# Projektuppgift

*Webbutveckling med .NET*

**Lyrics Quiz**

**Elias Eriksson**

Mittuniversitetet
MID SWEDEN UNIVERSITY

**Campus Härnösand** Universitetsbacken 1, SE-871 88. **Campus Sundsvall** Holmgatan 10, SE-851 70 Sundsvall.
**Campus Östersund** Kunskapens väg 8, SE-831 25 Östersund.
Phone: +46 (0)771 97 50 00, Fax: +46 (0)771 97 50 01.

**MITTUNIVERSITETET**
**Avdelningen för informationssystem och -teknologi**

**Författare:** Elias Eriksson, eler2006@student.miun.se
**Utbildningsprogram:** Webbutveckling, 120 hp
**Huvudområde:** Datateknik
**Termin, år: VT**, 2022

# Sammanfattning

En webbplats för musik quiz byggs med hjälp av webb ramverket ASP.NET MVC. För projektets databas hantering används entity-framework och identity-framework används för att underlätta hanteringen av registrerade användare.

Inloggade användare kommer själva att kunna lägga till en quiz på sidan och alla användare oavsett registrering kommer kunna utföra en quiz.

En Quiz startar med att användaren ser hela sång texten censurerad och målet är att lyckas gissa alla unika ord i en texten. För vardera ord som användaren lyckas gissa rätt så ändras de från att vara censurerade till visade i låt texten.

# Innehållsförteckning

# Terminologi

## Akronymer/Förkortningar

| | |
|---|---|
| ORM | Object-relational mapper |
| MVC | Model View Controller |

# 1 Introduktion

## 1.1 Bakgrund och problemmotivering

I detta projekt lanseras en ny plattform Lyrics Quiz för att quizza sång texter. Användare har möjligheten att ladda upp texter som andra användare sedan kan visa hur bra de har texten memorerad.

När användaren går in på en sång ska de se en censurerad version av låt texten och för varje ord det gissar rätt så visas de orden. Varje unikt ord behöver bara gissas en gång och användarens poäng beräknas som en kvot mellan mängden unika ogissade ord och mängden unika ord i texten. Denna beräkning sker när användaren antingen har gissat alla ord i texten eller när en sångs specificerade tidsgräns nås.

## 1.2 Detaljerad problemformulering

Webbplatsen kommer att byggas med ASP.NET 6 MVC, entity- och identity-framework med en databaskoppling till en postgres databas. Den interaktiva delen där användaren får gissa på ord kommer att byggas i Javascript. För att skicka data till Javascript applikationen kommer endpoints att byggas in i .NET applikationen som sedan kan efterfrågas med Fetch API.

Projektet kommer att publiceras tillgängligt på webben via en Ubuntu server bakom NGINX som reverse proxy server. Applikationens process kommer att startas med hjälp av systemd.

# 2 Teori

## 2.1 ASP.NET

ASP.NET är ett webbapplikations ramverk utvecklat utav Microsoft som gör det lätt att hantera webbtrafik med språk som stöds av .NET plattformen. I detta projekt används C# som språk. ASP.NET utökar den vanliga HTML standarden med Razor. Razor öppnar upp för möjligheten att använda C# i html på ett sätt som liknar PHP för att skapa dynamiska webbsidor. [1] För att börja skriva Razor öppnas antingen ett Razor kod block med @{ } eller så kan en ett specifikt uttryck skrivas ut i HTML med @(uttryck). Dessa Razor filer använder sig utav filändelsen .cshtml istället för .html. [2] ASP.NET ger även möjligheten att välja på några olika strukturer för att utveckla webbapplikationer. Antingen så kan ett projekt startas helt tomt och allt byggs från grunden men det finns även möjlighet att starta ett projekt som bland annat designmönstret MVC eller Razor pages. [3][4] I detta projekt kommer MVC mönstret att användas.

## 2.2 MVC

MVC (Model View Controller) är ett designmönster där applikationens data representeras av en modell. I detta fall när C# och en relationsdatabas används komme en klass användas för att representera en tabell där varje property klassen motsvarar en kolumn i en tabell eller en relation till en annan tabell. Dessa klasser blir applikationens modeller. [3][5]

Vyer används för att visa en viss data för användare. Om användaren som exempel går in på en sida för att se sin användarprofil kommer Vyn ta emot användardata och sedan representera det som HTML. Vyn innehåller alltså instruktioner om hur data visas för användaren. [3][5]

Kontrollern är den delen av applikationen som tar in någon form av input och dirigerar användaren till rätt del av applikationen. I en webbapplikation är det kontrollen som tar det anrop som görs som input och sedan avgör vilken vy användaren ska till. Hämtar den data som vyn kräver från modellen och ger vidare data till vyn.[3][5]

## 2.3 ORM

ORM (Object Relational Mapper) är en teknik som automatiserar processen av att hämta data från en data källa och sedan formatera det till ett objekt eller spara data från ett objekt till en data källa. Data källan kan variera men är väldigt ofta en databas. Genom att använda en ORM slipper utvecklaren själv implementera metoder för att hämta / spara data från / till databasen och formatera den, istället kan data hämtas / sparas via den den API given av en specifik ORM. [6]

Vissa ORM ramverk har även stöd för att översätta klasser till tabeller för en given databas och utföra migrationer mellan när datastrukturen ändrar sig under ut-

vecklings processen. Ett ORM som har stöd för migrationer är bland annat entity-framework som tillhör .NET plattformen samt SQLAlchemy. [7][8]

## 2.4 Entity-framework

Entity-framework är en ORM utvecklad av Microsoft och är en del av .NET plattformen men installeras separat. Genom att använda entity-framework underlättas arbetet med att sätta upp databastabeller, göra queries samt modifiera tabellerna när applikationens data modell ändras. [7]

Entity-framework är databas agnostiskt och har officiellt stöd för Microsofts egna SQLServer, SQLite, InMemory och Cosmos men gör det möjligt för andra utvecklare att utveckla utökat stöd så Microsoft länkar själva till andra paket som ger stöd till bland annat MySQL och PostgreSQL. Eftersom Entity-framework är databas agnostiskt är det lätt att byta från en till en annan databas. [9]

För att Skapa en tabell så skapas bara en ny klass. Vid utformningen av klassen så kommer viss namngivning på properties påverka hur entity-framework behandlar dom. Ett property som endast heter Id kommer att användas som primär nyckel i databastabeller. Om ett fällts typ är en annan klass Foo kommer entity-framework tolka det som en relation till Foo. För att representera en ”many” relation ändras typen från att vara Foo till List<Foo>. Dessa properties som utrycker relationer kallas för ”navigation properties”. För att specificera nyckeln till relationen kan ett property FooId specificeras och entity-framework kommer att automatiskt förstå att FooId är id:t till Foo. [7][10]

När modellerna för applikationen är skapade kan en migration göras med kommandot `dotnet ef migrations add <migrationName>` och sedan appliceras med kommandot `dotnet ef database update`.[11]

## 2.5 Identity-framework

Identity-framework är ett ramverk som hjälper till att hantera användares inloggning samt användarprofildata. Identity-framework bygger på entity-framework och ger tillgång till en klass IdentityUser som kan användas för att relatera annan data i applikationen till en viss användare. Genom att förlänga en modell med IdentityUser och sedan specificera navigation properties till olika modeller så kommer det skapas relationer mellan modellen och användaren i databas tabellerna. [12][13]

# 3 Metod

Projektet kommer att byggas på .NET plattformen med C# som språk och ASP kommer användas som ramverk. MVC är det designmönster från ASP som kommer att användas för att bygga upp sidan.

Jetbrains Rider kommer att användas som utvecklings miljö.

Applikationen kommer vara databasdriven och PostgreSQL kommer att användas som databas. Vilken annan relationsdatabas som entity-framework har stöd för skulle också fungera till projektet som exempelvis MySQL eller SQLite.

För att hantera databaskopplingen och formateringen av databas data till objekt i C# så kommer entity-framework användas som ORM. Detta gör det möjligt att definiera databas tabellerna som klasser som sedan entity-framework översätter till tabeller.

För att hantera användares inloggningar kommer identity-framework att användas.

Den byggda hemsidan kommer att validera enligt W3Cs HTML validator och färgvalet på hemsidan har som mål att uppfylla WCAG AA för kontrast av läsbar text. För att kontrollera kontrasterna används https://webaim.org/resources/contrastchecker/.

Viss data i projektet kommer att presenteras som tabeller. För att göra tabellerna responsiva och tillgängliga kommer CSS funktionen attr() att utnyttjas tillsammans med CSS regeln content. Genom att skriva ut kolumnernas rubriker som data- attribut kan kolumnens rubrik läsas in i CSS och appliceras framför innehållet i ett visst <td> element med CSS selectorn :before. Tabellhuvudet kan därför göras 0 px hög och varje tabellrad görs om till en liten box på mindre skärmar. Detta kommer att låta tabellen vara tillgänglig då tabellhuvudet fortfarande finns kvar i HTML koden och den extra texten som läggs till för andra användare räknas som styling och kommer inte påverka tillgängligheten.

# 4 Konstruktion

## 4.1 Uppsättning av projektet

Projektet startas som ett ASP.NET MVC projekt med individuell autentisering. Utöver de paket som installeras automatiskt av .NET efter detta installeras paketen:

- Microsoft.EntityFrameworkCore.Design
- Microsoft.VisualStudio.Web.CodeGeneration.Design
- Npgsql.EntityFrameworkCore.PostgreSQL
- Newtonsoft.Json
- Microsoft.EntityFrameworkCore.SqlServer
- Microsoft.VisualStudio.Web.CodeGeneration.Design
- Microsoft.AspNetCore.Mvc.NewtonsoftJson

## 4.2 Databasanslutning

För att upprätta en databasanslutning skapas filen QuizContext.cs i mappen Data/. QuizContext ärver från klassen IdentityDbContext och en konstruktor som tar DbContextOptions som parameter läggs till i klassen och option parametern skickas ner till super klassens konstruktor.

För att ansluta till PostgreSQL databasen överskrivs OnConfiguring(DbContextOptionBuilder) metoden i QuizContext över till att ladda in inloggningsuppgifter till databasen från en JSON fil och sedan läggs uppgifterna till genom att använda UseNpgsql metoden på DbContextOptionBuilder för att lägga till anslutningsuppgifterna som en textsträng på formatet: ”Host=host;Username=username;Password=passowrd;Database=database;” värderna för host, username, password och database läses in från den laddade JSON filen.

JSON filen namnges .credentials.json och läggs i projektets rot katalog och innehåller ett objekt med attributen Host, Role, Pwd och Db.

En ny användare skapas i postgres databasen med `create role username with login encrypted password ’pwd’;` samt en databas med `create database db with owner = username encoding = ’utf-8’;`. Uppgifterna username, pwd och db läggs in i filen .credentials.json. host läggs också in i filen och specificeras till localhost.

QuizContext läggs sedan till i Program.cs med
`builder.Services.AddDbContext<QuizContext>();`

## 4.3 Identity-framework scaffolding

Ett urval av identity-frameworks sidor scaffoldas:

- Manage/ChangePassword
- Manage/DeletePersonalData
- Manage/PersonalData
- Manage/_Layout
- Manage/_ManageNav
- Manage/_StatusMessage
- Login
- Logout
- Register

Sidorna scaffoldas med QuizContext och använder SQLServer inte SQLite. Efter att sidorna scaffoldas lägger .NET till extra kod i Program.cs som försöker ansluta till en SQLServer databas. Denna kod måste raderas då detta projekt använder PostgreSQL.

## 4.4 Modells

Alla projektets modeller läggs i en mapp Models som ligger i roten av projektet.

För att relatera en Quiz till en användare skapas en klass ApplicationUser som ärver från IdentityUser. En användare kan skapa flera quizzes så navigations propertyn av typen List<Quiz>? Läggs till på ApplicationUser och klassen Quiz skapas.

Quiz klassen relaterar till en användare och ges navigations propertyn ApplicationUser? och användar id:t med typen string? vilket är typen av IdentityUsers Id. Den data som lagras i en Quiz kommer att vara ett Id av typen Guid, ett namn av typen string?, låt texten av typen string? och en tidsgräns av typen int. Värdet på tidsgränsen motsvarar tiden i sekunder som användaren har till sitt förfogande att lösa quizzen. Varje quiz kommer också att ha flera resultat så en relations property av typen List<QuizResult>? läggs också till och klassen QuizResult skapas.

QuizResult klassen Relaterar till en Quiz och ges en navigations property av typen Quiz? och quiz id:t med typen Guid. QuizResults egna Id sätts till typen Guid, resultatet till typen double och tiden till typen int. I resultatet kommer en kvot

mellan mängden unika ogissade ord och mängden unika ord i sång texten lagras och tiden kommer innehålla antalet sekunder som användaren hade kvar när resultatet skickades in. För att spara de ord som saknades när en användare skickade in resultatet skapas en relation till MissingWord med navigations propertyn List<MissingWords>? och en klass MissingWord skapas.

MissingWord relaterar till en Quiz och ges en navigations property av typen QuizResult? och QuizResult id:t med typen Guid. MissingWords egna Id sätts till typen int och ord datan till typen string?.

## 4.5 Migrations

Med modellerna klara läggs dom till som DbSet i QuizContext. När alla modeller är tillagda kan en migration köras med `dotnet ef migrations add migrationsName` för att skapa migrationsfiler som sedan appliceras på databasen med `dotnet ef database update` vilket skapar alla tabeller.

## 4.6 Kontrollers

Alla kontrollers som skapas läggs i en mapp Controllers/ som ligger i roten av projektet.

### 4.6.1 RootController

En kontroller RootController skapas för att hantera standalone sidorna `How to play?` samt startsidan. RootControllern har en metod Index() som serverar startsidans vy och en metod Instructions() som serverar instructions sidan `How to play? `s vy.

### 4.6.2 QuizController

Kontrollern QuizController scaffoldas som en `MVC Controller with Views using entity framework` där namnet specificeras till QuizController, modellen till Quiz och databas kontexten till QuizContext.

Den genererade QuizContext klassen modifieras till att innehålla en klass QuizWithAvg som ärver från Quiz modellen. Detta för att lägga till en extra property av typen double som kommer innehålla medelvärdet på alla resultat relaterade till quizen i fråga som kommer beräknas i en query.

En metod string LyricsFormatter(string) läggs också till som returnerar en strängen den tar som input fast alla skiljetecken, text inom parenteser och hakparenteser är borttagna. Tecknet ’ som används för engelska sammandragningar tas inte bort. Skiljetecken ersätts med en tom sträng där regex mönstret `@"(?:\([^)]+\))|\.|!|\?|,|-|;|(?:\[[^)]+\])|"""` matchar. Textens sekvenser med whitespace ersätts till att vara ett enda mellanrum där mönstret `@"[\s\n]+"` matchar.

QuizControllerns scaffoldade Index() metod modifieras till att kunna utföra sökningar på det som finns i databasen. Detta görs genom att först lägga till parametern `string? search` till Index() metoden. En variabel definieras till typen IQueryable<Models.Quiz> och initialiseras olika beroende på öm söksträngen är tom /

null eller om den innehåller en sökning. Om det finns en sökning initieras en query på Quiz tabellen som kontrollerar om quizzens namn i lowercase innehåller sökningssträngen i lowercase. Om ingen sökning har utförts så initieras bara en sökning på Quiz tabellen. För att beräkna medelvärdet på quiz resultaten görs en mellanliggande query över QuizResult modellen med en group by på quiz id:t och medelvärdet över resultat kolumnen beräknas. Queryn utökas med mellansteget och samman med en join över quiz id:t och resultaten sparas i QuizWithAvg objekt. För att få med de objekt som inte har blivit avklarade än så utförs en query där utan mellanstegets join och istället filtreras QuizResultaten med resultat bort. Slutprodukten blir två listor med QuizWithAvg som slås samman och sorteras på quiz namnet. Denna lista blir data till Vyn.

QuizControllerns Create(Models.Quiz) metod modifieras lite. UserId plockas bort från post datan så Models.Quiz UserId initieras i med Id:t på den nuvarande inloggade användaren och sång texten rensas från skiljetecken. Kontrollerns modelstate rensas och valideras om. Resten av metoden är oändrad.

Edit(Models.Quiz) modifieras på samma sätt som Create(Models.Quiz). Modelstate återställs och UserId blir satt till nuvarande inloggade användarens och sång texten rensas från skiljetecken och modellen blir om validerad.

Resterande scaffoldade metoder ändras inte från deras original utseende. En extra metod Results(Guid) läggs dock till för att kunna visa Quiz resultaten. I Results görs en query över QuizResult för att hitta den givna Guid:t och navigations propertyn Quiz blir inkluderad.

### 4.6.3 ApiController

För att kunna ta del av datan i databasen med Javascript skapas två endpoints som kan efterfrågas med Javascripts Fetch API.

Controllern ApiController skapas och tilldelas en metod GetQuiz(string) som hämtar en Quiz med det givna id:t i strängen.

ApiControllern tilldelas även metoden PostResult(double, string, int, string) som förväntar sig post data om resultatet, quiz id:t, användarens återstående tid och orden som användaren hade kvar att gissa. Ett QuizResult objekt skapas med resultatet, quiz id:t och den återstående tiden. Detta objekt sparas sedan till databasen. När QuizResultatet finns sparat i databasen skapas MissingWord objekt av de återstående orden och resultat id:t och sparas sedan till databasen.

## 4.7 Program.cs

Med alla kontrollers klara görs de sista ändringarna i Program.cs. Eftersom projektet kommer ligga bakom en reverse proxy definieras en variabel rootUrl som tilldelas värdet ”aspdotnet/moment4” som kommer att agera som rot. För att definiera rootUrl till som rot används UsePathBase(string) och tilldelas rot urlen prefixad med /. Statiska filer konfigureras till att servas från /static med UseStaticFiles(string). MapControllerRouter används för att sätta upp URLerna till kontrol-

lerna och deras actions och mönstret som används är ”{controller}/{action}/{id?}” med standard kontrollen Root och standard action Index.

ApiControllern konfigureras till att inte använda det givna URL mönstret i Program.cs utan använder istället /api/quiz/ som dess action GetQuiz utökar med {id} och PostResult utökar med result/.

## 4.8 Vyer

När projektet först skapades tillkommer och efter scaffolding så har det lagts till vyer för olika undersidor samt att CSS ramverket bootstrap tillkommer. Bootstrap raderas men de andra Javascript biblioteken lämnas kvar då de används för form validering.

Shared/_Layout.cshtml filen är en av filerna som tillkommer av att starta ett MVC projekt och är basen för alla vyer i applikationen. I denna _Layout.cshtml fil finns en del referenser till bootstrap och dessa filer tas bort. Ett <base> element läggs även till i <head> med samma url aspdotnet/moment4 som vart definierad i Program.cs och alla länkar görs om till att vara relativa så att de använder urlen specificerad i <base> elementet.

I <head> skapas även ett Razor kod block för att läsa ut Javascript fil sökvägar ifrån ViewData för att kunna lägga till script taggar i <head> med typ module.

Menyn i _Layout.cshtml ändras från standarden och ändras till att länka till RootControllerns Index action samt Instructions action och QuizControllerns Index action.

För att läsa in bilder som logotypen så skapas en partial där SVG koden läggs in. CSS klasser läggs sedan till på SVG koden för att lätt kunna byta färg på bilderna vid behov via CSS.

### 4.8.1 Identity vyer

När identity vyerna blev scaffoldade skapades vyer för de sidor som valdes. _ManageNav.cshtml filen som skapas innehåller en undermeny för alla undersidor av identity-framework. Eftersom att projektet inte har stöd för mejlutskick och inloggning via andra tjänster så är en del av sidorna inte så användbara och länkarna till dessa undersidor tas bort. De undersidor som lämnas kvar är sidorna för att ändra lösenord samt sidorna för att hantera personlig data.

### 4.8.2 Kontroller Vyer

RootControllerns Index vy ändras till att innehålla allmän information om sidan och till dess Information vy så läggs det till beskrivningar för hur sidan används.

QuizControllerns Index vy ändras från att använda Quiz modellen till att använda QuizWithAvg klassen istället så att medelvärdet kan läggas till i den tabell som scaffoldas fram. Om det inte finns ett medelvärde för en viss quiz skrivs ”Not Completed” ut istället.

Varje <td> element i tabellen får också ett data-label attribut som innehåller <td> elementets rubrik. Detta läses sedan ut med CSS attr() funktion och skrivs ut med CSS content för att göra tabellen responsiv.

För att kunna utföra sökningar på Index vyn så läggs ett <form> med action get till. Formuläret har ett text fält med name attributet search som matchar Index actions parameter namn. Samt att det har en submitknapp.

Tabellens navigation till Vyn Details, Edit och Delete ändras så att namnet på quizzen länkar till Details och Edit samt Delete länkarna flyttas också till Details vyn.

Details vyn är den sida där användare utför quizzen så den ändras från att innehålla en lista av hur quizen ser ut till att innehålla ett form och quizens låt text fast censurerad. Sidan har också länkar till Edit och Delete vyerna om användaren är inloggad samt en länk tillbaka till Index vyn. Quizzens namn visas och det tidigare nämnda formuläret. Information som hur många ord som användaren har kvar att gissa samt tiden som är kvar visas också.

Den Javascript som behövs läggs till med ViewData så att den laddas in i <head> från _Layout.cshtml. När Javascripten har laddats kommer ett fetch anrop att göras till /api/quiz/id som motsvarar ApiControllerns GetQuiz action. Informationen som kommer tillbaka används för att bygga upp den censurerade låt texten med span taggar som får längden av ett vist ord i enheten ch i bredd för att ge en uppskattning på hur brett ordet ska vara. Tiden och hur många unika ord användaren har kvar att gissa skrivs också ut.

Tiden kommer att börja ticka så fort användaren gör sin första gissning och slutar när användaren trycker på knappen ”Submit Quiz” eller att tiden går ut. När quizzen är slut kommer ett fetch anrop till ApiControllerns PostResult action att göras där uppgifter som hur många procent som va avklarat, vilket quizzens id är, tiden som va kvar och vilka ord som saknades. Efter att fetch anropen har fått en respons från servern kommer användaren att bli omdirigerad till QuizControllerns Result action. Så användaren kan se sitt resultat.

Resultat Vyn visar användaren hela låt texten där varje ord som användaren gissat rätt på har en grön bakgrundsfärg och alla ord som användaren har missat har en röd bakgrundsfärg. Användaren kan också se andelen unika ord som den gissat rätt på. Det finns även navigering tillbaka till quizzen de just gjorde och en länk tillbaka till listan med quizzes.

I vyerna Create och Edit så tas det input fältet som specificerar användare id bort då det tas ifrån den inloggade användaren i QuizControllerns Create och Edit actions. Delete vyn är funktionellt orörd från hur den scaffoldades.

## 4.9 Host

För att hosta projektet klonas projektets git repository över till ubuntuservern och projektet byggs med `dotnet publish -c Release -o /destination/directory/` och en systemd unit fil ExecStart komandot sätts till `/destination/directory/Quiz –urls

”http://localhost:8088”`, unit filen läggs till i /etc/systemd/system/ mappen och systemd daemonen reloadas med `sudo systemctl daemon-reload` och unit filen ställs in till att starta vid system startup med `sudo systemctl enable unitfile.service` samt att den startas nu med `sudo systemctl start unitfile.service`. NGINX konfigureras till att skicka requests från urlen /aspdotnet/moment4/ till http://127.0.0.1:8088. Webbplatsen blir nu tillgänglig på urlen https://miun.eliaseriksson.io/aspdotnet/moment4/

# 5 Resultat

Webbplatsen Lyrics Quiz är skapad och finns publikt tillgänglig på urlen https://miun.eliaseriksson.io/aspdotnet/moment4/ och är hostad på en personlig Ubuntu server och NGINX används som reverse proxy server för att skicka requests till /aspdotnet/moment4/ vidare till applikationen. Applikationen är skapas med platformen .NET 6 med C# som språk och ASP som ramverk.

Webbplatsen tar hjälp av ramverken entity-framework och identity-framework för att hantera databaskopplingar och registrerade användare. Entity-framework kopplas upp till en PostgreSQL databas som körs på samma Ubuntu server som hostar webbapplikationen. För att entity-framework ska kunna ansluta till PostgreSQL används paketet Npgsql.EntityFrameworkCore.PostreSQL som hämtats från nuget.

Användare kan lägga till egna quizzes på hemsidan om de är inloggade och det finns instruktioner på sidan som beskriver hur det går till. När en quiz är tillagd kan vem som helst gå in på sidan och navigera till en quiz för att utföra den. När en quiz är avklarad kommer blir användaren omdirigerad för att kunna se sitt resultat.

# 6 Slutsatser

Den slutgiltiga sidan fungerar mycket bra och är den slutprodukt som va förväntad.

Under utvecklingsprocessen har det varit en del frågetecken till hur ASP hanterar vissa saker. Som exempel slutade användare kännas igen som inloggade vilket löstes med att skapa om sidan och lägga till kod successivt. Den kod som orsakade problemet identifierades aldrig.

Watchern som är inkluderad i dotnet är inte heller världens mest pålitliga verktyg och känner ibland inte av att kod har ändrats så den måste startas om. Detta skapar en del förvirring när kod som läggs till förväntas köras men som inte gör det. Queryn i QuizController Index action är speciellt känslig av någon anledning vilket har lett till en del krascher och errors som löses genom att starta om watchern.

MVC mönstret har varit väldigt smidigt att jobba med och tycker om strukturen som används.

Alltid så finns det något som kan förbättras och det finns det såklart på denna sida också. Det skulle möjligen vara önskvärt att flytta mer beräkningar till backenden och mer strikta regulationer på olika inputs.

# Källförteckning

[1] Microsoft ”What is ASP.NET” https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet

[2] Microsoft ”Razor syntax referens for ASP.NET Core” https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-6.0

[3] Microsoft ”ASP.NET MVC Pattern” https://dotnet.microsoft.com/en-us/apps/aspnet/mvc

[4] Microsoft ”Introduction to Razor Pages in ASP.NET Core” https://docs.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-6.0&tabs=visual-studio

[5] Geeksforgeeks ”MVC Design Pattern” https://www.geeksforgeeks.org/mvc-design-pattern/

[6] Microsoft ”Entity Framework Terminology” https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/terminology

[7] Microsoft ”Entity Framework overview” https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/overview

[8] SQLAlchemy ”The python SQL Toolkit and Object Relationl Mapper” https://www.sqlalchemy.org/

[9] Microsoft ”Database Providers” https://docs.microsoft.com/en-us/ef/core/providers/?tabs=dotnet-core-cli

[10] Microsoft ”Relationships” https://docs.microsoft.com/en-us/ef/core/modeling/relationships?tabs=fluent-api%2Cfluent-api-simple-key%2Csimple-key

[11] Microsoft ”Migrations Overview” https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli

[12] Microsoft ”Introduction to Identity on ASP.NET Core” https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-6.0&tabs=visual-studio

[13] Microsoft ”Identity model customization in ASP.NET Core”
https://docs.microsoft.com/en-us/aspnet/core/security/authentication/customize-identity-model?view=aspnetcore-6.0