC приложение будет разделено как минимум на три части:

* Модели, которые содержат или представляют данные, с которыми работают пользователи. Это могут быть простые модели представления, которые только представляют данные, передаваемые от контроллера представлению, или они могут быть доменными моделями, которые содержат данные домена, а также операции, преобразования и правила работы с этими данными.
* Представления, которые используются для того, чтобы обработать некоторые части модели в качестве пользовательского интерфейса.
* Контроллеры, которые обрабатывает входящие запросы, выполняют операции для модели и выбирают представления для показа пользователю.

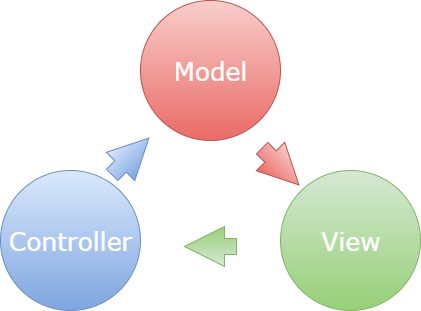


Рисунок 3.2 – MVC паттерн

Основная цель применения этой концепции состоит в отделении бизнес-логики (модели) от её визуализации (представления, вида). За счёт такого разделения повышается возможность повторного использования кода. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения. [11]

Модели являются определением вселенной, в которой работает приложение. В банковском приложении, например, модель представляет собой все в банке, что поддерживает приложение: то есть счета, главную книгу и кредитные лимиты для клиентов, а также операции, которые могут быть использованы для манипулирования данными в модели, такие как депонирование средств и осуществление изъятия со счетов.

Модель также несет ответственность за сохранение общего состояния и согласованности данных, например, она гарантирует, что все сделки будут добавлены в книгу, и что клиент не сможет снять больше денег, чем он может, или больше денег, чем имеет банк.

Модели также определяются тем, за что они не несут ответственность: модели не занимаются UI и обработкой запросов – это обязанности представлений и контроллеров. Представления содержат логику, необходимую для отображения элементов модели пользователю, и больше ничего. Они не имеют прямого понимания модели и никоим образом напрямую не сообщаются с моделью.

Контроллеры являются мостом между представлениями и моделью: запросы приходят от клиента и обслуживаются контроллером, который выбирает соответствующее представление для показа пользователю и, при необходимости, соответствующие действия, которые нужно выполнить с моделью.

Каждый элемент MVC архитектуры является четко определенным и автономным – это называется разделением понятий. Логика, которая манипулирует данными в модели, содержится только в модели; логика, которая отображает данные, находится только в представлении, а код, который обрабатывает запросы пользователей и вводные данные, содержится только в контроллере. С четким разделением обязанностей между каждой из частей ваше приложение будет легче поддерживать и расширять, независимо от того, насколько большим оно будет становиться.

MVC выступает в качестве паттерна для веб приложений в связи с тем, что взаимодействие пользователя с MVC приложением следует естественному циклу: пользователь совершает действие, в ответ на это приложение меняет свою модель данных и предоставляет пользователю обновленный вид. А затем цикл повторяется. Это очень удобно для веб-приложений, предоставляемых в виде серии HTTP запросов и ответов. Необходимость веб приложению объединять несколько технологий (например, базы данных, HTML и исполняемый код), как правило, разбивается на множество уровней или слоев. Моделей, которые вытекают из этих комбинаций, естественны для концепции MVC. [12]

Наиболее важной частью MVC приложения является доменная модель. Мы создаем модели путем выявления реальных лиц, операций и правил, которые существуют в отрасли или деятельности.

Они должны поддерживаться нашим приложением, и они известны как домен.

Затем мы создаем программное представление домена – доменную модель. Для наших целей доменная модель представляет собой набор типов C# (классы, структуры и т.д.), известных под общим названием доменные типы. Операции из домена представлены методами, определенными в доменных типах, а доменные правила выражаются в логике внутри этих методов, или, как мы видели в предыдущей главе, в применении C# атрибутов к методам. Когда мы создаем экземпляр доменного типа, чтобы представить определенный фрагмент данных, мы создаем доменный объект. Доменные модели, как правило, постоянны – есть много разных способов достижения этого, но реляционные базы данных остаются наиболее распространенным выбором.

Доменные объекты — это объекты в объектно-ориентированных компьютерных программах, выражающие сущности из модели предметной области, относящейся к программе, и реализующие бизнес-логику программы. [13]

Короче говоря, доменная модель является отдельным авторитетным определением бизнес данных и процессов внутри вашего приложения. Постоянная доменная модель также является внушительным определением состояния доменного представления.

Дипломный проект состоит из четырех слоев:

* Слой представления;
* Серверный слой;
* Слой сервисов;
* Слой доступа к данным.

Каждый слой реализует определенные функции и решает свои задачи. Все модули реализованы на языке C# c применением .NET Core 1.1.

## Слой представления

Слой представления — это тот уровень, с которым непосредственно взаимодействует пользователь. Этот уровень включает компоненты пользовательского интерфейса, механизм получения ввода от пользователя. Применительно к ASP.NET MVC на данном уровне расположены представления и все те компоненты, который составляют пользовательский интерфейс (стили, статичные страницы HTML, JavaScript), а также модели представлений, контроллеры, объекты контекста запроса.

При чем об ASP.NET MVC мы говорим прежде всего применительно к уровню представления, остальные же уровни могут быть реализованы независимо и могут использоваться в приложениях на других технологиях, как Windows Forms, WPF и т.д. И, как правило, все приложение в целом будет представлять решение (solution) в Visual Studio, а отдельные уровни - проекты. В то же время неверно полагать, что, если уровень обязательно должен соответствовать отдельному проекту. При необходимости мы можем раздробить один уровень на несколько проектов, главное, чтобы его функционал представлял единое логическое звено.

Традиционно слой представления реализовывался таким образом, что после очередного запроса на сервер возвращался HTML (см. рис. 3.3). Это приводило к обновлению всей страницы. Однако в последствии появился подход, который называется SPA (см. рис. 3.4). С помощью этого подхода и реализован слой представления дипломного проекта.

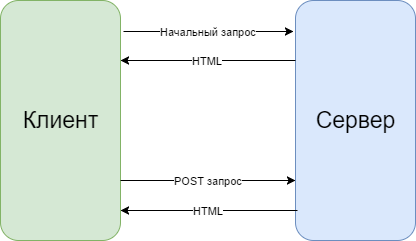


Рисунок 3.3 – Традиционный жизненный цикл страницы

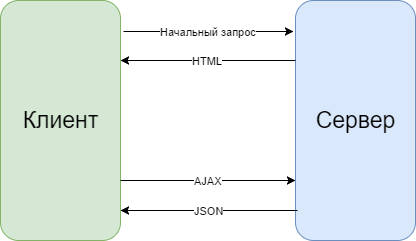


Рисунок 3.4 – Жизненный цикл в SPA

Single Page Application – сокращенно SPA, в переводе на русский язык означает «Приложение одной страницы». Другими словами, SPA – это web-приложение, размещенное на одной web-странице, которая для обеспечения работы загружает весь необходимый код вместе с загрузкой самой страницы. Приложение такого типа появились сравнительно недавно, с началом эры HTML5 и SPA является типичным представителем приложений на HTML5.

Как мы знаем, HTML5 это нечто иное как HTML + CSS3 + JavaScript. Таким образом, SPA - это приложения, написанные на языке JavaScript. И, следовательно, немного перефразировав предыдущие определение получаем:

«SPA – это web-приложение, размещенное на одной странице, которая для обеспечения работы загружает все JavaScript-файлы (модули, виджеты и т.д.), а также файлы CSS вместе с загрузкой самой страницы.»

Приложения на SPA отлично работают на устройствах как стационарных, так и мобильных. Персональные компьютеры, планшеты, смартфоны и простые телефоны (некоторые) могут беспрепятственно работать с сайтами построенных по принципу SPA. Итак, первый “плюс” – работа на большом количестве устройств.

Богатый пользовательский интерфейс, так называемый User Experience. Так как web-страница одна, построить богатый, насыщенный пользовательский интерфейс гораздо проще. Проще хранить информацию о сеансе, управлять состояниями представлений и управлять анимацией (в некоторых случаях).

SPA существенно (в разы) сокращает загрузку одного и того же контента снова и снова. Если портал (сайт) использует шаблон, то вместе с основным содержанием какой-либо страницы посетитель сайта обязательно загружает разметку шаблона. Да, кэширование данных на данном этапе развития WWW достигло высочайших результатов, но если нечего кэшировать, то и время, и ресурсы на это не тратятся.

В данном дипломном проекте SPA было реализовано с использование фреймворка Angular 2. Это cделано для улучшения взаимодействия с пользователем.

Angular представляет фреймворк от компании Google для создания клиентских приложений. Прежде всего он нацелен на разработку SPA-решений (Single Page Application), то есть одностраничных приложений. В этом плане Angular является наследником другого фреймворка AngularJS. В то же время Angular это не новая версия AngularJS, а принципиальной новый фреймворк.

Для работы с Angular необходимо установить сервер Node.js и пакетный менеджер npm, если они отсутствуют на рабочей машине. Для установки можно использовать программу установки node.js. Вместе с сервером она также установить и npm. При этом особого какого-то знания для работы с NodeJS и npm не требуется.

После установки node.js и npm необходимо установить и настроить Angular. Для этого нужно создать конфигурационные файлы.

Файл package.json устанавливает пакеты и зависимости, которые будут использоваться проектом.

Для определения кода приложения применяется язык TypeScript, поэтому в папке проекта был создан новый файл tsconfig.json. Данный файл определяет настройки для компилятора TypeScript.

Затем в папке проекта была создана папка app, где и находится само приложение.

Приложение Angular состоит из модулей. Модульная структура позволяет легко подгружать и задействовать только те модули, которые непосредственно необходимы. И каждое приложение имеет как минимум один корневой модуль. Поэтому в большинстве проектах на Angular 2 будет пристуствовать файл с модулем app.module.ts

Компоненты представляют основные строительные блоки приложения Angular 2. Каждое приложение Angular имеет как минимум один компонент. Компонент управляет отображением представления на экране.

## Серверный слой

Большинство крупных веб-сайтов используют программирование серверной части чтобы динамично отображать различные данные при необходимости, в основном взятые из базы данных, располагающейся на сервере и отсылаемые клиенту через некоторый код (например, HTML и JavaScript). Самая значительная польза программирования серверной части в том, то оно позволяет формировать контент веб-сайта под конкретного пользователя. Основным компонентом серверного слоя является API, по которому происходит общение между клиентом и сервером. Разработка API является одной из важнейших частей при проектирование серверного слоя. Ведь интерфейс API не должен часто меняться с течением времени, т.к. его изменение повлечет за собой изменение все клиентских приложений, которые работают с API (см. рис. 3.5).

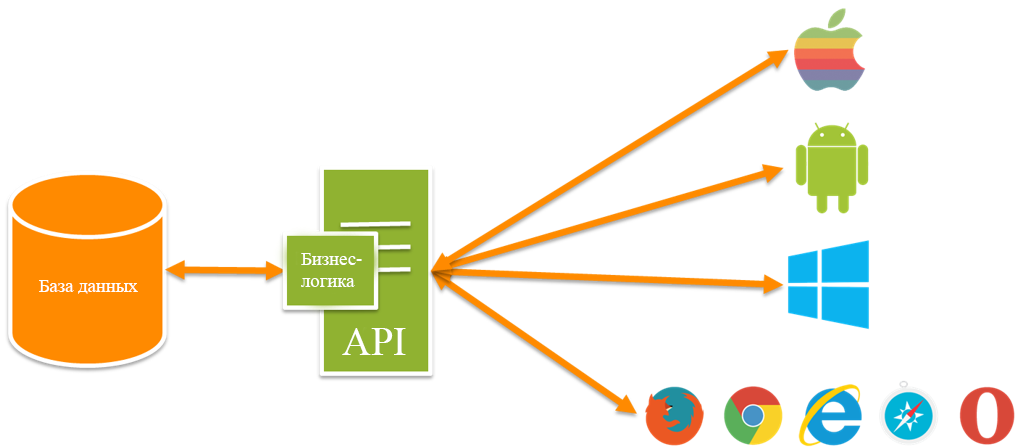


Рисунок 3.5 – Использование API множеством клиентских приложений

API – это, в первую очередь, интерфейс. Интерфейс, который позволяет разработчикам использовать готовые блоки для построения приложения. В случае веб-приложений, API может отдавать данные в отличном от стандартного HTML формате, благодаря чему им удобно пользоваться при написании собственных приложений. Сторонние общедоступные API чаще всего отдают данные в одном из двух форматов: XML или JSON. JSON намного более лаконичен и прост в чтении, чем XML, а сервисы, предоставляющие доступ к данным в XML-формате, постепенно отказываются от последнего.

На основе API строятся такие вещи, как карты 2GIS, всевозможные мобильные и десктопные клиенты для Twitter и Vkontakte. Все их функции стали возможными именно благодаря тому, что соответствующие сервисы имеют качественные и детально документированные API.

В данном дипломном проекте используется REST API.

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами.

В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.

Отсутствие дополнительных внутренних прослоек означает передачу данных в том же виде, что и сами данные. Т.е. мы не заворачиваем данные в XML или JSON и просто отдаем сами данные.

Каждая единица информации однозначно определяется URL – это значит, что URL по сути является первичным ключом для единицы данных. Т.е., например, третье упражнение из списка упражнений будет иметь вид /exercise/3. Отсюда и получается строго заданный формат. Причем совершенно не имеет значения, в каком формате находятся данные по адресу /exercise/3 – это может быть и HTML, и отсканированная копия в виде jpeg-файла, и документ Microsoft Word.

Как происходит управление информацией сервиса – это целиком и полностью основывается на протоколе передачи данных. Наиболее распространенный протокол конечно же HTTP. Для HTTP действие над данными задается с помощью методов: GET (получить), PUT (добавить, заменить), POST (добавить, изменить, удалить), DELETE (удалить). Таким образом, действия CRUD (Create-Read-Update-Delete) могут выполняться как со всеми 4-мя методами, так и только с помощью GET и POST.

Примеры:

* GET /exercise/ — получить список всех упражнений
* GET /exercise/3/ — получить упражнение номер 3
* PUT /exercise/ — добавить упражнение (данные в теле запроса)
* POST /exercise/3 – изменить упражнение (данные в теле запроса)
* DELETE /exercise/3 – удалить упражнение

Существуют так называемые REST-Patterns, которые различаются связыванием HTTP-методов с тем, что они делают. В частности, разные паттерны по-разному рассматривают POST и PUT. Однако, PUT предназначен для создания или изменения, для POST это не определено.

POST может использоваться одновременно для всех действий изменения:

* POST /exercise/ – добавить упражнение (данные в теле запроса)
* POST /exercise/3 – изменить упражнение (данные в теле запроса)
* POST /exercise/3 – удалить упражнение (тело запроса пустое)

Это позволяет иногда обходить неприятные моменты, связанные с неприятием PUT и DELETE.

Для реализации REST API в дипломном проекте была выбрана технология ASP.NET Core WebAPI. Web API представляет способ построения приложения ASP.NET, который специально заточен для работы в стиле REST.

Проект имеет следующую структуру:

* Dependencies: все добавленные в проект пакеты и библиотеки;
* wwwroot: этот узел (на жестком диске ему соответствует одноименная папка) предназначен для хранения статических файлов - изображений, скриптов JavaScript, файлов CSS и т.д., которые используются приложением. Цель добавления этой папки в проект по сравнению с другими версиями ASP.NET, состоит в разграничении доступа к статическим файлам, к которым разрешен доступ со стороны клиента и к которым доступ запрещен. Однако в случае ASP.NET Core WebAPI приложения эта папка остается пустой;
* Controllers: папка для хранения контроллеров, используемых приложением;
* appsettings.json: хранит конфигурацию приложения;
* bower.json: файл, который управляет клиентскими зависимостями (библиотеки JavaScript и CSS), которые подключаются через менеджер пакетов Bower;
* bundleconfig.json: файл, который содержит задачи по минификации используемых скриптов и стилей, которые выполняются при построении проекта;
* Program.cs: файл, определяющий класс Program, который инициализирует и запускает хост с приложением;
* Startup.cs: файл, определяющий класс Startup, с которого начинается работа приложения. То есть это входная точка в приложение;

## Слой доступа к данным

Слой доступа к данным позволяет абстрагировать остальную часть приложения от деталей реализации конкретной технологии для работы с БД.

Этот слой важно выделять отдельно. Ведь в какой-то момент может потребоваться сменить технологию хранения данных либо начать использовать новые технологии одновременно с текущими. Однако если при проектировании приложения его бизнес-логика не была отделена от работы с хранилищами данных, то смена инструмента хранения может привести к дорогостоящей и плохо управляемой миграции.

Проблему разделения бизнес-логики и работы с данными на уровне отдельного приложения решает архитектурный шаблон Data Access Layer (DAL).

Разрабатываемая концепция должна подготавливать приложения к плавной и технологичной смене СУБД в случае необходимости или, возможно, из соображений технологической модернизации и технологического развития.

В результате, с одной стороны, приложение может использовать специализированные средства хранения, получая к ним доступ через четко определенный программный интерфейс. С другой стороны, если потребуется замена какой-то конкретной СУБД на аналогичную, не придется осуществлять серьезные переделки в приложении. В крайнем случае — относительно недорогая адаптация к мелким особенностям новой технологии без необходимости пересмотра архитектуры.

Структура DAL — это набор программных средств, из которых может быть построен слой унифицированного доступа к данным, реализующий принцип «управляемого специализированного хранения данных» и фиксирующий его в жесткой форме программной архитектуры. DAL образует контролируемый «слой» между бизнес-приложениями и средствами хранения (базами данных).

Структура DAL представляет собой набор слабо связанных программных модулей для организации доступа к данным разных приложений. Слабая связанность модулей позволяет избежать проблем в синхронизации процессов разработки и обновления разных приложений и программного обеспечения DAL. Сами модули доступа могут иметь специфические реализации для различных базовых технологий, на которых разработаны бизнес-приложения.

При реализации слоя доступа к данным была выбрана технология Entity Framework Сore. Entity Framework Core позволяет разработчикам создавать приложения для доступа к данным, работающие с концептуальной моделью приложения, а не напрямую с реляционной схемой хранения. Цель состоит в уменьшении объема кода и снижении затрат на сопровождение приложений, ориентированных на обработку данных.  Благодаря ее возможностям был решен ряд проблем, связанных с реализацией слоя доступа к данным.

В данном дипломном проекте использовался подход Code First. Его суть заключается в том, что сначала пишется код модели на С#, а затем по нему генерируется база данных. Классы моделей, по которым генерируются таблицы в базе данных находиться в сборке Sport.Models.

Основная логика работы с базой данных реализована в классе SportContext, который располагается в сборке Sport.DAL. Данный класс предоставляет надо методов предназначенный для доступа к данным хранящимся в базе данных. Entity Framework предоставляет возможность работы с разными инфраструктурами доступа к данным. SportContext является наследником стандартного класса Entity Framework под названием DbContext. Entity Framework реализует паттерн репозиторий (Repository) и единица работы (Unit of Work), который позволяет разделить ответственность за постоянство бизнес-модели приложения. Благодаря ему мы изолируем логику непосредственной работы с базой данных от остального приложения. Это позволяет изменять SportContext под наши требования, без ущерба для остальной логики приложения.

Также на этом уровне реализована функциональность, предназначенный для миграции данных. Этот модуль отвечает за сохранение необходимых данных при переносе приложения на новую базу данных. Логика его работы заключаться в том, что при переходе на новую базу данных он заполняет ее прописанными в нем данными, необходимыми для полного функционирования системы.

## Слой сервисов

Когда мы анализируем домен и пытаемся определить основные объекты, составляющие модель, мы обнаруживаем, что некоторые аспекты домена не легко сопоставляются объектам. Объекты обычно обладают атрибутами, внутренним состоянием, которое управляется объектом, и демонстрируют поведение. Когда мы развиваем вездесущий язык, ключевые понятия домена вводятся в язык, а существительные языка легко сопоставляются объектам. Глаголы языка, связанные с их соответствующими существительными, становятся частью поведения этих объектов.

Но есть некоторые действия в домене, некоторые глаголы, которые, кажется, не принадлежат ни одному объекту. Они представляют собой особое поведение домена, поэтому их нельзя игнорировать или просто включать в некоторые объекты. Добавление такого поведения к объекту испортит объект, сделав его пригодным для функций, которые ему не принадлежат. Тем не менее, используя объектно-ориентированный язык, для этой цели мы должны использовать объект. У нас не может быть отдельной функции. Он должен быть привязан к какому-либо объекту. Часто такое поведение действует в нескольких объектах, возможно, разных классов. Например, для перевода денег с одного счета на другой; Должна ли эта функция находиться в учетной записи отправителя или получающей учетной записи? Когда такое поведение признается в домене, наилучшей практикой является объявление его как сервиса. Такой объект не имеет внутреннего состояния, а его цель - просто обеспечить функциональность для домена. Помощь, предоставляемая сервисом, может быть значительной, и служба может группировать связанные функции, которые обслуживают сущности. Гораздо лучше объявить службу в явном виде, потому что она создает четкое различие в домене, она инкапсулирует концепцию. Сервисы выступают в качестве интерфейсов, обеспечивающих операции. Сервисы являются общими в технических средах, но их можно использовать и на уровне домена. Сервис не относится к объекту, выполняющему эту службу. Таким образом, служба обычно становится точкой соединения для многих объектов. Это одна из причин, по которой поведение, естественно принадлежащее сервису, не должно включаться в объекты домена. Если такая функциональность включена в объекты домена, между ними создается плотная сеть ассоциаций и объектов, которые являются бенефициарами операций. Высокая степень сцепления между многими объектами является признаком плохой конструкции, потому что делает код трудным для чтения и понимания, и, что еще более важно, затрудняет изменение. Служба не должна заменять операцию, которая обычно относится к объектам домена.

В дипломном проекте логика сервисов изолирована от остальной логики программы. Это позволяет гарантировать, что изменения на уровне сервисов не повлечет изменения логики других уровней. Классы, отвечающие за данную логику, располагаются в сборке Sport.Services.

На данном этапе также необходимо задумать о данных, которые будут возвращать сервисы. Передача сущностей, полученных напрямую из базы данных не является лучшим решением, ввиду того что данные сущности часто содержать много лишней информации, и как правиле совершенно ненужной для клиента. Во избежание ненужных затрат на передачу ненужных данным в проекте были созданы модели представления (ViewModels). Это позволило уменьшить количество передаваемых данных за счет хранения в сущностях только данных необходимых для вызова информации. Так же такой подход упрощает сериализацию данным для удобной передачи по сети. Сущности такого типа хранят в себе только простые типы данных, а вместо ссылки на другие объекты только их идентификаторы. Как правило, на одну сущность уровня данных создаются две такие сущности. Одна с минимальным необходимым набором данных предназначенная для отображения сущности в списках. Вторая более развернутая для работы со связями между сущностей. Все сущности данного типа описаны в сборке Sport.ViewModels.

# СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА

В данном разделе будет поэтапно расписано создание web-приложения для дипломного проекта. Проект предназначен для людей, занимающихся спортом.

Данный проект должен отвечать следующим функциональным требованиям:

* необходима возможность просмотра всех упражнений, а также поиск по заданным параметрам;
* необходима возможность создания и редактирования программ тренировок;
* необходима возможность добавления программы тренировок в календарь;
* необходима возможность управления календарем, отображение и возможность изменения дней тренировок.

В начале создания проекта необходимо определить сущности, с которыми он будет работать. Данные сущности будут представлять основные объекты, участвующие в работе проекта. Можно выделить два типа сущностей это сущности-объекты и сущности-перечисления. Первые описывают устройство объекта, являясь своего рода чертежом для реального объекта предметной области. Вторые представляют собой перечисления разного характера будь то статусы или типы того или иного объекта.

Основные сущности, используемые в проекте:

* Базовая сущность (BaseEntity). Является базовым классом для всех сущностей. Сущность по определению должна содержать идентификатор. Поэтому в этом классе есть поле, в котором хранится идентификатор;
* Базовая сущность с именем (BaseNamedEntity). Наследник BaseEntity. Содержит одно новое поле, где хранится имя сущности.
* Упражнение (Exercise), основная сущность, с которой будет происходить работа. Непосредственно связана с объектом, для которого и составляется программа тренировок;
* Программа тренировок (TrainingPlan), сущность для представления программ тренировок;
* Тренировочный день (TrainingDay). Представляет собой тренировочный день. Программа тренировок хранит список тренировочных дней. Тренировочный день включает себя упражнения, который в этот день необходимо выполнить.
* Элемент тренировочного дня (TrainingDayItem), сущность для представления составной части тренировочного дня. В этой сущности указывается выполняемое упражнение, количество подходов и повторений, рабочие веса в каждом повторении.
* Программа питания (DietPlan), сущность для представления программ питания;
* Прием пищи (Meal). Программа питания ссылается на эту сущность. В программе питания хранится список приемов пищи. В приеме пищи указывается блюдо, количество калорий, белков, жиров и углеводов в нем;

В итоге выработки требований были созданы классы сущностей (см. рис. 4.1).

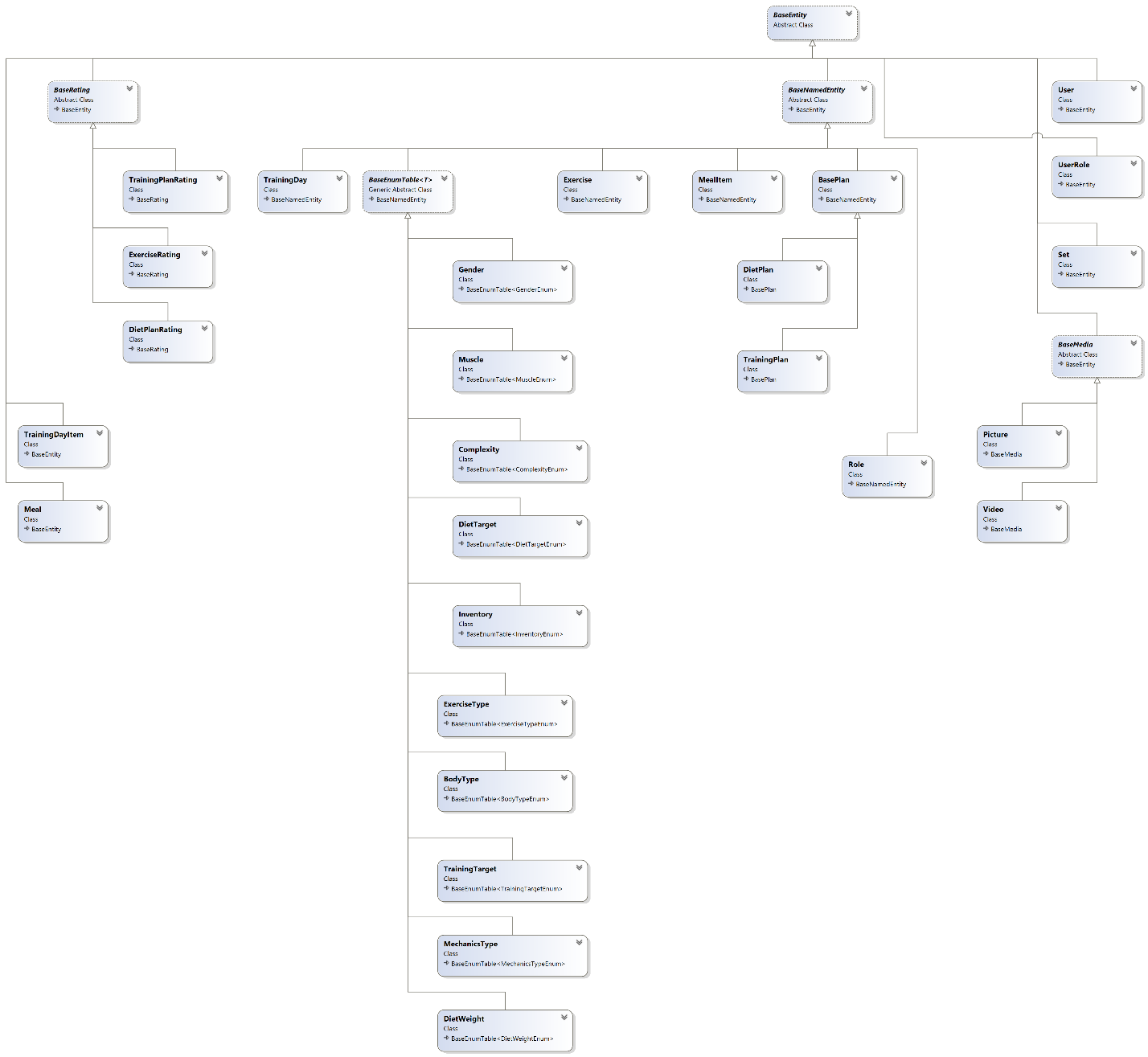


Рисунок 4.1 – Диаграмма классов сущностей

Для выработанных сущностей необходимо создать классы, которые будут их, представляют и создать таблицы в БД для их хранения. Работы на данном этапе проходят в двух сборках:

* Sport.Models, в нем содержатся классы, описывающие сущности;
* Sport.DAL, в нем реализуется логика работы с БД.

В дипломном проекте используются подход программирования Code First, предоставленным технологи Entity Framework, данный подход позволяет по созданным классам генерировать модели для хранения в БД. Данный подход сильно облегчает создание новых сущностей и изменения старых. Все что для этого необходимо – создать классы, описывающие сущно-сти, и зарегистрировать в SportContext. Все новые классы, описывающие сущности, должны наследоваться от класса BaseEntity, для которого создана логика предназначенная для работы с БД, в частности он содержит поле long Id, необходимое для идентификации сущности.

Код класса-сущности описывающего Упражнение:

namespace Sport.Models

{

public class Exercise : BaseNamedEntity

{

public string Description { get; set; }

public long? MechanicsTypeId { get; set; }

public MechanicsType MechanicsType { get; private set; }

public long? ExerciseTypeId { get; set; }

public ExerciseType ExerciseType { get; set; }

public long? ComplexityId { get; set; }

public Complexity Complexity { get; private set; }

public long? InventoryId { get; set; }

public Inventory Inventory { get; private set; }

public long? MuscleTargetId { get; set; }

public Muscle MuscleTarget { get; private set; }

public Video Video { get; set; }

public Picture MainMusclePicture { get; set; }

public ICollection<ExerciseRating> ExerciseRatings { get; set; }

public ICollection<ExercisePicture> ExercisePictures { get; set; }

public Exercise()

{

}

public Exercise(string name, string description, Complexity complexity, MechanicsType mechanicsType, ExerciseType exerciseType, Video video) : base(name)

{

Description = description;

Complexity = complexity;

MechanicsType = mechanicsType;

ExerciseType = exerciseType;

Video = video;

}

}

}

Код класса-сущности описывающего Программу тренировок:

namespace Sport.Models.Plan

{

public class TrainingPlan : BasePlan

{

public string Description { get; set; }

public int DurationMinutes { get; set; }

public TrainingTarget TrainingTarget { get; set; }

public BodyType BodyType { get; set; }

public ICollection<TrainingDay> TrainingDays { get; set; }

public ICollection<TrainingPlanRating> TrainingPlanRatings { get; set; }

}

}

Регистрация классов в SportContext происходит при его инициализации. После этого Entity Framework сам создаст необходимые таблицы и связи в базе данных.

public DbSet<Exercise> Exercises { get; set; }

public DbSet<ExerciseRating> ExerciseRatings { get; set; }

public DbSet<ExercisePicture> ExercisePictures { get; set; }

public DbSet<MechanicsType> MechanicsTypes { get; set; }

public DbSet<ExerciseType> ExerciseTypes { get; set; }

public DbSet<Muscle> Muscles { get; set; }

public DbSet<Inventory> Inventories { get; set; }

public DbSet<User> Users { get; set; }

public DbSet<Role> Roles { get; set; }

public DbSet<UserRole> UserRoles { get; set; }

public DbSet<Complexity> Complexities { get; set; }

Следующий этап разработки нового модуля заключается в создании методов для управления данными классов, которые были созданы на уровне доступа к данным. Данные методы предназначены для создания, удаления и получения данных из БД.

Реализации данной системы находиться в сборке Sport.Services, она содержит как интерфейсы, которые расположены в каталоге Interfaces, так и классы реализации, расположеных в Services.

Необходимые базовые методы, которые будут присутствовать у каждого сервиса, описываются в интерфейсе IBaseService.

namespace Sport.Services.Interfaces

{

public interface IBaseService<TEntity, TFilterModel>

where TEntity: BaseEntity

where TFilterModel: BaseFilterModel

{

IEnumerable<TEntity> Get(TFilterModel model = null);

TEntity Get(long id);

void Add(TEntity entity);

void Remove(TEntity entity);

void Remove(long id);

void Update(TEntity entity);

}

}

А реализации базовых методов определены в классе BaseService который является реализацией интерфейса IBaseService.

namespace Sport.Services.Services

{

public abstract class BaseService<TEntity, TFilterModel> : IBaseService<TEntity, TFilterModel>

where TEntity : BaseEntity, new()

where TFilterModel: BaseFilterModel

{

protected readonly SportContext \_sportContext;

protected readonly DbSet<TEntity> \_mainSet;

protected BaseService(SportContext sportContext)

{

\_sportContext = sportContext;

\_mainSet = sportContext.Set<TEntity>();

}

public virtual IEnumerable<TEntity> Get(TFilterModel model)

{

var set = \_mainSet.AsNoTracking();

if (model.GetAll == false)

{

set = set

.Skip(model.PageIndex \* model.PageSize.GetValueOrDefault())

.Take(model.PageSize.GetValueOrDefault());

}

return set.ToList();

}

public virtual TEntity Get(long id)

{

return \_mainSet.AsNoTracking().FirstOrDefault(e => e.Id == id);

}

public virtual void Add(TEntity entity)

{

\_sportContext.Add(entity);

}

public virtual void Remove(TEntity entity)

{

\_sportContext.Remove(entity);

}

public virtual void Remove(long id)

{

var removeEntity = new TEntity() { Id = id };

\_sportContext.Entry(removeEntity).State = EntityState.Deleted;

}

public virtual void Update(TEntity entity)

{

\_sportContext.Update(entity);

}

}

}

В дальнейшем все сервисы и их реализации наследуются от классов IBaseService и BaseService соответственно. При необходимости базовую реализацию методов можно переопределить.

Новый этап разработки происходит на серверном слое, его реализация распределена между двумя сборками:

* Sport.ViewModels, сборка, содержащая сущности, с помощью которых происходит обмен информации между клиентом и сервером;
* Sport.API, здесь находиться реализации API и серверной логики.

Первым делом необходимо создать сущности серверного уровня, данные сущности необходимы для обмена информацией между системами в частности между клиентом и сервером. Создание таких сущностей позволяет уменьшить объем данных, передаваемых между клиентом и сервером, а значит увеличить скорость работы портала.

Модель упражнения. В основном применяется для отображения краткой информации об упражнении в списке упражнений:

namespace Sport.ViewModels.Models.Exercise

{

public class ExerciseGetModel : IdBaseModel

{

public string Name { get; set; }

public MuscleModel MuscleTarget { get; set; }

public InventoryModel Inventory { get; set; }

public PictureModel Picture { get; set; }

}

}

Детализированная модель упражнения. Содержит в себе больше информации:

namespace Sport.ViewModels.Exercise

{

public class ExerciseGetByIdModel : IdBaseModel

{

public string Name { get; set; }

public string Description { get; set; }

public double Rating { get; set; }

public InventoryModel Inventory { get; set; }

public MechanicsTypeModel MechanicsType { get; set; }

public ExerciseTypeModel ExerciseType { get; set; }

public ComplexityModel Complexity { get; set; }

public MuscleModel MuscleTarget { get; set; }

public VideoModel Video { get; set; }

public PictureModel MainMusclePicture { get; set; }

public List<PictureModel> Pictures { get; set; }

}

}

Посредником между сущностями уровня данных и сущностями серверного уровня выступает AutoMapper, его задача состоит в автоматическом преобразовании объектов в другие. Благодаря использованию LINQ в своей основе AutoMapper является очень удобным и простым инструментом. Конфигурация маппинга объектов задается в MappingProfile который располагается в сборке Sport.API, также в нем можно задавать логику, по которой происходит преобразование объекта, в частности указывается какие поля к чему относятся и обрабатываются проверки на пустые поля. Стоит отметить, что AutoMapper позволяет преобразовывать как один объект, так и набор объектов. Благодаря этому еще больше упрощается работа по маппингу объектов.

CreateMap<ExercisePostModel, Exercise>()

.IgnoreAllPropertiesWithAnInaccessibleSetter()

.ForMember(dest => dest.ComplexityId, opts => opts.MapFrom(src => (long)src.Complexity));

CreateMap<ExerciseModel, Exercise>().ReverseMap();

CreateMap<Exercise, ExerciseGetModel>()

.ForMember(dest => dest.Picture, opts => opts.MapFrom(src => src.ExercisePictures.FirstOrDefault().Picture));

CreateMap<Exercise, ExerciseGetByIdModel>()

.ForMember(dest => dest.Pictures, opts => opts.MapFrom(src => src.ExercisePictures))

.ForMember(dest => dest.Rating, opts => opts.MapFrom(src => src.ExerciseRatings.Average(er => er.Value)));

CreateMap<ExercisePicture, PictureModel>()

.ForMember(dest => dest.Url, opts => opts.MapFrom(src => src.Picture.Url))

.ForMember(dest => dest.AltText, opts => opts.MapFrom(src => src.Picture.AltText));

Итак, наша разработка плавно переходит к месту, где встречаются все описанные изменения, а именно API. Под API подразумевается набор методов, через которые происходит общение клиент-сервер. Данный момент очень важен в разработке всех онлайн систем, ведь без грамотного API общение сервером может быть нет только затруднено, а иногда вообще невозможно.

В дипломном проекте общение между клиентом и сервером происходит по протоколу HTTP.

Все контроллеры наследуются от класса BaseController, в котором определены базовые методы. Эти методы могут быть переопределены в производных классах.

Код контроллера UsersController:

namespace Sport.API.Controllers

{

[Route("api/[controller]")]

public class UsersController : BaseController<User, UserFilterModel, UserModel, UserModel, UserPostModel, UserModel, IUserService, UsersController>

{

private readonly IJwtAuthService \_jwtAuthService;

public UsersController(IUserService userService, IJwtAuthService jwtAuthService, SportContext sportContext, IMapper mapper,

IStringLocalizer<UsersController> localizer, IOptions<UserOptions> userOptions) : base(userService, sportContext, mapper, localizer, userOptions)

{

\_jwtAuthService = jwtAuthService;

}

[AllowOnlyAnonymous]

[HttpPost("register")]

public IActionResult Register([FromBody] UserRegisterModel userRegister)

{

if (userRegister == null)

{

return BadRequest();

}

else if (ModelState.IsValid == false)

{

return BadRequest(ModelState);

}

else

{

var user = \_mainService.GetUser(userRegister);

if(user != null)

{

if(user.Login == userRegister.Login)

{

ModelState.AddModelError(nameof(user.Login), \_localizer["LoginAlreadyExists"]);

}

if(user.Email == userRegister.Email)

{

ModelState.AddModelError(nameof(user.Login), \_localizer["EmailAlreadyExists"]);

}

return BadRequest(ModelState);

}

var entity = \_mapper.Map<UserRegisterModel, User>(userRegister);

\_mainService.Add(entity);

\_sportContext.SaveChanges();

return Ok();

}

}

[AllowOnlyAnonymous]

[HttpPost("login")]

public IActionResult Login([FromBody] UserLoginModel user)

{

if (user == null)

{

return BadRequest();

}

else if (ModelState.IsValid == false)

{

return BadRequest(ModelState);

}

var foundUser = \_mainService.GetUser(user);

if (foundUser == null)

{

ModelState.AddModelError("", \_localizer["WrongCredentials"]);

return BadRequest(ModelState);

}

return Ok(\_jwtAuthService.GetSerializedToken(foundUser));

}

}

}

Завершающим этапом разработки нового модулю является создание UI части на клиенте и контролера для его работы. Данная задача сопоставима по сложности со всеми выше перечисленными, и занимает около половины всего времени необходимого для создания дипломного проекта.

Клиентская сторона работает на фреймворке Angular 2. Основным достоинством данного фреймворка является возможность делать одностраничные веб приложения. Парадигмой такого подхода является возможность использовать всю функциональность обычного сайта, но без перехода между страницами. То есть, сайт загружается один раз, а контент подгружается частями при необходимости. Это позволяет сильно укорить работу веб приложения.

Код компонента всех упражений:

export class ExercisesComponent implements OnInit {

errorMessage: string;

isShowMoreButton: boolean = true;

isFiltering: boolean = false;

pageIndex: number = 0;

exercises: ExerciseFilter[];

muscles: MuscleFilter[];

mechanicsTypes: MechanicsTypeFilter[];

exerciseTypes: ExerciseTypeFilter[];

inventories: InventoryFilter[];

complexities: ComplexityFilter[];

mode = 'Observable';

constructor(private exerciseService: ExerciseService,

private muscleService: MuscleService,

private mechanicsTypeService: MechanicsTypeService,

private exerciseTypeService: ExerciseTypeService,

private inventoryService: InventoryService,

private complexityService: ComplexityService) { }

ngOnInit() {

this.getExercises();

}

public getMore() {

this.pageIndex++;

this.getExercises(this.pageIndex);

}

public filterExercises() {

this.isFiltering = true;

let musclesChecked = this.muscles

.filter(muscle => muscle.checked === true)

.map(muscle => muscle.id);

this.exerciseService.getExercises(0, musclesChecked)

.subscribe(

(exercises) => {

this.exercises = exercises;

this.isFiltering = false;

},

error => this.errorMessage = <any>error);

}

public getExercises(pageIndex: number = 0) {

this.isShowMoreButton = false;

let musclesChecked = this.muscles != null

?

this.muscles

.filter(muscle => muscle.checked === true)

.map(muscle => muscle.id)

: null;

this.exerciseService.getExercises(pageIndex, musclesChecked)

.subscribe(

(exercises) => {

if (!this.exercises) {

this.exercises = exercises;

this.getMuscleFilters();

this.getMechanicsTypes();

this.getExerciseTypes();

this.getInventories();

this.getComplexities();

}

else {

this.exercises = this.exercises.concat(exercises);

}

this.isShowMoreButton = true;

},

error => this.errorMessage = <any>error);

}

public create(exercise: ExerciseFilter) {

this.exerciseService.create(exercise)

.subscribe(

exercise => this.exercises.push(exercise),

error => this.errorMessage = <any>error);

}

public delete(id: number) {

this.exerciseService.delete(id)

.subscribe(

exercise => this.exercises = this.exercises.filter(function (obj) {

return obj.id !== id;

}),

error => this.errorMessage = <any>error);

}

public update(exercise: ExerciseFilter) {

this.exerciseService.update(exercise)

.subscribe(

exercise => this.getExercises(),

error => this.errorMessage = <any>error);

}

}

В результате, когда пользователь откроет страницу с фильтрацией упражнений, отдельным упражнением или страницу с фильтрацией планов спортивного питания, он увидит то, что изображено на рис. 4.2, 4.3 и 4.4 соответственно.

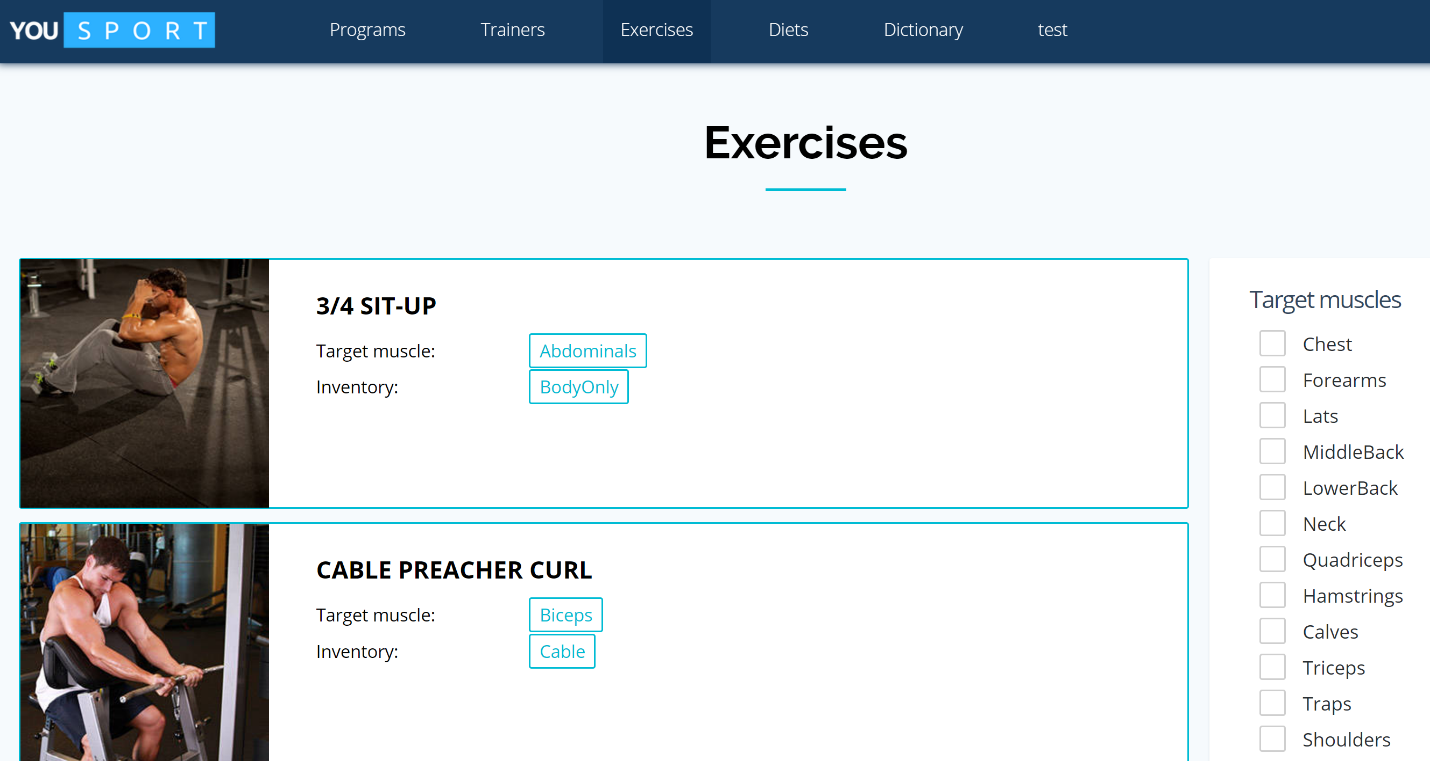


Рисунок 4.2 – Страница с упражнениями

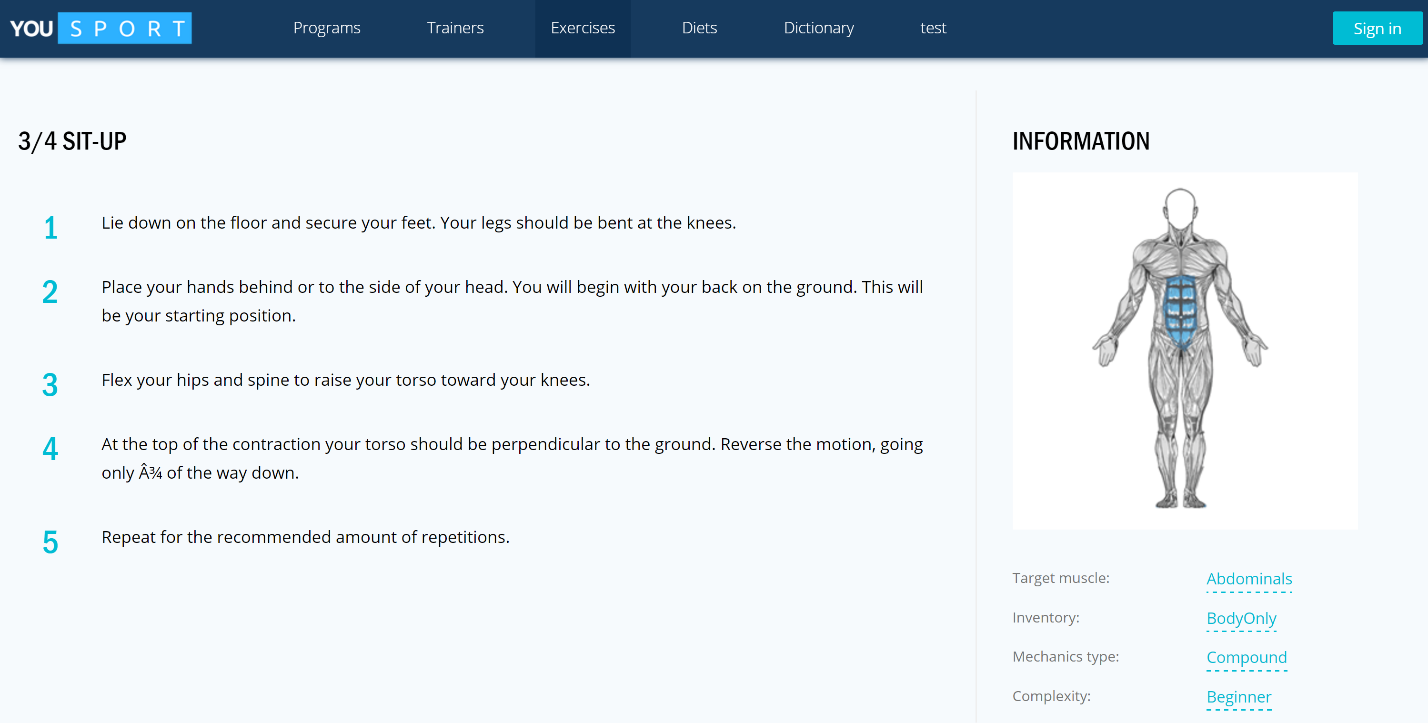


Рисунок 4.3 – Страница упражнения

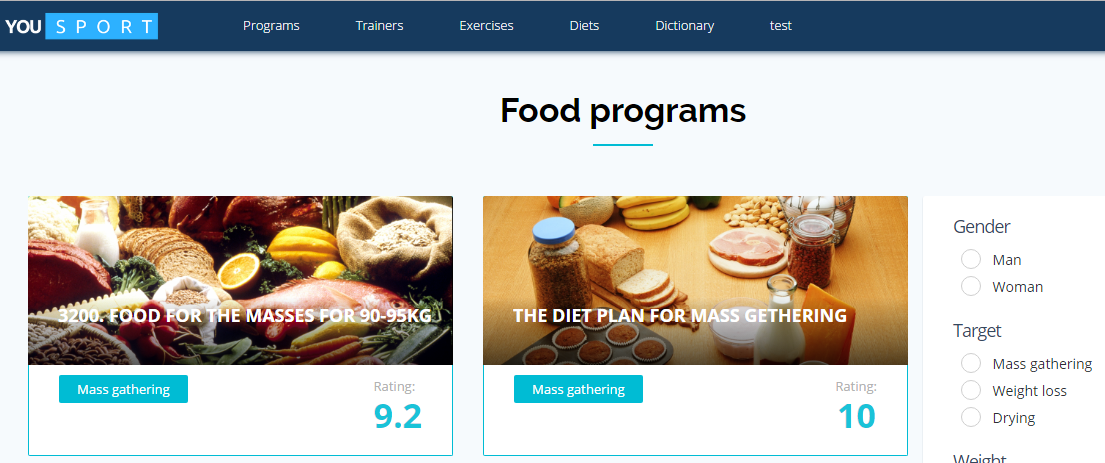


Рисунок 4.4 – Страница с программами питания

На этом и завершается работа по созданию портала для людей, занимающихся спортом. Хочется отметить, что приложение является сайтом с простым и интуитивно понятным интерфейсом. При разработке использовались только современные технологии.

Также следует отметить, что не стоят на месте и технологии, которые используются в проекте. Angular сейчас активно развивается. В скором будущем возможно появление новых версий этого фреймворка. И не исключено, что вслед за обновлением Angular, данный проект будет также переведен на новую версию этого фреймворка.

# ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## Модульное тестирование

Модульное тестирование (юнит-тестирование, unit testing) – это процесс в программировании, который позволяет проверить на корректность конкретные модули исходного кода. Суть заключается в том, чтобы писать тесты для каждого нетривиального метода или функции, что позволит очень быстро проверить, не привело ли какое-либо изменение исходного кода к появлению ошибок в уже оттестированных частях программы. Так же тестирование облегчает обнаружение у устранение таких ошибок. Цель юнит-тестирования – это изоляция отдельных частей программы и демонстрация того, что по отдельности эти части работоспособны. Данный тип тестирования в основном выполняется программистами.

Для модульного тестирования веб-приложения использовался фреймворк юнит-тестов XUnit.

Код тестов:

namespace Sport.Services.UnitTests

{

public abstract class BaseServiceUnitTest<TEntity, TFilterModel, TIService>

where TEntity : BaseEntity, new()

where TFilterModel : BaseFilterModel

where TIService : class, IBaseService<TEntity, TFilterModel>

{

[Fact]

public virtual void GetAll\_Correct()

{

var ex = Assert.Throws<InvalidOperationException>(() => Service.Get());

Assert.Equal(ex.Message, \_noTrackingError);

}

[Fact]

public virtual void Get\_Correct()

{

var ex = Assert.Throws<InvalidOperationException>(() => Service.Get(1));

Assert.Equal(ex.Message, \_noTrackingError);

}

[Fact]

public virtual void Add\_Correct()

{

Service.Add(Model);

MockContext.Verify(r => r.Add(Model), Times.Once);

}

[Fact]

public virtual void Update\_Correct()

{

Service.Update(Model);

MockContext.Verify(r => r.Update(Model), Times.Once);

}

[Fact]

public virtual void Delete\_Correct()

{

Service.Remove(Model);

MockContext.Verify(r => r.Remove(Model), Times.Once);

}

}

}

## Интеграционное тестирование

Интеграцио́нное тести́рование (англ. Integration testing, иногда называется англ. Integration and Testing, аббревиатура англ. I&T) — одна из фаз тестирования программного обеспечения, при которой отдельные программные модули объединяются и тестируются в группе. Обычно интеграционное тестирование проводится после модульного тестирования и предшествует системному тестированию.

Интеграционное тестирование в качестве входных данных использует модули, над которыми было проведено модульное тестирование, группирует их в более крупные множества, выполняет тесты, определённые в плане тестирования для этих множеств, и представляет их в качестве выходных данных и входных для последующего системного тестирования.

Целью интеграционного тестирования является проверка соответствия проектируемых единиц функциональным, приёмным и требованиям надежности. Тестирование этих проектируемых единиц — объединения, множества или группы модулей — выполняется через их интерфейс, с использованием тестирования «чёрного ящика».

Код тестов:

namespace Sport.Services.IntegrationTests

{

public abstract class BaseIntegrationTest<TEntity, TFilterModel, TGetModel, TGetByIdModel, TPostModel, TPutModel>

where TEntity: BaseEntity, new()

where TFilterModel: BaseFilterModel

where TGetModel : IdBaseModel

where TGetByIdModel : IdBaseModel

where TPostModel: BaseModel, new()

where TPutModel: IdBaseModel, new()

{

protected readonly ITestOutputHelper \_testOutputHelper;

protected readonly RestHelper<TEntity, TFilterModel, TGetModel, TGetByIdModel, TPostModel, TPutModel> \_restHelper;

protected readonly IMapper \_mapper;

protected BaseIntegrationTest(ITestOutputHelper testOutputHelper, HttpFixture httpFixture, string url)

{

\_testOutputHelper = testOutputHelper;

\_restHelper = new RestHelper<TEntity, TFilterModel, TGetModel, TGetByIdModel, TPostModel, TPutModel>(httpFixture, url);

var \_mapperConfiguration = new MapperConfiguration(cfg => { cfg.CreateMissingTypeMaps = true; });

\_mapper = \_mapperConfiguration.CreateMapper();

}

[Fact]

public virtual async Task<HttpResponseMessage> GetById\_Wrong()

{

var response = await \_restHelper.GetAsync(-1);

Assert.Equal(response.StatusCode, HttpStatusCode.NotFound);

return response;

}

protected async Task<HttpResponseMessage> GetById\_Correct(HttpResponseMessage responseAfterCreate)

{

var response = await \_restHelper.SendAsync(responseAfterCreate.Headers.Location.AbsoluteUri);

Assert.Equal(response.StatusCode, HttpStatusCode.OK);

var modelById = await ConvertHelper.ToModel<TEntity>(response);

Assert.NotNull(modelById);

return response;

}

[Fact]

public virtual async Task<HttpResponseMessage> Create\_Wrong()

{

var response = await \_restHelper.PostAsync(null);

Assert.Equal(response.StatusCode, HttpStatusCode.BadRequest);

return response;

}

[Fact]

public virtual async Task<HttpResponseMessage> Create\_Correct()

{

var model = new TPostModel();

var response = await \_restHelper.PostAsync(model);

Assert.NotEqual(response.StatusCode, HttpStatusCode.UnsupportedMediaType);

Assert.Equal(response.StatusCode, HttpStatusCode.Created);

return response;

}

[Fact]

public virtual async Task<HttpResponseMessage> Update\_Wrong()

{

var response = await \_restHelper.PutAsync(null);

Assert.Equal(response.StatusCode, HttpStatusCode.BadRequest);

var wrongModel = new TPutModel() { Id = -1 };

response = await \_restHelper.PutAsync(wrongModel);

Assert.Equal(response.StatusCode, HttpStatusCode.NotFound);

return response;

}

protected abstract Task<HttpResponseMessage> Update\_Correct(HttpResponseMessage responseAfterGetById);

[Fact]

public virtual async Task<HttpResponseMessage> Delete\_Wrong()

{

var response = await \_restHelper.DeleteAsync(-1);

Assert.Equal(response.StatusCode, HttpStatusCode.NotFound);

return response;

}

protected async Task<HttpResponseMessage> Delete\_Correct(long id)

{

var response = await \_restHelper.DeleteAsync(id);

Assert.Equal(response.StatusCode, HttpStatusCode.NoContent);

var modelAfterDelete = await ConvertHelper.ToModel<TEntity>(await \_restHelper.GetAsync(id));

Assert.True(modelAfterDelete == null, "Entity was not deleted from database");

return response;

}

[Fact]

public async Task<HttpResponseMessage> CRUD\_Correct()

{

var response = await Create\_Correct();

response = await GetById\_Correct(response);

var exerciseById = await ConvertHelper.ToModel<TEntity>(response);

response = await Update\_Correct(response);

response = await Delete\_Correct(exerciseById.Id);

return response;

}

}

}

# ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО СРЕДСТВА

Программное средство, разрабатываемое в рамках рассматриваемого дипломного проекта, предназначено для людей, занимающихся спортом. Основная цель заключается в упрощении организации тренировок как профессионалов, так и любителей. Проект будет предоставлять возможность выбора готовой программы тренировок руководствуясь ее рейтингом и отзывами пользователей, что позволит новичку выбрать наиболее подходящую для себя программу, либо составить собственную программу тренировок, если пользователь является профессионалом.

Главной особенностью проекта является минимизация действий пользователя. Основную работу пользователю нужно будет сделать в самом начале, при планировании своей программы тренировок. В дальнейшем программное средство будет само информировать его о начале очередного занятия. В дальнейшем пользователь сможет увидеть, как менялись его результаты от тренировки к тренировке. Для этого ему нужно будет в конце занятия менять показатели в выполненных упражнениях.

Написанное в ходе данного дипломного проекта программное средство предназначено для пользователей, увлеченных спортом. Воспользоваться программным средством смогут любые пользователи, в любой точке мира.

## Расчет затрат и отпускной цены программного средства

Объем ПО (строки исходного кода, LOC)



Таблица 6.1 – Перечень и объем функций программного модуля

|  |  |  |  |
| --- | --- | --- | --- |
| № функции | Наименование (содержание)  функции | Объем функции (LОС) | |
| По каталогу Vi | Уточненный  Vyi |
| 109 | Организация ввода/вывода информации в интерактивном режиме | 320 | 320 |
| 201 | Генерация структуры базы данных | 4300 | 4300 |
| 204 | Обработка наборов и записей баз данных | 2670 | 2670 |
| 206 | Обслуживание базы данных в интерактивном режиме | 6950 | 6950 |

Продолжение таблицы 6.1

|  |  |  |  |
| --- | --- | --- | --- |
| № функции | Наименование (содержание) функции | Объем функции (LОС) | |
| По каталогу Vi | Уточненный  Vyi |
| 207 | Манипулирование данными | 9550 | 9550 |
| 208 | Организация поиска и поиск в базе данных | 5480 | 3836 |
| 210 | Загрузка базы данных | 2780 | 2780 |
| 506 | Обработка ошибочных и сбойных ситуаций | 410 | 410 |
|  | Итого | 32460 | 30816 |

Среда разработки ПО (Visual Studio 2017, SQL Server Management Studio 2016), ПО функционального назначения. Vi = 30816 LOC.

Трудоемкость разработки ПО:

Категории сложности ПО – 3-я категория, Vi = 30816 LOC;  
Нормативная трудоемкость ПО (Tн) – 540 чел./дн.;

Коэффициент сложности (Kc) – 1,12;

Коэффициент, учитывающий степень использования при разработке

ПО стандартных модулей (Kт) – 0,6;

Коэффициент новизны разрабатываемого ПО (Kн) – 0,7;

Общая трудоемкость:

Tо = Tн⋅Kс⋅Kт⋅Kн = 540⋅1,12⋅0,6⋅0,7 = 254 чел./дн.;

Численность исполнителей проекта (Чp):

, где Tp− срок разработки проекта (лет);

Эффективный фонд времени работы одного работника (Фэф):

Фэф = Дг-Дп-Дв-До = 236 дн.;

Срок разработки установлен 4 мес. (Tp = 0,33 г.):

.

Основная заработная плата исполнителей на конкретное ПО рассчитывается по формуле:

Зоi = ∑ni=1 \* Зci \* Фрi \* К,

где n− количество исполнителей, занятых разработкой конкретного ПО;

Зci −среднедневная заработная плата i-го исполнителя (*д.е.*);

Фрi − плановый фонд рабочего времени i-го исполнителя (дн.);

К − коэффициент премирования (1,2).

Расчет основной заработной платы представлен в табл. 6.2.

Таблица 6.2 — Расчет основной заработной платы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категория исполнителя | Среднедневная заработная плата, руб. | Плановый фонд рабочего времени, дн. | Коэффициент премирования | Основная заработная плата, руб. |
| Разработчик программного средства на языке программирования C# | 45 | 97 | 1,2 | 5238 |
| Front-end разработчик | 40 | 97 | 1,2 | 4656 |
| Разработчик базы данных | 40 | 55 | 1,2 | 2640 |
| Итого с премией (20%), Зо | - | - | - | 12534 |

Дополнительная заработная плата исполнителей проекта определяется по формуле:



где НД – норматив дополнительной заработной платы (20%).

Дополнительная заработная плата составит:

Зд = 12534 ·20/100= 2506 руб.

Отчисления в фонд социальной защиты населения и на обязательное страхование (ЗС) определяются в соответствии с действующими законодательными актами по формуле:



где НСЗ – норматив отчислений в фонд социальной защиты населения и на обязательное страхование (34 +0,6%).

Зсз = (12534+2506) ·34,6 / 100= 5204.12 руб.

Расходы по статье «Машинное время» (РМ) включают оплату машинного времени, необходимого для разработки и отладки ПС, и определяются по формуле:

Рм = Цм \* Тч \* Ср,

где Цм – цена одного машино-часа;

Тч – количество часов работы в день;

Ср – длительность проекта.

Стоимость машино-часа на предприятии составляет 1,2 руб.. Разработка проекта займет 93 дня. Определим затраты по статье “Машинное время”:

Рм=1,2 ·8·93=892.8 руб.

Затраты по статье «Накладные расходы» (РН), связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных (экспериментальных) производств, а также с расходами на общехозяйственные нужды (РН), определяются по формуле



где НРН – норматив накладных расходов (50%).

Накладные расходы составят:

Рн= 12534 · 0.5=6267 руб.

Общая сумма расходов по всем статьям сметы (Сп) на ПО рассчитывается по формуле:



Сn =12534 + 2506 + 5204.12 + 892.8 + 6267 = 27404.72 руб.

Прибыль ПС рассчитывается по формуле:

,

где Ппс – прибыль от реализации ПС заказчику (руб.);

УР – уровень рентабельности ПС (25%);

СП – себестоимость ПС (руб.).

Ппс =27404.72 · 25/100=6851.18 руб.

Прогнозируемая отпускная цена ПС вычисляется по формуле:

.

Цп =27404.72 + 6851.18 = 34255.89 руб.

Налог на добавленную стоимость (НДСi):

НДС= ЦП \* НДС : 100,

где Hдc− норматив НДС (%).

НДС= 34255.89 \* 20% : 100% = 6851.18 руб.

Прогнозируемая отпускная цена (Цo):

Цo = Цп+ НДС

Цo = 34255.89 + 6851.18 = 41107.07 руб.

Кроме того, организация-разработчик осуществляет затраты на сопровождение ПС (РС), которые определяются по нормативу (НС), где Нс – норматив расходов на сопровождение и адаптацию (20%).



где Рс – расходы на сопровождение и адаптацию ПС в целом по организации (руб.);

СР – смета расходов в целом по организации без расходов на сопровождение и адаптацию (руб.).

Рс = 27404.72 · 20 / 100= 5480.94 руб.

## Оценка экономической эффективности применения программного средства у пользователя

Расчет экономии основных видов ресурсов в связи   
с использованием нового программного средства

Исходные данные для расчета экономии ресурсов в связи с применением нового программного средства

Таблица 6.3 – Исходные данные для расчета экономии ресурсов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Наименование | Обозначение | Единицы | Значение  показателя | | Наименование |
| в базовом варианте | в новом варианте |
| 1. Капитальные вложения, включая затраты пользователя на приобретение ПО | Kпp | д.е. |  | 85031,4 | Договор заказчика с разработчиком |
| 2. Затраты на освоение ПО | Кос | д.е. |  | 5061,4 | Договор заказчика с разработчиком |
| 3. Затраты на сопровождение ПО | Кс | д.е. |  | 10122,8 | Договор заказчика с разработчиком |
| 4. Затраты на укомплектование ВТ техническими средствами в связи с внедрением нового ПО | Ктс | д.е. |  | 6100,0 | Сметы затрат на внедрение |
| 5. Затраты на пополнение оборотных средств в связи с эксплуатацией нового ПО | Коб | д.е. |  | 2000,0 | Сметы затрат на внедрение |
| 6. Время простоя сервиса, обусловленное ПО, в день | П1, П2 | мин | 50 | 10 | Расчетные данные пользователя и паспорт ПО |
| 7. Стоимость одного часа простоя | Сп | д.е. | 30,1 | 30,1 | Расчетные данные пользователя и паспорт ПО |

Продолжение таблицы 6.3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Наименование | Обозначение | Единицы | Значение показателя | | Наименование |
| в базовом варианте | в новом варианте |
| 8. Среднемесячная ЗП одного программиста | Зсм | д.е. | 450,0 | 450,0 | Расчетные данные пользователя теля |
| 9. Коэффициент начислений на зарплату | Kнз |  | 1,5 | 1,5 | Рассчитывается по данным пользователя |
| 10. Среднемесячное  количество рабочих дней | Д р | день |  | 21,5 | Принято для расчета |
| 11. Количество типовых задач, решаемых за год | Зт1, Зт2 | задача | 1800 | 1800 | План пользователя |
| 12. Объем выполняемых работ | А1, А2 | задача | 1800 | 1800 | План пользователя |
| 13. Средняя трудоемкость работ на задачу | Tc1, Тс2 | Человеко-часов | 6,0 | 0,87 | Рассчитывается по данным пользователя |
| 14. Количество часов работы ты в день | Tч | ч | 8 | 8 | Принято для расчета расчета |
| 15. Ставка налога на прибыль | Hп | % |  | 18 |  |

Экономия затрат на заработную платув расчете на 1 задачу (Cзe):

,

где Зcм− среднемесячная заработная плата одного программиста (руб.);

Tc1, Tc2− снижение трудоемкости работ в расчете на 1 задачу (человеко-часов);

Tч− количество часов работы в день (ч);

Дp− среднемесячное количество рабочих дней.

Сзе = 896\*(6-0.87)/(8\*21.5) = 26.72 руб

Экономия заработной платы при использовании нового ПО (руб.):

Сз=СзеА2,

где Сз− экономия заработной платы;

А2− количество типовых задач, решаемых за год (задач).

Сз=26.72 \*1800=48093.75 руб.

Экономия с учетом начисления на зарплату (Сн):

*Cн= Cз****ּ*** 1,5 *=* 48093.75 ***ּ*** 1,5 *=* 72140.63 руб.

Экономия за счет сокращения простоев сервиса (Сс) рассчитывается по формуле:

,

где Дрг – плановый фонд работы сервиса (дней).

Сс = (50-10)\* 21.5\*602 / 60 = 431.43 руб.

Общая готовая экономия текущих затрат, связанных с использованием  
нового ПО (Со), рассчитывается по формуле:

Co= Cн + Cc

Co= 72140.63 + 431.43 = 72572.05 руб.

Расчет экономического эффекта.

Внедрение нового ПО позволит пользователю сэкономить на текущих затратах, т.е. практически получить на эту сумму дополнительную прибыль. Для пользователя в качестве экономического эффекта выступает лишь чистая прибыль − дополнительная прибыль, остающаяся в его распоряжении (ΔПЧ), которая определяется по формуле:

,

где Hп− ставка налога на прибыль (%).

ΔПЧ = 72572.05 – (72572.05 \* 0.18) = 59509.09 руб.

В процессе использования нового ПО чистая прибыль в конечном итоге  
возмещает капитальные затраты. Однако полученные при этом суммы результатов (прибыли) и затрат (капитальных вложений) по годам приводят к единому времени − расчетному году (за расчетный год принят 2017-й год) путем умножения результатов и затрат за каждый год на коэффициент дисконтирования α.

В данном примере используются коэффициенты: 2017 г. – 1, 2018-й – 0,8696, 2019-й – 0,7561, 2020 г. – 0,6575. Все рассчитанные данные экономического эффекта сводятся в таблицу 6.4.

Таблица 6.4 – Расчет экономического эффекта от использования нового программного средства

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Показатели | Единицы  измерения | Годы | | | |
| 2017 | 2018 | 2019 | 2020 |
| *Результаты* |  |  |  |  |  |
| Прирост прибыли за счет экономии затрат (Пч) | руб. |  | 59509.09 | 59509.09 | 59509.09 |
| То же с учетом фактора времени | руб. |  | 51749.1 | 44994.82 | 39127.22 |
| *Затраты* |  |  |  |  |  |
| Приобретение ПО (Кпр) | руб. | 85031,4 |  |  |  |
| Сопровождение (Кс) | руб. | 10122,8 |  |  |  |
| Всего затрат | руб. | 95154.2 |  |  |  |
| То же с учетом фактора времени | руб. | 95154.2 |  |  |  |
| *Экономический эффект* |  |  |  |  |  |
| Превышение результата над затратами | руб. | -95154.2 | 51749.1 | 44994.82 | 39127.22 |
| То же с нарастающим итогом | руб. | -95154.2 | -43405.09 | 1589.72 | 40716.94 |
| Коэффициент приведения | единицы | 1 | 0,8696 | 0,7561 | 0,6575 |

## Заключение

Реализация проекта ПО позволит заказчику снизить трудоемкость решения задач, сократить простои сервиса и минимизировать возникновение ошибок из-за человеческого фактора.

Все затраты заказчика окупятся на 2 год эксплуатации ПО.

Проект представляется эффективным и полезным для заказчика.

# ЗАКЛЮЧЕНИЕ

В результате работы над дипломным проектом был разработан портал для людей, занимающихся спортом.

Разработанное программное средство предназначено облегчить обычным людям планирование всего процесса занятий спортом.

В качестве инструментальных средств разработки была использована среда Microsoft Visual Studio 2017 Community. Данная среда разработки является последней версией проверенного временем программного продукта, обладающая всеми необходимыми инструментами для создания высокоэффективных и надежных приложений. Благодаря широким возможностям этого средства разработки, в процессе создания программного средства было достигнуто резкое повышение производительности труда программиста.

Портал разработан на языке C# и Typescript с использование технологий Angular 2, ASP.NET Core Web Api, ASP.NET Core MVC, Entity Framework Core.

Портал предоставляет следующие возможности:

* составление программ тренировок;
* планирование графика тренировок;
* ведение дневника тренировок;
* составление программ питания;

В процессе технико-экономического обоснования проекта по внедрению разработанного программного средства были получены результаты, свидетельствующие о том, что данный проект является экономически целесообразным и уже на втором году внедрения данного программного средства предприятие – заказчик начинает получать прибыль.

Основными достоинствами разработанного программного средства являются:

* простота в освоение, ввиду использования большого количества популярных и удобных технологий;
* высокая скорость работы портала;
* легкая расширяемость на любых уровнях архитектуры для добавления дополнительной функциональности;
* большой набор упражнений и программ тренировок под разные нужды людей;
* постоянные обновления, нацеленные на улучшение программного средства в целом.

Основными недостатками являются:

* узкая направленность портала;
* большие затраты ресурсов на сервере.

Дальнейшее развитие связано с созданием мобильных приложений. Это потребует меньших усилий за счет того, что уже есть работающее API и создаваться будет только клиентское приложение.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] Angular [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/AngularJS.

[2] TypeScript [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/TypeScript.

[3] .NET Framework [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/.NET\_Framework.

[4] MSDN [Электронный ресурс]. – Электронные данные. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/zw4w595w(v=vs.110).aspx.

[5] CIL [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Common\_Intermediate\_Language.

[6] JIT [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/JIT.

[7] Албахари, Джозеф. C# 6.0. Справочник. Полное описание / Джозеф Албахари, Бен Албахари. – 6-е изд. – ООО “И.Д. Вильямс”, 2016. – 1040 стр.

[8] C Sharp [Электронный ресурс]. – Электронные данные. – ­Режим доступа: http://ru.wikipedia.org/wiki/C\_Sharp.

[9] Entity Framework Core [Электронный ресурс]. – Электронные данные. – ­Режим доступа: https://metanit.com/sharp/entityframeworkcore/1.1.php.

[10] ASP.NET Core [Электронный ресурс]. – Электронные данные. – ­Режим доступа: https://metanit.com/sharp/aspnet5/1.1.php.

[11] MVC [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Model-View-Controller

[12] Фримен, Адам. ASP.NET MVC 5 с примерами на C# 5.0 для профессионалов. Полное описание / Адам Фримен. – 5-е изд. – ООО “И.Д. Вильямс”, 2015. – 736 стр.

[13] Доменный объект [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Доменный\_объект

# ПРИЛОЖЕНИЕ А

**Исходный текст программы**

namespace Sport.API.Controllers

{

public abstract class BaseController<TEntity, TFilterModel, TGetModel, TGetByIdModel, TPostModel, TPutModel, TService, TController> : Controller

where TEntity: BaseEntity

where TFilterModel: BaseFilterModel

where TGetModel: IdBaseModel

where TGetByIdModel: IdBaseModel

where TPostModel: BaseModel

where TPutModel: IdBaseModel

where TService: IBaseService<TEntity, TFilterModel>

where TController: Controller

{

protected readonly TService \_mainService;

protected readonly SportContext \_sportContext;

protected readonly IMapper \_mapper;

protected readonly IStringLocalizer<TController> \_localizer;

private readonly UserOptions \_userOptions;

public BaseController(TService mainService, SportContext sportContext, IMapper mapper, IStringLocalizer<TController> localizer, IOptions<UserOptions> userOptions)

{

\_mainService = mainService;

\_sportContext = sportContext;

\_mapper = mapper;

\_localizer = localizer;

\_userOptions = userOptions.Value;

}

[HttpGet]

public virtual IActionResult Get(TFilterModel filterModel)

{

if (filterModel.PageSize == null)

{

filterModel.PageSize = \_userOptions.DefaultPageSize;

}

var entities = \_mainService.Get(filterModel);

var models = \_mapper.Map<IEnumerable<TEntity>, List<TGetModel>>(entities);

return Ok(models);

}

[HttpGet("{id}")]

public virtual IActionResult Get(long id)

{

var entity = \_mainService.Get(id);

if (entity == null)

{

return NotFound();

}

var model = \_mapper.Map<TEntity, TGetByIdModel>(entity);

return Ok(model);

}

[Authorize(Roles = "user")]

[HttpPost]

public virtual IActionResult Post([FromBody]TPostModel value)

{

if (value == null)

{

return BadRequest();

}

else if (ModelState.IsValid == false)

{

return BadRequest(ModelState);

}

else

{

var entity = \_mapper.Map<TPostModel, TEntity>(value);

\_mainService.Add(entity);

\_sportContext.SaveChanges();

var modelForReturn = \_mapper.Map<TEntity, TGetModel>(entity);

return CreatedAtAction("Get", new { id = entity.Id }, modelForReturn);

}

}

[Authorize(Roles = "user")]

[HttpPut("{id}")]

public virtual IActionResult Put(long id, [FromBody]TPutModel value)

{

if (value == null || id != value.Id)

{

return BadRequest();

}

if (\_mainService.Get(id) == null)

{

return NotFound();

}

var entity = \_mapper.Map<TPutModel, TEntity>(value);

\_mainService.Update(entity);

\_sportContext.SaveChanges();

return NoContent();

}

[Authorize(Roles = "user")]

[HttpDelete("{id}")]

public virtual IActionResult Delete(long id)

{

if (\_mainService.Get(id) == null)

{

return NotFound();

}

\_mainService.Remove(id);

\_sportContext.SaveChanges();

return NoContent();

}

}

}

namespace Sport.API.Attributes

{

public class AllowOnlyAnonymousAttribute : AllowAnonymousAttribute, IAuthorizationFilter

{

public AllowOnlyAnonymousAttribute()

{

}

public void OnAuthorization(AuthorizationFilterContext context)

{

if(context.HttpContext.User.Identity.IsAuthenticated)

{

context.ModelState.AddModelError("", Resources.Attributes.AllowOnlyAnonymousAttribute.NotAnonymousUser);

context.Result = new BadRequestObjectResult(context.ModelState);

}

}

}

}

namespace Sport.Services.Services

{

public class UserService : BaseService<User, UserFilterModel>, IUserService

{

private readonly IRoleService \_roleService;

public UserService(SportContext sportContext, IRoleService roleService) : base(sportContext)

{

\_roleService = roleService;

}

public override IEnumerable<User> Get(UserFilterModel model)

{

return \_mainSet

.AsNoTracking()

.Where(u =>

(model.Login == null || u.Login == model.Login)

&& (model.Password == null || u.Password == model.Password))

.Include(u => u.UserRoles)

.ThenInclude(ur => ur.Role)

.ToList();

}

public User GetUser(UserLoginModel model)

{

return \_mainSet.AsNoTracking()

.Where(u => u.Login == model.LoginOrEmail && u.Password == model.Password)

.Include(m => m.UserRoles)

.ThenInclude(ur => ur.Role)

.FirstOrDefault();

}

public User GetUser(UserRegisterModel model)

{

return \_mainSet.AsNoTracking()

.Where(u => u.Login == model.Login || u.Email == model.Email)

.FirstOrDefault();

}

public override void Add(User entity)

{

base.Add(entity);

\_sportContext.UserRoles.Add(new UserRole

{

RoleId = \_roleService.GetUserRole().Id,

User = entity

});

}

}

}

namespace Sport.IntegrationTests.Helpers

{

public class RestHelper<TEntity, TFilterModel, TGetModel, TGetByIdModel, TPostModel, TPutModel>

where TEntity: BaseEntity

where TFilterModel: BaseFilterModel

where TGetModel : IdBaseModel

where TGetByIdModel : IdBaseModel

where TPostModel: BaseModel

where TPutModel: IdBaseModel

{

private string \_url;

private HttpClient \_client;

/// <summary>

///

/// </summary>

/// <param name="url">Example: "api/values"</param>

public RestHelper(HttpFixture httpFixture, string url)

{

\_client = httpFixture.Client;

\_url = url;

}

#region Requests

public async Task<HttpResponseMessage> GetAsync(TFilterModel model = null)

{

return await \_client.GetAsync(GetUrlWithQueryParameters(model));

}

public async Task<HttpResponseMessage> GetAsync(long id)

{

return await \_client.GetAsync(GetUrlWithId(id));

}

public async Task<HttpResponseMessage> PostAsync(TPostModel model)

{

return await \_client.PostAsync(\_url, ConvertHelper.ToJson(model));

}

public async Task<HttpResponseMessage> PostAsync<T>(T model, string actionRouteName)

{

string postUrl = \_url + "/" + actionRouteName;

return await \_client.PostAsync(postUrl, ConvertHelper.ToJson(model));

}

public async Task<HttpResponseMessage> PostAnonymousAsync<T>(T model, string actionRouteName)

{

string postUrl = \_url + "/" + actionRouteName;

return await SendAnonymousRequest(postUrl, () => { return \_client.PostAsync(postUrl, ConvertHelper.ToJson(model)); });

}

public async Task<HttpResponseMessage> PutAsync(TPutModel model)

{

long id = model != null ? model.Id : -1;

return await \_client.PutAsync(GetUrlWithId(id), ConvertHelper.ToJson(model));

}

public async Task<HttpResponseMessage> DeleteAsync(long id)

{

return await \_client.DeleteAsync(GetUrlWithId(id));

}

public async Task<HttpResponseMessage> SendAsync(string url)

{

var request = new HttpRequestMessage()

{

RequestUri = new Uri(url)

};

return await \_client.SendAsync(request);

}

private async Task<HttpResponseMessage> SendAnonymousRequest(string url, Func<Task<HttpResponseMessage>> request)

{

var authorizationHeader = \_client.DefaultRequestHeaders.Authorization;

\_client.DefaultRequestHeaders.Authorization = null;

var response = await request.Invoke();

\_client.DefaultRequestHeaders.Authorization = authorizationHeader;

return response;

}

#endregion

public async Task<bool> HasError(HttpResponseMessage response, string errorMessage)

{

return await HasError(response, errorMessage, 1);

}

public async Task<bool> HasError(HttpResponseMessage response, string errorMessage, int expectedErrorsCount)

{

var errors = await ConvertHelper.ToModel<Dictionary<string, string[]>>(response);

int actualErrorsCount = 0;

if(errors != null)

{

foreach (var value in errors.Values)

{

if (value.Contains(errorMessage))

{

actualErrorsCount++;

}

}

}

return actualErrorsCount == expectedErrorsCount;

}

public async Task<int> GetErrorsCount(HttpResponseMessage response)

{

var errors = await ConvertHelper.ToModel<Dictionary<string, string[]>>(response);

int errorsCount = 0;

foreach (var value in errors.Values)

{

foreach (var error in value)

{

errorsCount++;

}

}

return errorsCount;

}

private string GetUrlWithId(long id)

{

return $"{\_url}/{id}";

}

private string GetUrlWithQueryParameters(TFilterModel model)

{

if(model != null)

{

string url = $"{\_url}?";

var properties = model.GetType().GetTypeInfo().GetProperties();

foreach (var property in properties)

{

url += $"{property.Name.ToLower()}={property.GetValue(model)}&";

}

return url.Substring(0, url.Length - 1); //remove last ampersand (&) in url string

}

return \_url;

}

}

}

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Обозначение* | | | | | *Наименование* | *Дополнительные сведения* | | | | |
|  | | | | | Текстовые документы |  | | | | |
|  | | | | |  |  | | | | |
| БГУИР ДП 1–40 01 03 00 023 ПЗ | | | | | Пояснительная записка | 86 c | | | | |
|  | | | | | Отзыв руководителя |  | | | | |
|  | | | | | Рецензия |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | | Графические документы |  | | | | |
|  | | | | |  |  | | | | |
| ГУИР.400103.023 ПД1 | | | | | Диаграмма классов | Формат А1 | | | | |
|  | | | | | (контроллеров). Чертеж. |  | | | | |
|  | | | | |  |  | | | | |
| ГУИР.400103.023 ПД2 | | | | | Диаграмма классов | Формат А1 | | | | |
|  | | | | | (моделей). Чертеж. |  | | | | |
|  | | | | |  |  | | | | |
| ГУИР.400103.023 ПД3 | | | | | Диаграмма компонентов | Формат А1 | | | | |
|  | | | | | (front-end). Чертеж. |  | | | | |
|  | | | | |  |  | | | | |
| ГУИР.400103.023 ПЛ1 | | | | | Диаграмма деятельности. | Формат А1 | | | | |
|  | | | | | Плакат. |  | | | | |
|  | | | | |  |  | | | | |
| ГУИР.400103.023 ПЛ2 | | | | | Диаграмма вариантов | Формат А1 | | | | |
|  | | | | | использования. Плакат. |  | | | | |
|  | | | | |  |  | | | | |
| ГУИР.400103.023 ПЛ3 | | | | | Диаграмма базы данных | Формат А1 | | | | |
|  | | | | | Плакат. |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  | | | | |  |  | | | | |
|  |  |  |  |  | ***БГУИР ДП 1- 40 01 03 00 023 Д1*** | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| *Изм* | *Л.* | *№ докум.* | *Подп.* | *Дата* | ***Портал для людей, занимающихся спортом.***  ***Ведомость дипломного  проекта*** |  | | | *Лист* | *Листов* |
| ***Разраб.*** | | ***Волчецкий А.М.*** |  |  | *Т* |  |  | *86* | *86* |
| ***Пров.*** | | ***Богушевский Е.Л.*** |  |  | ***Информатика***  ***гр. 253504*** | | | | |
| ***Т.контр.*** | | ***Богушевский Е.Л.*** |  |  |
| ***Н.контр.*** | | ***Шиманский В.В.*** |  |  |
| ***Утв.*** | | ***Волорова Н.А.*** |  |  |