**ДИПЛОМНА РАБОТА**

**„Виртуален треньор по калистеника“**

**Ученик: Станимир Иванов Колев**

**Клас: 12 „е“ клас**

**Имейл: stanimirkolev\_e18@schoolmath.eu**

**Специалност: Системно програмиране**

**Научен ръководител: Дарина Брънчева**

**Имейл: darinabrancheva@schoolmath.eu**

**Длъжност: старши учител по информатика и информационни технологии**

1. Увод 3

1.1 Цели и задачи 4

1.2 Обхват 4

2. Използвани технологии 4

2.1 JetBrains Rider 4

2.2 .NET 5

2.3 C# 7

2.4 ASP.NET 7

2.4.1 ASP.NET Core 7

2.4.2 Web API 8

2.4.3 Blazor 8

2.5 Entity Framework Core (EF Core) 9

2.6 MediatR 9

2.6.1 CQRS 9

2.6.2 MediatR 10

2.6.3 ErrorOr 10

2.7 FluentValidation 11

2.8 AutoMapper 11

2.9 xUnit.NET 12

2.10 JavaScript Web Token (JWT) 12

2.11 JavaScript 12

2.12 Docker 12

2.13 Azure 13

2.13.1 Azure Kubernetes Service (AKS) 13

2.13.2 Azure Key Vault 14

2.13.3 Azure Service Bus 14

2.13.4 Azure Functions 15

2.14 Python 15

2.14.1 Keras 16

2.15 Open Neural Network Exchange (ONNX) 16

3. Архитектура 17

3.1 Микросървиси 17

3.2 Архитектура на базата данни 18

3.2.1 Аuthorization 18

3.2.2 Skeletal 21

3.2.3 Таблица Workout 26

3.3 Архитектура на проекта 30

3.3.1 Common 30

3.3.2 Microservice 31

3.3.3 UI 33

3.3.4 Tests 33

4. Разработка на приложението 33

4.1 Еntity 33

4.2 DbContext 34

4.3 Repository 34

4.4 CQRS 35

4.5 Controller 36

4.6 Validator 36

4.7 Тестове 37

4.8 Program.cs 37

5. Ръководство на потребителя 38

5.1 Стартиране на приложението 38

5.2 Razor компоненти 39

5.2.1 Регистрационна форма 39

5.2.2 Влизане в акаунт 40

5.2.3 Администраторски изглед за управление на упражненията 40

6. Използвана литература 41

Увод

В България спортът калистеника (съвкупност от двигателни физически упражнения, използващи основно тежестта на тялото като съпротивление) е изключително популярен, но поради природата на този спорт няма треньори, които да навигират и подпомагат развитието на навлизащите в света на калистениката спортисти. Необходимо е за трениращите да бъде изготвен тренировъчен план, който да следват, но и да бъде оценено тяхното изпълнение на дадено физическо упражнение. „Виртуален треньор по калистеника“ е подобно уеб приложение, което отговаря на нуждите от напътствия по време на тренировка.

Цели и задачи

С помощта на „Виртуален треньор по калистеника“, спортистите имат възможност да създадат с помощта на изкуствен интелект тренировъчен план, специално пригоден за тях, както и да подложат своето изпълнение на дадено упражнение на оценка, която да спомага за техния по-нататъчен успех в спорта. Приложението цели да изгради и теоретичните знания на спортистите, като им предостави голям набор от информация за калистениката и човешкото тяло.

Обхват

Софтуерът се състои от вход за трениращи, които ще могат да създават нови тренировки, да оценяват изпълнението си, да засичат времето на дадена тренировка; вход за разработчици, които ще могат да настройват работата на приложението; вход за администратори, които ще могат да управляват информацията за упражненията, мускулите и тъканите в човешкото тяло и др.

Използвани технологии

JetBrains Rider

Rider е интегрирана среда за разработка на софтуер (IDE), разработена от JetBrains. Използва се за разработка на десктоп приложения, мобилни приложения, уеб приложения и видеоигри. Rider използва платформи за разработка на софтуер като .NET Core, ASP.NET Core и Unity. Rider включва в себе си редактор на кодове, интегриран с известното разширение за Visual Studio – ReSharper, също разработено от JetBrains, което е компонент за завършване на код и кодово реформиране. Rider поддържа и Roslyn, платформата за .NET компилатори, което допълнително предлага на разработчика кодово реформиране.

Другите вградени инструменти включват дизайнер на форми за изграждане на графичен потребителски интерфейс (GUI), уеб дизайнер, дизайнер на класове и дизайнер на схема за бази данни. Също така може да се инсталират допълнителни приставки, които добавят функционалност и/или подобряват продуктивността на разработчика, като Azure Toolkit for Rider, например. Rider предоставя вградени системи за контрол на версията (VCS) - Git, Subversion, Mercurial и Perforce.

Поддържаните езици и платформи включват C#, F#, JavaScript, .NET Core, .NET Framework, .NET Standard, XML, HTML, CSS, Razor; поддръжката за други езици, като Rust, например, е достъпна чрез приставки. Средата за разработка Rider се заплаща, но има пробен тестов период от 30 дни. Таблицата по-долу сравнява Rider с неговия конкурент Visual Studio и посочва предимствата на продукта на JetBrains.

|  |  |  |
| --- | --- | --- |
| Характеристика | JetBrains Rider | Microsoft Visual Studio |
| Мултиплатформност | Поддържа Windows, Linux, MacOS | Поддържа само Windows и MacOS |
| Анализиране и завършване на кода | ReSharper в комбинация с Roslyn | Intellisense, минимални анализатори |
| Производителност | Бързо и леко с минимални системни изисквания | Бавно на по – стар хардуер, повече системни изисквания |
| Цена | Нужен е лиценз | Безплатно |

Таблица 1 – Сравнение между Rider и Visual Studio

.NET

.NET Core е новата версия на .NET Framework, която е безплатна платформа за създаване на софтуер с отворен код, създадена от Microsoft. След версия 5.0 Microsoft преименуват .NET Core на .NET. Технологията е мултиплатвормена и поддържаWindows, MacOS и Linux-базирани операционни системи. .NET се използва за създаване на приложения от всякакъв род и вид – мобилни приложение, десктоп приложения, уеб и облачни приложения, микросървиси, изкуствен интелект и др.

.NET платформата е пренаписана изцяло, за да е по-модулярна и с по-висока производителност, но спазва подобна на .NET Framework структура. Приложенията се пишат на езици от високо ниво, като C#, F#, C++/CLI, които се компилират до платформено-независим междинен език, наречен Common Intermediate Language (CIL). По време на изпълнение на конкретна програма с помощта на Just In Time (JIT) компилатор междинният код се свежда до машинни инструкции за текущата хардуерна платформа и операционна система. Таблицата по-долу сравнява .NET с неговия конкурент Java и посочва предимствата на продукта на Microsoft.

|  |  |  |
| --- | --- | --- |
| Характеристика | .NET | Java |
| Производителност | Според индекса на производителност на TechEmpower, .NET оглавява класациите за JSON сериализация, заявки към бази данни, манипулация на низове и др. | Java е известна със своята производителност като език от високо ниво, но според индекса на TechEmpower е много по неефективна, отколкото .NET. |
| Сигурност | .NET има вградена функционалност против кибер атаки като препълване на буфер и премахва уязвимости като SQL инжекции и Cross-Site Scripting (XSS). | Java проявявва много повече уязвимости в сигурността, отколкото .NET. Ярък пример за това е катастрофичната за много бизнеси уязвимост Log4j (CVE-2021-44228). |
| Общност | Голяма и активна общност, която доразвива платформата и оказва помощ. | Java също има активна общност, която доразвива платформата и оказва помощ на нови в сферата разработчици. |
| Екосистема | Голяма екосистема, много библиотеки както от самите Microsoft, така и от общността. | Java има голяма екосистема, включваща голям брой проекти и библиотеки с отворен код. |
| Web Assembly (WASM) | Поддържа WASM и WASI. | Поддържа WASM, но не и WASI. |

Таблица 2 – Сравнение между .NET и Java

C#

C# е най-често използваният език за програмиране за .NET. Проектиран от екипа на Андрес Хейлсбърг, С# е съобразен с особеностите и изискванията на платформата .NET. Основните характеристики на езика са:

* Силна типизация, която позволява на разработчиците да откриват грешки по време на процеса на компилация, а не на изпълнение, което води до намаляване на количеството на грешки.
* Автоматично управление на паметта с помощта на Garbage Collector (GC), което освобождава разработчиците от ръчно управление на оперативната памет.
* Следва обектно ориентирания подход и поддържа капсулация на данните, наследяване и полиморфизъм, които са нужни за създаването на модерни приложения.

С# е най-удачният избор за програмиране на .NET поради неговата лесна употреба, гъвкавост и продуктивност.

ASP.NET

ASP.NET Core

ASP.NET Core, наследникът на ASP.NET, е безплатен междуплаформен фреймуърк за разработка на модерни уеб приложения, услуги и приложно програмни интерфейси (API). ASP.NET е модулярен и изключително лек, позволявайки на разработчиците да използват само необходимите компоненти, правейки разработката на уеб приложения лесна. ASP.NET е създадено върху уеб сървъра Kestrel, който е оптимизиран да поддържа голям набор от конкурентни връзки и поставя основата на каналите за заявки и отговори. Таблицата по – долу илюстрира разликата между ASP.NET Core и Java Spring и посочва предимствата на продукта на Microsoft пред Spring.

|  |  |  |
| --- | --- | --- |
| Характеристика | ASP.NET Core | Java Spring |
| Мултиплатформен | Да | Да |
| REST API | Да | Да |
| MVC | Да | Да |
| Рендериране при сървъра | Да (чрез Razor) | Да (чрез Thymeleaf) |
| Рендериране при клиента | Да (чрез WASM) | Не |

Таблица 3 – Сравнение между ASP.NET Core и Java Spring

Web API

Уеб API е програмно-приложен интерфейс, предназначен за уеб сървър и/или уеб браузър. Главната цел на уеб API е да предостави стандартен интерфейс за достъпване и взаимодействие с данни и услуги на дадено уеб приложение, като същевременно интегрира данни от различни източници, улеснявайки създаването на сложни и децентрализирани системи. Уеб API се създават с помощта на протокола HTTP, JavaScript Object Notation (JSON), Extensible Markup Language (XML) и Representational State Transfer (REST) архитектурата. В контекста на уеб програмирането, API е съвкупност от HTTP извикващи съобщения, заедно с дефиниция на структурата на отговарящите съобщения, най-често с JSON и XML.

Blazor

Blazor Server и Blazor Web Assembly (WASM) са уеб фреймуърци за разработка на интерактивни уеб приложения, използвайки C# и .NET. Blazor премахва нуждата от JavaScript при на интерактивни уеб приложения, но позволява и лесно интегриране на съществуващ JavaScript код.

От една страна, Blazor може да изпълнява рендерирането и логиката на сървъра и да актуализира потребителския интерфейс чрез SignalR (библиотека за комуникация в реално време, базирана на TCP). Сървърното изпълнение на Blazor допринася за повече сигурност и запазва интегритета на данните. Началната заявка е сравнително малка по размер, тъй като съдържа прекомпилирани компоненти, които осъществяват връзката със сървъра.

От друга страна, Blazor WASM изпълнява приложенията изцяло при клиента, което позволява режим на работа изцяло офлайн. Кодът на приложението се компилира към Web Assembly – стандарт, поддържан от всеки модерен браузър. Началната заявка е голяма по размер, понеже цялото приложение се тегли наведнъж.

Entity Framework Core (EF Core)

Entity Framework Core (EF Core) е обектно-релационен картограф (ORM), който позволява използването на обектно ориентирания подход при работа с данни, съхранявани в релационни източници на данни. Създаден от Microsoft през 2016 г., EF Core е наследник на Entity Framework, но е проектиран да бъде по-лесен за употреба и предлага по-добра производителност спрямо своя предшественик. Entity Framework Core поддържа голям набор от доставчици на бази данни, сред които SQL Server, MySQL, Azure Cosmos Db и др. Таблицата по – долу показва сравнение между EF Core и втората най-популярна ORM технология в екосистемата на .NET - Dapper.

|  |  |  |
| --- | --- | --- |
| Характеристика | Entity Framework Core | Dapper |
| Тип ORM | Пълнофункционален ORM | Микро ORM |
| Заявки | Интегрирани заявки в езика (LINQ) | SQL – базирани заявки и запазени процедури |
| Производителност | По – бавно от Dapper | По – бързо от EF Core |
| Картографиране | Автоматично | Ръчно |
| Интегритет на данните | Осъществява интегритет на данните чрез транзакции | Няма вграден механизъм за запазване на интегритета на данните |

Таблица 4 – сравнение между EF Core и Dapper

Въпреки очевидното превъзходство в производителността на Dapper, продуктът на Microsoft предлага повече функционалност и запазва цялостта на данните, което е приоритет в изграждането на едно приложение.

MediatR

CQRS

Разделяне на отговорностите за команди и запитвания (CQRS) е архитектурен шаблон, който цели логически да раздели операциите за четене и промяна върху данните. В традиционните монолитни приложения един и същи домейн модел се използва и за четене, и за промяна, което води до намаляване на производителността и скалируемостта.

MediatR

MediatR е библиотека, създадена от Джими Богарт и осъществява шаблона посредник (Mediator). Шаблонът посредник се използва за намаляване на обвързаността между отделните компоненти, като същевременно им позволява да комуникират помежду си чрез централизиран посредник. MediatR се използва за осъществяване на CQRS архитектурата, но и за изпращане на уведомления и събития. Би блиотеката поддържа синхронна и асинхронна обработка на данните.

MediatR разделя допълнително всяка обработка на заявка (request), уведомление (notification) и поточна зявка (stream request). Заявките са предназначени да бъдат получени от една обработваща единица (handler), която връща изходните си данни обратно на изпращача чрез посредника. От друга страна, уведомленията са сходни със събитията, тъй като могат да бъдат получени от повече от една обработваща единица, която обаче не може да връща данни обратно на изпращача. Поточните заявки поддържат потоци, което обуславя тяхната популярност в сферата на уеб API, тъй като потоците позволяват стрийминг на данни, допринасящ за изграждането на приложения с голяма производителност.

ErrorOr

ErrorOr е .NET библиотека, разработена от Амичай Мантинбанд, която предоставя механизъм за обработка на грешки в стила на функционалното програмиране. ErrorOr позволява навързването на операции една след друга и обработва грешките елегантно чрез опционални и резултатни типове. Библиотеката намаля излишния код и спомага за създаването на чист и лесен за поддръжка софтуер.

FluentValidation

FluentValidation е .NET библиотека за създаване на строги правила за валидиране на данни. Главната цел на тази библиотека е да изчисти кода на домейна и да предостави централизирано място, където се съхранява логиката за валидиране. FluentValidation може лесно да бъде интегриран с MediatR, чрез внедряването на валидационна логика в каналната архитектура на MediatR, което позволява валидация на командите и заявките.

FluentValidation реализира валидацията на данни чрез валидатори, които представляват класове, описващи валидното състояние на точно един клас. Библиотеката предоставя често срещани методи за валидация, сред които проверки за празна стойност, имейл, телефон и др., но предоставя и механизъм за описване на   
по-сложни валидационни методи чрез ламбда изрази.

AutoMapper

AutoMapper е сред най-използваните картографи (mapper) в .NET екосистемата. Неговата главна цел е да преобразува един тип обект в друг тип обект, чрез ясно описани правила. AutoMapper позволява преобразувания между обекти от един и същи тип, обекти от различен тип, вложени свойства и персонализирани фабрични методи за   
по-сложни случаи.

Конфигурацията за мапирането обикновено се извършва по време на инициализацията на приложението чрез профили, които са логическо разделение на отделните преобразувания на база домейн модел. AutoMapper използва компилирани изрази за картографиране по време на изпълнение на програмата, което е форма на оптимизирано отражение. Това позволява на AutoMapper да избегне понижаване на производителността, което се получава при традиционните техни на картографиран, изцяло базирани на отражение. AutoMapper компилира кода за мапиране в делегат, който може да бъде изпълнен директно.

xUnit.NET

xUnit.net е безплатен инструмент с отворен код за тестване на единица за .NET. Създаден от оригиналния автор на NUnit инструментът е лицензиран под Apache License 2.0.

JavaScript Web Token (JWT)

JSON Web Token (JWT) е стандард за създаване на сигурни жетони, удостоверяващи дигитална самоличност и най-често се използват при уеб приложенията. JWT е жетон, който сам осигурява своя интегритет и съдържа информация за даден потребител и неговите права в една информационна система.   
Най-често тази технология се използва при автентикацията и авторизацията на заявки, направени към уеб API. JWT е дигитално подписан, което осигурява неговата неговата автентичност и предотвратява промяна от злонамерени лица. Структурата на JWT (Фигура 4) се състои от заглавна част, съществени данни и подпис.

JavaScript

JavaScript е интерпретируем език за програмиране, поддържан от всеки модерен уеб браузър. Разработен от от Брендан Айк, JavaScript се използва за добавяне на функционалност към статична HTML страница, а с помощта на NodeJS може да бъде използван като сървърен език. В Интернет пространствтото съществува огромен набор от библиотеки с отворен код, създадени на JavaScript, който не са пренаписани на Blazor. Между двете технологии обаче съществува гъвкава интероперация, която позволява лесното използване на съществуващ JavaScript код директно от Blazor (независимо Server или WASM), както и обратната връзка.

Docker

Docker е платформа, която позволява на разработчиците да пакетират, разпращат и изпълняват приложения в контейнеризирана среда. Контейнерът представлява лек, самостоятелен и изпълним пакет от софтуер, който включва всичко необходимо за изпълнението на дадена програма като код, библиотеки, системни инструменти и настройки.

Docker предоставя централизиран регистър за запазване и споделяне на контейнерни изображения, което води до лесна дистрибуция и колаборация с други лица. Docker се интегрира лесно с други популярни инструменти като Kubernetes, правейки го ключов компонент от модерната софтуерна разработка.

Hyper-V е платформа за виртуализация, вградена в Windows, която позволява изпълнението на няколко виртуални машини върху един и също хост. Тази виртуална машина предоставя изолирана среда, която Docker да използва за изпълнение на контейнери. Самите контейнери не са виртуални машини, а по-скоро ненатоварващи процеси, изолирани чрез мрежови namespace-и.

От 2004 г. Docker поддържа и платформата Windows Subsystem for Linux 2 (WSL2) за бакенд на контейнери като заместител на Hyper-V. WSL2 е ненатоварваща Линукс виртуална машина, която лесно се интегрира с операционната система на хоста.

Azure

Azure е доставчик на облачни услуги, разработен от Microsoft, и предоставя на потребителите възможност да създават, разгръщат и управляват приложения и услуги чрез глобална мрежа от центрове за данни. Azure включва голям набор от услуги като виртуални машини, бази данни, пространство, когнитивни услуги и др.

Azure Kubernetes Service (AKS)

Kubernetes е платформа за управление на контейнери, която автоматизира разгръщането, скалирането и управлението на контейнеризирани приложеиня. Тя позволява лесното разгръщане на приложения върху клъстер от устройства. Kubernetes има богат набор от функционалности, включително автоматизирани развития и откати, балансиране на натоварването и автоматично възстановяване на приложенията.

Azure Kubernetes Service (AKS) е система за управление на контейнеризирани приложения, която е базирана на Kubernetes. Създадена от Microsoft, тя позволява лесно разгръщане на апликации, като същевременно скрива инфраструктурата отдолу.

Azure Key Vault

Azure Key Vault е услуга, предоставящ сигурен механизъм за запазване и управление на криптографски ключове, тайни и сертификати. Разработена от Microsoft, тя оперира върху хардуерни модули за сигурност (HSM), които представляват физични устройства, специфично създадени да предоставят по – голямо ниво на сигурност за чувствителни криптографски операции. Процесите на създаване на асиметрични ключове, криптиране и декриптиране се извършват в HSM, което предпазва криптографските тайни от нежелан достъп и промяна.

Azure Service Bus

Azure Service Bus е облачно-базирана услуга за съобщения, която предостава надеждна и сигурна комуникация между различни приложения и услуги. Технологията изолзва шаблони за комуникация Публикуване-Абониране и Точка-До-Точка за осъществяване на комуникация.

Azure Service Bus предлага два типа съобщителни единици – тема и опашка. Опашката се използва за комуникация Точка-До-Точка между един изпращач и един получател. Изпращачът изпраща съобщения по опашката, докато получателят чете съобщения от същата опашка. От друга страна, темите се използват за комуникация тип Публикуване-Абониране, където един изпращач може да изпрати съобщение на много абонати.

Azure Service Bus предоставя надеждна инфраструктура за изпращане на съобщения, която гарантира интегритет на съобщенията и запазване на техния ред. Продуктът на Microsoft запазва изпратената информация докато получателят не е готов да я приеме. Също така тази технология поддържа транзакции, което осигурява получаването на всички съобщения наведнъж или нито едно.

Azure Functions

Azure Functions е изчислителна услуга, която оперира на безсървърния принцип, което е облачен изчислителен модел, където доставчикът на облачни услуги управлява инфраструктурата и автоматично алокира ресурси, необходими за изпълнението на приложението. Това позволява на разработчика да се фокусира единствено върху създаването на разгръщаем код, вместо да ръчно да настройва и управлява сървър за неговото изпълнение.

Продуктът на Microsoft позволява създаването на събитийно-ориентирани приложения чрез малки частици код, наречени функции, които могат да бъдат изпълнени след различни типове събития – HTTP заявки, съобщения в опашка, актуализация на бази данни и др. Също така Azure Functions се интегрират с голяма част от услугите на Azure и системи от трети страни, предоставяйки на разработчика голям избор от задвижващи механизми.

Освен задвижващи механизми, в Azure Functions съществува и концепцията за връзки. Връзките са метод за свързване на входните и изходните данни на функцията. Например, изходните данни могат да бъдат свързани към SQL сървър, опашка, HTTP отговор и др.

Python

Python е интерпретируем език от високо ниво, който се фокусира върху лесната употреба и четимостта на кода. Създаден от Гуидо ван Росум през 1991, Python е един от най-популярните езици поради своята простота.

Python е препоръчваният език за създаване на изкуствени интелект, поради богатия набор от функции, сред които динамични типове, автоматично управление на паметта и голяма стандартна библиотека. Също така съществува огромен брой библиотеки, предназначени за научни дейности и разработка на изкуствен интелект, сред които NumPy, SciPy, Pandas и др. Тези библиотеки позволяват на разработчика да изпълнява сложни изчисления с лекота, правейки Python препоръчваният език за разработка на изкуствен интелект.

Keras

Keras е популярна библиотека с отворен код, която се използва за изграждане и трениране на невронни мрежи с Python. Тя е проектирана да бъде лесна за използване, силно модулярна и разшируема. Keras се използва за решаване на проблеми като класификация на изображения, откриване на обекти, обработка на езици и др. Keras предоставя прост интерфейс за създаване и трениране на невронни мрежи, като абстрактира голяма част от сложността на TensorFlow – библиотека, върху която е базиран Keras.

TensorFlow е библиотека с отворен код, разработена от Google. Тя е предназначена за изграждане на всякакъв вид решения – от проста линейна регресия до невронни мрежи. Също така поддържа голям набор от инструменти за трениране на различни типове невронни мрежи – конволюционни невронни мрежи (CNN), рекурсивни невронни мрежи (RNN), трансформатори и др.

Keras поддържа богат набор от технологии, сред които дълготрайна кратковременна памет (LSTM), която е специфично проектирана да запомня зависимости в последователни данни като език, реч, времеви редове и др. Създадени през 1997 г. от Хохрайтер и Шмидхубер, LSTM мрежите успяват да запомнят или забравят информация от предишни времеви стъпки чрез използването на специализирани клетки за памет и три вида вратички (входни, изходни и забравителни), които контролират потока на данни през мрежата. LSTM мрежите се тренират най-често с помощта на техники за оптимизация по градиент и обратно разпространение във времето (BPTT). Предимството на LSTM пред традиционния рекурсивен модел се крие във възможността на LSTM да запазва зависимостите в данните за дълги периоди от време, което допринася за създаване на по – точен модел.

Open Neural Network Exchange (ONNX)

Open Neural Network Exchange (ONNX) е отворен формат за представяне на модели на невронни мрежи, поначало разработен от Microsoft. Целта на ONNX е да предостави стандартизиран формат за описание на невронни мрежи, който да бъде използван от различни библиотеки и платформи. Това позволява на разработчиците да тренират даден модел с една библиотека, но да го разгърнат с друга. ONNX поддържа голям набор от библиотеки, сред които TensorFlow, PyTorch и Keras.

ONNX третира моделите като съвкупност от изчислителни графи, които представят операциите и потока на данни между тях. Тези графи се запазват във формата protobuf, който е платформено-независим двоичен формат, който се използва за сериализация и десериализация на данни. Освен изчислителните графи, ONNX запазва параметрите (тежестите) на даден модел под формата на тенсори, които представляват многоизмерни масиви. Запазването на тенсорите във формата protobuf позволява за лесното разгръщане на модела на различни платформи и устройства.

Архитектура

В тази част на дипломната работа ще бъде разгледана основната концепция на работата на системата и технологиите, свързани с разработката ѝ.

Микросървиси

Микросървисите са архитектурен подход на разработка на софтуер, при който приложението е съвкупност от услуги (сървиси). Всяка услуга е автономна – може да бъде разработвана, разгърната и мащабирана без това да влияе на някоя от другите услуги в системата. Микросървисите не споделят код или конкретни имплементации помежду си, комуникацията между тях се осъществява чрез добре дефинирани API. Всяка услуга е специализирана – съдържа определена функционалност и е фокусирана върху решаването на даден проблем.

Освен разделение на ниво бизнес функционалност, при микросървис архитектурата е често срещана практика и използването на отделна база данни за всеки микросървис, което допринася за независимостта и изолацията на всяка услуга, подобрява гъвкавостта, позволявайки всяка услуга да използва най – подходящата технология, както и повишава нивото на сигурност и намалява риска от изтичане на данни от системата.

Graphical user interface, application

Description automatically generated

Архитектура на базата данни

Аuthorization

Graphical user interface, application

Description automatically generated

Авторизацията е имплементирана с помощта на технологията Identity – системата за автенрикация и авторизация на .NET. Тъй като приложението използва авторизация основана на твърдения (claims) и JWT, по-долу са разгледани само използваните таблици. Останалите таблици – AspNetRoles, AspNetUserTokens, AspNetUserLogins, AspNetUserRoles и AspNetRoleClaims се създават по подразбиране от Identity, но не се използват при авторизация основана на твърдения.

Таблица AspNetUsers

|  |  |  |
| --- | --- | --- |
| Име на колоната | Тип на колоната | Описание |
| CreatedOn | datetime2 | Дата и час на създаване |
| UpdatedOn | datetime2 | Дата и час на последната актуализация |
| UserName | nvarchar(256) | Потребителско име |
| NormalizedUserName | nvarchar(256) | Нормализирано потребителско име |
| Email | nvarchar(256) | Имейл |
| NormalizedEmail | nvarchar(256) | Нормализиран имейл |
| EmailConfirmed | bit | Потвърждение на имейл |
| PasswordHash | nvarchar(max) | Хеширана парола |
| SecurityStamp | nvarchar(max) | Уникална стойност, използвана при мултифакторната автентикация и др. |
| ConcurrencyStamp | nvarchar(max) | Уникална стойност, използвана при конкурентното актуализиране на потребителя |
| PhoneNumber | nvarchar(max) | Телефон |
| PhoneNumberConfirmed | bit | Потвърждение на телефон |
| TwoFactorEnabled | bit | Включена мултифакторна автентикация |
| LockoutEnd | datetimeoffset | Дата и час на заключване на потребителя, ако то съществува |
| LockoutEnabled | bit | Заключване на потребителя |
| AccessFailedCount | int | Брой на отказани достъпи |
| Id | nvarchar(450) | Първичен ключ |

Таблица 5

Таблицата AspNetUsers съдържа основната информация за отделните потребители. Полето Id служи като първичен ключ на таблицата, представлява уникален идентификатор за всеки отделен обект от таблицата. Полетата CreatedOn и UpdatedOn съдържат датата и часа на създаване и последна актуализация на данните на потребителя. Полето UserName съдържа името на потребителя. Полето NormalizedUserName съдържа нормализираното име на потребителя. Полето Email съдържа имейла на потребителя, а полето NormalizedEmail – нормализираната стойност на имейла. EmailConfirmed представлява потвърждението на имейла. Полето PasswordHash съдържа хешираната парола. SecurityStamp и ConcurrencyStamp са уникални стойности, използвани при операции за сигурността и конкурентната актуализация на стойности. Полето PhoneNumber съдържа телефона на потребителя, а PhoneNumberConfirmed – потвържението на телефона. TwoFactorEnabled се отнася за включването/изключването на двуфакторна автентикация. LockoutEnabled показва наличието на заключване на потребителя, а ако има такова неговия край се записва в LockoutEnd. Полето AccessFailedCount съдъжа броя на неуспешни опити за влизане. Полето Id служи като първичен ключ на таблицата, представлява уникален идентификатор за всеки отделен обект от таблицата.

Таблица AspNetClaims

|  |  |  |
| --- | --- | --- |
| Име на колоната | Тип на колоната | Описание |
| Id | int | Първичен ключ |
| UserId | nvarchar(450) | Идентификатор на потребителя, външен ключ |
| ClaimType | nvarchar(max) | Тип на твърдението |
| ClaimValue | nvarchar(max) | Стойност на твърдението |

Таблица 6

Талбицата AspNetUserClaims съдържа информацията за всяко твърдение за даден потребител. Полето Id служи като първичен ключ на таблицата, представлява уникален идентификатор за всеки отделен обект от таблицата. Полето UserId е външен ключ към таблицата AspNetUsers, съдържа идентификатора на потребителя. Полето ClaimType представлява типът на даденото твърдение, а в ClaimValue се запазва неговата стойност.

Skeletal

Graphical user interface

Description automatically generated

Фигура 3

Таблица Authors

Таблицата Authors съдържа информация за всеки автор и произлиза от таблицата AspNetUsers на услугата Authorization (виж 3.2.1.1 и Фигура 3), като запазва единствено UserName(потребителското име) и ExternalId. Полето ExternalId представлява първичния ключ на записа в базата данни, от която той произлиза.

Таблица MuscleGroups

|  |  |  |
| --- | --- | --- |
| Име на колоната | Тип на колоната | Описание |
| Id | Int | Първичен ключ |
| Name | Nvarchar(max) | Име на мускулната група |
| Description | Nvarchar(max) | Описание на мускулната група |
| CreatedOn | datetime2 | Дата и час на създаване |
| UpdatedOn | datetime2 | Дата и час на последна актуализация |

Таблица 7

Таблицата MuscleGroups съдържа информация за всяка мускулна група в човешкото тяло. Полето Id служи като първичен ключ на таблицата, представлява уникален идентификатор за всеки отделен обект от таблицата. Полето Name съдържа името на мускулната група, а Description – нейното описание. Полетата CreatedOn и UpdatedOn съдържат датата и часа на създаване и последна актуализация на данните за всяка мускулна група.

Таблица Muscles

|  |  |  |
| --- | --- | --- |
| Име на колоната | Тип на колоната | Описание |
| Id | int | Пъвичен ключ |
| GroupId | int | Идентификатор на мускулната група, външен ключ |
| Name | nvarchar(max) | Име на мускула |
| Description | nvarchar(max) | Описание на мускула |
| CreatedOn | datetime2 | Дата и час на създаване |
| UpdatedOn | datetime2 | Дата и час на последна актуализация |
| TypeOneFiberPercentage | real | Процент на влакна от първи тип |
| TypeTwoFiberPercentage | real | Процент на влакна от втори тип |
| TypeThreeFiberPercentage | real | Процент на влакна от трети тип |

Таблица 8

Таблицата Muscles съдържа информация за всички мускули в човешкото тяло. Полето Id служи като първичен ключ на таблицата, представлява уникален идентификатор за всеки отделен обект от таблицата. Полето GroupId е външен ключ към таблицата MuscleGroups, съдържа идентификатора на мускулната група. Полето Name съдържа името на мускула, а Description – неговото описание. Полетата CreatedOn и UpdatedOn съдържат датата и часа на създаване и последна актуализация на данните за всеки мускул. Полетата TypeOneFiberPercentage, TypeTwoFiberPercentage и TypeThreeFiberPercentage съдържат процентното съотношение съответно на влакна от първи, втори и трети тип.

Таблица Fibers

|  |  |  |
| --- | --- | --- |
| Име на колоната | Тип на колоната | Описание |
| Id | Int | Първичен ключ |
| Name | nvarchar(max) | Име на мускулното влакно |
| Description | nvarchar(max) | Описание на мускулното влакно |
| MotorUnitType | nvarchar(max) | Tип на моторната единица |
| TwitchSpeed | nvarchar(max) | Скорост на съкращение на мускула |
| TwitchForce | nvarchar(max) | Сила на съкращение |
| ResistanceToFatigue | nvarchar(max) | Устойчивост на умора |
| CreatedOn | datetime2 | Дата и час на създаване |
| UpdatedOn | datetime2 | Дата и час на последна актуализация |

Таблица 9

Таблицата Fibers съдържа информация за различните мускулни влакна в човешкото тяло. Полето Id служи като първичен ключ на таблицата, представлява уникален идентификатор за всеки отделен обект от таблицата. Полето Name съдържа името на мускулното влакно, а Description – неговото описание. Полетата CreatedOn и UpdatedOn съдържат датата и часа на създаване и последна актуализация на данните за всяко мускулна влакно.

Таблица Exercises

|  |  |  |
| --- | --- | --- |
| Име на колоната | Тип на колоната | Описание |
| Id | int | Първичен ключ |
| Name | nvarchar(max) | Име на упражнението |
| Description | nvarchar(max) | Описание на упражнението |
| AuthorId | int | Идентификатор на автора, външен ключ |
| CreatedOn | datetime2 | Дата и час на създаване |
| UpdatedOn | datetime2 | Дата и час на последна актуализация |
| Video | varbinary(max) | Съдържа видео на изпълнението на упражнението |
| Thumbnail | varbinary(max) | Съдържа заглавната снимка |
| ExecutionSteps | nvarchar(max) | Съдържа стъпки с инструкции за изпълнение |
| IsWeighted | bit | Указва използването на допълнителна тежест |

Таблица 10

Таблицата Exercises съдържа информация за различните физически упражнения. Полето Id служи като първичен ключ на таблицата, представлява уникален идентификатор за всеки отделен обект от таблицата. Полето AuthorId е външен ключ към таблицата Authors, съдържа идентификатора на автора. Полето Name съдържа името на упражнението, а Description – неговото описание. Полетата CreatedOn и UpdatedOn съдържат датата и часа на създаване и последна актуализация на данните за всяко упражнение. Полетата Video и Thumbnail съдържат съответно видеоклип и заглавна снимка на упражнението в двоичен формат. Полето ExecutionSteps съдържа инструкции за изпълнение, а IsWeighted указва използването на допълнителна тежест за упражнението.

Таблица ExerciseMuscle

|  |  |  |
| --- | --- | --- |
| Име на колоната | Тип на колоната | Описание |
| ExerciseId | int | Идентификатор на упражнение, външен ключ |
| TargetsId | int | Идентификатор на мускул, външен ключ |

Таблица 11

Таблицата ExerciseMuscle служи за осъществяване на връзка тип много към много между упражненията и мускулите. Полето ExerciseId е външен ключ към таблицата Exercises, съдържа идентификатора на упражнението. Полето TargetsId е външен ключ към таблицата Muscles, съдържа идентификатора на мускула.

Таблица Workout

**Diagram

Description automatically generated**

Фигура 4

Таблица Users

Таблицата Users съдържа информация за всеки автор и произлиза от таблицата AspNetUsers на услугата Authorization(виж 3.2.1.1 и Фигура 4), като запазва допълнително само ExternalId. Полето ExternalId представлява първичния ключ на записа в базата данни, от която той произлиза.

Таблица Exercises

Таблицата Exercises съдържа информация за всяко упражнение и произлиза от таблицата Exercises на услугата Skeletal (виж 3.2.2.5 и Фигура 4), като запазва допълнително само ExternalId. Полето ExternalId представлява първичния ключ на записа в базата данни, от която той произлиза.

Таблица Workouts

|  |  |  |
| --- | --- | --- |
| Име на колоната | Тип на колоната | Описание |
| Id | Int | Първичен ключ |
| UserId | Int | Идентификатор на потребителя, външен ключ |
| Name | Nvarchar(max) | Име на тренировката |
| Notes | Nvarchar(max) | Бележки |
| Type | Nvarchar(max) | Тип на тренировката |
| CompletedOn | datetime2 | Дата и час на завършване |
| CreatedOn | datetime2 | Дата и час на създаване |
| UpdatedOn | datetime2 | Дата и час на последна актуализация |

Таблица 12

Таблицата Workouts съдържа информация всяка тренировка. Полето Id служи като първичен ключ на таблицата, представлява уникален идентификатор за всеки отделен обект от таблицата. Полето UserId е външен ключ към таблицата Users, съдържа идентификатора на потребителя. Полето Name съдържа името на тренировката, а Notes – бележки към нея. Полетата CreatedOn и UpdatedOn съдържат датата и часа на създаване и последна актуализация на данните за всяка тренировка. Полето CompletedOn съдържа датата и часа на завършване на тренировка.

Таблица Sets

|  |  |  |
| --- | --- | --- |
| Име на колоната | Тип на колоната | Описание |
| Id | Int | Първичен ключ |
| ExerciseId | Int | Идентификатор на упражнението, външен ключ |
| WorkoutId | Int | Идентификатор на тренировката, външен ключ |
| Notes | Nvarchar(max) | Бележки към серията |
| TargetReps | Int | Желаните повторения |
| CompletedReps | Int | Изпълнените повторения |
| CreatedOn | datetime2 | Дата и час на създаване |
| UpdatedOn | datetime2 | Дата и час на последна актуализация |

Таблица 13

Таблицата Sets съдържа информация всяка серия. Полето Id служи като първичен ключ на таблицата, представлява уникален идентификатор за всеки отделен обект от таблицата. Полето WorkoutId е външен ключ към таблицата Workouts, съдържа идентификатора на тренировката. Полето ExerciseId е външен ключ към таблицата Exercises, съдържа идентификатора на упражнението. Полето Notes съдържа бележки към тренировката. Полетата TargetReps и CompletedReps съдържат съответно желаните и изпълнените повторения. Полетата CreatedOn и UpdatedOn съдържат датата и часа на създаване и последна актуализация на данните за всяка тренировка.

Таблица Muscles

Таблицата Мuscles съдържа информация за всеки мускул и произлиза от таблицата Muscles на услугата Skeletal (виж 3.2.2.3 и Фигура 4), като запазва допълнително само ExternalId. Полето ExternalId представлява първичния ключ на записа в базата данни, от която той произлиза.

Таблица ExerciseMuscle

Таблицата ExerciseMuscle ExerciseMuscle служи за осъществяване на връзка тип много към много между упражненията и мускулите и произлиза от таблицата Muscles на услугата Skeletal (виж 3.2.2.6 и Фигура 4).

Таблица Preferences

|  |  |  |
| --- | --- | --- |
| Име на колоната | Тип на колоната | Описание |
| Id | int | Идентификатор на потребителя |
| ExerciseRestTime | float | Почивка между упражненията |
| SetRestTime | float | Почивка между сериите |
| CreatedOn | datetime2 | Дата и час на създаване |
| UpdatedOn | datetime2 | Дата и час на последна актуализация |

Таблица 14

Таблицата Preferences съдържа предпочитанията на даден потребител за тренировка. Полето Id е външен ключ към таблицата Users, съдържа идентификатора на потребителя. Полетата ExerciseRestTime и SetRestTime съдържат съответно почивките между упражненията и сериите. Полетата CreatedOn и UpdatedOn съдържат датата и часа на създаване и последна актуализация на данните за всяка тренировка.

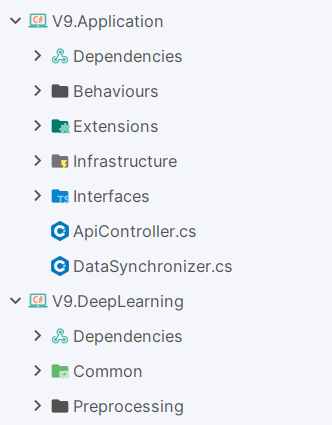
Архитектура на проекта

Graphical user interface

Description automatically generated with medium confidenceGraphical user interface, application

Description automatically generated

Common

** Graphical user interface, application

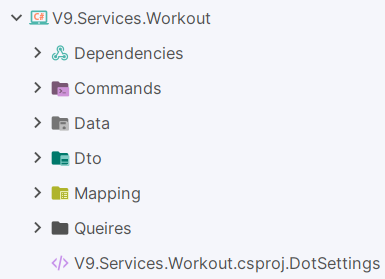
Description automatically generated**

В Common се намират проекти, касаещи всяка услуга. Във V9.Domain се съдържат базови класове относно съхраняваните данни и събития. V9.Application съдържа основни CQRS канали, разширения за услуги и сигурност, основното репозитори, както и интерфейси за инфраструктурата. V9.Infrastructure съдържа конкретни имплементации на интерфейсите за инфраструктурата и доставчиците на облачни услуги. Във V9.DeepLearning се съдържат основни манипулации на данни, свързани с изкуствен интелект.

Microservice

Всяка услуга следва обща структура, която се състои от два до четири проекта, сред които: проект, който съдържа цялата бизнес логика, валидиране, обработка и т.н; проект, който предоставя тази логика чрез API; безсървърен проект, който слуша за събития; проект, който съдържа изкуствения интелект и обработва неговия вход и изход. Тъй като всички услуги следват обща структура, по-долу е разгледан само Workout, тъй като включва и четирите типа проекти.

Проектът V9.Services.Workout съдържа CQRS логиката (Commands и Queries), връзката с базите данни (контекст класа, репозитории, модели на обектите), мапиране и обекти за трансформация.



Проектът V9.Services.Workout.Api представлява RESTful API и съдържа контролери, конфигурация на сървъра и компилираните ONNX модели.

Graphical user interface, application

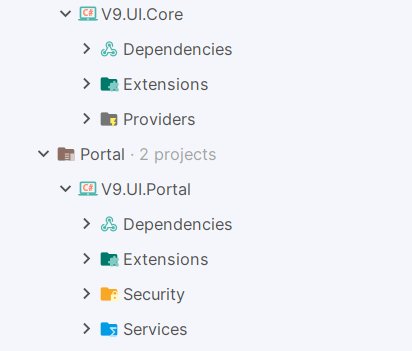
Description automatically generated

Проектът V9.Services.Workout.DeepLearning съдържа inference модела на изкуствения интелект, описания за неговите тенсори, входни и изходни данни, както и помощни обработчици на данни. Проектът V9.Services.Workout.Serverless е безсървърно приложение, което съдържа логика за мапиране на обекти и слушаща функционалност към облачна опашка. Този проект отговаря за обработката на събития, създадени и публикувани от другите микроуслуги.

Graphical user interface, application

Description automatically generated

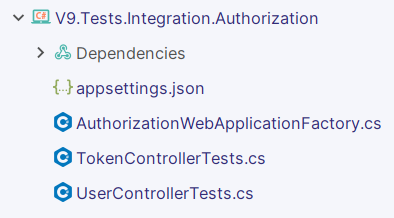
UI

** Graphical user interface, application

Description automatically generated**

Проектът V9.UI.Core съдържа основна функционалност на потребителския интерфейс, като разширения за сигурност и доставчици на автентикация. Проектът V9.UI.Portal съдържа повече разширения за сигурност, както и услуги за комуникация с останалите микросървиси. Проектът V9.UI.Portal.Blazor предоставя потребителски интерфейс чрез технологията Blazor.

Tests

Показаният проект на фигурата представлява интеграционните тестове на микросървиса Authorization. Той съдържа уеб фабрика и тестове за отделните контролери.

Разработка на приложението

Еntity

Entity класовете описват различните обекти от базата данни и техните характеристики. Показаният клас Exercise наспедява EntityBase<T>, който бива наследяван и от останалите Entity обекти. Базовият клас съдържа уникален идентификатор, чийто тип бива посочен чрез generic, както и дата и час на създаване и на последна промяна на даденото Entity.

Graphical user interface, text, application

Description automatically generated

DbContext

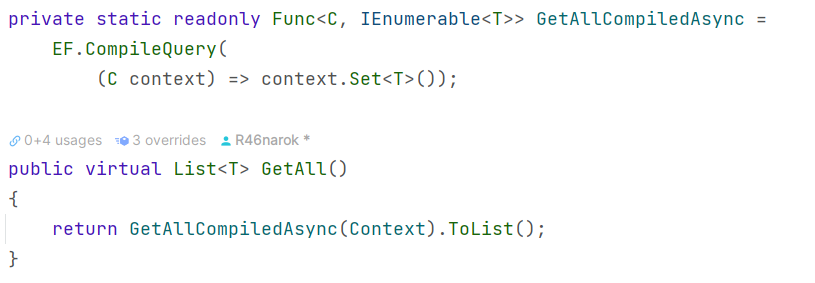
Показаният клас SkeletalDbContext представлява клас, който описва таблиците в базата данни и указанията при създаването на базата и връзката с нея. Показаният метод OnModelCreating съдържа инструкции, които се изпълняват при първоначално създаване на базата:

Graphical user interface, text, application, chat or text message

Description automatically generated

Repository

Repository класовете описват основните CRUD операции върху базата данни. Съществува базов клас RepositoryBase<Т, К, C>, който бива наследяван от всички repository класове. Generic параметъра “T” обуславя типа Entity, “К” представя типа на първичния ключ, а “C” – типа на DbContext класа. RepositoryBase имплементира основните CRUD операции чрез функционалността за компилирани заявки на Entity Framework Core, което допринася за по – голяма производителност докато запазва гъвкавостта на ORM технологията. Показаната компилирана заявка и метод са част от RepositoryBase и служат за извличането на всички записи от дадена таблица.



CQRS

Заявките и командите съдържат бизнес логиката на приложението. Всяка една от тях има входни данни, които наследяват интерфейса IRequest от MediatR, чийто generic параметри указват изходните данни на заявката или командата. Изходните данни са обгърнати в ErrorOr, което позволява използването на същия механизъм за обработка на грешки в каналите на MediatR при бизнес логиката. Съществува и един обработващ клас, отговарящ за обработката на точно една команда или заявка. Показана е заявката за четене на потребител по потребителско име.



Controller

Контролерите са клас, който взима необходимата информация от заглавната част или тялото на HTTP заявката, извикват обработваща функционалност и връщат HTTP отговор. Показаният метод е част от WorkoutController, използва JWT авторизация и служи за създаване на тренировки.

Text

Description automatically generated

Validator

Validator класовете осъществяват валидацията на данните на ниво команди и заявки (CQRS). Имплементират се чрез .NET библиотеката FluentValidation. FluentValidation изисква от разработчика да създаде свой валидатор, който служи за валидиране на точно една комманда или заявка, като наследява AbstractValidator<>. Един от валидаторите изглежда по следния начин:

Graphical user interface, text

Description automatically generated

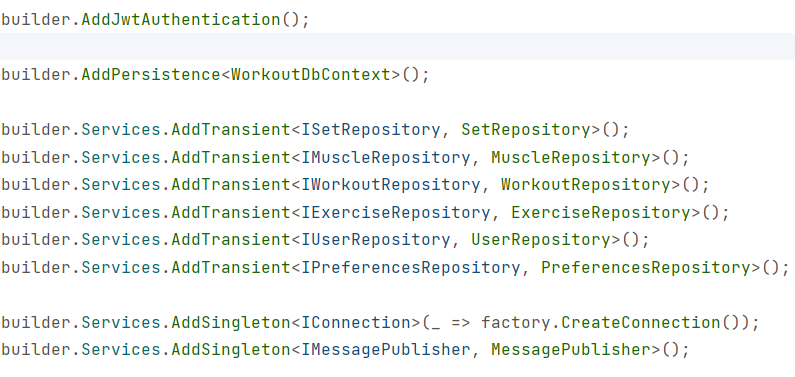
Тестове

Интеграционните тестове са реализирани чрез библиотеката Bogus за генериране на реалистични тестови данни. Използвайки механизма WebApplicationFactory, за всеки IClassFixture се инициализира целия уеб runtime и се създава клиент за взаимодействие с него.



Program.cs

Файлът Program.cs на всяка микроуслуга съдържа основна конфигурация и стартира уеб приложението. Показаният фрагмент е част от услугата Workout и илюстрира добавянето услуги към контейнера за инжектиране на зависимости (Dependency injection), сред които JWT автентикация, репозитории и механизми за публикуване на съобщения към опашки.



Ръководство на потребителя

Стартиране на приложението

За стартиране на приложението локално е нужно да бъдат инсталирани следните технологии: .NET 7, Project Tye, както и Docker. След като всички тези технологии са инсталирани е нужно да се изпълнят следните стъпки:

1. Стартиране на композираната конфигурация за SQL Server, намираща се в tools/mssqlserver чрез: **$docker compose up -d**.
2. Стартиране на Tye конфигурацията в главната директория на проекта, съдържаща всички микроуслуги, както и потребителския интерфейс чрез: **$tye run**.

Изчаква се стартирането на всяка услуга, преди да се продължи към следващата стъпка. След пълната инициализация, приложението е достъпно на [**http://localhost:5241**](http://localhost:5241).

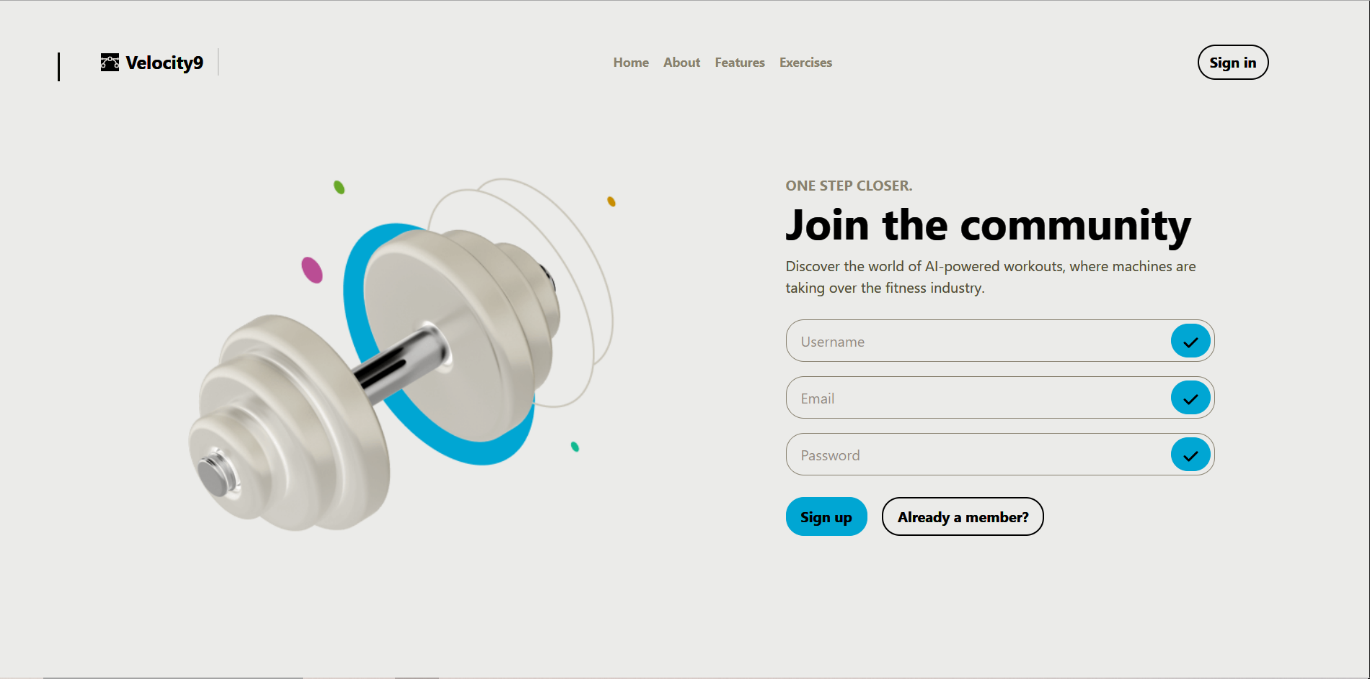
Вторият метод за стартиране на приложението е разгръщането му в Azure. За тази цел е необходимо да бъде инсталиран инструментът Terraform на HashiCorp. След това е нужно да бъде изпълнено **$terraform apply** в директорията deploy, както и да бъде подадено на входа следните характеристики на Azure профил:

1. Уникалния идентификатор на принципа на обслужване (Service Principle)
2. Тайният ключ на Service Principle
3. Уникалния идентификатор на наемателя (Tenant)
4. Уникалния идентификатор на Azure абонамент (subscription)
5. SSH ключ

Razor компоненти

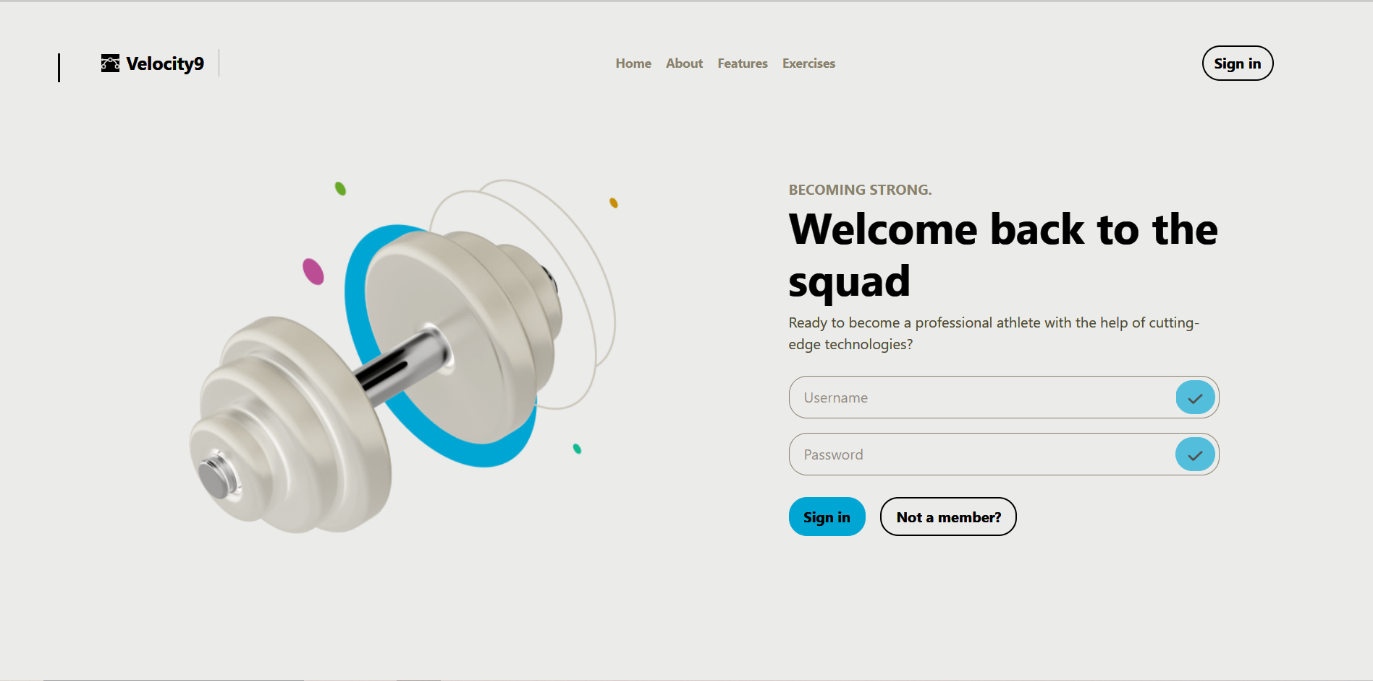
В основата на уеб приложението стои концепцията за компоненти, които представляват изгледите на проекта. Част от тях са:

Регистрационна форма

****

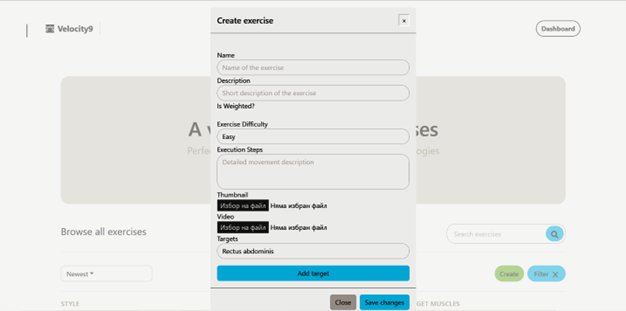
Потребителят може да създаде свой профил в системата чрез формата за регистрация. За тази цел е необходимо той да въведе уникално потребителско име, валиден имейл, както и парола. Паролата трябва да бъде поне 6 символа и да съдържа поне една главна и малка буква, цифра, както и специален символ. С натискане на бутона “Sign up” ще бъде изпълнена процедурата по създаване на профил в системата. Ако потребителят има вече съществуващ профил и иска да влезе в него, е нужно да бъде натиснат бутона “Already a member?” за препращане към страницата за вход в системата.

Влизане в акаунт

****

Потребителят може да влезе във вече съществуващ профил чрез формата за вход. За тази цел е необходимо той да въведе своето потребителско име и парола. С натискане на бутона “Sign up” ще бъде изпълнена процедурата по влизане в профил. Ако потребителят няма профил и иска да създаде такъв, е нужно да бъде натиснат бутона “Not a member?” за препращане към страницата за регистрация в системата.

Администраторски изглед за управление на упражненията

****

Администраторът може да управлява упражненията в системата. Показаният компонент служи за създаване на упражнение в системата. За тази цел администраторът трябва да въведе уникално име на упражнението, кратко описание за него, както и допълнителна информация – добавена тежест, стъпки за изпълнение, корица и видео към упражнението, както и трудност и мускули, които натоварва упражнението. С бутона „Save changes“ новосъздаденото упражнение се запазва в системата, а с бутона „Close“ – се затваря администраторското меню.

Използвана литература

<https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet>

<https://jwt.io/introduction>

<https://code-maze.com/cqrs-mediatr-fluentvalidation/>

<https://docs.automapper.org/en/latest/>

<https://learn.microsoft.com/en-us/azure/?product=popular>

<https://www.techempower.com/benchmarks/#hw=ph&test=plaintext&section=data-r21>

<https://www.docker.com/>

<https://en.wikipedia.org/wiki/JavaScript>

<https://en.wikipedia.org/wiki/Python_(programming_language)>

<https://onnx.ai/onnx/intro/concepts.html>

<https://keras.io/about/>

<https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet>

<https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html#overview>

<https://microservices.io/>

<https://intellipaat.com/blog/what-is-lstm/#:~:text=LSTM%20stands%20for%20long%20short,especially%20in%20sequence%20prediction%20problems>.