# Увод

Технолошки развој утицао је на све аспекте савременог живота, укључујући и начин на који појединци воде рачуна о свом здрављу и физичкој активности. Савремени темпо живота, стрес, недовољно кретања и неправилна исхрана све више утичу на здравствено стање становништва, због чега се јавља потреба за практичним и доступним рjешењима која могу подстаћи и подржати кориснике у усвајању здравих животних навика.

У том контексту, дигитална рjешења, попут веб и мобилних апликација, добијају све већи значај. Они омогућавају корисницима да на једноставан начин прате своју физичку активност, надгледају промjене тjелесне тежине, планирају тренинге и исхрану, те прате напредак током времена. Управо на тим принципима заснива се овај пројекат – израда веб апликације за праћење физичке активности и мjерење тjелесних параметара.

Развијена апликација, осмишљена као фитнес платформа, корисницима пружа функционалности као што су регистрација и пријава, креирање личног профила са могућношћу додавања фотографије и личних података, израчунавање индекса тjелесне масе (BMI) и базалног метаболизма (BMR), као и визуелно праћење промjене тjелесне тежине кроз дијаграме. Једна од важних функционалности јесте персонализовани садржај, који се прилагођава полу корисника, чиме се побољшава релевантност препорука и савјета. Такође, апликација омогућава приказ и управљање програмима тренинга и исхране, које уноси и ажурира администратор (тренер), чиме се обезбјеђује флексибилност и континуирано прилагођавање потребама корисника.

Технолошки гледано, апликација је реализована примјеном савремених и широко коришћених алата у развоју веб апликација: **ASP.NET Core** на серверској страни и **React.js** на клијентској. Као база података коришћена је **SQLite**, што омогућава лаку имплементацију и брз развој. Архитектура апликације је модуларна, што омогућава лако одржавање, проширивост и накнадно додавање нових функционалности.

Циљ овог рада је приказати процес развоја једне функционалне и интуитивне веб апликације која доприноси побољшању здравственог стања корисника кроз подршку у физичкој активности и праћењу кључних тјелесних параметара. Поред тога, рад истражује могућности интеграције технолошких рјешења у свакодневне активности, као и потенцијал за њихову примјену у области превентивне медицине и здравог стила живота.

# Фитнес и значај праћења здравља

## 2.1. Физичка активност и здравље

Физичка активност представља један од кључних фактора у одржавању и унапрјеђењу општег здравља. Редовно кретање има доказано позитиван утицај на кардиоваскуларни, респираторни, мишићно-коштано-зглобни и ментални систем човјека. Свјетска здравствена организација (СЗО) препоручује најмање 150 минута умјерене или 75 минута интензивне физичке активности недељно за одрасле особе, како би се смањио ризик од хроничних болести попут дијабетеса типа 2, хипертензије, гојазности и срчаних обољења. У савременом свијету, у којем доминирају седентеран начин живота и прекомјерно коришћење дигиталних уређаја, све је већа учесталост здравствених проблема који су директно повезани са недостатком физичке активности. Гојазност, смањена мишићна маса, болови у кичми, проблеми са држањем тијела, али и анксиозност и депресија – све су то стања на које физичка активност има снажан превентивни и терапијски утицај. Поред физичких бенефита, физичка активност има снажан утицај на психичко здравље. Редовно вјежбање доприноси бољем расположењу, већем самопоуздању и смањењу нивоа стреса. Такође, активност поспјешује лучење ендорфина и допамина – хормона који директно утичу на осећај среће и задовољства. Међутим, да би физичка активност била ефективна и одржива, потребно је да буде правилно планирана, праћена и прилагођена индивидуалним способностима, циљевима и здравственом стању особе. У том контексту, праћење физичке активности постаје веома важан сегмент здравог начина живота. Праћење омогућава увид у напредак, идентификацију навика, праћење резултата и мотивацију за наставак активности. Данас, захваљујући развоју технологије, праћење активности више не зависи од стручњака или скупе опреме – бројне дигиталне апликације омогућавају корисницима да самостално биљеже своје резултате, мјере тјелесне параметре као што су тежина, BMI (индекс тјелесне масе) и BMR (базални метаболизам), и добијају препоруке на основу својих података. Управо у том правцу развијају се и апликације попут оне описане у овом раду – које имају за циљ да приближе здрав начин живота свима, учине га доступним, праћеним и прилагођеним. Такве апликације омогућавају не само контролу и самопроцјену, већ и подстицање одговорности према сопственом здрављу, што је у основи модерне превентивне медицине.

## 2.2. Потребе и навике корисника у дигиталном добу

Савремено доба карактерише интензивна дигитализација свих аспеката људског живота – од комуникације, образовања и рада, до бриге о сопственом здрављу. Корисници данас живе у брзом, технолошки напредном окружењу, где се информације размјењују у реалном времену, а мобилни уређаји представљају централни алат за обављање свакодневних активности. У таквом окружењу, појављују се и нове потребе – за брзом, доступном и персонализованом подршком, укључујући и област физичке активности и здравља. Повећање свјести о важности здравог начина живота довело је до пораста интересовања за алате који могу помоћи у његовом одржавању. Корисници траже једноставне и ефикасне начине за праћење свог здравственог стања, мјерење напретка и добијање препорука без потребе за сталним одласком код стручњака. Овакве потребе су доприњеле експанзији веб и мобилних апликација намијењених фитнесу, исхрани и менталном здрављу. Модерни корисник очекује да технологија буде прилагођена његовом начину живота – да буде интуитивна, визуелно атрактивна, мобилна и функционална. У контексту фитнес апликација, то подразумијева могућност персонализованог програма вјежбања, евидентирање тјелесних параметара, анализу напретка кроз дијаграме и графиконе, као и мотивационе елементе – подсјетнике, похвале, изазове и циљеве. Такође, важно је да апликација буде прилагођена различитим профилима корисника – како онима који се баве спортом рекреативно, тако и онима који тек улазе у свијет физичке активности. Дигитални корисници уједно цијене и могућност интеракције – било са стручњацима (као што су тренери или нутриционисти), било са другим корисницима. Због тога су све популарније платформе које, осим индивидуалног праћења, омогућавају комуникацију, размјену искустава, савјете и подршку. Оваква заједница корисника доприноси континуитету у физичкој активности, јер оснажује мотивацију и осјећај припадности. Не мање важно, све је израженија потреба за приватношћу и сигурношћу података. Корисници желе да информације о њиховом здравственом стању остану повјерљиве, те су функционалности као што су login/регистрација, персонализовани профил и заштита података постале стандард.

У закључку, навике и потребе корисника у дигиталном добу обликују начин на који се приступа фитнесу и здрављу. Фитнес апликације данас не служе само за биљежење података, већ представљају савремене алате за образовање, мотивацију, самопроцјену и подршку – у складу са очекивањима генерација које расту у дигиталном свијету.

# 3. ASP.NET Core

## 3.1 Основне карактеристике

ASP.NET Core представља модерни, унакрсно-платформски веб оквир отвореног кода који је развила компанија Microsoft као потпуну реконструкцију претходне верзије ASP.NET технологије, са јасном намјером да одговори на све захтјевније потребе савременог веб развоја. Он је дизајниран тако да подржава изградњу свих врста веб апликација – од једноставних веб сајтова и RESTful API-ја до сложених enterprise рјешења и микросервисних архитектура. Једна од најважнијих карактеристика овог оквира је његова унакрсна платформа, што значи да се апликације развијене у ASP.NET Core-у могу извршавати на више оперативних система, укључујући Windows, Linux и macOS. Ово не само да доноси већу флексибилност тимовима у избору развојног окружења, већ утиче и на смањење трошкова инфраструктуре и већу приступачност у продукционом постављању апликација. ASP.NET Core је од самог почетка развијан са фокусом на високе перформансе, што га чини једним од најбржих оквира за веб развој. Перформансе се постижу кроз минималистичку архитектуру, асинхроно програмирање као стандард, брзо компајлирање и ефикасну обраду HTTP захтева. Сама архитектура је модуларна – у апликацију се додају само они пакети који су заиста неопходни, што значајно смањује величину пројекта, унапрјеђује брзину учитавања и смањује потенцијалне безбједносне ризике. Уместо монолитног приступа, ASP.NET Core омогућава програмерима да кроз NuGet пакет менаџер додају само функционалности које су потребне за конкретну апликацију, чиме се добија велика флексибилност. Једна од централних одлика ASP.NET Core-а је подршка за middleware компоненте – мање, независне јединице које обрађују HTTP захтјеве у ланцу. Овај механизам омогућава програмерима да креирају прилагођене логике за логовање, аутентификацију, обраду грешака и многе друге процесе, и то на врло интуитиван начин. Middleware приступ подржава separation of concerns и значајно поједностављује структуру апликације. Додатно, оквир пружа снажну и уграђену подршку за Dependency Injection (DI), што омогућава лако управљање зависностима међу класама. DI је саставни део ASP.NET Core-а и не захтјева никакве спољне библиотеке, што омогућава једноставну имплементацију и лакше писање тестова, као и одржавање читљивог и одвојеног кода. Сигурност је такође приоритет у оквиру ASP.NET Core-а. Оквир подржава све модерне стандарде за заштиту података – укључујући аутентификацију преко Identity система, подршку за JSON Web Token (JWT), као и интеграцију са OAuth2 и OpenID Connect протоколима. Ово је од посебног значаја за апликације које рукују осјетљивим личним подацима корисника, као што је то случај са фитнес апликацијама, где је неопходно осигурати безбједност података о здравственом стању, физичкој активности и личним навикама. Уз то, ASP.NET Core омогућава лаку интеграцију са различитим алатима за праћење, логовање и мониторинг, што додатно доприноси стабилности и поузданости апликације током рада у реалном окружењу. Такође, оквир се савршено уклапа у cloud-native приступе, што значи да је у потпуности прилагођен за рад у окружењима као што су Microsoft Azure, AWS или Google Cloud. Компатибилан је са алатима као што су Docker, Kubernetes и Helm, чиме се обезбјуеђује висока скалабилност и флексибилност у управљању ресурсима. Подршка за CI/CD процесе је такође добро развијена, што омогућава брзо и сигурно постављање нових верзија апликације. Коначно, ASP.NET Core је оквир који се континуирано развија, захваљујући активној заједници отвореног кода и сталној подршци од стране Microsoft-а. Ово значи да програмери имају приступ обимној документацији, великом броју туторијала, као и бројним библиотекама и компонентама које омогућавају убрзан развој функционалних и ефикасних рјешења. У оквиру овог рада, управо је овај оквир изабран као темељ за реализацију фитнес апликације која омогућава пријаву и регистрацију корисника, визуелизацију њихових параметара здравља, праћење физичке активности и персонализоване препоруке, јер ASP.NET Core у сваком аспекту задовољава критеријуме безбједности, перформанси и флексибилности неопходне за овакву врсту софтверског система.

## 3.2 Архитектура и примјена у веб апликацијама

ASP.NET Core је осмишљен као флексибилан и модуларни оквир који омогућава изградњу различитих типова веб апликација. Његова архитектура се темељи на концептима као што су middleware, dependency injection, routing и separation of concerns. Овакав приступ омогућава висок степен контроле над понашањем апликације и лакоћу у управљању комплексним сценаријима, као што су аутентификација, обрада захтјева и повезивање са базом података. Основу сваке ASP.NET Core апликације чини **middleware pipeline**, односно низ компоненти који обрађују HTTP захтеве у низу. Сваки middleware може да модификује захтјев или одговор, прослиједи га даље или прекине обраду. Ово омогућава креирање прилагођене логике као што су логовање, обрада грешака, CORS политика или аутентификација. **Dependency Injection (DI)** је уграђен механизам у ASP.NET Core и представља стандардни начин управљања зависностима. Кроз DI се обезбеђује да се компоненте лако тестирају, конфигуришу и одржавају. У типичној апликацији, контролери, сервиси и репозиторијуми се региструју у DI контејнеру и ASP.NET Core брине о њиховој иницијацији. Архитектура ASP.NET Core такође подржава различите шаблоне пројеката, од којих су најзначајнији **MVC (Model-View-Controller)**, **Web API** и **Razor Pages**. У оквиру овог рада, примјењени су **ASP.NET Core MVC** и **Web API**, при чему је Web API коришћен за комуникацију са frontend-ом React апликације, а MVC приступ за организацију backend логике. Такође, безбједност апликације реализована је коришћењем **ASP.NET Core Identity**, система за управљање корисницима, улогама и аутентификацијом. За рад са базом података коришћен је **Entity Framework Core**, ORM (Object-Relational Mapping) оквир који омогућава рад са базом користећи .NET објекте.

Сљедећа потпоглавља дају детаљан преглед наведених компоненти и њихову примјену у оквиру фитнес апликације.

### 3.2.1 ASP.NET Core MVC

ASP.NET Core MVC представља један од основних архитектонских модела подржаних у оквиру ASP.NET Core оквира, заснован на добро познатом и широко примјењивом шаблону **Model-View-Controller (MVC)**. Овај архитектонски образац уводи јасну подјелу одговорности унутар апликације, што омогућава бољу организованост кода, лакше тестирање, одржавање и проширивост апликације током њеног животног циклуса. Основна идеја иза овог шаблона је раздвајање апликационе логике (контролер), пословне логике и података (модел) те приказа (поглед), чиме се омогућава паралелан развој и независно тестирање појединачних компоненти. **Модел (Model)** представља логички слој апликације који дефинише структуру података и имплементира пословну логику. У контексту фитнес апликације, модели попут User, Exercise, Workout, MealPlan, ProgressLog и други описују ентитете у домену праћења здравог живота. Ови модели су директно повезани са релационом базом података преко Entity Framework Core (EF Core) ORM алата. EF Core омогућава аутоматску синхронизацију модела са базом података, што значајно олакшава рад програмера, јер се табеле и везе међу њима дефинишу кроз C# класе. Модел такође може садржати атрибуте за валидацију података ([Required], [StringLength], [Range] итд.), што омогућава аутоматску провјеру исправности података прије него што се унесу у базу. **Контролер (Controller)** служи као посредник између модела и погледа, односно, он обрађује HTTP захтјеве, комуницира са моделима и просљеђује резултате ка одговарајућим погледима или API одговорима. Контролери су C# класе које наслиједе Controller или ControllerBase, и садрже тзв. *action методе*, које се мапирају на руте дефинисане у апликацији. На пример, UserController обрађује логин и регистрацију корисника, WorkoutController управља креирањем и приказом тренинга, а ProgressController биљежи и приказује напредак корисника. Ове контролер класе директно сарађују са сервисима који инкапсулирају пословну логику, чиме се постиже чистији и лакше тестирани код. **Поглед (View)** је визуелни део апликације који приказује податке кориснику. У традиционалном ASP.NET Core MVC приступу, погледи се пишу у Razor синтакси и користе .cshtml датотеке. Ипак, у овом раду frontend је имплементиран помоћу React библиотеке, што значи да погледи нису Razor датотеке већ компоненте у JavaScript-у које комуницирају са backend-ом путем HTTP позива. И поред тога, логика контроле података, приступа моделима и рутинга остаје у духу MVC принципа, иако се сам приказ података измјешта на frontend дио апликације. Једна од кључних компоненти ASP.NET Core MVC је **routing механизам**, који дефинише на који начин се HTTP захтјеви мапирају на методе унутар контролера. Routing се конфигурише унутар Program.cs датотеке помоћу метода као што су MapControllerRoute, где се дефинише структура URL-а и подешавају параметри као што су име контролера, акције и ID вриједности. Ова флексибилност омогућава једноставно дефинисање чистих, читљивих и SEO-прилагођених URL адреса. ASP.NET Core MVC такође пружа уграђену подршку за **Model Binding**, што значи да подаци из HTTP захтјева (као што су JSON објекти, формe, query string параметри и сл.) могу бити аутоматски мапирани у објекте дефинисане унутар апликације. Уз то, подржана је и **Model Validation**, што омогућава валидацију тих објеката без писања додатне логике. Ово је посебно корисно при раду са обрасцима као што су регистрација, пријава, унос новог тренинга или исхране, где систем може аутоматски обавестити корисника уколико је неки податак неважећи или недостаје. Још једна важна карактеристика овог архитектонског модела је могућност примјене **Dependency Injection (DI)** механизма, који је уграђен у саму срж ASP.NET Core оквира. DI омогућава лако убризгавање зависности као што су сервиси, репозиторијуми или логички слојеви у контролере, чиме се добија модуларнији, тестабилнији и лакше одржив код. На пример, IUserService или IWorkoutService се могу регистровати у Program.cs и касније користити у контролерима без потребе за мануелним инстанцирањем објеката. У случају фитнес апликације, ASP.NET Core MVC модел омогућио је да се логика као што су регистрација и пријава корисника, обрада уноса исхране и физичке активности, биљежење напретка и генерисање извештаја организује на структуриран начин. Свака функционална цјелина је смјештена у засебан контролер, модели су пажљиво дефинисани тако да одговарају захтјевима домене, а валидација и руковање грешкама омогућавају безбједан и поуздан рад апликације. Захваљујући свим овим компонентама, ASP.NET Core MVC се показао као изузетно моћан и поуздан архитектонски оквир за развој веб апликација, који се једнако добро прилагођава традиционалним приступима са Razor View-има, као и савременим SPA (Single Page Application) рјешењима као што је React, који је коришћен у овом пројекту.

### 3.2.2 ASP.NET Core Web API

ASP.NET Core Web API представља оквир за изградњу HTTP сервиса који клијентске апликације (попут веб, мобилних или десктоп апликација) могу да користе за размјену података путем интернета. У контексту савремених веб апликација, посебно у SPA (Single Page Application) архитектури, као што је овај пројекат у коме је frontend реализован у React-у, Web API служи као backend сервис који омогућава комуникацију и размјену података између корисника и система. ASP.NET Core Web API користи **REST архитектонски стил** (Representational State Transfer), који подразумијева рад са ресурсима путем стандардних HTTP метода: GET, POST, PUT, DELETE и слично. Сваки ресурс, као што је корисник, тренинг, напредак или план исхране, има своју URL адресу и јасно дефинисану операцију коју клијент може да изврши над тим ресурсом.

* **Структура и рад Web API-ja**

ASP.NET Core Web API омогућава изградњу RESTful сервиса, при чему се комуникација одвија путем HTTP протокола. Контролери који насљеђују класу ControllerBase користе се за дефинисање API ендпоинта и обраду HTTP захтјева. За разлику од MVC архитектуре гдје се користе View-и за приказ података, Web API враћа искључиво податке у формату као што је JSON, што је оптимално за интеракцију са frontend клијентима као што су React или Angular апликације. Свака метода унутар контролера представља одређену руту, која се дефинише преко атрибута као што су [HttpGet], [HttpPost], [HttpPut] и [HttpDelete]. Пријем података врши се помоћу механизама као што су [FromBody], [FromForm] или [FromQuery], у зависности од врсте захтјева и садржаја који се шаље.

**Пример: Регистрација корисника**

У наставку је приказан примјер методе за регистрацију новог корисника. Метод се налази унутар AuthController класе и означен је атрибутом [HttpPost("register")], што значи да обрађује POST захтјеве упућене на руту /api/auth/register.



### Анализа метода:

* **Рута и тип захтјева:** Атрибут [HttpPost("register")] дефинише да метод одговара на POST захтјев за руту /register.
* **Улазни подаци:** Атрибут [FromBody] указује да ће подаци доћи у тијелу HTTP захтјева, у JSON формату, и биће аутоматски мапирани у DTO објекат RegisterDTO.
* **Валидација:** Провјера ModelState.IsValid осигурава да су сви потребни подаци исправно послати. У супротном, враћа се HTTP статус 400 (BadRequest).
* **Обрада:** Успјешна регистрација се обрађује позивом сервисне методе RegisterAsync која садржи пословну логику.
* **Изузеци:** У случају грешке током обраде, она се хвата преко try-catch блока и враћа се детаљна порука у оквиру BadRequest одговора.

Овај примјер илуструје кључне концепте Web API-ја у ASP.NET Core-у: јасну Separation of Concerns архитектуру, рад са DTO објектима, коришћење сервисног слоја, као и обраду захтјева и одговора у складу са REST принципима.

### 3.2.3 ASP.NET Core Identity

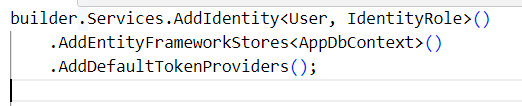
ASP.NET Core Identity је флексибилан и проширив систем за управљање корисницима и њиховом аутентификацијом и ауторизацијом. Он представља оквир (framework) који омогућава креирање, пријављивање, одјављивање, верификацију е-поште, ресетовање лозинке и друге функционалности везане за безбједност корисничког налога у веб апликацијама. У оквиру ASP.NET Core пројеката, Identity се може интегрисати уз минималан напор и конфигурисати према потребама апликације. Он подразумијевано користи Entity Framework Core за приступ подацима, што значи да се све корисничке информације чувају у бази података у облику табела као што су AspNetUsers, AspNetRoles, AspNetUserRoles, итд.

#### Основне компоненте Identity система:

* **User:** Представља ентитет корисника. Наслеђује се из класе IdentityUser, а може се проширити додавањем сопствених поља (нпр. Име, Презиме, Пол).
* **Role:** Представља улоге (нпр. Администратор, Корисник) које се могу додијелити корисницима ради контроле приступа.
* **UserManager:** Сервис који пружа методе за рад са корисницима (креирање, ажурирање, брисање, промјена лозинке, верификација итд.).
* **SignInManager:** Задужен за пријављивање корисника, провјеру лозинке, рад са колачићима (cookies) и безбједносним токенима.

#### Интеграција у пројекат

У Program.cs (или Startup.cs у старијим верзијама) настаје конфигурација сервиса:



Овдjе User представља класу која насљеђује IdentityUser, а ApplicationDbContext контекст базе података који такође насљеђује IdentityDbContext<User>.

#### Примјер коришћења ASP.NET Core Identity у регистрацији

У сљедећем примјеру се користи Identity за регистрацију новог корисника преко UserManager сервиса. DTO објекат RegisterDTO садржи потребне податке као што су корисничко име, е-пошта и лозинка.