28. maj 2019

Gruppe 2
Semesterprojekt 4
Ingeniørhøjskolen Aarhus

Photobook
Test

Deltagere		
Navn	**Initialer**	**Studienummer**
Andreas Harfeld Jakobsen	AHJ	201608930
Oskar Thorin Hansen	OTH	201700132
Morten Rosenquist	MLR	201706031
Jonas Hingeberg Hansen	JHH	201508749
Troels Østergaard Bleicken	TØB	201705588
Frederik Fuglsang Midtgaard	FFM	201704982
Nina Nguyen	NN	201506554

Indhold

Indledning

Dette dokument indeholder test informationer i forbindelse med Photobook. Det indeholder tanker, fremgangsmåde og resultater. Der bliver set på modul-, integration-, system- og accepttest. Dokumentet er skrevet med en formodning om, at læseren har kendskab til systemet opbygning og generel software test terminologi. For at få den forståelse refereres der til systemarkitekturen[1]. Testene er som resten af systemet også blevet implementeret agilt. Derfor er de tilhørende test blevet udført når de relevante dele af systemet er blevet implementeret.

1 Modultest

Der gennemføres unit test for de dele af systemet, hvor det giver mening. De dele af systemet, der modul testes er Repository, WebAPI og backend af MobilApp.

Repository

Der bliver undersøgt hvorvidt de implementeret CRUD metoder interagerer med databasen som forventet. Der bliver ikke testet op imod en MSSQL database, men istedet op imod en database, der oprettes i memory'en ved brug af Entity Framework Core's InMemory. Testene er ikke pæne unit test's idet, for at se om der sker de rigtige ændringer i databasen, så bliver den nødt til at blive tilgået ved brug af en anden metode. Men det vurderes som tilstrækkeligt, idet alternative er, at det først bliver testet ved integrationen med de andre dele.

WebAPI

Det skal nævnes at det ikke er alle controllerer der bliver testet i Unit testen. Som nævnt i SystemArkitekturen [2] er der to pakker i WebAPI solution, selve API'et og så en administrator side. Det er blevet valgt at der kun testes den del der er essentiel for systemet, alt API delen. Derfor er det kun AccountController, EventController og PictureController som testes.
Derudover bliver AccountController ikke testet i UnitTest da den primære funktionalitet i controlleren er fra Identity Core som forventes at være gennemtestet i forvejen da det er et officielt framework bakket op af Microsoft.Desuden har den af den grund også UserManager som afhængighed og den er for kompleks til at blive faket. AccountController bliver testes i integrations test.
De to controllere eventcontroller og picturecontroller modultestes da de begge holder den primære funktionalitet i API'et.

Da den primære funktion af web API'et er at håndtere kommunikationen og overførslen af data mellem delsystmerne. Det vigtigste at teste for API'et er altså at den kalder sine afhængigheder som forventet samt med de forventede værdier. Problemet i at Unit Teste API'et er at så snart andre delsystemer inkluderes bliver det til en integrationstest. Netop for at undgå at API'et integrationstestes fejlagtigt er det valgt at fake samtlige afhængigheder og blandt andet bruge dem som mocks for at kunne aflæse kaldende til dem og hvad de blev kaldt med. Til at opsætte en testsuite bruges Nuget pakken NUnit, og til at fake afhængighederne

[1] Bilag/SystemArkitektur.pdf
[2] Bilag/Systemarkitektur.pdf - afsnit 'Arkitektursignifikante designpakker'

bruges pakken NSubstitute.

Størstedelen af kompleksiteten i at teste systemet kom fra at fake afhængigheder. Specielt det at skulle undgå at bruge test computerens eget filsystem under billede oprettelse og sletning, samt det at skulle finde en måde at returnere den bruger som er logget ind som også ville tillade at fake det, da det kræver at der er et interface. For at kunne fake det lokale filsystem er de funktioner som gør brug af det blevet pakket ind i en FileSystem klasse, som har et tilhørende interface. Desuden laves et interface og en klasse til at pakke bruger funktionaliteten ind. Denne klasse indeholder ud over funktionerne også en Http context som bruges til at finde den bruger som er logget ind. Implementeringen af repositories til databasen bruger interfaces og er derfor ligetil at fake.

I starten af de brugte testsuites er defineret nogle test objekter som bruges til at definere retur værdier fra de kaldte afhængigheder, dernæst bruges disse test objekter til at vide hvilke værdier der kan forventes både som returværdier og som parametre i testsne. På figur. 1 ses et eksempel på hvordan kaldende til afhængigheder testes.

```
[Test]
0 references | Andreas, 10 days ago | 1 author, 1 change | 0 exceptions
public async Task GetPictureIds_EventRepo_GetEventByPinCalled()
{
    //Arrange
    _eventRepo.GetEventByPin(_testEvent.Pin).Returns(_testEvent);


    //Act
    await _uut.GetPictureIds(_testEvent.Pin);

    //Assert
    await _eventRepo.Received(1).GetEventByPin(_testEvent.Pin);
}
```

Figur 1: Test af afhængighedskald fra GetPictureIds() action, i picturecontroller

Desuden testes jo også om de forventede returværdier bliver returneret. Et eksempel på dette kan ses nedenfor på figur. 2 . Det kan også ses på figuren at der ikke bare testes for at den rigtige postcode kommer retur, men også om de kommer retur med de forventede værdier.

```
[Test]
0 references | Andreas, 10 days ago | 1 author, 1 change | 0 exceptions
public async Task GetPictureIds_ReturnsOk()
{
    //Arrange
    _testEvent.Pictures = new List<Picture> {_testPicture};
    _eventRepo.GetEventByPin(_testEvent.Pin).Returns(_testEvent);


    //Act
    var response = await _uut.GetPictureIds(_testEvent.Pin);
    var statCode = response as OkObjectResult;
    var result = statCode.Value as PicturesAnswerModel;

    //Assert
    Assert.That(statCode, Is.Not.Null);
    Assert.That(statCode.StatusCode, Is.EqualTo(200));
    Assert.That(result, Is.Not.Null);
    Assert.That(result.PictureList.Contains(_testPicture.PictureId));
}
```

Figur 2: Test af returværdi fra GetPictureIds() action, i picturecontroller

MobilApp

I mobilappen laves der unit tests for både modeller og viewmodeller, i henhold til MVVM arkitekturen.
I viewmodellerne testes der kun de steder det giver mening. Det betyder, at det ikke er alt funktionaliteten i viewmodellerne der testes. Blandt andet testes databindingen mellem viewmodellerne og view'et ikke, da dette ikke er muligt. Derimod testes dette visuelt.

Der er valgt kun at teste nogle dele af mobilappen, der findes hensigtsmæssige at teste. Derfor vil modeller, der blot indeholder get og set metoder ikke blive unit testet.
Nogle af klasserne indeholder også funktionalitet, der snakker sammen med hardware på den enhed appen kører på. Dette er ikke muligt at teste, og unit testes derfor ikke.

Der vælges ikke at unitteste viewmodellerne, da det er vurderet at udbyttet af dette ikke er tilstrækkeligt, ifht. udarbejdelsen af testene. Det skyldes da viewmodellerne indeholder funktionalitet, der er svær at teste eller som ikke er mulig at teste (eksempelvis databinding). Derimod integrationstestes viewmodellerne med de modeller de bruger, der findes hensigtsmæssige at integrationsteste i mellem.

Først unit testes modellerne og det verificeres at disse virker som forventet. Derefter foretages der integrationstest mellem viewmodellerne og modellerne. Så for at teste, at viewmodellerne virker efter hensigten, gøres dette ved at integrationsteste disse med modellerne.

Hardwaremodeller
Alle hardwaremodels er (som navnet antyder) tæt knyttet til hardwarekomponenter på mobilen. Derfor er de alle svære eller umulige at teste, medmindre alt bliver faket. Hvis alt fakes, vil testene blive ligegyldige, fordi der kun testes på data, der er skabt af testeren.
Dog er næsten alle harware modeller baseret på funktionalitet fra en enkelt NuGet package. Derfor kan det antages, at modellerne fungere, fordi det kan antages at de underliggende NuGet packages er blevet testet. Klassen MemoryManager er ikke baseret på en enkelt NuGet package, men derimod NuGet packagen PCLStorage og C#'s File klasse. Begge klasser kan antages som unit testet og resten af funktionaliteten kan "testes"run time igennem Debug.WriteLine.
App'en har en enkelt DependencyService - klassen IFileDirectoryAPI. Denne klasse finder de korrekte filstier for både Android og iOS run time og er derfor også svær at teste fra en PC.

Webapplikation

Der har i pensum for semesteret ikke indgået emner omkring test af hjemmesider som baseret på html og javascript, så der er derfor ikke oprettet nogle automatiske tests til webapplikationen. WebApplikationen er først og fremmest testet ved visuelle tests. Her er det testet hvorvidt applikationen fungerede, som det var forventet. Hvis dette ikke var tilfældet, er koden debugget ved hjælp af debuggeren i Google Chrome, hvorefter fejlen er forsøgt fundet for den pågældende funktion.

De mange asynkrone kald som fremkommer med adskillige fetch kald til web API'et har gjort det vigtigt at teste rækkefølgen af kald i systemet. Dette har været gjort med network analyse i Google Chrome, hvor

det er muligt visuelt at se hvornår de forskellige kald starter og slutter. Dette har været brugt til at teste hvorvidt async/await og brugen af promises har fungeret, og verificere at funktionaliteten har været som ønsket

Derudover er der lavet bruger test, hvor andre medlemmer af gruppen har benyttet webapplikationen, for at teste hvorvidt brugergrænsefladen har været fornuftigt at benytte. Her er respons fra gruppe medlemmer taget op, og brugergrænsefladen er tilpasset kommentarene, eks. er routingen fra Host Sign Up ændret, sådan at brugeren automatisk logger ind. Denne form for tests har ikke været teknisk, men har bidraget meget til at forbedre brugergrænsefladen meget.

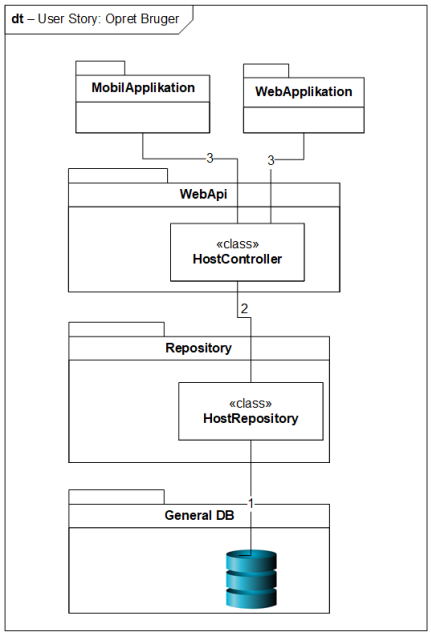
Det havde for webapplikationen været en stor fordel at benytte automatiske tests. Når der har været ændret i systemet, har det altid været usikkert hvorvidt det har medført fejl andre steder i koden. Der er eks. funktioner, som benyttes flere steder, og når disse er ændret, har det været umuligt at overskue konsekvenser som dette har kunne medfører. Dette vil derfor være en vigtig ting at implementerer i et fremtidigt videre arbejde eller et andet projekt.

2 Integrationstest

Der udføres ikke integrationstest internt i nogle pakker af systemet. Men grænsefladerne imellem pakkerne testes struktureret. Systemet er blevet udviklet agilt og derfor er dele af pakkerne i systemet også blevet integrationstestet agilt. Der er blevet udført en integrationstest til hver user story beskrevet i kravspecifikationen [3].

Der vil blive gennemgået hvordan integrationstesten af en user story er blevet udført. Der ses på user story'en Opret Bruger. Der kan på figur 3 ses et dependency tree, der viser afhængighederne inkluderet i user story'en.

[3] Bilag/Kravspecifikation.pdf



Figur 3: Dependency tree for systemet for user story: Opret Bruger

Der er valgt at variere i abstraktionsniveauet for de forskellige pakker for at holde træet så overskueligt som muligt. Der kan også ses hvilke rækkefølge grænsefladerne testes i, det er nummeringen på figuren.

Forklaring af hvordan de forskellige trin udføres:

- Trin 1: Repository'et kobles på den rigtige database. Der udføres lignende test af hvad der blev udført i modul testen af repository'et.
- Trin 2: WebAPI kobles på. Der testes ved brug af Postman.
- Trin 3: Web- og MobilApplikation kobles på. Der testes visuelt på hhv. webApplikationen og mobilApplikationen.

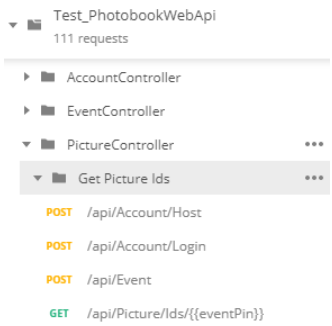
2.1 Trin 1 - Repository og General database

Der udføres automatiske test med NUnit, hvor repositoriet laver kald op imod en MSSQL DB. Der udføres generelle CRUD kald for alle repository's. Databasen struktur indeholder en cyklisk reference og bliver derfor tilgået igennem en transaktion ved manipulering af data. Derfor er der også udført ekstra tests, der tester adfærden ved manipulation. Det vigtigste er at se at om entiteters relationer opfører sig som forventet ved sletning.

2.2 Trin 2 - Tilkobing af WebAPI

Til integrationstest af WebAPI anvendes programmet Postman. Postman er et program der kan lave Http kald og vise hvilket response der kommer. Når der bliver testet med postman testes der kald direkte på det release der ligger på serveren. Det betyder at der er ikke noget der bliver faket og det er en "Big Bang"integrations

test. Postman bliver brugt til integrations test for at man kan teste af kaldene kommer ind til de forventede actions. Det er blackbox tests da der kun testes på hvilke response koder der kommer tilbage i forskellige situationer.
I postman kan man lave collections af kald. I vores system er testene delt op efter controllers. Hver test er inde i hver deres mappe under hver controller mappe. Dette er fordi hver test indeholder ARRANGE, ACT og ASSERT trin. I det der bliver vist på figur 4 er der en test der indeholder 4 requests. Her er de tre første kald en del af ARRANGE trinnet. Det sidste kald er ACT og ASSERT trinnet det vil sige det er her der er en test skrevet til som set på figur 5



Figur 4: Viser en test collection i postman

```
1 pm.test("Status code is
      200", function () {
2     pm.response.to.have
          .status(200);
3 });
```

Figur 5: Viser test kode for en test, her bliver assertet at response koden er OK(200)

Som nævnt under unit test af WebAPI er det kun den del der er essentiel for systemet som bliver testet. Admin siden bliver ikke testet da den mere har været til os selv for at se hvad der bliver kaldt i loggen og se hvad der bliver oprettet i databasen.

2.3 Trin 3 - Tilkobling af Web- og Mobilapplikation

Web- og MobilApplikation kobles på det resterende system. Der observeres om applikationerne opfører sig som forventet i forhold til de funktionelle krav specificeret i kravspecifikationen.

Webapplikation

Til integrationstest af webapplikation og WebAPI, er der benyttet network analyse i Google Chrome. Her har de enkelte netværks kald testest, hvor det sendte og modtagende data er checket efter. Det har også været muligt at teste hvilken respons status som WebAPI har returneret. Dette har været stærkt medvirkende til at der er blevet fundet fejl på både webapplikationen og WebAPI. Integrationen af systemerne er sket gradvist som de er blevet udviklet, og er ikke lavet som en endelig test. Dette har også medvirket til at systemerne har været lette at fejlfinde på i processen, da mængden af ting som kunne fejle, hele tiden blev

holdt på et minimalt niveau. Dette er klart at foretrække fremfor at teste to fuldt udviklede systemer samtidigt.

Et eksempel på fejl fundet i integrations testen er fejl vedrørende CORS policy og credentials. CORS policy omhandler Cross-Origin Resource Sharing. Det er en blokering, som sikrer at fremmede systemer ikke kan tilgå serveren uden tilladelse. Fetch kaldende i systemet krævede at der var en CORS tilladelse, så blev løst ved at tilføje deployment adressen for webapplikationen samt localhost:1337 for development testing, til en CORS white list i WebAPI.

Credentials problemet var et lignende problem, som vedrørte at sende identity cookies mellem server og klient. Uden dette var det ikke muligt at tjekke om en klient var logget ind, eller om der var tale om en host eller guest. Dette skulle tillades på serveren og fetch kald skulle have attributten credentials="include", hvorefter at systemet fungerede som ønsket.

Dette var browser specifikke fejl, som ikke tidligere var opdaget i systemet, og en strukturet integration af systemet var derfor stærkt påvirkende til at det var muligt at løse udfordringerne.

3 Accepttest

Efter afsluttet integrationstest af systemet udføres der accepttest. Accepttesten er beskrevet i accepttestspecifikationen[4].

3.1 Funktionelle

Accepttesten udføres ud fra specifikationen i Accepttestspecifikationen og udføres med både webapp og mobilapp på samme tid. Resultatet af testen kan ses på tabel 1.

[4] Bilag/Accepttestspecifikationen.pdf

Tabel 1: Accepttest

Egenskab	Ok/Fail
Opret bruger	Ok
Logge ind - som Host	Ok
Logge ind - som Guest	Ok
Upload billeder	Ok
Tag og upload billede	Ok
Download billeder	Ok
Oprette event	Ok
Log ud	Ok
Se billeder	Ok
Slette billeder	Ok
Slette event	Ok
Logge ind som administrator	Ok

Som det kan ses på tabellen, lykkes det at gennemføre alle tests og systemet fungerer derfor efter hensigten.

3.2 Brugrænseflader

Brugergrænsefladerne stemmer overens med skitserne illustreret i kravspecifikationen.

3.3 Ikke-funktionelle

Tabel 2: Accepttest af ikke funktionelle krav

Nr.	Krav	Test/Udførsel	Forventet Resultat	Faktiske Resultat	Vurdering
1	Synkronisering gennem systemet (MobilApp til WebApp) skal tage 1 minut +/- 5 sek	Der logges ind på samme bruger på mobil- og webapp. Der oprettes et event og stopur startes. Siden/appen refreshes efter 60 sek.	Eventet er tilgængeligt	Event er tilgængeligt	OK
2	Starttid af MobilApp skal tage 30 sek +/- 1 sek	MobilApp startes på smartphone. Stopur startes. Der observeres om applikationen er startet efter 30 sek.	Applikationen er startet på telefonen	Applikationen er startet på telefonen	OK
3	Start af WebApp skal tage 30 sek +/- 1 sek	WebApp åbnes i en browser. Stopur startes. Der observeres om siden er loadet efter 30 sek.	Siden er loadet på browseren.	Siden er loadet på browseren	OK
4	Når der er uploadet et billede skal det være tilgængeligt under event efter max. 30sek +/- 1 sek	Der logges ind på samme event og uploades et billede. Stopur startes. Der ses om billedet er under eventet efter 30 sek.	Billedet kan ses.	Billedet kan ses	OK
5	Når man vil se alle billeder på Mobil- og WebApplikation, så må det maks tage 5 sek +/- 1 sek inden du ser det første billede	Der logges ind på et event på mobil-/webApp. Stopur startes. Der observeres om alle billeder er vist efter 5 sek.	Der ses et billede	Der ses et billede	OK
6	WebAppen skal have et responsivt design fra opløsningerne: (375 x 667 til 1080 x 1920)	WebAppen åbnes i en browser og størrelsen ændres til hhv. (375 x 667) og (1080 x 1920)	Webapplikationen har et responsivt design	Webapplikationen har et responsivt design	OK
7	MobilAppen skal have et responsivt design til-/fra landscape mode	MobilAppen åbnes på en telefon. Telefon ligges ned.	MobilAppen har et responsivt design	MobilAppen har et responsivt design	OK
8	WebAppen skal virke på forskellige browsers	WebAppen åbnes i Chrome og Microsoft Edge	WebAppen virker i begge browsers	WebAppen virker i begge browsers	OK
9	Web- og MobilApplikationen skal kunne bruges af en person, der ser den for første gang	En førstegangs bruger prøver Mobil-/WebAppen	Brugeren mener, at de begge er intuitive	Brugeren mener, at de begge er intuitive	OK
10	MobilAppen skal kunne opere på forskellige OS'er	MobilAppen hentes ned på Android og IOS og afprøves	MobilAppen virker på begge OS'er	Der er kun testet på Android	FEJL