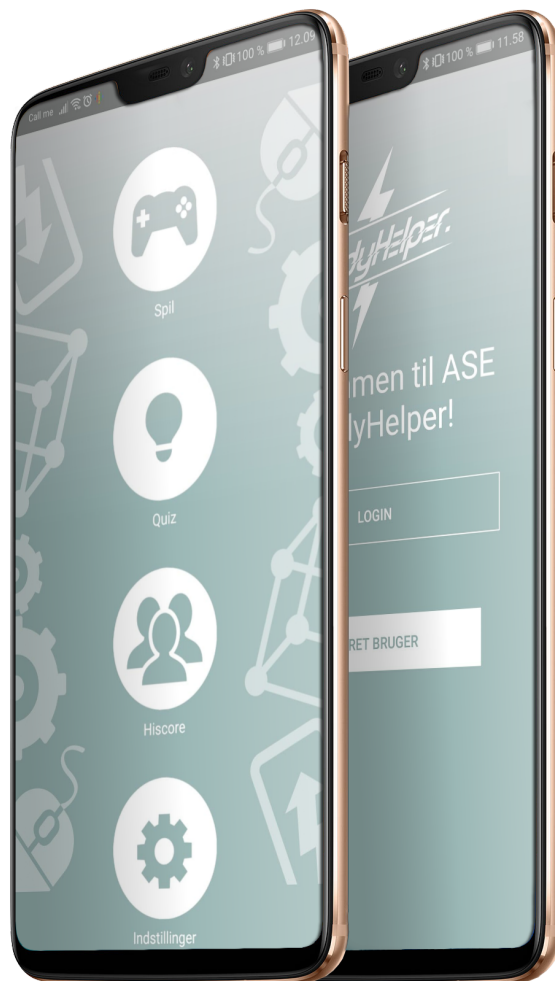


ASE StudyHelper Bilag



I4PRJ

GRUPPE 8

AARHUS UNIVERSITET

DEN 19. DECEMBER 2018



Titel: ASE StudyHelper

Projekt: I4PRJ

Projektperiode: Sep 2018 - Dec 2018

Projektgruppe: Gruppe 8

Deltagere:

Viktor Søndergaard, studentID. 201610466
Tobias Damm-Henrichsen, studentID. 201611660
Simon Møller Hindkær, studentID. 201609970
Christoffer Kjellerup Jakobsen, studentID. 201610457
Marius Linding Møller, studentID. 201610460
Kasper Gnutzmann Andersen, studentID. 201607263

Vejledere:

Henrik Daniel Kjeldsen

Anslag: ca. 110.000

Sidetæl: 83

Afsluttet 19-12-2018

Indholdsfortegnelse

Kapitel 1 Indledning	1
1.1 Problembeskrivelsen	1
1.2 Projektbeskrivelse	1
1.3 Systembeskrivelse	1
1.4 Ordliste	2
1.5 Produktmæssige hovedansvarsområder	2
Kapitel 2 Kravspecifikation	3
2.1 Aktør Diagram	3
2.2 User stories	4
Kapitel 3 Analyse	5
3.1 Mobil GUI	5
3.1.1 Valg af framework	5
3.1.2 Valg af design patterns	6
3.2 Valg af databaser	7
3.2.1 RESTful design	7
Kapitel 4 Arkitektur	8
4.1 Overordnet domænemodel for systemet	8
4.2 Sekvensdiagram med udgangspunkt i “user story 2 - Login”	8
Kapitel 5 Design	10
5.1 Mobil GUI	10
5.1.1 MVVM	10
5.1.2 User Interface	11
5.2 Quiz feature	24
5.2.1 MVVM anvendt på Quiz feature	24
5.2.2 MVC (Strukturering af databaser)	26
5.2.3 Quiz databasen	27
5.3 Login feature	29
5.3.1 MVVM på Login feature	30
5.3.2 MVC på Login feature	30
Kapitel 6 Implementering	32
6.1 Mobil GUI	32
6.1.1 Metodebeskrivelse for GUI klasserne	32
6.1.2 LoginPage.xaml.cs	32
6.1.3 LoginPage.xaml.cs	32
6.1.4 MainPage.xaml.cs	33
6.1.5 NewUserPage.xaml.cs	34

6.1.6	OmOsPage.xaml.cs	34
6.1.7	QuizConclusionPage.xaml.cs	34
6.1.8	QuizPageDemo.xaml.cs	34
6.1.9	SearchQuizPageSelectCategory.xaml.cs	35
6.1.10	SearchQuizPageSelectQuiz.xaml.cs	35
6.2	Login Feature - GUI	36
6.2.1	NewUserViewModel	36
6.2.2	LoginViewModel.cs	37
6.2.3	ResetHighscoreViewModel	38
6.2.4	LogOutViewModel	38
6.2.5	NewUserBindingModel.cs	38
6.2.6	DeleteUserViewModel	39
6.2.7	ApiServices.cs	39
6.2.8	QuizCategoryScoreBindingModel.cs	41
6.2.9	CategoryScoreModel.cs	43
6.2.10	ChangePasswordBindingModel.cs	43
6.2.11	NewUserBindingModel.cs	44
6.2.12	CategoryScoreModel.cs	44
6.2.13	User.cs	45
6.3	Login Feature - database	46
6.3.1	CategoryScoresController.cs	46
6.4	Quiz Services	48
6.4.1	QuizDBServices.cs	48
6.5	Quiz Feature: QuizModel	49
6.5.1	PropertyChangedModel.cs	49
6.5.2	Option.cs	49
6.5.3	Question.cs	49
6.5.4	Quiz.cs	50
6.6	Quiz Feature: MVVM Implementering	52
6.6.1	QuizViewModel - DataBinding, PropertyChanged Events og Com- mands	52
6.6.2	SearchQuizViewModel	57
6.6.3	QuizConclusionViewModel	61
6.7	Quiz feature - databasen	64
6.7.1	QuizRepository	64
6.7.2	QuizController	66
6.7.3	QuizToAPIController	68
6.7.4	Quiz	69
6.7.5	Question	70
6.7.6	Option	70
6.8	Quiz Feature - Diskussion	71
Kapitel 7 Test		72
7.1	Unittest	72
7.1.1	Unit test - Quiz Feature	72
7.2	Integrationtest	75

7.2.1	Continuous Integration	75
7.3	REST API hjælpeværktøjer	75
Kapitel 8	Accepttest	77
8.1	Beskrivelse	77
Kapitel 9	Udviklingsværktøjer	81
9.1	Software	81
9.2	Web services / apps	81
9.3	NuGet Packages for VS	82
9.4	Hardware	82
Kapitel 10	Litteratur	83

1.1 Problembeskrivelsen

Kender du følelsen? Hundredvis af sider, tusindvis af ord - de bibeltynde sider får selv de mest engagerede studenter til at smide håndklædet i ringen. Mange studerende på IHA har problemer med at få læst op til forelæsningerne, om dette skydes kedeligt materiale, manglende motivation, eller noget helt tredje, vides ikke. Med mange studerende til få undervisere bliver det naturligvis svært at tilgodese hver enkelts behov. Der må da findes en måde at hjælpe sig selv?

1.2 Projektbeskrivelse

Med udgangspunkt i at optimere indlæringen, vil vi forsøge at komme de tykke bøger til livs, ved at tilbyde en alternativ læringsmetode, hvor det høje faglige niveau, stadig bliver opretholdt. Vi vil udvikle en mobilapp, hvis formål er at give brugeren en mere effektiv og spændende indlæring. Mobilappen, ASE StudyHelper, vil som navnet antyder primært fokusere på en indlæringsplatform, hvor den enkelte bruger kan supplere sine mangler og forbedre sig i fag der findes på ASE (Aarhus School of Engineering). Applikationen vil blive gjort sjov, spændende og attraktiv, ved hjælp af forskellige konkurrencer, features, highscores mm.

Appen udvikles i Xamarin, som gør det muligt at inddrage mange faglige aspekter fra samtlige fag, hvor der fokuseres på brugerens oplevelse. Ud fra dette udarbejdes user stories, hvor systemets funktionalitet og krav specificeres.

1.3 Systembeskrivelse

StudyHelper ASE er en android applikation som er tiltænkt til at supplere læringen på Aarhus School of Engineering. Brugeren kan via en login menu oprette ny eller logge ind på en eksisterende bruger. Herefter vil der være adgang til hovedmenu, hvor brugeren kan vælge diverse features. Dette kunne f.eks. være en quizmode, hvor brugeren starter ud med at vælge det emne der er relevant. Herefter vil der komme en række spørgsmål, hvor der svares i form af multiple choice. Brugeren vil blive præsenteret for spørgsmålene, et ad gangen, og til sidst bliver brugeren evalueret og tildelt en pointscore ud fra tiden der er blevet brugt på hvert spørgsmål og antallet af korrekte svar. Der vil også være andre former for spil/quiz/indlæringsfeatures, hvor pointtildelingen selvfølgelig vil være tilpasset til den givne feature. Features som f.eks. quizmode, anvendes til lagring af data, hvor der gives mulighed for følge egen progression, samt udfordring i form af konkurrence med andre, via

highscores. Dette opstilles som sagt i applikationens hovedmenu, der gør tilføjelsen af flere features mulig og hjælper til at gøre systemet skalerbart.

1.4 Ordliste

ASE: Aarhus School of Engineering

IHA: Ingeniørhøjskolen Aarhus Universitet

User: Består af en brugerprofil, der lagrer et username, et password, samt en highscore.

Bruger: Består af en person, der ønsker at få en user. Denne kan blandt andet bruges til login og Highscore.

Speedreading: En teknik der hjælper med at forbedre ens læseegenskaber, så man læser og forstår en tekst hurtigere.

Highscore: En score der bliver udregnet efter en bruger har lavet en quiz. Selve udregning bliver gjort af en algoritme.

GUI: Grafisk brugerflade.

1.5 Produktmæssige hovedansvarsområder

Navn	Område
Christoffer(CKJ)	GUI og grafisk design
Marius (MSLM)	Login feature
Tobias (TDH)	Quiz feature
Viktor(VS)	Test og udviklingsværktøjer
Simon(SMH)	Quiz feature
Kasper(KGA)	Quiz feature

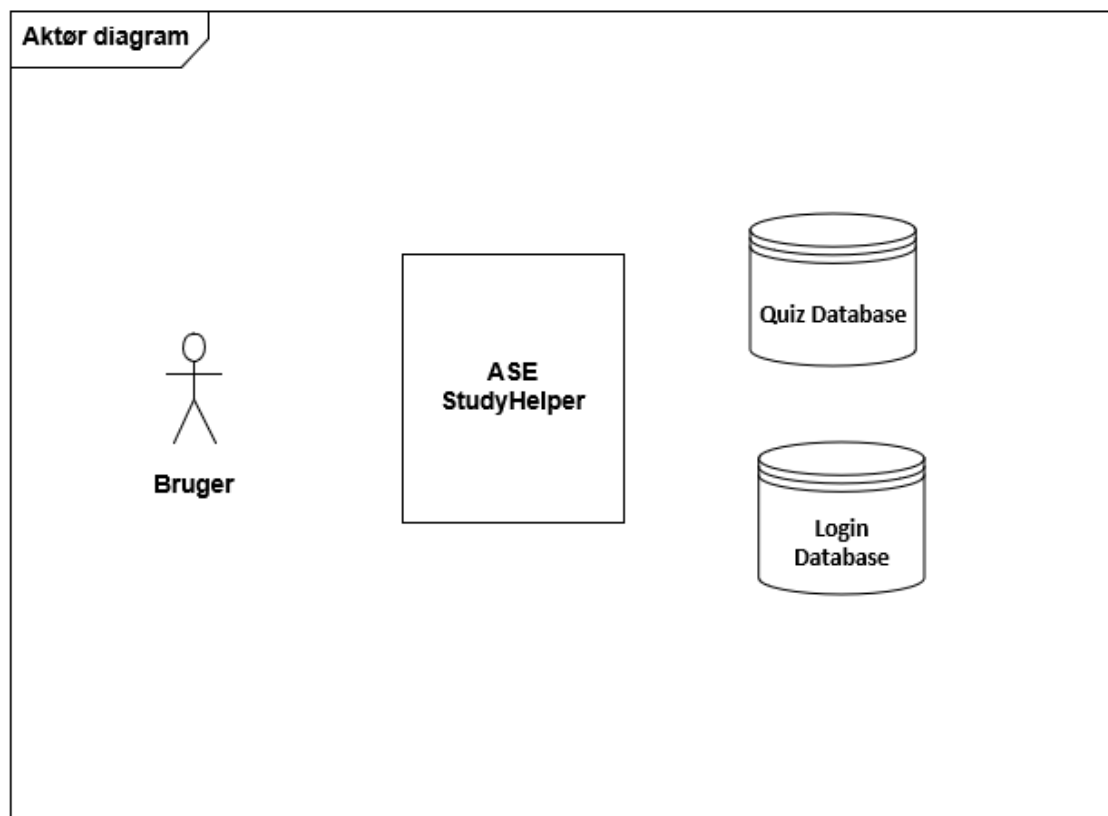
Tabel 1.1. Forklarer hvem der har haft hvilke produktmæssige hovedansvarsområder

Kravspecifikation 2

Kravspecifikationen vil gennemgå de funktionelle og ikke funktionelle krav gennem diagrammer og User stories.

2.1 Aktør Diagram

På figur 2.1 ses vores aktør diagram



Figur 2.1. Her ses vores aktør kontekst diagram. Hvor vi har en bruger der bruger vores system og vores system der kontakter database serveren for at indhente brugerens data.

Aktør beskrivelse

Bruger: Er den person der bruger appen ASE StudyHelper.

Sekundær aktør beskrivelse

Databaseserver: Her bliver det data ASE StudyHelper skal bruge opbevares

2.2 User stories

Da vi startede projektet skulle vi finde ud af hvad appen skulle have af funktionalitet. Dette beskrives i en række korte user stories, hvor vi som udviklere kan vente med at præcisere til implementeringen. Dette sker ved at der skrives en accepttest for den funktionalitet vores user stories beskriver. Til dette var vi brugt Gherkin sporget.

1. Opret "User"

Som ny bruger ønsker jeg kunne oprette et user login og et password for at få adgang til applikationen

2. Login

Som bruger ønsker jeg at kunne login med min eksisterende user.

3. Logud

Som user ønsker jeg at kunne logge ud.

4. Ændre password

Som user ønsker jeg at kunne ændre mit password

5. Følg progression

Som bruger ønsker jeg at kunne følge min users progression.

6. Sammenlign score

Som user ønsker jeg at kunne sammenligne min pointscore med andre brugere.

7. Tilføj eget indhold

Som user ønsker jeg at kunne tilføje egne quiz emner med tilhørende spørgsmål, hvor disse kan anvendes af mig selv og andre.

8. Kør quiz

Som user ønsker jeg at kunne tage en quiz for at teste min viden i kurser ved ASE

- a) Som user ønsker at få præsenteret et spørgsmål og fire svarmuligheder
- b) Som user ønsker jeg at have 20 sekunder til at vælge mit svar på et spørgsmål
- c) Som user ønsker jeg efter en afsluttet quiz at få præsenteret min score for denne quiz

9. Udregn highscore

Når jeg som bruger har fuldført en quiz ønsker jeg at systemet udregner en highscore for mit resultat.

10. Giv indhold en bedømmelse

Når jeg som bruger har fuldført noget indhold ønsker jeg at give det en bedømmelse

3.1 Mobil GUI

3.1.1 Valg af framework

I idefasen af projektet nåede vi frem til fire hovedpunkter for udviklingen af vores applikation.

1. Vi ville udvikle applikationen i et cross-platform miljø, så vi kunne skrive appen i ét programmeringssprog, og derefter kompilere til både iOS og Android.
2. Vi ville skrive appen i .NET frameworket i C#, da dette var det gennemgående programmeringssprog på gældene semester.
3. Mobilapplikationen skulle køre native på enhederne, og ikke køre i en "wrapper".
4. Vi ville hovedsageligt bruge et GUI-relateret design pattern, som det gennemgående pattern, i udviklingen af applikationen.

Der er flere muligheder for cross-platform udvikling, og vi havde valget mellem at bruge nogle forskellige miljøer til udviklingen. Vores overvejelser faldt på Android Studio, React Native, og Xamarin Forms.

Android Studio er det bygget til udelukkende at skrive til Android enheder. Her er Java det sprog som skrives i. Dette gør at der ikke behøves at være en masse ekstra frameworks til at kompilere applikationerne, da Android er selv er programmeret i Java. Android Studio og Java var derfor ikke det optimale valg for os, da det hverken tillod os at kryds-kompilere, eller tillod os at skrive i .NET.

React Native er et Javascript bibliotek, som gør det muligt at krydskompilere til flere forskellige mobile enheder. Det kører native på enhederne, hvilket var et af vores fokuspunkter, men hvis man udvikler applikationer i React Native, så skal man skrive i Javascript, og dette ville vi gerne undgå.

Xamarin Forms er et framework til udvikling af mobil applikationer. Det er bygget ovenpå .NET frameworket, så programmeringssproget her er C#. Samtidig kan det kompilere til flere enheder, herunder Android, iOS og UWP. Selvom man her skriver i C# vil det kompileres sådan, at det kører native på enhederne.

	Cross-Platform	Native	C# .NET
Android Studio	X	✓	X
React Native	✓	✓	X
Xamarin Forms	✓	✓	✓

Med disse pointer i overvejelse, har vi valgt Xamarin Forms som udviklingsmiljø. Her kan vi både skrive i C# .NET, samt udvikle applikationer som kører native på enhederne, og udvikle til multiple platforme.

3.1.2 Valg af design patterns

Da vi havde valgt framework, skulle vi også beslutte os omkring hvilket GUI design pattern vi ville bruge. Vi havde enkelte krav til det pattern vi skulle vælge.

- Vi ville have et design pattern der var nemt at teste
- Vi ville have et pattern der virkede godt med Databindings

Vi har i vores undervisning arbejdet med 3 forskellige GUI design patterns Model, View, Controller **MVC**, Model View Presenter **MVP**, og Model, View, ViewModel **MVVM**. En kort beskrivelse af de forskellige patterns fordele og ulemper, er listet her.

Fordele og ulemper GUI design patterns		
Navn	Fordele	Ulemper
MVC	<ul style="list-style-type: none"> • Model og View hver for sig • Model er nem og teste 	<ul style="list-style-type: none"> • Svær at teste • View og Controller er meget tæt knyttet • Meget vedligeholdelse i Controller • Ofte meget store Controller
MVP	<ul style="list-style-type: none"> • View får kun af vide hvad den skal vise og ikke hvordan • Nemt at teste Model 	<ul style="list-style-type: none"> • Meget vedligeholdelse i Presenter • Skal bruge et Virtual view
MVVM	<ul style="list-style-type: none"> • Testbarhed • Lav kobling • Udvikler arbejdsflow • Genbrugeligt. • Virker godt med XAML og Data bindings 	<ul style="list-style-type: none"> • Overdrivelse • Svært • Komplekst

Tabel 3.1. Fordele og ulemper i GUI design Patterns

Når vi tog disse pointer i overvejelser valgte vi at gå med MVVM pattern til applikationen. Især da MVVM pattern er udviklet med XAML i baghovedet¹, er dette godt til udvikling af applikationer.

¹<https://bit.ly/2QNAUXB> - From Data Bindings to MVVM Microsoft DOCS

3.2 Valg af databaser

Til valget af databaser skulle der besluttes hvilken tilgang vi ville anvende. Vi vidste at relationelle databaser typisk er statiske, anvender Structured Query Language og at data gemmes i tabeller. Dokumentdatabaser derimod er mere ustrukturerede og gemt skemafrit i JSON-filer.

Her blev det hurtigt besluttet vi ville prøve at anvende begge typer. Begge typer er gode valg, men har også begge sine fordele og ulemper. Disse fordele og ulemper vil blive uddybet i det tilhørende design afsnit.

Dertil vidste vi også lidt om hvad vi ønskede vores applikation skulle kunne. Vi ønskede en måde at gemme brugerdata, highscores og en form quiz. Her kunne vi hurtigt lave nogle antagelser om hvordan brugerdata/highscore skulle lagres, hvorimod quiz strukturen var mere uklar. Derfor blev det besluttet at en dokumentdatabase skulle indeholde quiz funktionaliteten og login/highscore skulle gemmes i en relationel database.

Til dokumentdatabasen bruges Azure Cosmos DB SQL API med foranliggende REST API som blev udviklet med et MVC pattern, hvorpå der var lagt et repository pattern ovenpå. Den relationelle database anvendes ligeså et REST API, dog uden et repository pattern.

3.2.1 RESTful design

Vores REST API anvender HTTP request til CRUD-operationerne. Vi kan således tilgå/ændre/slette vores quiz objekter fra Azure Cosmos databasen. Dette tillader brugerne at anmode om ressourcer som for eksempel kan fremkalde et svar formateret i for eksempel JSON. Hvordan dette implementeres uddybes senere.

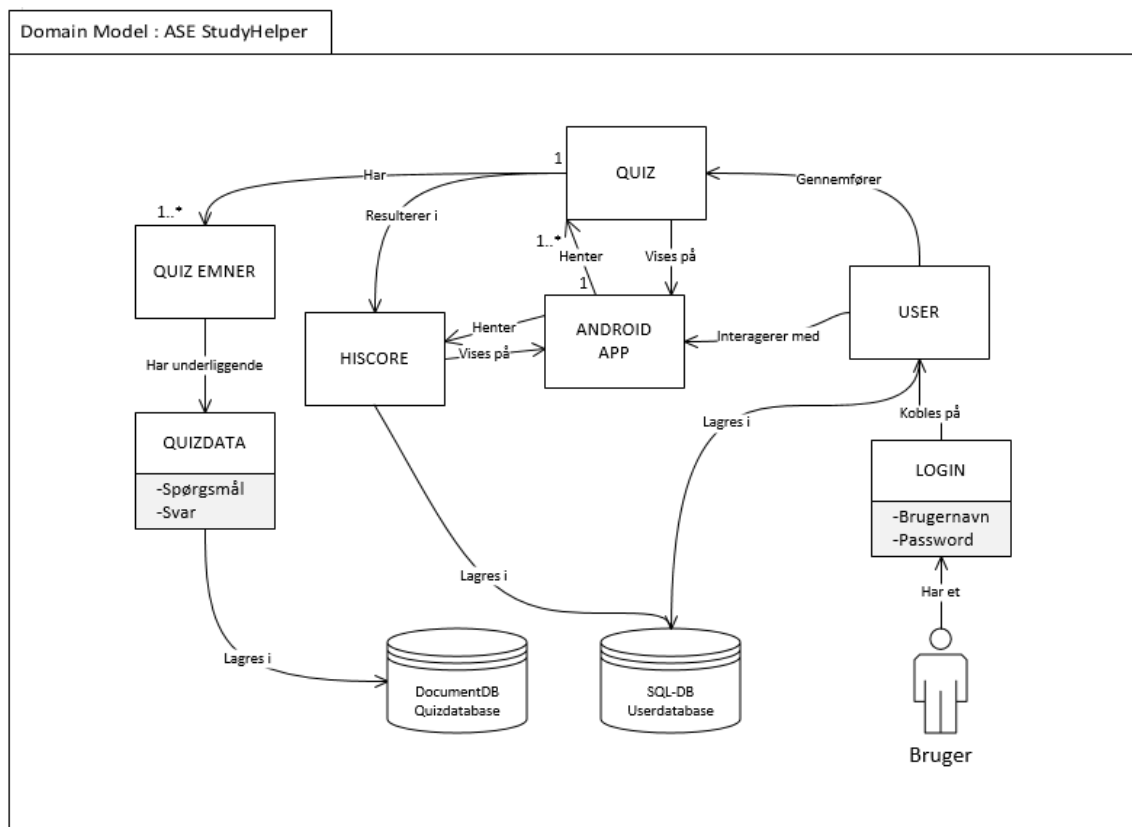
REST API er udgjort af forskellige lag, hvor hvert enkelt lag har specifik funktionalitet og ansvar. Disse forskellige lag arbejder sammen og skaber et hierarki der gør applikationen mere modulær og skalerbar. Sammenlignes med MVC Pattern, har hvert lag også sine egne ansvarsområder. Her fokuserer controller klasserne på de indkommende actions, view på hvordan data skal vises og modelklasserne på hvordan data skal formateres.

Denne tilgang og MVC frameworket har hjulpet os med udarbejdelsen af quiz databasen, hvis design samt fordele beskrives i afsnit 5.2.2 og afsnit 5.2.3.

Arkitektur 4

4.1 Overordnet domænemodel for systemet

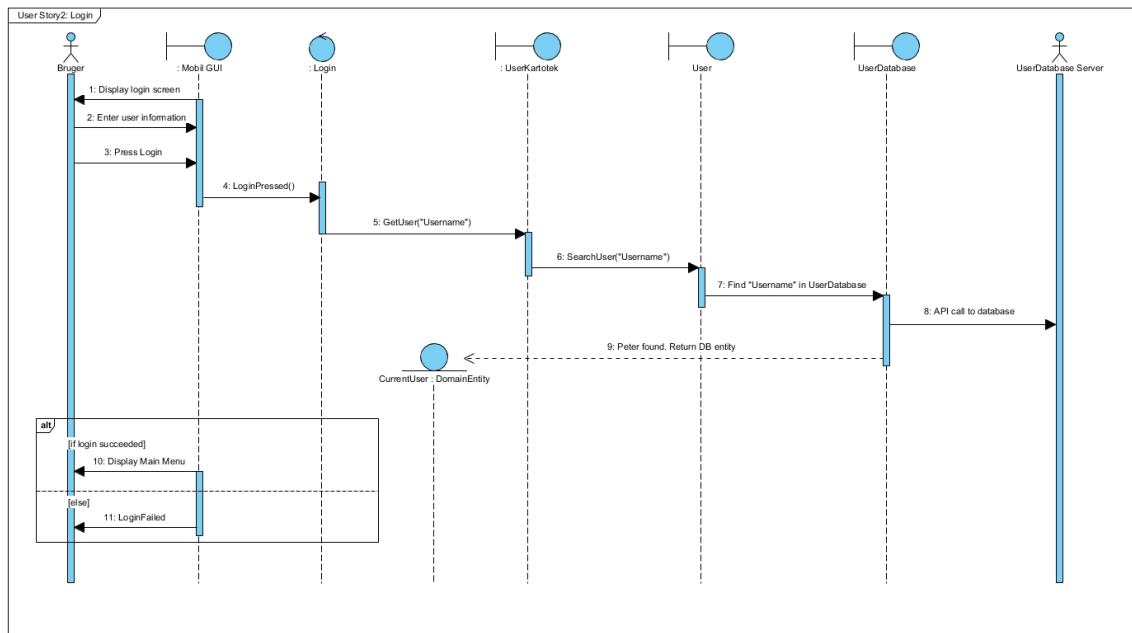
For at kunne forstå systemet og kommunikationen mellem APP'en, databaserne og brugeren er domænemodellen på figur 4.1 udarbejdet. Her ses kommunikationslinjen fra brugerens interaktion med GUI'en helt ned til databasen.



Figur 4.1. Overordnet domænemodel for vores system

4.2 Sekvensdiagram med udgangspunkt i “user story 2 - Login”

Til større forståelse af systemet og de mulige aktører er nedenstående sekvensdiagram udarbejdet det kan ses på figur 4.2. Diagrammet tager udgangspunkt i user story nr. 2 login. Her ses den primære aktør brugeren og hvordan den pågældende user story afspejles i systemet.



Figur 4.2. Sekvensdiagram over User Story 2 - login

5.1 Mobil GUI

5.1.1 MVVM

Til udviklingen af applikationen, benytter vi som sagt et MVVM design pattern. MVVM består, som nævnt tidligere, af tre dele. Model, View og ViewModel. Disse tre dele, indeholder tre forskellige lag i vores applikation.

Model - Denne del indeholder vores business objects, som bruges til at enkapsle data og opførsel af applikationsdomænet. Dette er klasser som f.eks. Quiz, Question og Option.

View - Dette lag, indeholder logik der er med til at definere strukturen, layoutet og udseendet af det som brugeren ser på skærmen. I Xamarin, kommer view til udtryk ved en "page". View er oftest kodet i XAML, og skal indeholde så lidt code-behind som muligt, selvom dette kan være svært hvis man vil implementere noget visuelt, der kan være svært at kode i XAML, såsom animationer.

ViewModel - ViewModel laget indeholder alt logikken for applikationen. Den er specielt designet til view, og består af en klasse med properties der repræsenterer Views state. Den indeholder også metoder der implementerer logikken bag View.

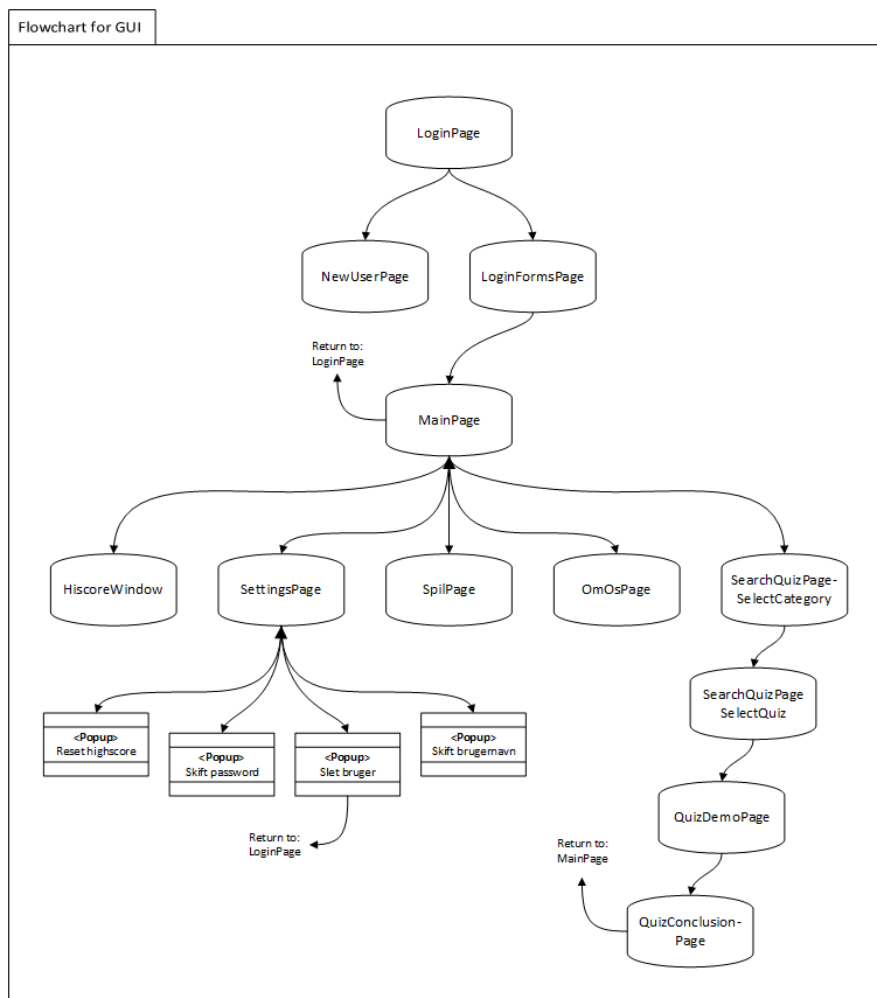
Selvom vores viewmodel lyder meget som code-behind, er dette ikke tilfældet. Code-behind er nemlig tæt koblet, med xamarin, hvilket gør det svært at unitteste Code-behind. ViewModel er dog en smule som Code-Behind, men det skal modificeres, så det har minimal kobling til Xamarin. Ved at gøre dette, får vi det der hedder et POCO (Plain Old CLR Object), som man let kan unitteste på. Det kræver dog også en del mere kode, at skulle decouple code-behind, så det bliver til en ViewModel, hvilket gør at MVVM ikke altid er gavnligt.

5.1.2 User Interface

GUI designet, også kaldet vores viewlag, er udarbejdet med henblik på brugervenlighed. Vi gik efter at gøre applikationen så intuitiv som muligt for brugeren. Vores applikation indeholder en længere række sider, hhv:

- LoginPage
- LoginFormsPage
- NewUserPage
- MainPage
- SpilPage
- HiscoreWindow
- OmOsPage
- SettingsPage
- SearchQuizPageSelectCategory
- SearchQuizPageSelectQuiz
- QuizDemoPage
- QuizConclusionPage

Der navigeres rundt i disse pages på kryds og tværs, og for at illustrere hvordan dette hænger sammen, har vi lavet et flowchart.



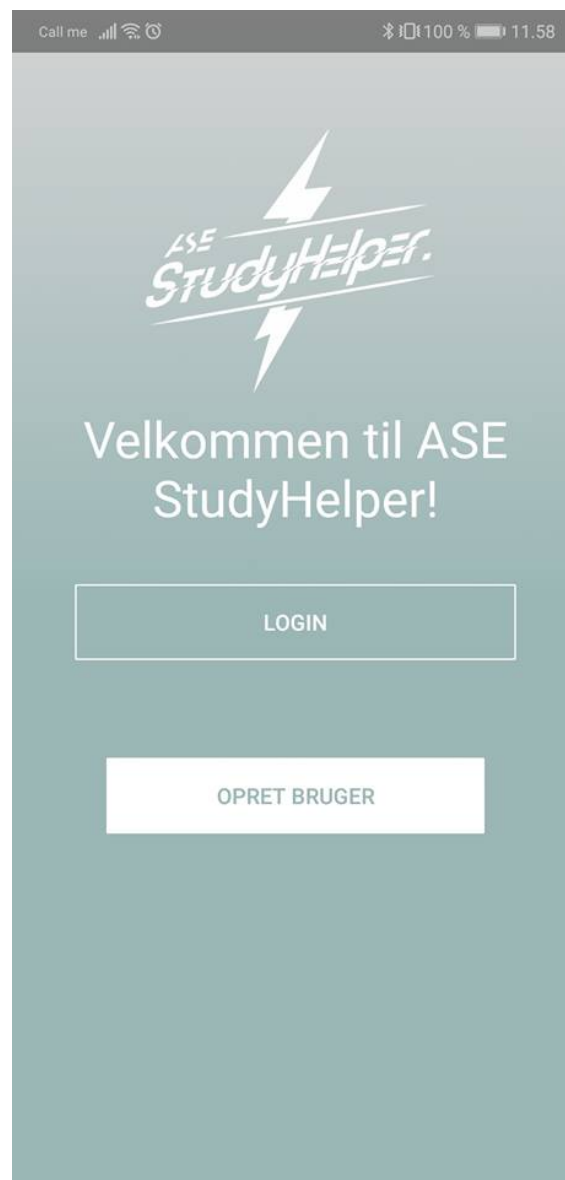
Figur 5.1. Screenshot af LoginPage

Funktionaliteten af applikationens forskellige pages, bliver forklaret i dybden nedenfor.

LoginPage - Applikationen starter ud på vores login page, hvor brugeren har muligheden for enten at logge ind med sin eksisterende user, eller at oprette en ny user på applikationen. Hvis man vælger at logge ind, bliver man navigeret videre til LoginFormsPage. Hvis man vælger at oprette en ny bruger, bliver man navigeret videre til NewUserPage.

Navigerer fra: Intet (Dette er default startpage når man åbner appen, hvis man **IKKE** er logget ind)

Navigerer til: LoginFormsPage, NewUserPage

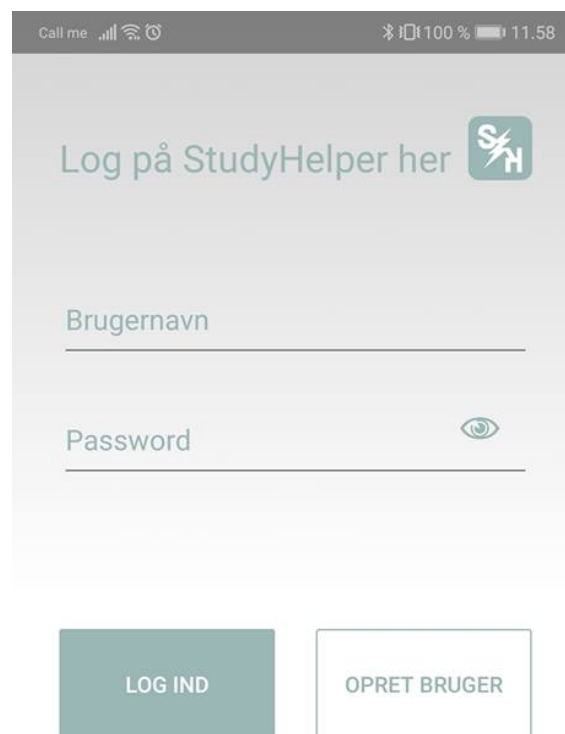


Figur 5.2. Screenshot af LoginPage

LoginFormsPage - Denne side er dedikeret til at brugeren kan logge ind med en allerede eksisterende user. Der skal indtastes et username og password på denne side, som skal stemme overens med en user som ligger på databasen. Siden beskriver hvad du skal udfylde for at logge ind, og hvis du glemmer at udfylde f.eks. password, fortæller siden dig, at du har glemt at udfylde det pågældne felt. Hvis dit username og password stemmer overens med en user på databasen, når du trykker på login, bliver du navigeret videre til MainPage.

Navigerer fra: LoginPage

Navigerer til: MainPage

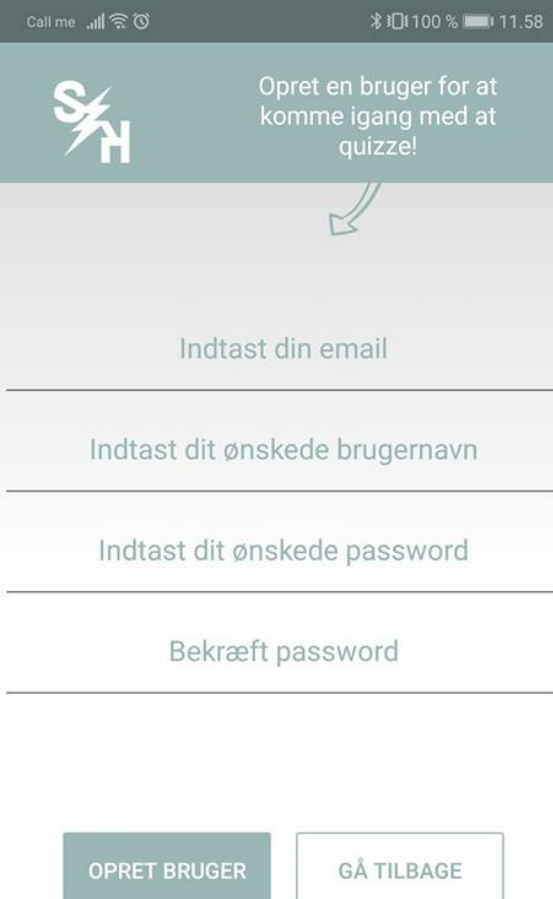


Figur 5.3. Screenshot af LoginFormsPage

NewUserPage - På denne side, har brugeren en mulighed for at oprette en ny user. På siden skal der indtastes en e-mail adresse, som bliver tilknyttet user, et username og et password, som skal indeholde et stort bogstav, et lille bogstav, et tal og et tegn. Hvis det indtastede username allerede findes på databasen, kommer der en popup besked, der informerer brugeren om at username allerede eksisterer.

Navigerer fra: LoginPage

Navigerer til: LoginPage



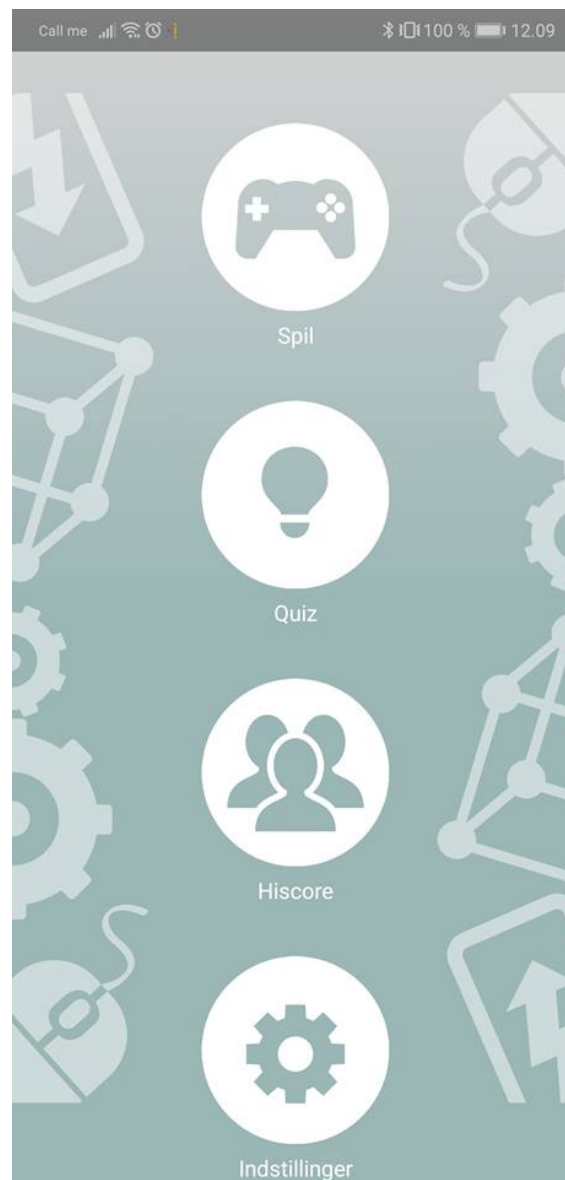
The screenshot shows a mobile application interface for creating a new user. At the top, there is a status bar with 'Call me', signal strength, Wi-Fi, and battery level at 100% with the time 11:58. Below the status bar is a header with the 'SH' logo on the left and the text 'Opret en bruger for at komme igang med at quizze!' on the right. A curved arrow points from the header text to the first input field. The form consists of four input fields with labels: 'Indtast din email', 'Indtast dit ønskede brugernavn', 'Indtast dit ønskede password', and 'Bekræft password'. At the bottom, there are two buttons: 'OPRET BRUGER' (a solid teal button) and 'GÅ TILBAGE' (a white button with a teal border).

Figur 5.4. Screenshot af NewUserPage

MainPage - MainPage fungerer som en hovedmenu gør på de fleste apps. Man kan på siden danne sig et overblik over hvilke features ASE-Studyhelper har. Dette inkluderer forskellige knapper, som hhv. navigerer til: HiscoreWindow, SpilPage, OmOsPage, SearchQuizPageSelectCategory eller SettingsPage. Det er også herfra, at brugeren har mulighed for at logge ud. Hvis brugeren trykker på logud, bliver user logget ud af applikationen og der bliver navigeret tilbage til LoginPage.

Navigerer fra: LoginFormsPage (Dette er default startpage når man starter appen, hvis man er logget ind)

Navigerer til: HiscoreWindow, OmOsPage, SettingsPage, SearchQuizPageSelectCategory, SpilPage, LoginPage

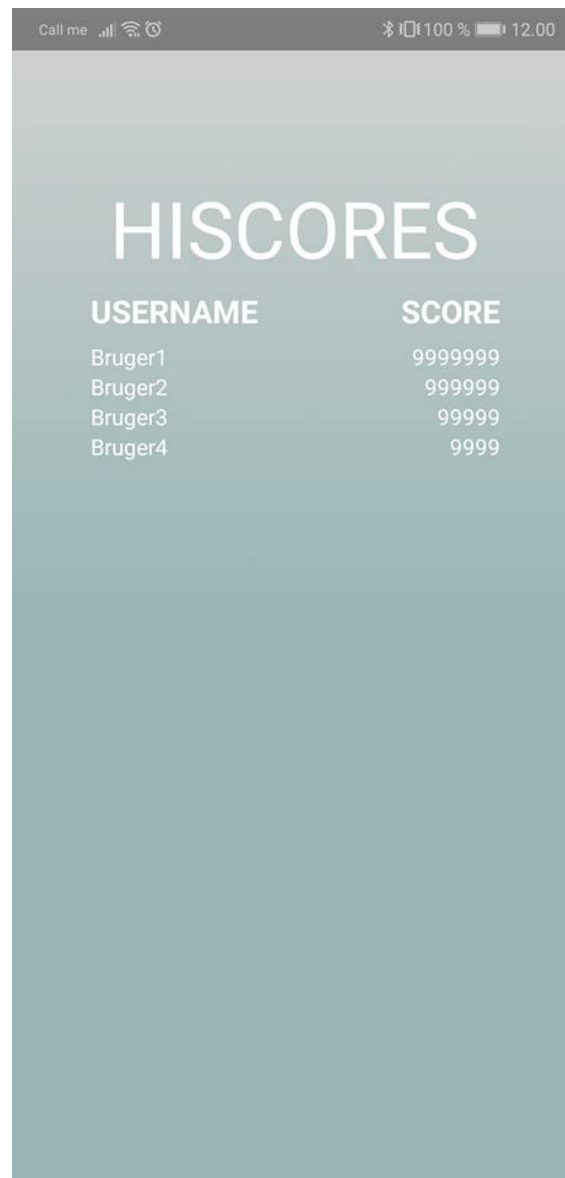


Figur 5.5. Delvist screenshot af MainPage

HiscoreWindow - Her kan brugeren se en oversigt over de hiscores som alle users på applikationen har opnået. Der er intet interaktivt på denne side, den bliver udelukkende brugt til at give en oversigt over hiscores.

Navigerer fra: MainPage

Navigerer til: Intet



Figur 5.6. Screenshot af HiscoreWindow

OmOsPage - Muligheden for at læse om udviklerholdet der har udarbejdet denne app, kan læses på denne page. Der findes intet interaktivt på siden, den bliver udelukkende brugt til at distribuere information om os, til brugeren.

Navigerer fra: MainPage

Navigerer til: Intet

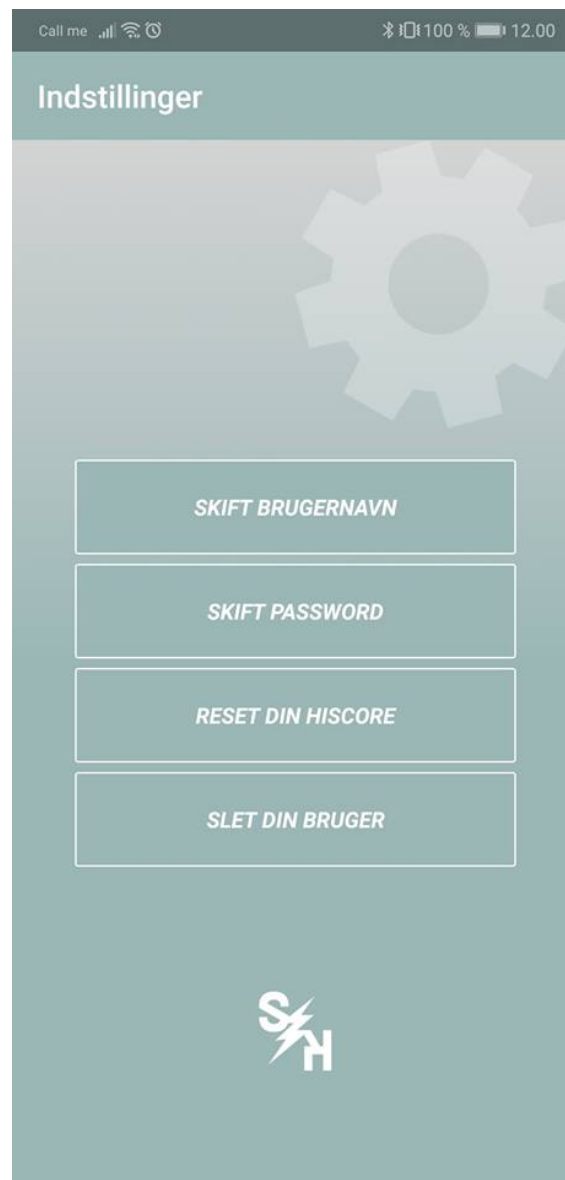


Figur 5.7. Delvist screenshot af OmOsPage

SettingsPage - På denne side, har brugeren mulighed for at ændre indstillinger i applikationen. Der er på siden mulighed for at skifte password, skifte brugernavn, slette bruger og resette alle users highscores. Hvis brugeren trykker på "Slet user"knappen, navigeres der tilbage til loginpage.

Navigerer fra: MainPage

Navigerer til: LoginPage

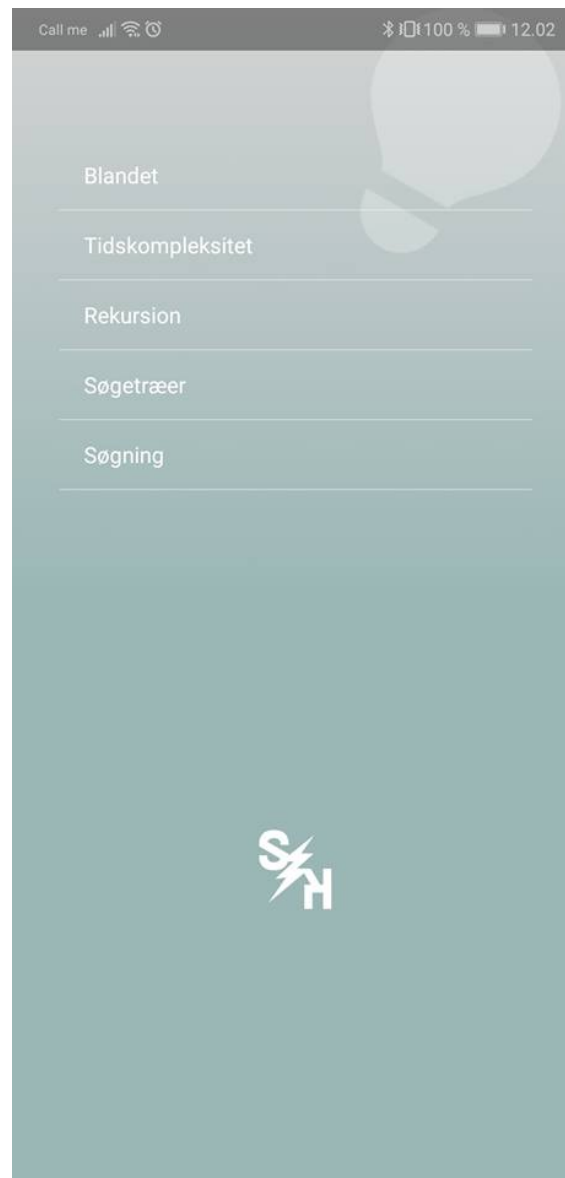


Figur 5.8. Screenshot af SettingsPage

SearchQuizPageSelectCategory - Her vælger brugeren hvilken quiz kategori han/hun gerne vil igang med. Når brugeren har valgt en kategori, bliver brugeren navigeret videre til SearchQuizPageSelectQuiz.

Navigerer fra: MainPage

Navigerer til: SearchQuizPageSelectQuiz

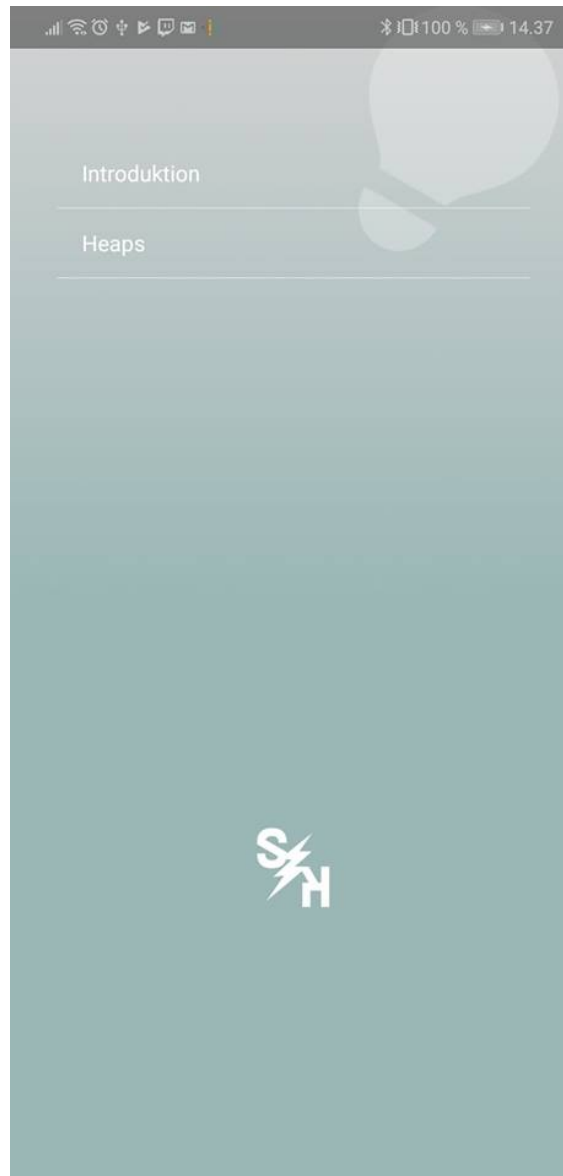


Figur 5.9. Screenshot af SearchQuizPageSelectCategory

SearchQuizPageSelectQuiz - Her vælger brugeren den specifikke quiz han/hun gerne vil igang med. Når brugeren har valgt en quiz, bliver brugeren navigeret videre til QuizDemoPage.

Navigerer fra: SearchQuizPageSelectCategory

Navigerer til: QuizDemoPage

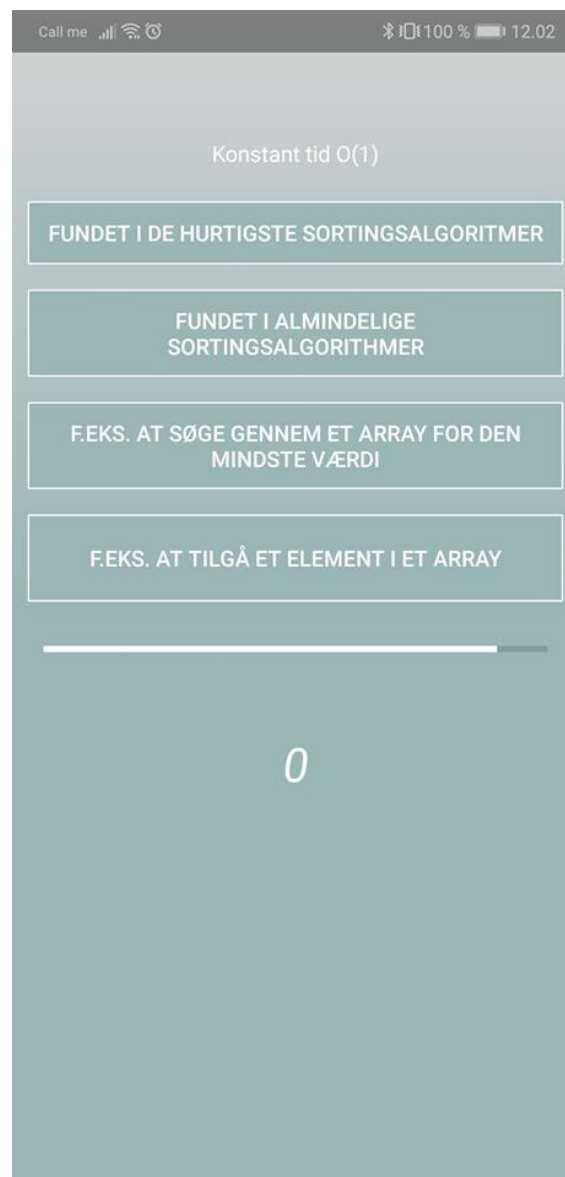


Figur 5.10. Screenshot af SearchQuizPageSelectQuiz

QuizDemoPage - Det er på denne side, at selve quizzen foregår. Her vil der fremkomme et spørgsmål, til den valgte kategori, med 4 tilhørende svarmuligheder, hvori det ene af svarmulighederne vil være det korrekte. Siden opdateres hver gang der bliver svaret på et spørgsmål. Til hvert spørgsmål, indgår der på siden en timer, som indikerer tiden brugeren har til at svare på spørgsmålet. Herudover kan brugeren følge med i antallet af point, der er opnået indtil videre. Når der ikke er flere spørgsmål tilbage i quizzen, bliver brugeren navigeret videre til QuizConclusionPage.

Navigerer fra: SearchQuizPageSelectCategory

Navigerer til: QuizConclusionPage, QuizDemoPage

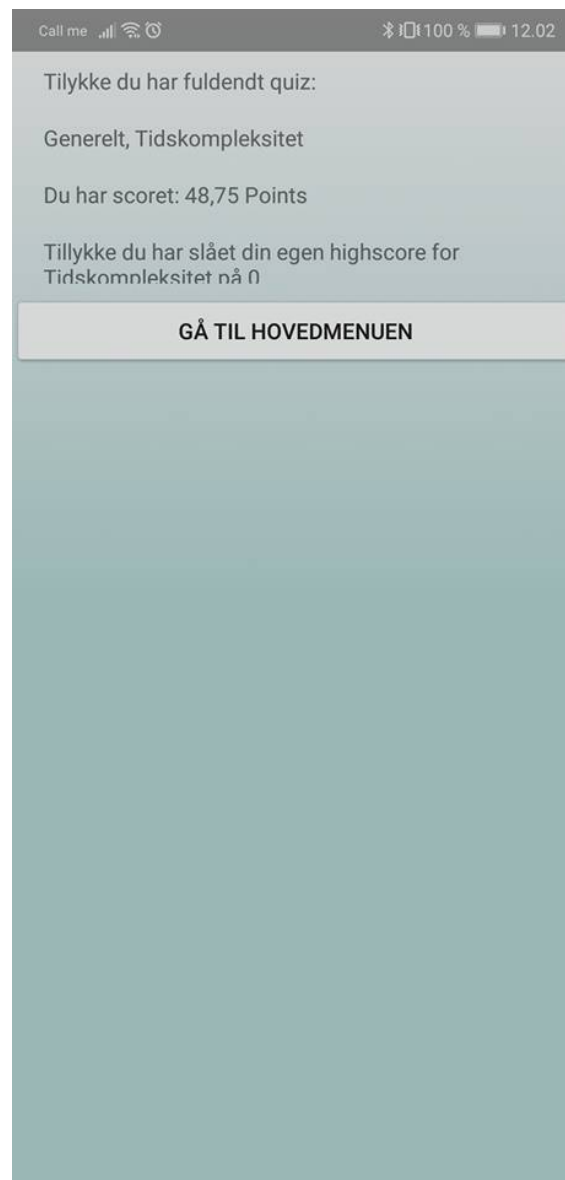


Figur 5.11. Screenshot af QuizDemoPage

QuizConclusionPage - Denne page fungerer som en afslutning på quizzen, hvor brugeren bliver informeret om hvilken quiz han/hun netop har gennemført, samt den samlede score der blev opnået i quizzen. På siden findes en knap, hvorpå der står "Naviger tilbage til hovedmenu". Hvis den knap trykkes, bliver man navigeret tilbage til MainPage.

Navigerer fra: QuizDemoPage

Navigerer til: MainPage

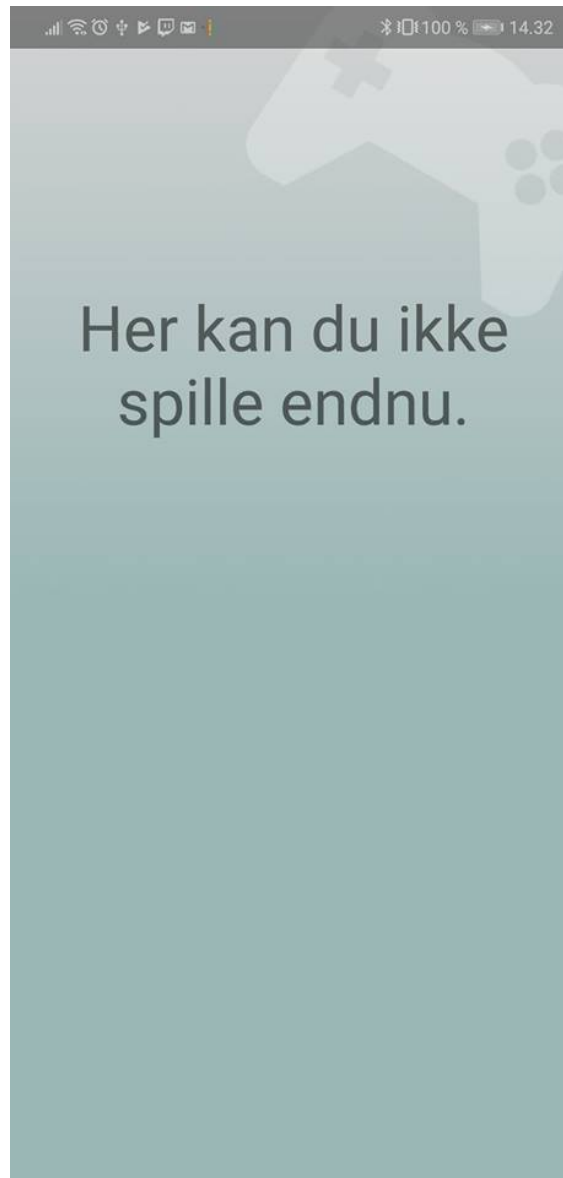


Figur 5.12. Screenshot af QuizConclusionPage

SpilPage - SpilPage skal indeholde en række spilfunktioner. Denne er dog ikke blevet udviklet endnu.

Navigerer fra: MainPage

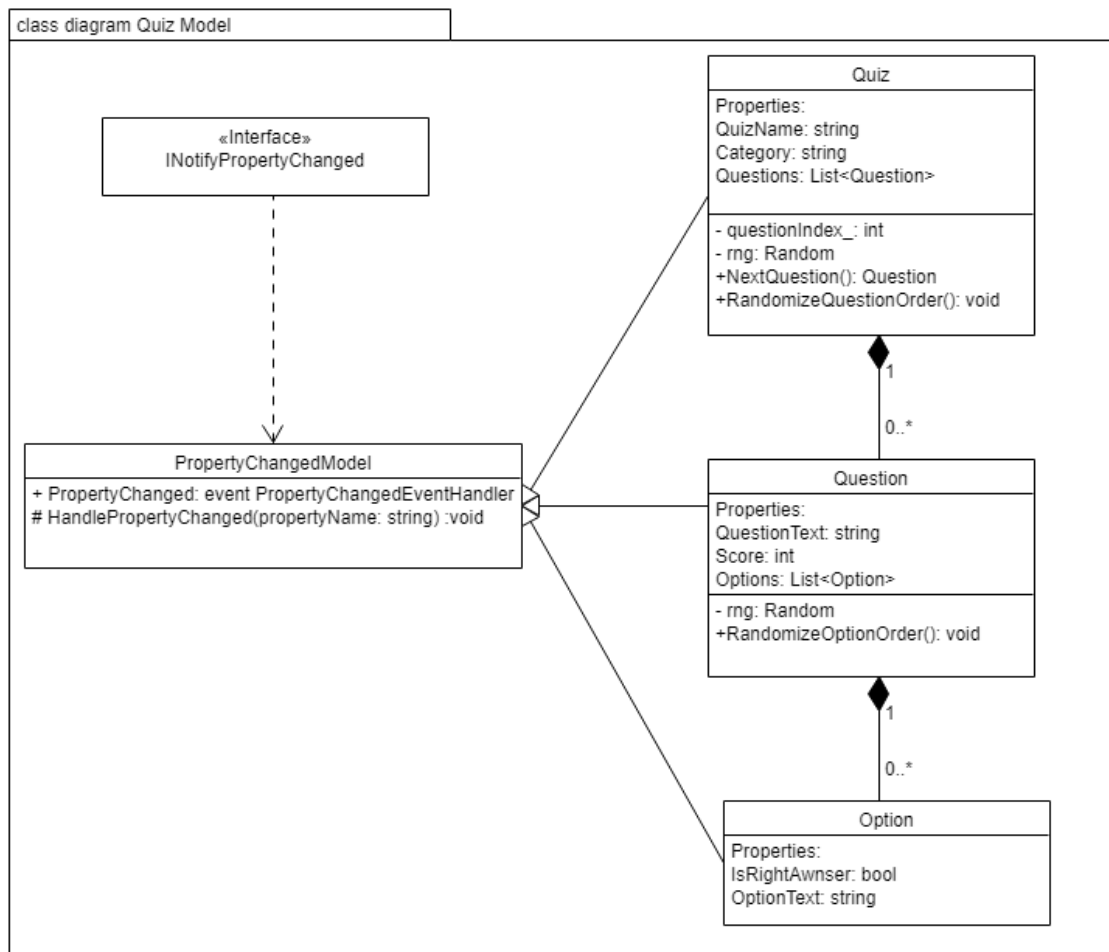
Navigerer til: Intet



Figur 5.13. Screenshot af SpilPage

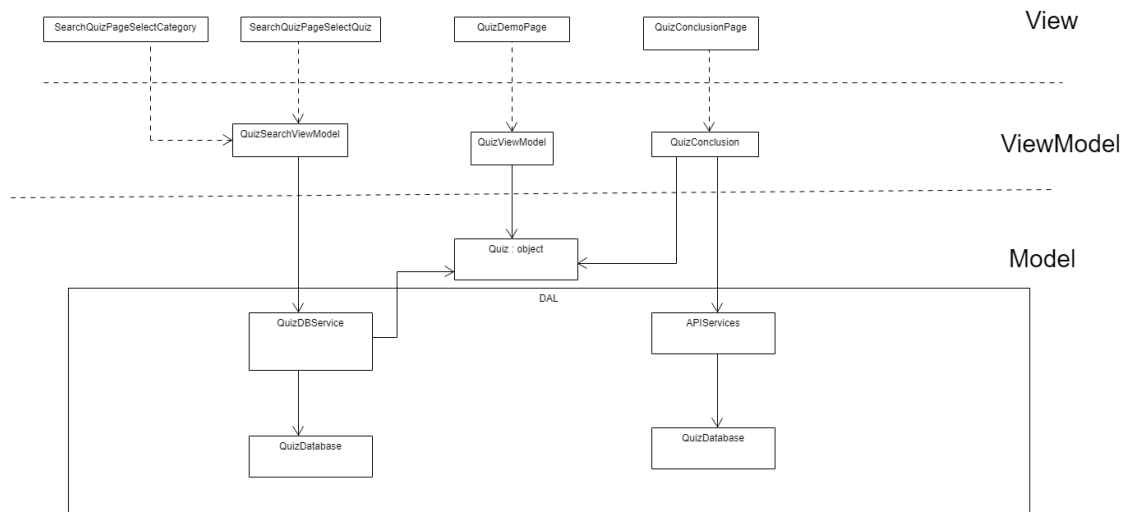
5.2 Quiz feature

5.2.1 MVVM anvendt på Quiz feature



Figur 5.14. klasse diagram for Quiz Model

Modellaget kan ses på figur 5.14 ovenfor. Dette består tre domain klasser: Quiz, Question og Option. Implementingen for disse kan findes under afsnittet Implementering - Quiz Feature: QuizModel 6.5



Figur 5.15. MVVM arkitektur anvendt på Quiz-modellen

Som det kan ses på figur 5.15 oven for, er MVVM blevet anvendt på projektet således at der er implementeret separation of concerns. I View-laget er der for quiz feature defineret 4 views. hhv.

- SearchQuizSelectCategory
- SearchQuizSelectQuiz
- QuizConclusionPage
- QuizDemoPage

SearchQuizSelectCategory og SearchQuizSelectQuiz er to views med hver sin funktion, i forhold til samme opgave at udvælge den quiz, user ønsker at besvare. De er derfor begge forbundet til den viewModel der har dette ansvar. SearchQuizViewModel. QuizDemoPage har til opgave, at give user mulighed for at besvare den udvalgte quiz og er derfor forbundet til QuizViewModel, der netop kun har dette ansvar. Til sidst har QuizConclusionPage opgaven at oplyse user om status, den endelige score og evt. highscoreopdatering, efter at den valgte quiz er blevet fuldendt. Dette view er derfor forbundet til den viewModel der udviklet til dette formål, med navnet QuizConclusionViewModel.

Adskillelsen af ansvar opstår i fordelingen mellem de tre viewModels. Kun SearchViewModel har adgang til quiz-dokumentdatabasen gennem QuizDBService-klassen. Denne viewModel stiller for den første page, muligheden for søge efter og præsentere en liste af unikke kategorier blandt alle de quizzes, som er tilgængelige på databasen. For den næste search-Page stiller viewModelen muligheden for at load og præsentere en liste af quizzes, som deler den valgte kategori på tidligere side.

QuizViewModel har ingen forbindelse til databasen, da dette ikke er nødvendigt. Den faciliterer kun logikken der er nødvendig, for at brugeren kan besvare en quiz. Den udvalgte quiz injekter altså til QuizViewModel, ved overgang fra SearchQuizPageSelectQuiz. Når det sidste spørgsmål i en quiz er blevet besvaret, bliver quizen behandlet i QuizViewModel samt dennes Totale score og dens QuizConclusionViewModel. QuizViewModel har heller

ingen adgang til databasen, hvor highscore gemmes. Den er altså helt isoleret og kender kun til det Quiz-objekt den får injectet.

QuizConclusionViewModel som injekter den gennemførte quiz, og userens opnåede score, kan herefter, via dets instans af ApiService, sammenligne den opnåede score med den gemte highscore for den aktuelle user. Hvis den nye score er højere, opdateres databasen. Den implementerede arkitektur faciliterer altså isoleret adgang til projektets databaser, i det omfang de er nødvendige.

5.2.2 MVC (Strukturering af databaser)

Til udviklingen af de to REST API'er, som vi benytter til at få fat i vores databaser, benytter vi et MVC design pattern. MVC består af tre dele. Model, View og Controller. Disse tre dele, indeholder tre forskellige lag i vores applikation. Til udvikling af quiz databasen, bliver der også anvendt et repository pattern, sammen med MVC, men dette vil ikke blive forklaret i denne sektion, der refereres i stedet til sektionen omkring Quiz Databasen i afsnit 5.2.3.

Model - Modellaget er med til at strukturere vores data på en pålidelig måde, og forbereder den på at udvikle sig efter Controllers instruktioner.

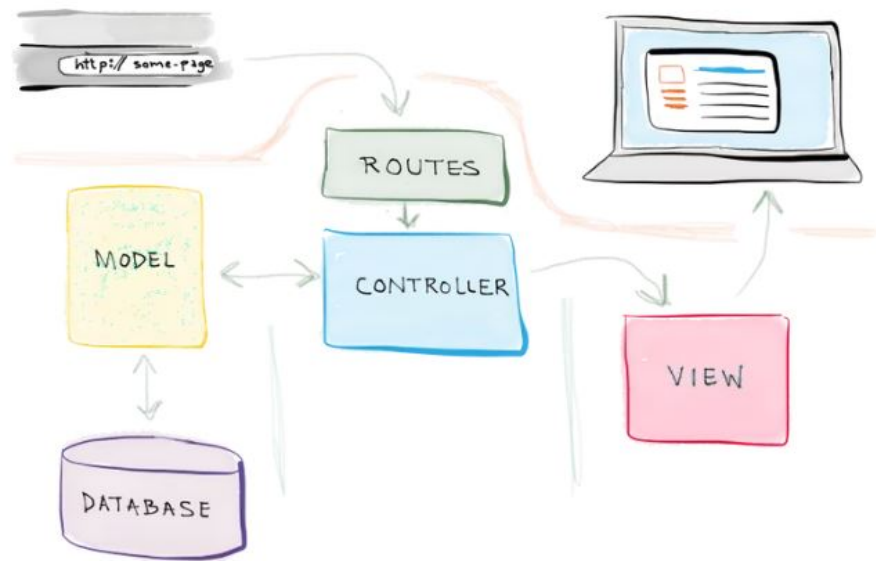
View - Viewet indeholder logik der er med til at definere strukturen, layoutet og udseendet af det som brugeren ser på skærmen. I vores tilfælde har vi ikke brugt tid på at implementere et view, men i stedet har vi anvendt swaggers user interface ¹, til at teste vores funktioner.

Controller - Controlleren er et nøgle-element i MVC. Det er den der tager i mod brugerkommandoerne, og ud fra dem sender kommandoer til modellaget, om at modeldataen skal opdateres. Samtidig sender den også instruktioner til view, om at den skal opdatere interfacet.

I diagram form, vil det se ud som i figur 5.16 ².

¹<https://swagger.io/tools/swagger-ui/>

²<https://bit.ly/2BaFyoM> - Billedet er lånt af blog.codeanalogies.com



Figur 5.16. MVC i diagramform

5.2.3 Quiz databasen

Til afviklingen af quizfunktionen der skal anvendes på ASE-StudyHelper udvikles en Azure Cosmos DB. Foran databasen udføres et REST API som bruger Repository pattern. Før vi går i gang med beskrivelsen, af udviklingen af ovenstående uddybes REST API og Repository pattern samt vores anvendelse af disse og Azure portalen.

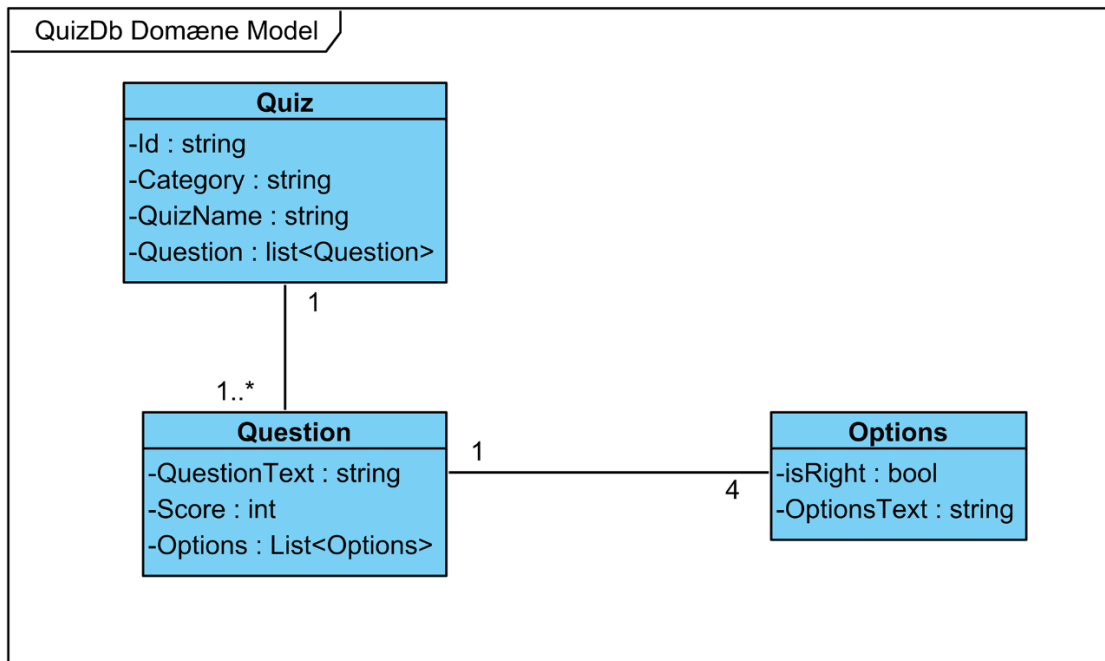
Som nævnt tidligere anvender vores REST API HTTP request til CRUD-operationerne. Vi kan således tilgå/ændre/slette vores quiz objekter fra Azure Cosmos databasen. Dette tillader brugerne at anmode om ressourcer som for eksempel kan fremkalde et svar formateret som JSON.

Der bruges også et Repository pattern. Repository kan ses som en mediator mellem domænet og data mapping layers. Nogle af fordelene ved dette er at man slipper for at duplikere query logik. Videre afkobler det vores applikation fra frameworks som entity framework.

Databasen der bruges, er en Azure Cosmos DB, der er installeret op Microsofts Azure portal. Fra denne bruges URI og primary key, og der kan i øvrigt holdes øje med databasen i forbindelse med test. Her blev der først fastlagt en struktur for det dokument der skulle gemmes i databasen. Dette hjalp med at lave nogle antagelser, ift. multiplicitet og tilhørsforhold af de forskellige entiteter. Her kom vi frem til at en quiz kan have 0 til mange spørgsmål og et spørgsmål skal have 4 valgmuligheder. Til dette blev nedenstående domænemodel udarbejdet, hvor der så kunne laves en JSON-model af denne. Denne tilgang er måske lidt unødvendig i arbejdet med NoSQL, men var på daværende tidspunkt det eneste vi havde lært.

En af fordele ved en NoSQL er nemlig at dokumentdatabaser har et dynamisk skema og kan lagre data på mange forskellige måder. Denne fleksibilitet tillader os faktisk at lave dokumenter uden strukturen skal planlægges nøje først. Dette giver os mere frihed og

tillader hvert enkelt dokument at have sin egen struktur. Så selvom det måske var lidt unødvendigt blev der alligevel opstilt domænemodel se figur 5.17 og JSON-model se figur 5.18



Figur 5.17. Domænemodel quiz DB

```

{id": "02d19547-6b10-4502-8db9-c503b8502480",
"Category": "Rekursion",
"QuizName": "Generelt",
"Question": [
  {
    "QuestionText": "Rekursion",
    "Score": 40,
    "Options": [
      {
        "IsRight": true,
        "OptionText": "Bruges til at vise tilsyneladende komplekse problemer kan løses"
      },
      {
        "IsRight": false,
        "OptionText": "Bruges med fordel til at tilgå enkelte elementer i et array"
      },
      {
        "IsRight": false,
        "OptionText": "Har aldrig risiko for lange beregningstider og stort hukommelses"
      },
      {
        "IsRight": false,
        "OptionText": "Ingen muligheder passer"
      }
    ]
  }
],
},

```

Figur 5.18. JSON-model quiz DB

Ud fra den opstillede domænemodel og JSON-dokument kunne vi definere de modelklasser som modsvarer JSON dokumentet. Disse modelklasser er dem der bruges til REST API'et.

Udviklingsforløbet med dertilhørende klasser, funktioner og funktionsbeskrivelser beskrives i implementeringsafsnittet og vi vil nu komme ind på hvad der gør vores design RESTful samt fordelene ved dette. Disse fordele gør applikationen mere fleksibel og giver os frihed til hurtigt at ændre efter behov. Ifølge datamatikeren og skaberen af REST Dr. Roy Thomas Fielding er der en række aspekter man skal være opmærksom hvis man ønsker at gøre sin applikation RESTful³. Disse aspekter og tanker i forhold til vores system er beskrevet nedenunder.

Client-server - I vores applikation skal det være muligt at lave ændringer til databasen og dens design uden at det har indflydelse på vores mobilapplikation. Det skal ligeså være muligt at modificere mobilapplikationen uden at skulle omstrukturere hele databasen. Mobilapplikation(client) og databaseapplikation(server) skal altså kunne vokse og udvikle sig individuelt. Denne "separation of concerns" afkobler hver applikation og gør dem mere skalerbare.

Generisk interface - Et andet vigtigt aspekt til at afkoble client fra server er at have en ensartet grænseflade. Dette tillader applikationen at udvikle sig individuelt uden at være koblet op på services, modeller og actions. Dette interface giver altså en standardiseret måde at kommunikere mellem server og client. I vores tilfælde ved at bruge HTTP med URI ressourcer, CRUD(Create, Read, Update, Delete) og JSON.

Stateless - REST API's er stateless, hvilket betyder at hvert kald kan udføres uafhængigt af hinanden. Det betyder at hvert kald har alt data nødvendigt til at kunne udføres succesfuldt. Det tillader serveren kun at afhænge af den data der gives med hvert kald. Disse informationer kunne være API key, access token, bruger id etc. Dette gør vores applikation mere pålidelig, ved at serveren ikke afhænger af en række af kald til at f.eks. at få opdateret et dokument i databasen.

Lagdelt system - REST API er udgjort af forskellige lag, hvor hvert enkelt lag har specifik funktionalitet og ansvar. Disse forskellige lag arbejder sammen og skaber et hierarki der gør applikationen mere modulær og skalerbar. Sammenligner vi med vores egen implementering, hvor der er anvendt MVC, har hvert lag også sine egne ansvarsområder. Her fokuserer controllerklasserne på de indkommende actions, view på hvordan data skal vises og modelklasserne på hvordan data skal formateres.

Hvordan de forskellige lag af arkitekturen arbejder sammen til at bygge et hierarki, der gør applikationen mere skalerbar og modulær skulle gerne være afspejlet i tilhørende afsnit i implementeringen.

5.3 Login feature

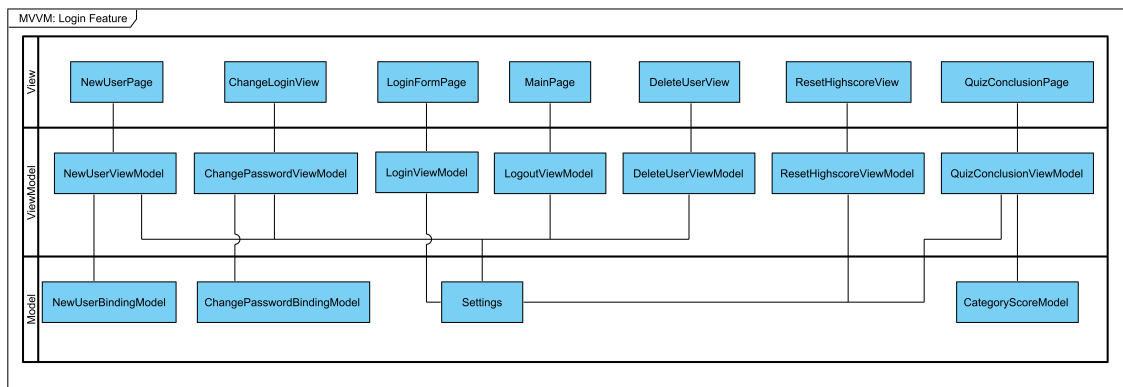
I dette afsnit vil designet af Login feature blive beskrevet. Login feature består af to databaser og et RestAPI som er deployet på azure. Udover det indeholder Login

³https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (Kapitel 5) Besøgt 12.12.2018

feature noget logik i AseStudyHelper appen. I MVVM afsnittet på Login feature er det underforstået at det er Login features design på AseStudyHelper bliver beskrevet, ligesom i MVC afsnittet er det databasernes og RestAPIs design.

5.3.1 MVVM på Login feature

I AseStudyHelper er kommunikationen, mellem AseStudyHelperDB, QuizScoresDB, applø-
gik, og GUI, designet med MVVM. På figur 5.19 gives der et overblik over Login Feature's
design i AseStudyhelper.

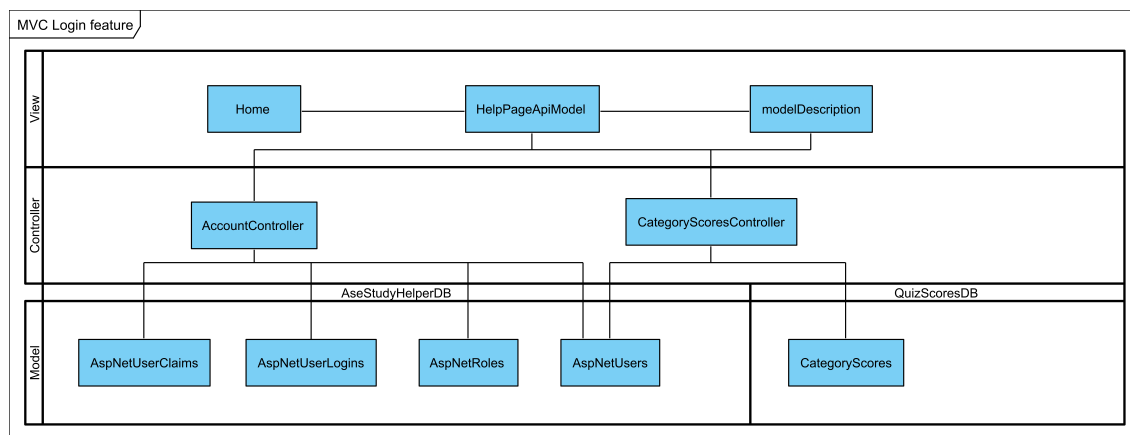


Figur 5.19. Overblik over Login Feature's MVVM design

På figur 5.19 er de forskellige views, viewmodels og models vist og hvad de snakker sammen med. Udover de klasser der er vist på figur fig: MVVMLoginFeature, bruger Login Feature også en klasse, ApiService, der står for kommunikation mellem databasernes RestAPI og Login Feature's viewmodels.

5.3.2 MVC på Login feature

Login feature RestAPI og databaser er udviklet i ASP.NET MVC. Her er det muligt at definere databasens domænemodel, view og controller. På figur 5.20 gives der et overblik over MVC designet.



Figur 5.20. MVC design for Login feature

På figur 5.20 ses de forskellige lag, Login feature består af. Fra toppen ses view, i midten controller, og i bunden Model. Modellaget er her opdelt, det skal forstås som to forskellige databaser, en med brugerinformation og en med quiz scorer.

Implementering 6

6.1 Mobil GUI

6.1.1 Metodebeskrivelse for GUI klasserne

Klasser som kun har konstruktører, og ingen yderligere funktioner er udeladt.

GUIen i vores mobil applikation er primært skrevet i HTML i XAML filer. Her er alt det visuelle lavet, og funktionaliteten bag dette sker i .cs filerne, som er tilhørende XAML filerne. I .cs filerne bliver de forskellige metoder til fx. knapper, animationer mm., implementeret. Når der trykkes på en knap i appen, så vil knappens tryk opfanges af en EventHandler i tilhørende .cs fil, og denne EventHandler kan så eksekvere noget kode.

Derudover, så gør vi meget brug af DataBinding i XAML, for at holde os indenfor rammerne i vores design pattern (MVVM).

Følgende er en metodebeskrivelse for alle de metoder som er i .cs filerne i GUI klasserne (View klasserne).

6.1.2 LoginPage.xaml.cs

```
1 | private void ShowPass_OnTapped(object sender, EventArgs e) |
```

Parametre: object, EventArgs

Returværdi: Intet

Beskrivelse: EventHandler som bruges til at vise/skjule password som er skrevet ved at ændre på IsPassword-parameteret i Placeholderen for passwordet.

```
1 | private async void CreateUser_Clicked(object sender, EventArgs e) |
```

Parametre: object, EventArgs

Returværdi: Intet

Beskrivelse: EventHandler som kalder PushModalAsync for at navigere til NewUserPage.

6.1.3 LoginPage.xaml.cs

```
1 | private async void Login_Clicked(object sender, EventArgs e) |
```

Parametre: object, EventArgs

Returværdi: Intet

Beskrivelse: EventHandler som kalder PushModalAsync for at navigere til LoginFormPage.

```
1 | private async void Opret_Bruger_Clicked(object sender, EventArgs e)
```

Parametre: object, EventArgs

Returværdi: Intet

Beskrivelse: EventHandler som kalder PushModalAsync for at navigere til NewLoginPage.

6.1.4 MainPage.xaml.cs

```
1 | private async void hiscore_OnTapped(object sender, EventArgs e)
```

Parametre: object, EventArgs

Returværdi: Intet

Beskrivelse: EventHandler som kalder PushModalAsync for at navigere til HiscoreWindow.

```
1 | private async void quiz_OnTapped(object sender, EventArgs e)
```

Parametre: object, EventArgs

Returværdi: Intet

Beskrivelse: EventHandler som laver en ny SearchQuizPageSelectCategory, og navigerer, via. PushModalAsync, til den som en NavigationPage.

```
1 | private async void spil_OnTapped(object sender, EventArgs e)
```

Parametre: object, EventArgs

Returværdi: Intet

Beskrivelse: EventHandler som kalder PushModalAsync for at navigere til SpilPage.

```
1 | private async void settings_OnTapped(object sender, EventArgs e)
```

Parametre: object, EventArgs

Returværdi: Intet

Beskrivelse: EventHandler som laver en ny SettingsPage, og navigerer, via. PushModalAsync, til den som en NavigationPage.

```
1 | private async void omos_OnTapped(object sender, EventArgs e)
```

Parametre: object, EventArgs

Returværdi: Intet

Beskrivelse: EventHandler som kalder PushModalAsync for at navigere til OmOsPage.

6.1.5 NewUserPage.xaml.cs

```
1 | private async void GoBack_Clicked(object sender, EventArgs e)
```

Parametre: object, EventArgs

Returværdi: Intet

Beskrivelse: EventHandler som kalder PopModalAsync for at navigere tilbage til MainPage.

6.1.6 OmOsPage.xaml.cs

```
1 | private async void ScrollViewTo(object sender, EventArgs e)
```

Parametre: object, EventArgs

Returværdi: Intet

Beskrivelse: EventHandler som programmatisk scroller ned til det givne Y-koordinat vha. funktionen ScrollToAsync.

6.1.7 QuizConclusionPage.xaml.cs

```
1 | public void goback(object sender, EventArgs e)
```

Parametre: object, EventArgs

Returværdi: Intet

Beskrivelse: EventHandler navigere tilbage til MainPage.

6.1.8 QuizPageDemo.xaml.cs

```
1 | async void onQuizCompleted(object sender, EventArgs e)
```

Parametre: object, EventArgs

Returværdi: Intet

Beskrivelse: EventHandler navigere tilbage til QuizConlusionPage.

6.1.9 SearchQuizPageSelectCategory.xaml.cs

```
1 | public SearchQuizPageSelectCategory(SearchQuizViewModel viewModel = null)
```

Parametre: SearchQuizViewModel

Returværdi: Intet

Beskrivelse: Constructor, hvis en SearchQuizViewModel bliver givet, gemmes den i en privat variabel, ellers oprettes en ny. Derefter sættes dataContext til view model'en, til sidst loades kategorierne.

```
1 | protected async void listCategorySelected(object sender, SelectedItemChangedEventArgs e)
```

Parametre: object, SelectedItemEventArgs

Returværdi: Intet

Beskrivelse: EventHandler for valgt list item. Navigere til SearchQuizPageSelecQuiz med den valgte kategori som parameter.

6.1.10 SearchQuizPageSelectQuiz.xaml.cs

```
1 | public SearchQuizPageSelectQuiz(SearchQuizViewModel viewModel , string category =null)
```

Parametre: SearchQuizViewModel, string

Returværdi: Ingen

Beskrivelse: Constructor, SearchQuizViewModel fra parameterlisten gemmes den i en privat variabel, Derefter sættes dataContext til view model'en, til sidst loades quizzerne for den kategori, som gives i parameterlisten.

```
1 | protected async void listQuizSelected(object sender, SelectedItemChangedEventArgs e)
```

Parametre: object, SelectedItemChangedEventArgs

Returværdi: ingen

Beskrivelse: EventHandler for valgt list item. Navigere til QuizPageDemo med den valgte quiz som parameter.

SettingsPage.xaml.cs

```
1 | private void Button_Clicked(object sender, EventArgs e)
```


Parametre: object, EventArgs

Returværdi: Ingen

Beskrivelse: Kalder PopupNavigation.Instance.PushAsync, som laver et popup-vindue hvori man kan ændre sit login.

```
1 | private void ResetHighscore(object sender, EventArgs e)
```

Parametre: object, EventArgs

Returværdi: Ingen

Beskrivelse: Kalder PopupNavigation.Instance.PushAsync, som laver et popup-vindue hvori man kan resette sin highscore.

```
1 | private void Delete_OnClicked(object sender, EventArgs e)
```

Parametre: object, EventArgs

Returværdi: Ingen

Beskrivelse: Kalder PopupNavigation.Instance.PushAsync, som laver et popup-vindue hvori man kan slette sin bruger.

6.2 Login Feature - GUI

6.2.1 NewUserViewModel

Navn	Type	Funktionalitet
Email	String	Anvendes til at registrere brugerens ønskede email
Username	String	Anvendes til at registrere brugerens ønskede brugernavn
Password	String	Anvendes til at registrere brugerens ønskede kodeord
ConfirmPassword	String	Anvendes til at bekræfte brugerens ønskede kodeord
message	String	Anvendes til at binde en besked til view. Når denne ændres kaldes OnPropertyChanged() metoden så der opdateres til viewet
NewUserCommand	ICommand	Når der kaldes get på denne metode valideres på en række aspekter. Herunder om der er forbindelse, om brugerinformationerne er udfyldt korrekt og der eventuelt er udfyldt forkert. Herefter registreres bruger med RegisterAsync og brugeren gives besked herom

Tabel 6.1. Variabler i NewUserViewModel klassen

```
1 | public bool IsEverythingFilled()
```

Parametre: Ingen

Returværdi: bool

Beskrivelse: Metoden tjekker om brugeren har udfyldt de ønskede informationer korrekt.

```
1 public string WhatIsNotFilled()
```

Parametre: Ingen

Returværdi: String

Beskrivelse: Metoden hvilke informationer brugeren har udfyldt forkert og der returneres en besked herom.

```
1 [NotifyPropertyChangedInvoker]
2 protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)
```

Parametre: CallerMemberNameAttribute som tillader os at finde metode eller variablenavn som kalder metoden.

Returværdi: Ingen

Beskrivelse: Giver besked om hvorvidt noget har ændret sig eller ej.

6.2.2 LoginViewModel.cs

Navn	Type	Funktionalitet
Username	String	Indeholder username for brugeren der vil logge ind.
Password	String	Indeholder password for brugeren der vil logge ind.
LoginResponse	String	Sætter _loginresponse og kalder PropertyChanged eventet
LoginCommand	ICommand	Indeholder logikken for login, commanden bliver eksekveret når der bliver tappet på log ind.
PropertyChanged	event	PropertyChanged
_apiServices	ApiServices	Instantiere et nyt objekt af ApiService
_loginresponse	string	Indeholder en besked til brugeren med evt. fejl i login, ingen internetforbindelse eller succesfuldt login.

Tabel 6.2. Variabler i LoginViewModel klassen

```
1 LoginViewModel()
```

Parametre: ingen

Returværdi: Ingen

Beskrivelse: Intalisere konstrukteren

```
1 public bool IsEverythingFilled()
```

Parametre: ingen

Returværdi: bool

Beskrivelse: Tjekker at der både er kommet et password og et brugernavn.

```
1 public string WhatIsNotFilled()
```

Parametre: ingen

Returværdi: Ingen

Beskrivelse: Fortæller hvis brugernavn og password ikke er fyldt

```
1 | virtual void \#OnPropertyChanged ()
```

Parametre: ingen

Returværdi: Ingen

Beskrivelse: Sender et event hvis kaldt

6.2.3 ResetHighscoreViewModel

```
1 | public ICommand ResertHighscoreCommand {get; }
```

Beskrivelse: Indeholder logikken for at slette highscoren. Kommandoen bliver eksekveret når der bliver trykket på "reset highscore" i settingsmenuen. Når den trykkes, slettes en userss highscore på databasen.

```
1 | public string Response { get; set; }
```

Beskrivelse: Bliver oprettet til at indeholde den tekststreng, der fortæller om der er oprettet forbindelse til User databasen, når ResertHighscore kommandoen bliver udført.

```
1 | private readonly ApiService
```

Beskrivelse: Bliver anvendt for at oprette en ny ApiService, så klassen kan tilgås.

6.2.4 LogOutViewModel

```
1 | public ICommand LogOutCommand {get; }
```

Beskrivelse: Indeholder logikken for at logge ud. Kommandoen bliver eksekveret når der bliver trykket på "log ud" i mainmenuen. Når den trykkes, logger en bruger ud.

```
1 | public string Response { get; set; }
```

Beskrivelse: Bliver oprettet til at indeholde den tekststreng, der fortæller om der er oprettet forbindelse til User databasen, når ResertHighscore kommandoen bliver udført.

```
1 | private readonly ApiService
```

Beskrivelse: Bliver anvendt for at oprette en ny ApiService, så klassen kan tilgås.

6.2.5 NewUserBindingModel.cs

```
1 | public string Email { get; set; }
```

Beskrivelse: set sætter string Email = value. Get returnerer 'string Email'. Email bruges til at gemme en e-mailadresse, når der oprettes en ny bruger.

```
1 | public string Username { get; set; }
```

Beskrivelse: set sætter string Username = value. Get returnerer 'string Username'. Username bliver brugt til at gemme et brugernavn, når der oprettes en ny bruger.

```
1 | public string Password { get; set; }
```

Beskrivelse: set sætter string Password = value. Get returnerer 'string Password'. Password bruges til at gemme et kodeord, når der oprettes en ny user.

```
1 | public string ConfirmPassword { get; set; }
```

Beskrivelse: Set sætter string ConfirmPassword = value. Get returnerer 'string ConfirmPassword'. ConfirmPassword bruges til at gemme det bekræftende spørgsmål, når der skal oprettes en user.

6.2.6 DeleteUserViewModel

```
1 | public ICommand DeleteUserCommand {get; }
```

Beskrivelse: Indeholder logikken for at slette user. Kommandoen bliver eksekveret når der bliver trykket på "Slet bruger" i settings menuen. Når den trykkes, slettes useren.

```
1 | public string Response { get; set; }
```

Beskrivelse: Bliver oprettet til at indeholde den tekststreng, der fortæller om der er oprettet forbindelse til User databasen, når ResetHighscore kommandoen bliver udført.

```
1 | private readonly ApiService
```

Beskrivelse: Bliver anvendt for at oprette en ny ApiService, så klassen kan tilgås.

6.2.7 ApiService.cs

```
1 | internal async Task<bool> RegisterAsync(string username, string email, string password, string  
    confirmPassword)
```

Parametre: string, string, string, string

Returværdi: bool

Beskrivelse: Opretter en ny user på User Databasen. Dataen useren får er hhv. et username, en email og et password, hvor username og password, bliver anvendt til at logge ind.

```
1 | public async Task<string> LoginAsync(string username, string password)
```

Parametre: string, string

Returværdi: string

Beskrivelse: Sender en request til userdatabasen. I requesten er der tre parametre, username, password og grant_type. Herefter kommer User Databaen med en response der indeholder et access token, token type, username, udløbsdatoen og udstedsdato. Hvis Api kaldet returnere en 2xx kode, gemmes udløbsdatoen i settings og accesstoken bliver returneret som en string.

```
1 | public async Task<List<CategoryScoreModel>> GetHighScoreForQuiz()
```

Parametre: Ingen

Returværdi: List<CategoryScoreModel>

Beskrivelse: Henter alle highscores , og returnerer dem i listen "highscores".

```
1 | public async Task<bool> PostHighscore(CategoryScoreModel model)
```

Parametre: CategoryScoreModel

Returværdi: bool

Beskrivelse: Funktion der bliver brugt til at ligge en ny highscore op på User databasen.

```
1 | public async Task<bool> PutHighscore(CategoryScoreModel model)
```

Parametre: CategoryScoreModel

Returværdi: bool

Beskrivelse: Opdaterer en allerede eksisterende highscore på User databasen og returnerer en statuskode for hvordan opdateringen er gået.

```
1 | public async Task<bool> DeleteHighscores()
```

Parametre: Ingen

Returværdi: bool

Beskrivelse: Sletter alle highscores, på user databasen, for den user der er logget ind.

```
1 | public async Task<bool> ChangePassword(ChangePasswordBindingModel model)
```

Parametre: ChangePasswordBindingModel

Returværdi: bool

Beskrivelse: Bliver brugt til at opdatere et password, på user databasen, for den user der er logget ind.

```
1 | public async Task<bool> DeleteUser()
```

Parametre: Ingen

Returværdi: bool

Beskrivelse: Sletter den user der er logget ind, samt userens data.

6.2.8 QuizCategoryScoreBindingModel.cs

```
1 | public int Id { get; set; }
```

Parametre: get, set

Returværdi: get returnerer 'int Id'

Beskrivelse: set sætter int Id = value.

```
1 | public string Category { get; set; }
```

Parametre: get, set

Returværdi: get returnerer 'string Category'

Beskrivelse: set sætter string HighScore = value.

```
1 | public int HighScore { get; set; }
```

Parametre: get, set

Returværdi: get returnerer 'int HighScore'

Beskrivelse: set sætter int HighScore = value.

```
1 | public int UserId { get; set; }
```

Parametre: get, set

Returværdi: get returnerer 'int UserId'

Beskrivelse: set sætter int UserId = value.

6.2.9 CategoryScoreModel.cs

```
1 | public int Id { get; set; }
```

Beskrivelse: Set sætter int Id = value. Get returnerer 'int Id'. Bruges til at gemme Id'et for useren.

```
1 | public string Category { get; set; }
```

Beskrivelse: Set sætter string Category = value. Get returnerer 'string Category'. Anvendes til at gemme den kategori, useren er ved at få en highscore i.

```
1 | public int HighScore { get; set; }
```

Beskrivelse: Set sætter int HighScore = value. Get returnerer 'int HighScore'. Bruges til at gemme en highscore i en kategori.

```
1 | public bool IsTotalHighscore { get; set; }
```

Beskrivelse: Set sætter string IsTotalHighscore = value. Get returnerer 'bool IsTotalHighscore'. IsTotalHighscore fortæller om den highscore vi har fat i, er den totale highscore for useren.

```
1 | public string UserId { get; set; }
```

Beskrivelse: Set sætter string UserId = value. Get returnerer 'string UserId'. UserId bliver brugt til at hente en user, når der skal sættes en Hiscore for en user, i en bestemt kategori.

6.2.10 ChangePasswordBindingModel.cs

Denne klasse bruges til at give brugeren muligheden for at skifte sit password til et nyt.

Navn	Type	Funktionalitet
OldPassword	String	Anvendes til at validere brugerens gamle password, som sikkerhed for at lave et nyt.
NewPassword	String	Anvendes til at registrere brugerens nye ønskede password.
ConfirmPassword	String	Anvendes til at gentage det ønskede password for at sikre sig de er ens.

```
1 | public string OldPassword { get; set; }
```

Parametre: get, set

Returværdi: get returnerer 'string OldPassword'

Beskrivelse: Set sætter string OldPassword = value. Get returnerer 'string OldPassword'. OldPassword bruges til at gemme en users nuværende password, når en bruger ønsker at skifte password for sin user.

```
1 | public string NewPassword { get; set; }
```

Beskrivelse: Set sætter string NewPassword = value. Get returnerer 'string NewPassword'. NewPassword bruges til at gemme det nye kodeord, når en bruger ønsker at oprette et nyt kodeord til sin user.

```
1 | public string ConfirmPassword { get; set; }
```

Beskrivelse: set sætter string ConfirmPassword = value. Get returnerer 'string ConfirmPassword'. ConfirmPassword bruges til at gemme det bekræftende kodeord, når der skal ændres kodeord for user.

6.2.11 NewUserBindingModel.cs

```
1 | public string Email { get; set; }
```

Beskrivelse: set sætter string Email = value. Get returnerer 'string Email'. Email bruges til at gemme en e-mailadresse, når der oprettes en ny bruger.

```
1 | public string Username { get; set; }
```

Beskrivelse: set sætter string Username = value. Get returnerer 'string Username'. Username bliver brugt til at gemme et brugernavn, når der oprettes en ny bruger.

```
1 | public string Password { get; set; }
```

Beskrivelse: set sætter string Password = value. Get returnerer 'string Password'. Password bruges til at gemme et kodeord, når der oprettes en ny user.

```
1 | public string ConfirmPassword { get; set; }
```

Beskrivelse: Set sætter string ConfirmPassword = value. Get returnerer 'string ConfirmPassword'. ConfirmPassword bruges til at gemme det bekræftende spørgsmål, når der skal oprettes en user.

6.2.12 CategoryScoreModel.cs

```
1 | public int Id { get; set; }
```

Beskrivelse: Set sætter int Id = value. Get returnerer 'int Id'. Bruges til at gemme Id'et for useren.

```
1 | public string Category { get; set; }
```

Beskrivelse: Set sætter string Category = value. Get returnerer 'string Category'. Anvendes til at gemme den kategori, useren er ved at få en highscore i.

```
1 | public int HighScore { get; set; }
```

Beskrivelse: Set sætter int HighScore = value. Get returnerer 'int HighScore'. Bruges til at gemme en highscore i en kategori.

```
1 | public bool IsTotalHighscore { get; set; }
```

Beskrivelse: Set sætter string IsTotalHighscore = value. Get returnerer 'bool IsTotalHighscore'. IsTotalHighscore fortæller om den highscore vi har fat i, er den totale highscore for useren.

```
1 | public string UserId { get; set; }
```

Beskrivelse: Set sætter string UserId = value. Get returnerer 'string UserId'. UserId bliver brugt til at hente en user, når der skal sættes en Hiscore for en user, i en bestemt kategori.

6.2.13 User.cs

```
1 | public int UserHiscore { get; set; }
```

Beskrivelse: set sætter int UserHiscore = value. get returnerer 'int UserHiscore'. UserHiscore bruges til at gemme en users Hiscore.

```
1 | public virtual int UserID { get; set; }
```

Beskrivelse: set sætter int UserID = value. get returnerer 'int UserID'. UserID bruges til at gemme en users ID.

```
1 | public virtual string Username { get; set; }
```

Beskrivelse: set sætter string Username = value. get returnerer 'string Username'. Username bruges til at gemme en users brugernavn.

```
1 | public virtual string Password { get; set; } |
```

Beskrivelse: set sætter string Password = value. get returnerer 'string Password'. Password bruges til at gemme en users kodeord.

```
1 | public int fontSizeforUser { get; set; } |
```

Beskrivelse: set sætter string fontSizeforUser = value. get returnerer 'int fontSizeforUser'. FonSizeforUser, bruges til at gemme skriftstørrelsen på en bestemt users Hiscore.

```
1 | public int calculateFontSize() |
```

Parametre: Ingen

Returværdi: int newsize

Beskrivelse: Udregner en ny fontstørrelse, der bliver anvendt i highscore. Jo højere score, jo større fontsize.

```
1 | public User(string _username, string _password) |
```

Parametre: string, string

Returværdi: Intet

Beskrivelse: Konstruktor for User.

6.3 Login Feature - database

6.3.1 CategoryScoresController.cs

```
1 | public IQueryable<CategoryScore> GetCategoryScoresForCurrentUser() |
```

Parametre: Ingen

Returværdi: IQueryable<CategoryScore>

Beskrivelse: Bliver kaldt med et authorizationtoken i header, som bilver brugt til userclaim. Funktionen retunere en liste af scorere som tilhører brugeren, hvis authorizationtoken er givet med i headeren.

```
1 | public IQueryable<CategoryScore> GetCategoryScores() |
```

Parametre: ingen

Returværdi: IQueryable<CategoryScore>

Beskrivelse: Retunere alle scorer på databasen.

```
1 | public IHttpActionResult GetCategoryScore(int id) |
```

Parametre: int

Returværdi: IHttpActionResult

Beskrivelse: Getter en CategoryScore med Id tilsvarende parameteren

```
1 | public IHttpActionResult PutCategoryScore(int id, CategoryScore categoryScore) |
```

Parametre: int, CategoryScore

Returværdi: IHttpActionResult

Beskrivelse: Oprette en CategoryScore med id tilsvarende parameteren

```
1 | public IHttpActionResult PostCategoryScore(CategoryScore categoryScore) |
```

Parametre: CategoryScore

Returværdi: IHttpActionResult

Beskrivelse: Oprette en ny CategoryScore på databasen

```
1 | public IHttpActionResult DeleteCategoryScore(int id) |
```

Parametre: int

Returværdi: IHttpActionResult

Beskrivelse: Sletter en CategoryScore med id tilsvarende parameteren

```
1 | private bool CategoryScoreExists(int id) |
```

Parametre: int

Returværdi: bool

Beskrivelse: Tjekker om der findes en CategoryScore med id tilsvarende parameteren

6.4 Quiz Services

6.4.1 QuizDBServices.cs

```
1 | public QuizDBServices()
```

Parametre: Ingen

Returværdi: Intet

Beskrivelse: Konstruktor for QuizDBServices.

```
1 | public async Task<List<Quiz>> GetAllQuizzesAsync()
```

Parametre: Ingen

Returværdi: null eller List<Quiz> Quizzes

Beskrivelse: Henter alle quizzer på quiz databasen, og returnerer en liste af disse quizzer, hvis funktionen modtager true fra "IsSuccessStatusCode". Hvis "IsSuccessStatusCode" er null, returnerer funktionen null.

```
1 | public async Task<List<Quiz>> GetQuizzesByCategoryAsync(string category)
```

Parametre: String

Returværdi: null eller List<Quiz> Quizzes

Beskrivelse: Henter en liste af quizzer med samme kategori fra quiz databasen, og returnerer en liste af disse quizzer, hvis funktionen modtager true fra "IsSuccessStatusCode". Hvis "IsSuccessStatusCode" er null, returnerer funktionen null.

6.5 Quiz Feature: QuizModel

6.5.1 PropertyChangedModel.cs

Som det blev nævnt under design afsnittet er der til Quiz Feature blevet implementeret en data model til definition af De quizzer som skal fremvises i appen, modellen kunne ses figur 5.14. Følgende er implementeringen af disse klasser:

PropertyChanged er implementeringen af INotifyPropertyChanged interfacet, og de andre 3 Quiz model klasser nedarver fra denne klasse. Følgende er metodebeskrivelserne for PropertyChangedModel-klassen:

```
1 | protected void HandlePropertyChanged([CallerMemberName]string propertyName = "")
```

Parametre: string

Returværdi: Intet

Beskrivelse: Invoker PropertyChangedEventHandler med et nyt PropertyChangedEventArgs med string propertyName som parameter. - Implementerer INotifyPropertyChanged interfacet.

Option-klassen definerer svarmuligheder til et givet spørgsmål i en quiz. Dette består af en tekst, det egentlige svar, og en boolean for om svaret er korrekt. Følgende er metodebeskrivelserne for Option-klassen:

6.5.2 Option.cs

```
1 | public bool IsRight { get; set; }
```

Beskrivelse: Set kaldes kun, hvis value er anderledes fra den gemte værdi for propertyen. Herefter kaldes HandlePropertyChanged. Get returnerer 'bool IsRight'. IsRight fortæller om en valgmulighed er den rigtige valgmulighed til et givent spørgsmål. Hvis IsRight = true, er denne valgmulighed det rigtige svar til spørgsmålet. Hvis IsRight = false, er denne valgmulighed det forkerte svar til spørgsmålet.

```
1 | public string OptionText { get; set; }
```

Beskrivelse: Set kaldes kun, hvis value er anderledes fra den gemte værdi for propertyen. Herefter kaldes HandlePropertyChanged. Get returnerer 'string OptionText'. OptionText bruges til at gemme teksten for en valgmulighed.

6.5.3 Question.cs

Question definerer spørgsmålene i quiz, og er dennes egentlige indhold. Question består tekst, det egentlige spørgsmål, en scoreværdi og en liste af svarmuligheder svarmulighederne. - I designet af quizen er der for hvert spørgsmål inkluderet 4 svarmuligheder. Følgende er metodebeskrivelserne for Question-klassen:

```
1 | public Question()
```

Parametre: Ingen

Returværdi: Intet

Beskrivelse: Kontruktor for Question.

```
1 | public string QuestionText { get; set; }
```

Beskrivelse: Set kaldes kun, hvis value er anderledes fra den gemte værdi for propertyen. Herefter kaldes HandlePropertyChanged. Get returnerer 'string QuestionText'. QuestionText bruges til at gemme teksten til et spørgsmål.

```
1 | public int Score { get; set; }
```

Beskrivelse: Set kaldes kun, hvis value er anderledes fra den gemte værdi for propertyen. Herefter kaldes HandlePropertyChanged. Get returnerer 'int Score'. Score bruges til at gemme et spørgsmåls score.

```
1 | public List<Option> Options { get; set; }
```

Beskrivelse: Set kaldes kun, hvis value er anderledes fra den gemte værdi for propertyen. Herefter kaldes HandlePropertyChanged. Get returnerer 'List<Option> Options'. Options bruges til at gemme 4 valgmuligheder, for et specifikt spørgsmål.

```
1 | public void RandomizeOptionOrder()
```

Parametre: Ingen

Returværdi: Intet

Beskrivelse: Bliver anvendt til at tilfældiggøre rækkefølgen af Options.

6.5.4 Quiz.cs

Quiz klassen er den primære entity i vores model, den består af et navn, en kategori og en liste af spørgsmål. Følgende er metodebeskrivelserne for Quiz-klassen:

```
1 | public Quiz()
```

Parametre: Ingen

Returværdi: Intet

Beskrivelse: Konstruktor for Quiz.

```
1 | public string Id { get; set; }
```

Beskrivelse: Set kaldes kun, hvis value er anderledes fra den gemte værdi for propertyen. Herefter kaldes HandlePropertyChanged. Get returnerer 'string Id'. Id bruges til at gemme et ID for en specifik quiz.

```
1 | public string Category { get; set; }
```

Beskrivelse: Set kaldes kun, hvis value er anderledes fra den gemte værdi for propertyen. Herefter kaldes HandlePropertyChanged. Get returnerer 'string Category'. Category bruges til at gemme en kategori på en quiz.

```
1 | public string QuizName { get; set; }
```

Beskrivelse: Set kaldes kun, hvis value er anderledes fra den gemte værdi for propertyen. Herefter kaldes HandlePropertyChanged. Get returnerer 'string QuizName'. Quizname bruges til at gemme navnet på en quiz.

```
1 | public List<Question> Question { get; set; }
```

Beskrivelse: Set kaldes kun, hvis value er anderledes fra den gemte værdi for propertyen. Herefter kaldes HandlePropertyChanged. get returnerer 'List<Question> Question'. Bruges til at gemme en liste af spørgsmål, i quizklassen.

```
1 | public Question NextQuestion()
```

Parametre: Ingen

Returværdi: Question nextQuestion eller null

Beskrivelse: Returnerer null, hvis der ikke er flere Questions i arrayet. Returnerer det Question nextQuestion, hvis der er flere Questions i arrayet.

```
1 | public void RandomizeQuestionOrder()
```

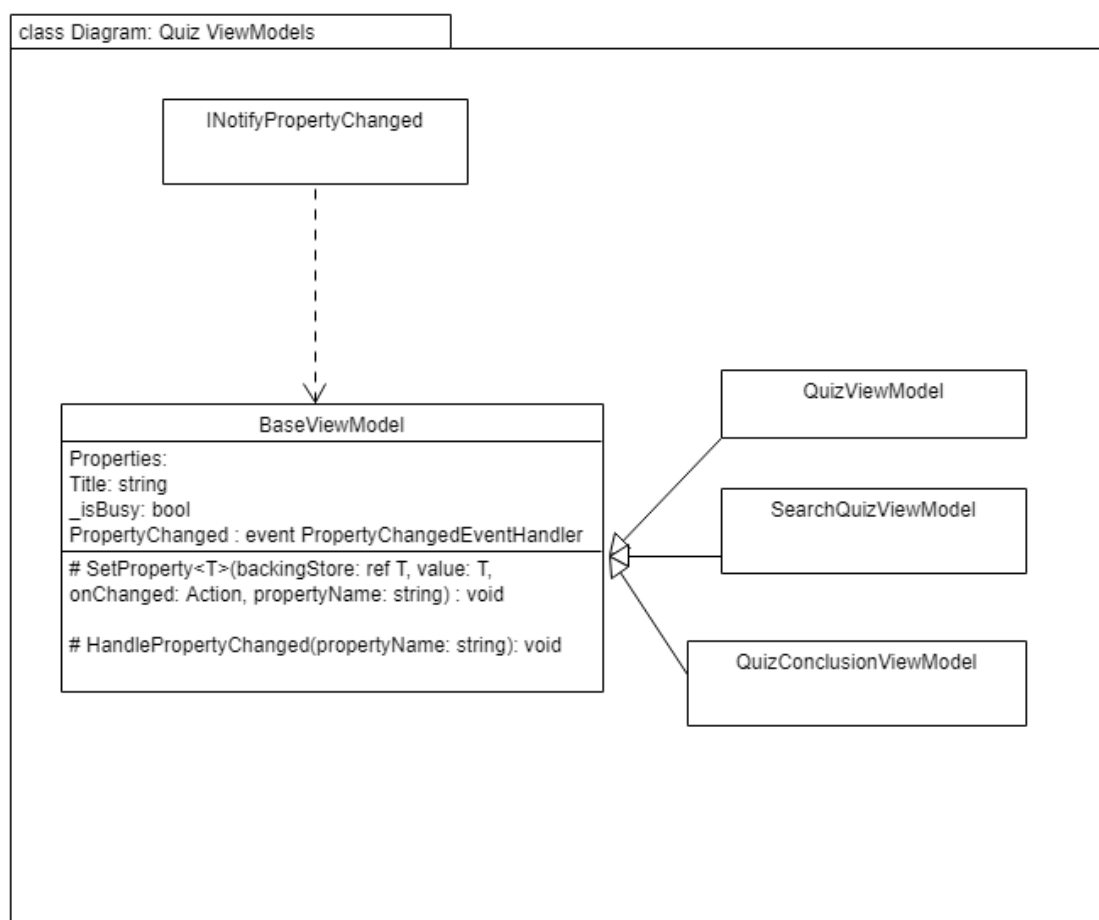
Parametre: Ingen

Returværdi: Intet

Beskrivelse: Bliver anvendt til at tilfældiggøre rækkefølgen af Questions.

6.6 Quiz Feature: MVVM Implementering

Der er tre primære koncepter der hjælper os med at implementere denne overstående MVVM-arkitektur, og den lave kobling mellem view-laget og model-laget gennem viewModels. Disse er databinding, PropertyChanged events (implementering af INotifyPropertyChanged) og Commands. I XAML filerne for quiz features views databindes GUI-elementer til properties udstillet fra viewModels. Gennem PropertyChanged events, vil disse elementer gennem viewModel blive opdateret, hvis de bundne properties ændres. Ligeledes kan de avendes til at manipulere modellen, igen gennem ViewModels. Commands tillader os ligeledes gennem XAML-filerne at delegere til metodekald i vores viewModel. På figur 6.1 kan et klassediagram for Quiz ViewModel-klasserne ses.



Figur 6.1. Klassediagram for Quiz ViewModels

6.6.1 QuizViewModel - DataBinding, PropertyChanged Events og Commands

Herunder vil vi anvende QuizDemoPage viewet og QuizViewModel, som eksempel til at forklare disse koncepter.

BaseViewModel.cs

BaseViewModel, hvis UML-beskrivelse kan ses på figur 6.1, er implementeringen af `INotifyPropertyChanged` interfacet for ViewModel-klasserne, og de andre 3

viewModels nedarver fra denne klasse. Klassen stammer fra kode eksemplet givet i "QuickStart: Build a MongoDB API Xamarin.Forms app with .NET and the Azure portal".¹ Denne guide har været inspiration til udvikling af ViewModel-klasserne. Følgende er metodebeskrivelserne for BaseViewModel-klassen:

```
1 | protected void SetProperty<T>(ref T backingStore, T value, Action onChanged = null,
    | [CallerMemberName] string propertyName = "")
```

Parametre: ref T, T, Action, string

Returværdi: Intet

Beskrivelse: Sammenligner to objekter. Er de ikke identiske, kaldes HandlePropertyChanged metoden på det objekt der gives til metoden til med ref keyword (sådan denne reference ikke er const). Dette gør det muligt at sammenligne komplekse objekter, som Quiz med en liste af Question.

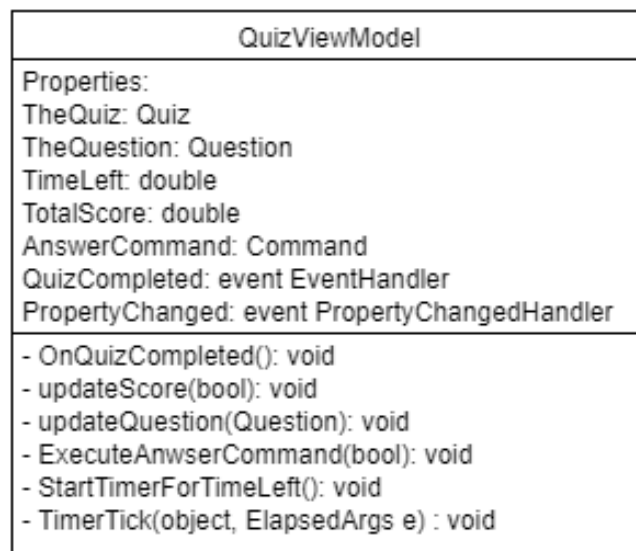
```
1 | protected void HandlePropertyChanged(string propertyName = "") =>
```

Parametre: string

Returværdi: Intet

Beskrivelse: HandlePropertyEventHandler, sender et changedProperty event, med et nyt PropertyChangedEventHandler objekt med propertyName som parameter.

QuizViewModel.cs



Figur 6.2. UML for QuizViewModel

¹<https://bit.ly/2QVc4Fa> - QuickStart: Build a MongoDB API Xamarin.Forms app with .NET and the Azure portal, BaseViewModel

QuizViewModel er den klasse, som er ansvarlig for præsentation af den Quiz som er blevet videresendt fra SearchQuizViewModel. Neden for følger metodebeskrivelserne for QuizViewModelklassen som ses på figur 6.2:

```
1 | public Quiz TheQuiz {get; set; }
```

Beskrivelse: Indeholder en bestemt quiz, med tilhørende data.

```
1 | public Question TheQuestion {get; set; }
```

Beskrivelse: Indeholder et bestemt spørgsmål, med tilhørende data.

```
1 | public Command AnswerCommand {get; set; }
```

Beskrivelse: Bruges til at svare i quizen.

```
1 | public double Timeleft {get; set; }
```

Beskrivelse: Bruges til at gemme den tid der er tilbage af spørgsmålet i quizen.

```
1 | public double TotalScore {get; set; }
```

Beskrivelse: TotalScore bruges til at gemme den totale score for en quiz.

```
1 | public QuizViewModel(Quiz quiz)
```

Parametre: Quiz

Returværdi: Ingen

Beskrivelse: Konstruktor for Quiz.

```
1 | protected virtual void OnQuizCompleted()
```

Parametre: Ingen

Returværdi: Intet

Beskrivelse: Kaldes når en Quiz er afsluttet. Sender et event til QuizPageDemo for at trigger view skifte til QuizConclusionPage.

```
1 | void ExcuteAnswerCommand(bool isRightAnswer)
```

Parametre: bool

Returværdi: Ingen

Beskrivelse: Kalder updateScore med same parameter som metoden selv kaldes med. Derefter udhentes det næste spørgsmål fra TheQuiz med dennes NextQuestion metode. Hvis der ikke returneres null tilføjes rækkefølgen af det hentede Question objekts Options, med Questions RandomizeOptionOrder og TheQuestion opdateres til det nye Question objekt. - Returneres der null fra GetNextQuestion kaldet, kaldes OnQuizCompleted i stedet for at trigger view skifte.

```
1 | void updateScore(bool isRightAnswer)
```

Parametre: bool

Returværdi: Ingen

Beskrivelse: Opdaterer TotalScore afhængig af parameteren. Scoren opdateres kun, hvis parameteren er true.

```
1 | void updateQuestion(Question newQuestion)
```

Parametre: Question

Returværdi: Ingen

Beskrivelse: Opdaterer alle TheQuestion properties til at matche dem i newQuestion. Desuden resitter den property'en TimeLimit til 1.

```
1 | void startTimerForTimeLeft()
```

Parametre: Ingen

Returværdi: Ingen

Beskrivelse: Starter en timer, med et interval på 500, for denne subscribes der med en ElapsedTimeEventHandler, som kalder metoden TimerTick.

```
1 | void TimerTick(object sender, ElapsedEventArgs e)
```

Parametre: object, ElapsedEventArgs

Returværdi: Ingen

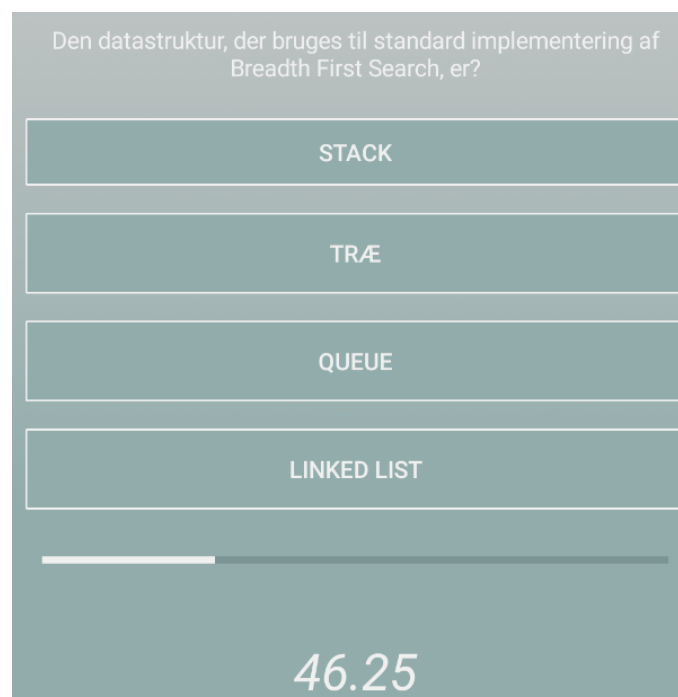
Beskrivelse: Dette er eventHandler metoden for Timer events. Hvis TimeLeft er større end 0 trækkes der 0.025 fra TimeLeft, ellers kaldes ExecuteAnswerCommand med false som parameter.

Som det kan ses på figur 6.3 binder Text-property'en for en label i dette view til QuestionText property'en for QuizViewModelens TheQuestion property. På samme måde binder vi for 4 knapper (2 ses på billedet) til indexes i TheQuestions liste af Options. Gennem PropertyChanged event opdateres alle disse elementer tekst, når viewModellen ændrer sin TheQuestion property til et nyt spørgsmål. På samme måde kan det ses, at der for knapper bindes til ViewModellens AnswerCommand, når der trykkes på disse. Desuden bindes enkeltes Options IsRight property som parametre til dette. Denne Command delegerer til QuizViewModels ExecuteAnswerCommand, som tager en bool som parameter, og validerer om der svaret korrekt. Resultat er, at den eneste kode, vi har behov for i code-behind filerne er at Views DataContext sættes til den relevante viewModel. På figur 6.4 kan det endelige udtryk ses:

```
<Label Grid.Row="1"
      Grid.RowSpan="1"
      Grid.ColumnSpan="3"
      HorizontalTextAlignment="Center"
      VerticalTextAlignment="Center"
      Margin="10,10,10,10"
      Text="{Binding Path=TheQuestion.QuestionText}"
      TextColor="White"/>

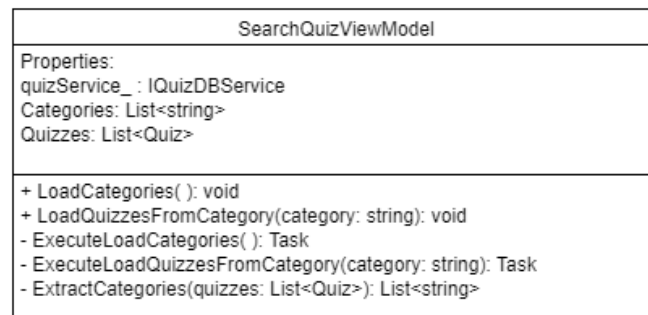
<Button Grid.Row="2" Grid.ColumnSpan="3"
        TextColor="White"
        Text="{Binding TheQuestion.Options[0].OptionText}"
        BackgroundColor="#9ab7b6"
        Margin="10,5,10,5"
        Command="{Binding AnswerCommand}" CommandParameter="{Binding TheQuestion.Options[0].IsRight}"
        BorderWidth="1" BorderRadius="1" BorderColor="White"/>
<Button Grid.Row="3" Grid.ColumnSpan="3"
        TextColor="White"
        Text="{Binding Path=TheQuestion.Options[1].OptionText}"
        BackgroundColor="#9ab7b6"
        Margin="10,5,10,5"
        Command="{Binding AnswerCommand}" CommandParameter="{Binding TheQuestion.Options[1].IsRight}"
        BorderWidth="1" BorderRadius="1" BorderColor="White"/>
```

Figur 6.3. Kode snippet fra QuizPageDemo, eksempel på databinding for quiz featuren Views



Figur 6.4. Demonstration af databinding til gui-elementer for QuizPageDemo View

6.6.2 SearchQuizViewModel



Figur 6.5. UML for SearchQuizViewModel

SearchQuizViewModel er den klasse, som er ansvarlig for at tilgå Quiz databasen via QuizDBService-klassen, som set på figur 6.1, samt præsentere de kategorier og quizzes, der hentes derfra. Neden for følger metodebeskrivelserne for SearchQuizViewModelklassen som ses på figur 6.5

```
1 public List<string> Categories {get; set; }
```

Beskrivelse: Bruges til at gemme en liste af strings, som er vores kategorier for vores quizzes.

```
1 public List<Quiz> Quizzes {get; set; }
```

Beskrivelse: Bruges til at gemme en liste af quizzes.

```
1 public SearchQuizViewModel(string title, IQuizDBService quizService)
```

Parametre: string, IQuizDBService

Returværdi: Intet

Beskrivelse: Konstruktor for SearchQuizViewModel

```
1 public async void LoadCategories()
```

Parametre: Ingen

Returværdi: Ingen

Beskrivelse: Wrapper metode for ExecuteLoadCategories.

```
1 public async void LoadQuizzesFromCategory(string category)
```

Parametre: string

Returværdi: Ingen

Beskrivelse: Wrapper metode for ExecuteLoadQuizzesFromCategory

```
1 | async Task ExecuteLoadQuizzesFromCategory(string category)
```

Parametre: string

Returværdi: Task

Beskrivelse: Asynkront kaldes GetQuizzesQuizzesByCategoryAsync på quizService_ med parameteren givet til metoden. returværdien, en liste af databasens quizzes med den valgte kategori gemmes i Quizzes.

```
1 | public async Task ExecuteLoadCategories()
```

Parametre: Ingen

Returværdi: Task

Beskrivelse: Asynkront kaldes GetQuizzesQuizzesAsync på quizService_. returværdien, en liste af databasens quizzes gemmes i en lokal variabel. Herefter kaldes ExtractCategories med denne liste som parameter. Resultatet gøres i Categories.

```
1 | List<string> ExtractCategories(List<Quiz> quizzes)
```

Parametre: List<Quiz>

Returværdi: List<string>

Beskrivelse: Ekstraherer unikke kategorier fra listen af quizzes i metodeparameteren og returnerer en liste med disse kategorier.

Følgende listing 6.1 er XAML-koden for SearchQuizPageSelectCategory, som viser hvorledes dette view databinder til viewModel:

Listing 6.1. SearchQuizPageSelectCategory.xaml - Databinding to SearchQuizView

```
1 | <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
2 |             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
3 |             x:Class="App1.View.SearchQuizPageSelectCategory"
4 |             Title="Vlg kategori"
5 |             BackgroundColor="#9ab7b6"
6 |             xmlns:ffimageloading="clr-namespace:FFImageLoading.Forms;assembly=FFImageLoading.Forms"
7 |             xmlns:xfg="clr-namespace:XFGloss;assembly=XFGloss">
8 |
9 |     <xfg:ContentPageGloss.BackgroundGradient>
10 |         <xfg:Gradient Rotation="180">
11 |             <xfg:GradientStep StepColor="LightGray" StepPercentage="0" />
12 |             <xfg:GradientStep StepColor="#9ab7b6" StepPercentage=".5" />
13 |         </xfg:Gradient>
14 |     </xfg:ContentPageGloss.BackgroundGradient>
```

```

15
16     <ContentPage.Content>
17         <Grid>
18
19             <ffimageloading:CachedImage
20                 Margin="0,0,0,0"
21                 VerticalOptions="Start"
22                 HorizontalOptions="End"
23                 Rotation="30"
24                 Opacity="0.25"
25                 Scale="2"
26                 Source = "quiz_bg"
27             />
28
29
30
31             <Grid Margin="30">
32
33                 <StackLayout HorizontalOptions="CenterAndExpand">
34                     <ActivityIndicator HorizontalOptions="Center" Color="White" IsRunning="{Binding
35                         IsBusy}" />
36                     <!--<Label VerticalTextAlignment="Center" TextColor="White">Vlg en
37                         Kategori</Label>-->
38
39                     <ListView
40                         HorizontalOptions="Center"
41                         SeparatorColor="White"
42                         ItemsSource="{Binding Categories}"
43                         x:Name="LViewCategories">
44
45                         <ListView.ItemTemplate>
46                             <DataTemplate>
47                                 <TextCell TextColor="White"
48                                     Text="{Binding .}">
49                             </TextCell>
50                         </DataTemplate>
51                     </ListView.ItemTemplate>
52                 </ListView>
53             </StackLayout>
54
55             <Image Scale="0.35" Grid.Row="1" HorizontalOptions="Center" VerticalOptions="Center"
56                 Source="Icon" />
57
58         </Grid>
59     </ContentPage.Content>
60 </ContentPage>

```

Følgende listing 6.2 er XAML-koden for SearchQuizPageSelectQuiz, som viser hvorledes dette view også databinder til viewModel:

Listing 6.2. SearchQuizPageSelectCategory.xaml - Databinding to SearchQuizView

```

1 <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
2     xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
3     x:Class="App1.View.SearchQuizPageSelectQuiz"
4     BackgroundColor="#9ab7b6"
5     Title="Vlg quiztype"
6     xmlns:ffimageloading="clr-namespace:FFImageLoading.Forms;assembly=FFImageLoading.Forms"
7     xmlns:xfg="clr-namespace:XFGloss;assembly=XFGloss">
8
9     <xfg:ContentPageGloss.BackgroundGradient>
10         <xfg:Gradient Rotation="180">
11             <xfg:GradientStep StepColor="LightGray" StepPercentage="0" />
12             <xfg:GradientStep StepColor="#9ab7b6" StepPercentage=".5" />
13         </xfg:Gradient>

```

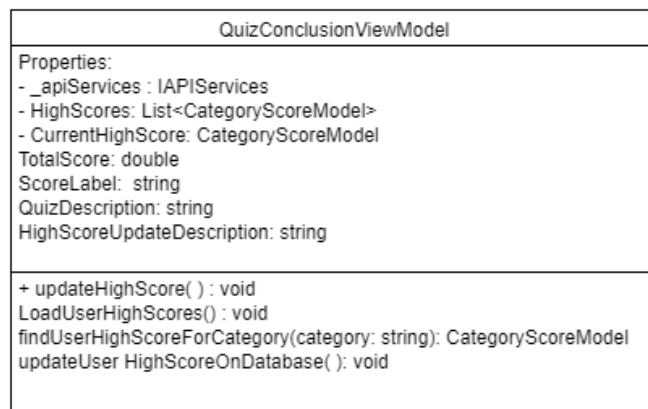


```

14     </xfg:ContentPageGloss.BackgroundImage>
15     <ContentPage.Content>
16
17         <Grid>
18
19             <ffimageloading:CachedImage
20                 Margin="0,0,0,0"
21                 VerticalOptions="Start"
22                 HorizontalOptions="End"
23                 Rotation="30"
24                 Opacity="0.25"
25                 Scale="2"
26                 Source = "quiz_bg"
27             />
28
29
30
31             <Grid Margin="30">
32                 <Grid.RowDefinitions>
33                     <RowDefinition/>
34                     <RowDefinition/>
35                 </Grid.RowDefinitions>
36
37
38
39                 <StackLayout Grid.Row="0" HorizontalOptions="CenterAndExpand">
40                     <ActivityIndicator HorizontalOptions="Center" Color="White" IsRunning="{Binding IsBusy}"
41                         />
42                     <ListView HorizontalOptions="Center"
43                         ItemsSource="{Binding Quizzes}"
44                         SeparatorColor="White"
45                         x:Name="LViewQuizzes">
46                         <ListView.ItemTemplate>
47                             <DataTemplate>
48                                 <TextCell
49                                     TextColor="White"
50                                     Text="{Binding QuizName}"
51                                 />
52                             </DataTemplate>
53                         </ListView.ItemTemplate>
54                     </ListView>
55                 </StackLayout>
56
57                 <Image Scale="0.35" Grid.Row="1" HorizontalOptions="Center" VerticalOptions="Center"
58                     Source="Icon" />
59
60
61             </Grid>
62         </Grid>
63     </ContentPage.Content>
64 </ContentPage>

```

6.6.3 QuizConclusionViewModel



Figur 6.6. UML for QuizConclusionViewModel

QuizConclusionViewModel er den klasse, som er ansvarlig for at tilgå QuizScore databasen via APIServices-klassen, som set på figur 6.1, samt præsenterer resultatet af den udførte quiz, og hvorvidt higscoren på databasen er opdateret. Nedenfor følger metodebeskrivelserne for QuizConckusionViewModelklassen som ses på figur 6.6

```
1 | private IAPIServices _apiServices {get; set; }
```

Beskrivelse: Indeholder en et APIServices objekt til kommunikation til Quiz Score Databasen.

```
1 | public Quiz CompletedQuiz {get; set; }
```

Beskrivelse: Indeholder den quiz du lige har gennemført, med tilhørende data.

```
1 | public string ScoreLabel{ get; set; }
```

Beskrivelse: Indeholder en tekst, der fortæller dig hvordan du har klaret dig i quizen, ud fra din score.

```
1 | public string QuizDescription{get; set; }
```

Beskrivelse: Indeholder den tekst der beskriver hvordan du har klaret det i quizen.

```
1 | public string HighScoreUpdatedDescription{get; set; }
```

Beskrivelse: Indeholder den tekst som beskriver, hvorvidt man har slået sin egen highscore for den gennemførte kategori. Er dette ikke tilfældet bliver denne string tom.

```
1 | public QuizConclusionViewModel(Quiz quiz, double score)
```

Parametre: Quiz, double

Returværdi: Ingen

Beskrivelse: Constructor, sætter CompletedQuiz og TotalScore properties ud fra de parametre som constructoren modtager

```
1 | private void updateHighScore()
```

Parametre: Ingen

Returværdi: Ingen

Beskrivelse: Kalder LoadHighScores metoden, således der i HighScores gemmes en liste af CategoryScoreModel, svarende til userens highscores. Herefter sætte CurrentHighScore med findUserHighScoreForQuizCategory. Hvis TotalScore er større en scoren gemt i CurrentHighScore kaldes updateHighScoreOnDatabase.

```
1 | private void LoadHighScores()
```

Parametre: Ingen

Returværdi: Ingen

Beskrivelse: På _apiService kaldes GetHighScoreForCurrentUser og resultatet gemmes i HighScores property'en.

```
1 | private void findUserHighScoreForQuizCategory()
```

Parametre: Ingen

Returværdi: Ingen

Beskrivelse: Finder og returner den CategoryScoreModel som har samme kategori som den afsluttede quiz.

```
1 | private void updateUserHighScoreOnDatabase()
```

Parametre: Ingen

Returværdi: Ingen

Beskrivelse: Hvis CurrentHighScore ikke er null sættes HighScore property'en for denne til TotalScore og på _apiService kaldes PutHighscore med CurrentHighScore. - Er CurrentHighScore null, eksisterede der ingen

CategoryScoreModel for kategorien (users første udførsel af quiz med denne kategori) laves et nyt objekt med quizzens kategori og TotalScore. Herefter kaldes PostHighScore med dette objekt på `_apiService`.

Følgende listing 6.3 er XAML-koden for SearchQuizPageSelectCategory, som viser hvorledes dette view databinder til viewModel:

Listing 6.3. QuizConclusionPage.xaml - Databinding to QuizConclusionViewModel

```

66 <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
67     xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
68     xmlns:xfg="clr-namespace:XFGloss;assembly=XFGloss"
69     x:Class="App1.View.QuizConclusionPage">
70
71     <xfg:ContentPageGloss.BackgroundGradient>
72         <xfg:Gradient Rotation="180">
73             <xfg:GradientStep StepColor="LightGray" StepPercentage="0" />
74             <xfg:GradientStep StepColor="#9ab7b6" StepPercentage=".5" />
75         </xfg:Gradient>
76     </xfg:ContentPageGloss.BackgroundGradient>
77
78     <ContentPage.Content>
79         <StackLayout>
80             <Grid>
81                 <Grid.RowDefinitions>
82                     <RowDefinition Height="30"/>
83                     <RowDefinition Height="30"/>
84                     <RowDefinition Height="30"/>
85                     <RowDefinition Height="30"/>
86                     <RowDefinition Height="30"/>
87                 </Grid.RowDefinitions>
88                 <Label Grid.Row="1" Margin="20,0,0,0">Tillykke du har fuldenendt quiz: </Label>
89                 <Label Grid.Row="2" Margin="20,0,0,0" Text="{Binding QuizDescription}"/>
90                 <Label Grid.Row="3" Margin="20,0,0,0" Text="{Binding ScoreLabel}"/>
91                 <Label Grid.Row="4" Margin="20,0,0,0" Text="{Binding HighScoreUpdatedDescription}"/>
92                 <Button Grid.Row="5" Text="G til hovedmenuen" Clicked="goback"/>
93             </Grid>
94         </StackLayout>
95     </ContentPage.Content>
96 </ContentPage>

```

6.7 Quiz feature - databasen

Som nævnt tidligere har vi til udviklingen af databaseservicen brugt Repository pattern og MVC, hvor hvert lag har deres eget ansvar. Dette passer godt til REST API designs, hvor de forskellige lag af arkitekturen arbejder sammen til at bygge et hierarki, der gør applikationen mere skalerbar og modulær. Dette skulle gerne være afspejlet i de nedenstående klassebeskrivelser.

6.7.1 QuizRepository

Denne klasse implementerer interfacet `IQuizRepository` som indeholder signaturen på CRUD-metoderne. Dette interface findes i source koden og vil ikke være beskrevet yderligere. `QuizRepository` indeholder et antal af variabler og funktioner og disse beskrives nedenunder.

Navn	Type	Funktionalitet
Endpoint	String	Anvendes som URI service endpoint
Key	String	Anvendes som AUTH-KEY
DatabaseId	String	Bruges af <code>URIFactory.CreateDocumentUri()</code> metoden – der givent database, collection og document id laver et dokument link
CollectionId	String	Bruges af <code>URIFactory.CreateDocumentUri()</code> metoden – der givent database, collection og document id laver et dokument link
client	DocumentClient	Anvendes til at skabe en client-side repræsentation af en Azure Cosmos DB service

Tabel 6.3. Variabler i `QuizRepository` klassen

```
1 public QuizRepository()
```

Parametre: Ingen.

Returværdi: Ingen.

Beskrivelse: Står for at initialisere en ny instans af `Microsoft.Azure.Documents.Client.DocumentClient` klassen. Til dette bruges det specificerede Azure Cosmos DB service endpoint og godkendelsesnøgle. Denne client kan derefter anvendes til at konfigurere og udføre forespørgsler mod vores quiz tjeneste². Yderligere kaldes `CreateDatabaseIfNotExistsAsync().Wait()` og `CreateCollectionIfNotExistsAsync().Wait()`, der som navnene antyder opretter database og collection – hvis disse ikke allerede findes. Disse kaldes med `Task.Wait()`, da dette kan tage lang tid at oprette og sikrer derfor at der ventes til de er færdige med at eksekvere.

Disse funktioner vil være beskrevet i detaljer i nedenstående funktionsbeskrivelser.

```
1 public async Task<T> GetQuizAsync(string id)
```

Parametre: Id som anvendes som dokument id.

Returværdi: `Task<T>` som repræsenterer en asynkron operation der kan returnere en generisk type `T`. I vores tilfælde `Quiz` klassen fra modelklasserne.

Beskrivelse: Prøver at lave et dokument repræsenteret fra Azure Cosmos DB servicen³. Dette dokument læses og returneres som en generisk type `T`, hvor vi anvender vores quizklasse fra arbejdet fra domænemodells analysen. Dette

²<https://bit.ly/2EpNLJY> - DocumentClient Class for the Azure Cosmos DB service

³<https://bit.ly/2C2HRM6> - Documentation for Document Class for Azure Cosmos DB service

gøres med `ReadDocumentAsync(UriFactory.CreateDocumentUri(DatabaseId, CollectionId, id))` metoden, hvor der igen givet database, collection og document id laves et dokument link. Denne operation kan tage et lille stykke tid og køres derfor asynkront for ikke at blokere.

```
1 | public async Task<IEnumerable<T>> GetQuizByCategory(Expression<Func<T, bool>> predicate)
```

Parametre: `Expression<Func<T, bool>` predicate der anvendes som filter metode til kunne søge i databasen ud fra kategori.

Returværdi: `Task<IEnumerable<T>` som repræsenterer en asynkron operation der kan lave en iteration over en kollektion af en generisk type `T`. I vores tilfælde en liste af Quiz klassen fra modelklasserne.

Beskrivelse: Opretter en `IDocumentQuery<T>` query ved `CreateDocumentQuery<T>(...)`, der sørger for metoder der understøtter query paginering og asynkron eksekvering i en Azure Cosmos DB service⁴. Her udnyttes filter metoden og `IDocumentQuery<T>.HasMoreResults` så der hentes korrekt baseret på predicate.

```
1 | public async Task<IEnumerable<T>> GetAllQuizzesAsync()
```

Parametre: Ingen.

Returværdi: `Task<IEnumerable<T>` som repræsenterer en asynkron operation der kan lave en iteration over en kollektion af en generisk type `T`. I vores tilfælde en liste af Quiz klassen fra modelklasserne.

Beskrivelse: Opretter en `IDocumentQuery<T>` query ved `CreateDocumentQuery<T>(...)`, der sørger for metoder der understøtter query paginering og asynkron eksekvering i en Azure Cosmos DB service. Her udnyttes `IDocumentQuery<T>.HasMoreResults` så der hentes korrekt.

```
1 | public async Task<Document> CreateQuizAsync(T quiz)
```

Parametre: Generisk type `T`, hvor `T` er en klasse.

Returværdi: `Task<T>` som repræsenterer en asynkron operation der kan returnere en generisk type `T`. I vores tilfælde Quiz klassen fra modelklasserne.

Beskrivelse: Metoden laver et dokument ved `CreateDocumentAsync()` som en asynkron operation i vores Azure Cosmos DB service. Dette tillader givet parameteren er korrekt at oprette et JSON-dokument tilsvarende til modelklassen for quiz objektet⁵. Til dokument link anvendes `UriFactory.CreateDocumentCollectionUri(DatabaseId, CollectionId)`, der skabes igen ved givent database og kollektions id.

```
1 | public async Task<Document> UpdateQuizAsync(string id, T quiz)
```

Parametre: Generisk type `T`, hvor `T` er en klasse og string til dokument id.

Returværdi: `Task<T>` som repræsenterer en asynkron operation der kan returnere en generisk type `T`. I vores tilfælde Quiz klassen fra modelklasserne.

Beskrivelse: Metoden opdaterer et dokument ved `ReplaceDocumentAsync()`⁶ som en asynkron operation i vores Azure Cosmos DB service. Dette tillader givet parameteren er korrekt at opdatere et dokument på databasen. Til dokument link anvendes `UriFactory.CreateDocumentCollectionUri(DatabaseId, CollectionId, id)`, der skabes igen ved givent database, kollektions og dokument id.

⁴<https://bit.ly/2B6Ft5v> - `IDocumentQuery<T>` interface

⁵<https://bit.ly/2C48xMC> - `DocumentClient.CreateDocumentAsync`

⁶<https://bit.ly/2EeRCZ6> - `DocumentClient.ReplaceDocumentAsync`

```
1 | public async Task DeleteQuizAsync(string id)
```

Parametre: String til dokument id.

Returværdi: Ingen.

Beskrivelse: Metoden sletter et dokument asynkront fra vores DB service ved hjælp af DocumentClient.DeleteDocumentAsync() metoden⁷. Igen anvendes UriFactory til at skabe dokument link.

```
1 | private async Task CreateDatabaseIfNotExistsAsync()
```

Parametre: Ingen.

Returværdi: Ingen.

Beskrivelse: Prøver først at læse en database igen fra Azure Cosmos DB service. Dette gøres endnu gang asynkront. Hvis databasen ikke findes kaldes CreateDatabaseAsync()⁸ metoden med database id. Denne laver databaseressourcen.

```
1 | private async Task CreateCollectionIfNotExistsAsync()
```

Parametre: Ingen.

Returværdi: Ingen.

Beskrivelse: Prøver først at læse en kollektion med ReadDocumentCollectionAsync() metoden. Denne tager linket til DocumentCollection med som parameter og det udføres igen asynkront. Hvis kollektionen ikke findes kaldes CreateDocumentCollectionAsync()⁹ metoden med database id og kollektions id.

6.7.2 QuizController

Controlleren er tilrettet til at bruge det før beskrevne repository hvor logikken til CRUD-operationerne er implementeret. Controller klasserne fokuserer på indkomende handlinger og det hjælper ligeledes på opdelingen af ansvar og på gøre applikationen mere skalerbar og modulær.

Navn	Type	Funktionalitet
QuizRepository	IQuizRepository<Quiz>	Anvender vores repository tilgang og tillader at bruge metoderne beskrevet tidligere.

Tabel 6.4. Variabler i QuizController klassen

```
1 | [ActionName("Index")]
2 | public async Task<IActionResult> Index()
```

Parametre: Ingen.

Returværdi: IEnumerable<Quiz> som viewresult objekt der giver et view af svaret.

⁷<https://bit.ly/2z0yb6w> - DocumentClient.DeleteDocumentAsync

⁸<https://bit.ly/2RVDjwV> - DocumentClient.CreateDatabaseAsync(...)

⁹<https://bit.ly/2B7iB5P> - DocumentClient.CreateDocumentCollectionAsync

Beskrivelse: Kalder GetAllQuizzesAsync() metoden fra repository klassen. Denne henter som beskrevet tidligere en liste af quiz objekter. Repræsenterer yderligere resultatet af en action metode¹⁰.

```
1 [ActionName("Create")]
2 public async Task<ActionResult> CreateAsync()
```

Parametre: Ingen.

Returværdi: Returnere et viewresult objekt er giver et view af svaret.

Beskrivelse: Definere en kontrakt der repræsenterer resultatet af en action metode.

```
1 [HttpPost]
2 [ActionName("Create")]
3 [ValidateAntiForgeryToken]
4 public async Task<ActionResult> CreateAsync([Bind("QuizName,Category,Id,Question")] Quiz quiz)
```

Parametre: En instans af BindAttribute klassen der bruges til at angive og anvende model niveau metadata¹¹.

Returværdi: Returnere et viewresult objekt er giver et view af svaret.

Beskrivelse: Definere en kontrakt der repræsenterer resultatet af en action metode. Der vurderes på Controller-Base.ModelState¹² og hvis denne er valid kaldes CreateQuizAsync() fra repository klassen.

```
1 [HttpPost]
2 [ActionName("Edit")]
3 [ValidateAntiForgeryToken]
4 public async Task<ActionResult> EditAsync([Bind("QuizName,Category,Id,Question")] Quiz quiz)
```

Parametre: En instans af BindAttribute klassen der bruges til at angive og anvende model niveau metadata.

Returværdi: Returnere et viewresult objekt er giver et view af svaret.

Beskrivelse: Definere en kontrakt der repræsenterer resultatet af en action metode. Der vurderes på Controller-Base.ModelState og hvis denne er valid kaldes UpdateQuizAsync() med quiz id og quiz objekt fra repository klassen.

```
1 [ActionName("Edit")]
2 public async Task<ActionResult> EditAsync(string id)
```

Parametre: String med dokument id.

Returværdi: Returnere et viewresult objekt er giver et view af svaret.

Beskrivelse: Hvis id er lig null returneres en fejl 400 BadRequest. Hvis ikke kaldes GetQuizAsync() med dokument id. Hvis denne ikke findes returneres en fejl 404 NotFound.

```
1 [ActionName("Delete")]
2 public async Task<ActionResult> DeleteAsync(string id)
```

Parametre: String med dokument id.

Returværdi: Returnere et viewresult objekt er giver et view af svaret.

¹⁰<https://bit.ly/2EpWXhs> - ActionNameAttribute Class

¹¹<https://bit.ly/2Em6d5W> - BindAttribute Class

¹²<https://bit.ly/2PwDc90> - ControllerBase.ModelState Property

Beskrivelse: Hvis id er lig null returneres en fejl 400 BadRequest. Hvis ikke kaldes GetQuizAsync() med dokument id. Hvis denne ikke findes returneres en fejl 404 NotFound.

```
1 [HttpPost]
2 [ActionName("Delete")]
3 [ValidateAntiForgeryToken]
4 public async Task<ActionResult> DeleteConfirmedAsync([Bind("Id")] string id)
```

Parametre: En instans af BindAttribute klassen der bruges til at angive og anvende model niveau metadata.

Returværdi: Returnere et viewresult objekt er giver et view af svaret.

Beskrivelse: Metoden kalder DeleteQuizAsync() metoden med dokument id.

```
1 [ActionName("Details")]
2 public async Task<ActionResult> DetailsAsync(string id)
```

Parametre: String med dokument id.

Returværdi: Returnere et viewresult objekt er giver et view af svaret.

Beskrivelse: Metoden kalder GetQuizAsync() metoden med dokument id.

6.7.3 QuizToApiController

Controlleren er tilrettet til at bruge det før beskrevne repository og logikken til CRUD-operationerne er implementeret. Controller klasserne fokusere på indkommende handlinger og det hjælper ligeledes på opdelingen af ansvar og på gøre applikationen mere skalerbar og modulær.

Der sættes også en route ¹³og produces¹⁴.

Navn	Type	Funktionalitet
QuizRepository	IQuizRepository<Quiz>	Anvender vores repository tilgang og tillader at bruge metoderne beskrevet tidligere.

Tabel 6.5. Variabler i QuizToApiController klassen

```
1 [HttpGet]
2 public async Task<IEnumerable<Quiz>> Get()
```

Parametre: Ingen.

Returværdi: IEnumerable<Quiz>.

Beskrivelse: Kalder GetAllQuizzesAsync() metoden fra repository klassen som returnere en liste med quiz objekter.

```
1 [HttpGet("{category}", Name = "GetByCategory")]
2 public async Task<IEnumerable<Quiz>> GetByCategory(string category)
```

¹³<https://bit.ly/2zU0f8v> - RouteAttribute(String) Constructor

¹⁴<https://bit.ly/2C5nP3J> - ProducesAttribute Constructors

Parametre: String med kategori.

Returværdi: IEnumerable<Quiz>.

Beskrivelse: Kalder GetQuizByCategory() metoden fra repository klassen som returnere en liste med quiz objekter baseret på parameteren.

```
1 [HttpGet("{id}", Name = "GetByID")]
2 public async Task<ActionResult> GetByID(string id)
```

Parametre: String med kategori.

Returværdi: Returnere et OKResult¹⁵ objekt der giver et view af svaret.

Beskrivelse: Hvis id er lig null returneres en fejl 400 BadRequest. Hvis ikke kaldes GetQuizAsync() med dokument id. Hvis denne ikke findes returneres en fejl 404 NotFound.

```
1 [HttpPost]
2 public async Task<ActionResult> Post([Bind("Id,Category,QuizName,Question")] Quiz quiz)
```

Parametre: En instans af BindAttribute klassen der bruges til at angive og anvende model niveau metadata.

Returværdi: Returnere et viewresult objekt er giver et view af svaret.

Beskrivelse: Der vurderes på ControllerBase.ModelState og hvis denne er valid kaldes CreateQuizAsync() med quiz objekt.

```
1 [HttpPut("{id}")]
2 public async Task<ActionResult> Put([FromRoute] string id, [FromBody] Quiz quiz)
```

Parametre: Parametre med route-data til dokument id og request body med quiz objekt.

Returværdi: Returnere et OKResult objekt der giver et view af svaret.

Beskrivelse: Der vurderes først på ControllerBase.ModelState og dokument id. Hvis der er fejl i disse, returneres en fejl 400 BadResponse. Derefter kaldes UpdateQuizAsync() metoden fra repository klassen med dokument id og quiz objekt. Ved fejl returneres en fejl 404 NotFound.

```
1 [HttpDelete("{id}")]
2 public async Task<IActionResult> Delete([FromRoute] string id)
```

Parametre: Parametre med route-data til dokument id.

Returværdi: Returnere et OKResult objekt der giver et view af svaret.

Beskrivelse: Der vurderes først dokument id. Hvis der er fejl i denne, returneres en fejl 400 BadResponse. Derefter kaldes GetQuizAsync() metoden fra repository klassen med dokument id. Ved fejl returneres en fejl 404 NotFound. Herefter kaldes DeleteQuizAsync() metoden. Ved fejl returneres igen en fejl 404 NotFound.

6.7.4 Quiz

Modelklasse der styrer hvordan data skal formateres.

¹⁵<https://bit.ly/2Em8TR2> - ControllerBase.Ok

Navn	Type	Funktionalitet
Id	String	JsonProperty med property navnet id.
Category	String	JsonProperty med property navnet Category.
QuizName	String	JsonProperty med property navnet QuizName.
Question	List<Question>	JsonProperty med property navnet Question.

Tabel 6.6. Variabler i Quiz model klassen

6.7.5 Question

Modelklasse der styrer hvordan data skal formateres.

Navn	Type	Funktionalitet
QuestionText	String	JsonProperty med property navnet QuestionsText.
Score	Int	JsonProperty med property navnet Score.
Options	List<Option>	JsonProperty med property navnet Options.

Tabel 6.7. Variabler i Question model klassen

6.7.6 Option

Modelklasse der styrer hvordan data skal formateres.

Navn	Type	Funktionalitet
OptionText	String	JsonProperty med property navnet OptionText.
IsRight	Bool	JsonProperty med property navnet IsRight.

Tabel 6.8. Variabler i Option model klassen

6.8 Quiz Feature - Diskussion

Det er lykkedes projektgruppen at få implementeret en funktionel quiz feature. Dette involverer udvælgelse af quiz, først efter kategori dernæst specifikt, dette gennem en QuizService som giver adgang til dokumentdatasen, hvor quizzene er gemt som individuelle dokumenter (nærmere beskrevet under databaseafsnittet for quiz feature). Der udover at give user mulighed for at besvare den udvalgte quiz og til sidst få konklusion på, hvordan denne besvarelse er gået forhold til egen tidligere bedste præstation.

Gennem arbejdet har projektgruppen nået de krav som i MOSCOW-analysen defineret som must have. Projektgruppen har ønsket også have implementeret en feature som ville give users mulighed for at uploadede brugergenerede quizzes til databasen via det udviklede api. Dette er dog ikke lykkedes.

Udfordringen lå heri med arbejdet omkring dokumentdatabasen, som tog længere tid at udvikle end forventet. NoSQL og dokumentdataser har været pensum, som der først er blevet undervist i sent på semesteret, hvorfor projektgruppen også tidligere forsøgte at undersøge emnet på egen hånd. Der er bl.a. tidligere i forløbet forsøgt anvendelse af Mogo Api til azure dokument database, men dette endte ikke i en funktionel implementering. Udfordringen i at opdele arbejde i database udvikling og view/viewModel udvikling til quiz feature og evt. brugergeneret quiz har været usikkerhed omkring hvilket api vi kunne stille til rådighed for appen, et samlet quiz objekt, kun en liste af spørgsmål, kunne denne have mere end en svar mulighed pr. spørgsmål? Dette gjorde en fast definition af model-laget svær at definere. Derfor har udviklingen af separate quiz features (udførsel og generering) været besværligt at arbejde på sideløbende med udviklingen af databasen. Da gruppen havde fået forståelse for rest api blev færdiggørelsen af quiz featuren dog hurtigt fulendt.

Hvordan er implementeringen af MVVM gået?

Det er lykkedes projektgruppen, efter egen vurdering, at få implementeret MVVM-arkitekturen, med single responsibility, for hver af de tre viewModels. Det blev i starten af processen overvejet, om der skulle implementeres en controller klasse til styring af Model-laget og view-laget. Det ville altså være muligt i stedet for MVVM at implementere MVC-arkitektur hvor en controller klasse er ansvarlig for alt logik.

MVVM-arkitekturen har dog nogle fordele i forhold til MVC, først at der er bedre separation of concerns gennem MVVM, hvor logikken er opdelt mellem de tre lag, model, viewModel og view. View kan som tidligere beskrevet ikke direkte tilgå model-laget, men er ansvarlig for den nødvendige logik for at gui-elementerne fungerer og kan interagere med brugeren. ViewModel står for præsentation logikken, og kan sende events både til View-laget og model-laget, denne fungerer som en abstraktion af model-laget. Model-laget kender hverken til View eller ViewModel, men implementere sin egen logik. I en MVC-struktur med en kontroller risikerer man hurtigt at få lavet et såkaldt god object, da kontrolleren skal være ansvarligt for alt logik, og den bliver meget svær at teste.

Delkonklusion Det er lykkedes projektgruppen at udvikle en quiz feature bestående af de to delelementer:

- En database der stiller de oprettede quizzes til rådighed for appen.
- En GUI feature i appen som giver mulighed for at vælge og besvare en

7.1 Unittest

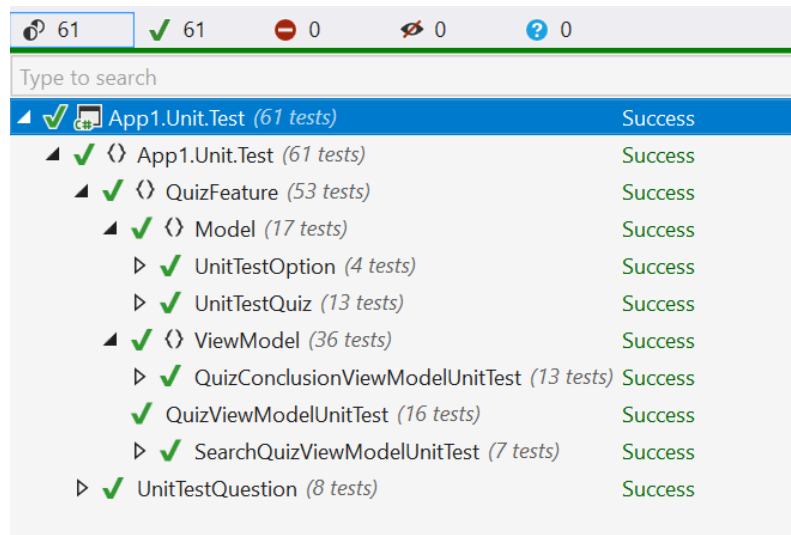
7.1.1 Unit test - Quiz Feature

En af fordelene ved anvendelse af MVVM arkitekturen er den lave mængde kode i code-behind filerne. Anvendelsen af viewModels gør logikken testbart, som ellers er svært at teste, hvis implementeret i code-behind. For Quiz featuren i dette projekt er foruden test af database api'et blevet opstillet unittest for modelklasserne, Option, Question og Quiz samt de udviklede viewModels til Quiz Feature, SearchQuizViewModel, QuizViewModel og QuizConclusionViewModel.

UnitTest af Model Formålet med unittestene af Modelklasserne havde til hensigt at demonstrere og teste, udførelse af get og særligt set-logikken for modelklassernes properties. At disse kun ændres, hvis parameteren for set-udførslen er forprskellig fra tidligere værdi, og at deri dette set-kald sendes et PropertyChanged event på denne ændrede property, som senere anvendes i databinding af disse klasser gennem viewModels.

UnitTest af ViewModels Første segment af de udførte unittest på de tre Quiz viewModels har ligesom for selve modelklasserne til formål at INotifyPropertyChanged interfacet er korrekt implementeret og at disse viewModels stiller et PropertyChanged event til rådighed for det view som skal databinde til denne viewModel. Derudover er der blevet udført unittest mere specifikt for hver enkelt viewModel:

For QuizViewModel blev håndteringen kald af AnswerCommand, som er denne viewModels primære logik, testet. Det blev valideret at når metoden kaldes med false parameter (et forkert svar eller intet svar) så forøges brugerens scorer ikke, og ligeledes tjekket at det er tilfældet at den forøges når metoden kaldes med true (et korrekt svar blev valgt). Det blev test desuden testet at uanset om der blev afgivet korrekt eller forkert (inklusive intet) svar at det nuværdene spørgsmål herefter opdateres med det næste spørgsmål for den pågældende quiz, samt at der sendes et quizCompleted i de tilfælde der ikke er flere spørgsmål i den aktuelle quiz. For SearchQuizViewModel blev der foruden propertyChanged test blevet unit test, hvorvidt den for det første view vil load og præsentere alle de unikke kategorierne for de quizzes, som er gemt på dokumentdatabasen. For QuizConclusionViewModel er det blevet test at HighscoreUpdatedDescription opdateres korrekt. Nedenfor på figur 7.1 kan unit test rapporten ses, samt der findes en oversigt over de udført unit tests:



Figur 7.1. Rapport over udførte unit tests

Der er blevet brugt NUnit-Framwork til at unit teste klasserne og til test af ViewModel-klasserne er NSubstitute blevet anvendt til at mocke Service-klasserne APIServices og QuizDBService.

Neden for er de udførte unit test blevet præsenteret i tabelform:

Test Navn	IsRightChangedToNewValue_PropertyChangedInvoked
Beskrivelse	Tester hvorvidt propertyChanged eventet sendes efter Option objektets IsRight property
Resultat	Succes: IsRight property er blevet sat korrekt, og propertyChanged er blevet invoked en
Test Navn	IsRightChangedToSameValue_PropertyChangedNotInvoked
Beskrivelse	Tester hvorvidt propertyChanged eventet kun sendes en enkelt efter Option objektets Is
Resultat	Succes: IsRight property er blevet sat korrekt, men propertyChanged er kun blevet invo
Test Navn	OptionTextChangedToNewString_PropertyChangedInvoked
Beskrivelse	Tester hvorvidt propertyChanged eventet sendes efter Option objektets OptionText pro
Resultat	Succes: OptionText property er blevet sat korrekt, og propertyChanged er blevet invoke
Test Navn	OptionTextChangedToSameString_PropertyChangedNotInvoked
Beskrivelse	Tester hvorvidt propertyChanged eventet kun sendes en enkelt efter Option objektets O
Resultat	Succes: OptionText property er blevet sat korrekt, men propertyChanged er kun blevet
Test Navn	NextQuestionWhenNotLastQuestion_ReturnsQuestion
Beskrivelse	Tester at NextQuestion returner et Question objekt, da der er flere Questions tilbage i l
Resultat	Succes: Der returneres et Question objekt.
Test Navn	NextQuestionWhenLastQuestion_ReturnsNull
Beskrivelse	Tester at NextQuestion returner et null, da der ikke er flere Questions tilbage i listen
Resultat	Succes: Der returneres null
Test Navn	SearchCategoriesCalled_CategoriesPropertyChangedCalled
Beskrivelse	Tester hvorvidt propertyChanged eventet sendes, når LoadCategories kaldes, da det æn
Resultat	Succes: PropertyChanged er blevet invoked.
Test Navn	LoadCategories_OnlyUniqueCategoriesLoaded
Beskrivelse	Tester at der returnes en liste med unikke kategorier, hvis kaldet til databasen returnere
Resultat	Succes: Der returneres en liste kun med unikke kategorier.
Test Navn	LoadQuizzesFromCategoryCalled_CategoriesPropertyChangedCalled
Beskrivelse	Tester hvorvidt propertyChanged eventet sendes, når LoadCategories kaldes, da det æn
Resultat	Succes: PropertyChanged er blevet invoked.
Test Navn	WrongAnswerGivenScoreIsNotIncreased
Beskrivelse	Tester at TotalScore ikke øges, hvis der svares forkert
Resultat	Succes: TotalScore er ikke forøget.
Test Navn	WrongAnswerGivenTheQuestionUpdated
Beskrivelse	Tester at TheQuestion property bliver updateret selvom der svares forkert.
Resultat	Succes: TheQuestion opdateres
Test Navn	WrongAnswerGivenLastQuestionQuizCompletedEventCalled
Beskrivelse	Tester QuizCompleted eventet kaldes, selvom der svares forkert på sidste spørgsmål (Ell
Resultat	Succes: QuizCompleted eventet bliver kaldt.
Test Navn	RightAnswerGivenScoreIsIncreased
Beskrivelse	Tester at TotalScore øges, hvis der svares korrekt
Resultat	Succes: TotalScore er forøget.

For alle de medtagede klasser er propertyChanged event calls blevet testet men undladt af tabellen.

Logik der ikke unit testes Der har været noget logik, der ikke er blevet testet via unit test. Dette er f.eks. randomizing af rækkefølgen gældende for Options og Questions, da det er svært at teste for tilfældighed. Derudover der ikke lavet unittest for nedtælling af TimeLeft property'en for QuizViewModel samt selve Databindingen af alle ViewModels, da disse inkluderer en realtidskomponent. Dette betyder dog ikke disse implementeringer ikke er blevet testet. De er blevet valideret ved at debugge application ved brug af emulator

7.2 Integrationtest

7.2.1 Continuous Integration

I projektet har der ikke været et stort behov for integrations test som kendt fra tidligere projekter. Dette skyldes der i projektet har været anvendt en type af continuous integration (CI). Anvendelsen af projektstyring i form af git og muligheden for at teste på en virtuel enhed, emulator, har givet projektgruppen mulighed for med det samme at se implementeringen af nyt tilføjet kode og test heraf. Det er gennem test på denne emulator det har været muligt at teste og validerer de implementeringer, som tidligere nævnt er blevet udeladt af selve unittestene.

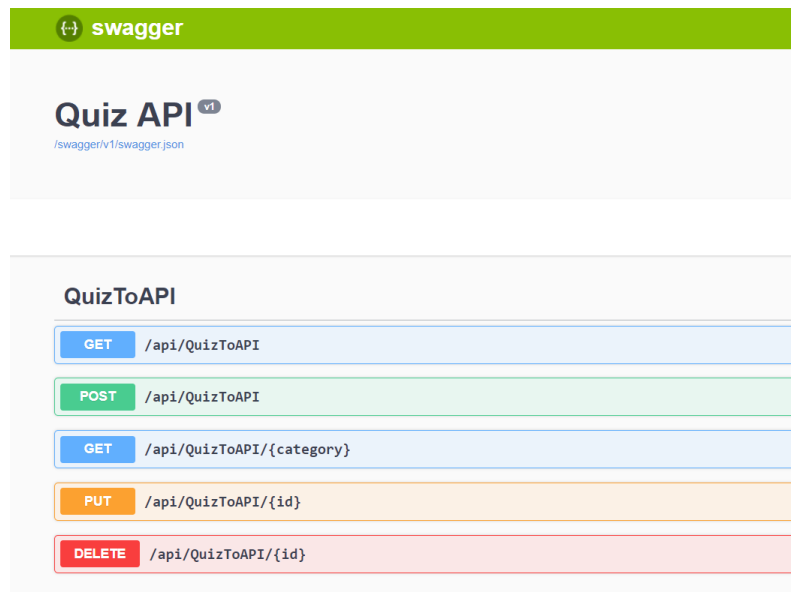
1

7.3 REST API hjælpeværktøjer

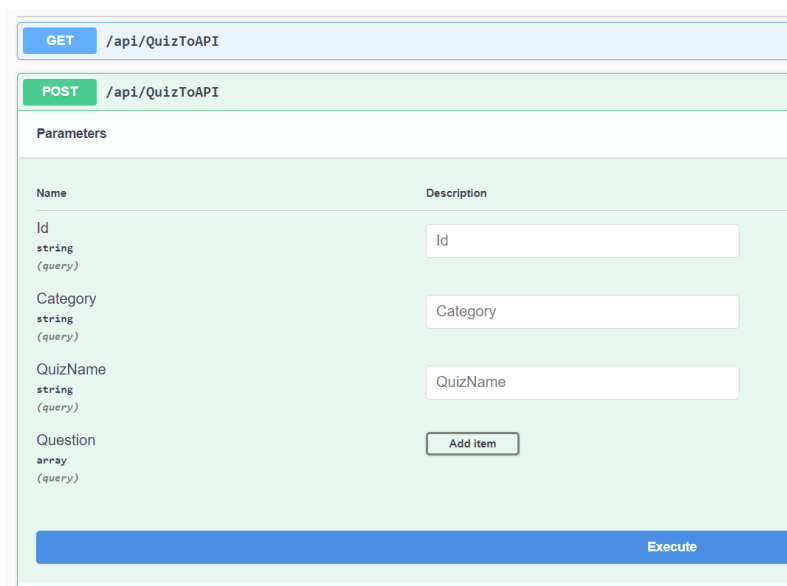
Til test af CRUD-operationerne på databaserne er der i applikationerne installeret forskellige hjælpeværktøjer. I dokumentdatabasen er hjælpeværktøjet Swagger brugt, hvor der i den relationelle er brugt Postman. Begge gør API udvikling nemmere ved at præsentere vores API simpelt.

Nedenunder ses eksempler fra test Swagger.

¹FiXme Note: INDSÆT BILLEDE af emulator på quizPage



Figur 7.2. Demonstration af REST API med Swagger



Figur 7.3. Demonstration af REST API post med Swagger

Ved disse har vi altså en form for integreret test klient, hvor vi som udviklere hurtigt kan sende requests og få svar på disse, hvormed vi slipper for en masse fejl og testkode.

Man kunne have testet på andre måder for eksempel ved at "Load teste" vores REST API's, men grundet applikationen anvendes som prototype anså vi Swagger/Postman som fyldestgørende.

8.1 Beskrivelse

For at være i stand til at udføre denne accepttest, er der brug for følgende:

- 1 Android smartphone eller en emulator af denne

De følgende testscenarier, er blevet udarbejdet for hver enkelt user story, med en definition af konkret testdata. Kravet for accepttesten, er at selv en person der ikke ved noget om applikationen, stadig ville kunne få accepttesten i hånden og udføre testscenarierne. Beskrivelsen af disse testscenarier er blevet udarbejdet ved hjælp af en Gherkin syntax¹.

1. Opret "User"

Egenskab: Oprette en User

Som ny bruger ønsker jeg kunne oprette et user login og et password, for at få adgang til applikationen.

Baggrund:

Givet at brugeren har ASE StudyHelper installeret på telefonen og ikke har en user.

Oprette user:

Når brugeren ønsker at oprette en user

Så åbner brugeren applikationen på sin mobil

Så trykker han på knappen "Opret User"

Så indtaster han sit ønskede brugernavn og password

Så trykker brugeren på knappen "Opret"

Og user er blevet oprettet med det ønskede brugernavn og password

Resultat: Accepteret

2. Login

Egenskab: Login

Som bruger ønsker jeg at kunne logge ind med user login og password, for at få adgang til applikationens indhold.

Baggrund:

Givet at brugeren har oprettet sig som "User", står på Login siden, og nu har både et user login og et password.

Scenarie: Login

Når brugeren ønsker at logge ind på applikationen

Så skriver brugeren sit brugernavn i User-feltet

Så skriver brugeren sit password i Password-feltet

Og trykker på login

¹<http://docs.behat.org/en/v2.5/guides/1.gherkin.html>

Resultat: Accepteret

3. Logud

Egenskab: Logud

Som bruger ønsker jeg at kunne logge ud fra min user.

Baggrund:

Givet at brugeren er logget ind på sin user, og befinder sig på hovedmenuen.

Scenarie: Logud

Når brugeren ønsker at logge ud fra applikationen

Så navigerer brugeren til indstillinger-ikonet

Så trykker brugeren på indstillinger-ikonet

Og trykker brugeren på logud

Resultat: Accepteret

4. Ændre password

Egenskab: At ændre password

Som bruger ønsker jeg at kunne skifte mit allerede eksisterende password.

Baggrund:

Givet at brugeren allerede har oprettet en user, og befinder sig på hovedmenuen.

Scenarie: Ændre password

Når brugeren ønsker at ændre sit password

Så navigerer brugeren til indstillinger-ikonet i hovedmenuen

Så trykker brugeren på indstillinger-ikonet

Og trykker brugeren på "skift password"

Resultat: Accepteret

5. Følg progression

Egenskab: At følge progression

Som bruger ønsker jeg at kunne følge min users progression.

Baggrund:

Givet at brugeren allerede har oprettet en user, og gennemført mindst 1 quiz.

Scenarie: Følg progression

Når brugeren ønsker at følge sin progression

Så navigerer brugeren til highscore-ikonet i hovedmenuen

Så trykker brugeren på highscore-ikonet

Og observerer sin progression på skærmen"

Resultat: Fejlet

6. Sammenlign score

Egenskab: At sammenligne sin highscore

Som bruger ønsker jeg at kunne sammenligne min highscore med andre users.

Baggrund:

Givet at brugeren allerede har oprettet en user, og gennemført mindst 1 quiz.

Scenarie: Sammenlign score

Når brugeren ønsker at sammenligne sin highscore med andre users

Så navigerer brugeren til highscore-ikonet i hovedmenuen

Så trykker brugeren på highscore-ikonet

Og sammenligner sin highscore med de andre users på listen"

Resultat: Fejlet

7. Tilføj eget indhold

Egenskab: At oprette sin egen quiz

Som bruger ønsker jeg at kunne oprette min egen quiz, med tilhørende spørgsmål, hvor denne quiz kan anvendes af mig selv og andre.

Baggrund:

Givet at brugeren allerede har oprettet en user.

Scenarie: Tilføj eget indhold

Når brugeren ønsker at tilføje sit eget indhold

Så navigerer brugeren til quiz-ikonet

Så trykker brugeren på quiz-ikonet

Så trykker brugeren på "opret quiz"

Så indtaster brugeren spørgsmål og svar på sin egen quiz

Så trykker brugeren på afslut

Og quizen er nu oprettet

Resultat: Fejlet

8. Kør quiz

Egenskab: At quizze

Som bruger ønsker jeg at kunne tage en quiz om ingeniørfaglig viden.

Baggrund:

Givet at brugeren allerede har oprettet en user.

Scenarie: Kør quiz

Når brugeren ønsker at kunne tage en quiz for at teste sin viden i kurser ved ASE

Så navigerer brugeren til quiz-ikonet i hovedmenuen

Så trykker brugeren på quiz-ikonet

Så trykker brugeren på den ønskede kategori han/hun ønsker at quizze i

Så får brugeren præsenteret et spørgsmål og 4 valgmuligheder

Så svarer brugeren på spørgsmålet indenfor 20 sekunder

Så fremkommer et nyt spørgsmål, og det tidligere step gentages indtil der ikke er flere spørgsmål

Og trykker på afslut quiz for at få præsenteret sin score

Resultat: Accepteret

9. Udregn highscore

Egenskab: At udregne highscore

Som bruger ønsker jeg at der bliver udregnet en highscore for hver quiz for mig.

Baggrund:

Givet at brugeren allerede har oprettet en user og har svaret på alle spørgsmål i en quiz.

Scenarie: Udregn quiz

Når brugeren ønsker at få udregnet sin highscore

Så navigerer brugeren til quiz-ikonet i hovedmenuen

Så trykker brugeren på quiz-ikonet

Så trykker brugeren på den ønskede kategori han/hun ønsker at quizze i

Så får brugeren præsenteret et spørgsmål og 4 valgmuligheder

Så svarer brugeren på spørgsmålet indenfor 20 sekunder

Så fremkommer et nyt spørgsmål, og det tidligere step gentages indtil der ikke er flere spørgsmål

Og trykker på afslut quiz for at få præsenteret den udregnede highscore

Resultat: Accepteret

10. Bedøm indhold

Egenskab: At bedømme indholdet

Som bruger ønsker jeg at kunne bedømme de quizzes, jeg har gennemført.

Baggrund:

Givet at brugeren allerede har oprettet en user og har svaret på alle spørgsmål i en quiz.

Scenarie: Bedøm indhold

Når brugeren ønsker at bedømme en quiz

Så navigerer brugeren til quiz-ikonet i hovedmenuen

Så trykker brugeren på quiz-ikonet

Så trykker brugeren på "bedøm quiz" for den quiz der er ønsket at bedømme

Og giver quizen en score fra 1-5

Resultat: Fejlet

Udviklingsværktøjer 9

9.1 Software

Visual Studio

Alt programmering er forgået i Visual Studio. Herunder har vi også brugt VS' indbyggede Android emulator til at debugge.

Visual Paradigm

Brugt til at designe arkitekturen for softwaren og databaserne. Samtidig brugt til ERD diagrammer.

Adobe Photoshop CC

Brugt til at lave logoerne og ikonerne til appen.

Adobe Illustrator CC

Brugt til at lave vektorer til animationerne.

Adobe After Effects CC

Brugt til at lave animationerne samt at render dem til JSON filer.

Github Desktop

Brugt som GUI for git og for at holde styr på de forskellige branches.

Git

Brugt som versionsstyringssoftware, og til at interagere med GitHub.

Postman

Brugt til at teste rest api'et for login databasen

Swagger Swashbuckle

Brugt til at teste rest api'et for quiz databasen

9.2 Web services / apps

Google Drive

Brugt til at dele tidlige dokumenter, som mødeplan mm.

Github

Her har vi haft vores repositories til appen.

OverLeaf V2

Brugt til at opsætte bilag, rapport og procesbeskrivelse i LaTeX. Her har vi alle kunne redigere og skrive på samme tid, ligesom i Google Docs.

Microsoft Azure

Brugt til at hoste vores databaser, både SQL og NoSQL.

9.3 NuGet Packages for VS

XFGloss

Brugt til at lave gradients til baggrunden af alle pages i appen.

Lottie

Brugt til at render .JSON animationer native.

NUnit

Brugt til unit og integrationstest.

Xamarin Forms

Frameworket som appen er bygget i. Gør det muligt at lave Android apps skrevet i C# i .Net.

FFImageLoading

Brugt til at gemme billederne cached i rammerne, for at effektivisere appen.

NSubstitute

Brugt til tests, for at lave substitutes i NUnit.

.Net Framework

Dette er frameworket som appen og login databasen er blevet programmeret i.

.Net Core

Heri er REST API'et for quiz databasen udviklet.

Azure DocumentDB

Brugt til at kunne connecte til Azure fra appen.

Swagger - Swashbuckle

Brugt til at teste og udvikle rest api'et for quiz databasen.

9.4 Hardware

Huawei P10 Lite

Brugt som primær debugging device og for at teste applikationen.

Huawei P20 Pro

Brugt som sekundær debugging device og for at teste applikationen.

Litteratur 10

1. <https://bit.ly/2QNAUXB> - From Data Bindings to MVVM Microsoft DOCS
2. [urlhttps://swagger.io/tools/swagger-ui/](https://swagger.io/tools/swagger-ui/)
3. <https://bit.ly/2BaFyoM> - Billedet er lånt af blog.codeanalogies.com
4. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (Kapitel 5) Besøgt 12.12.2018
5. <https://bit.ly/2QVc4Fa> - QuickStart: Build a MongoDB API Xamarin.Forms app with .NET and the Azure portal, BaseViewModel
6. <https://bit.ly/2EpNLJY> - DocumentClient Class for the Azure Cosmos DB service
7. <https://bit.ly/2C2HRM6> - Documentation for Document Class for Azure Cosmos DB service
8. [urlhttps://bit.ly/2B6Ft5v](https://bit.ly/2B6Ft5v) - IDocumentQuery<T> interface
9. <https://bit.ly/2C48xMC> - DocumentClient.CreateDocumentAsync
10. <https://bit.ly/2EeRCZ6> - DocumentClient.ReplaceDocumentAsync
11. <https://bit.ly/2z0yb6w> - DocumentClient.DeleteDocumentAsync
12. <https://bit.ly/2RVDjwV> - DocumentClient.CreateDatabaseAsync
13. <https://bit.ly/2B7iB5P> - DocumentClient.CreateDocumentCollectionAsync
14. <https://bit.ly/2EpWXhs> - ActionNameAttribute Class
15. <https://bit.ly/2Em6d5W> - BindAttribute Class
16. <https://bit.ly/2PwDc90> - ControllerBase.ModelState Property
17. <https://bit.ly/2zU0f8v> - RouteAttribute(String) Constructor
18. <https://bit.ly/2zU0f8v> - RouteAttribute(String) Constructor
19. <https://bit.ly/2Em8TR2> - ControllerBase.Ok
20. <http://docs.behat.org/en/v2.5/guides/1.gherkin.html>