Skrevet af: Benjamin, Jari, Henrik, Jacob & Daniel

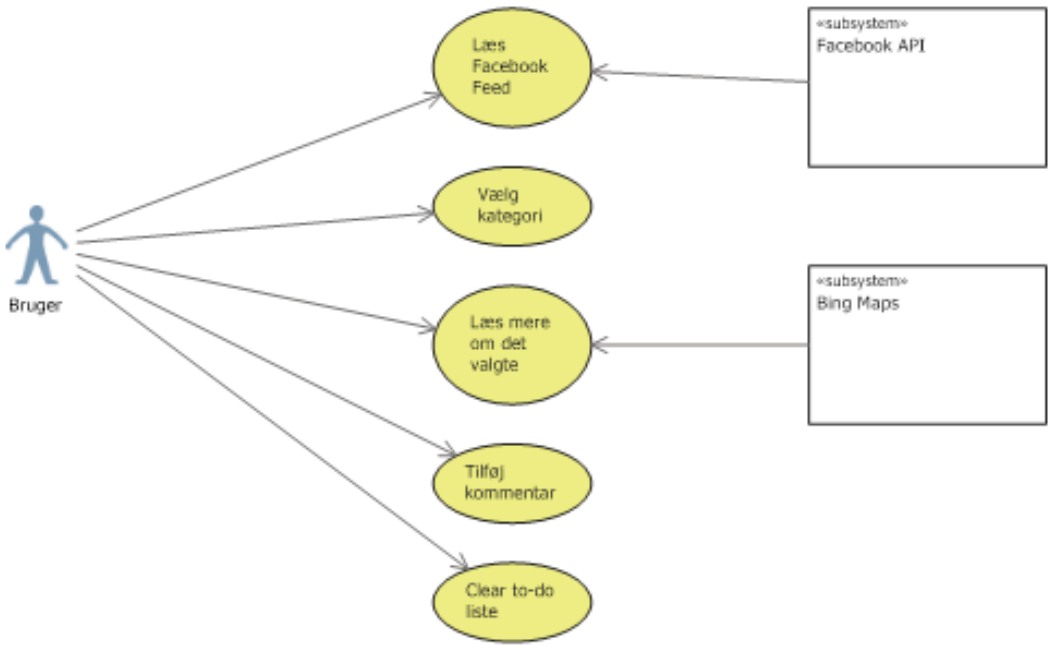
#VisitRoskilde

Dokumentation

Indhold

Softwaredesign

Use Case diagram



Vi har i vores Use Case beskrivelser taget udgangspunkt i at formulere brief Use Case, hvor Use Case beskrivelsen i korte vendinger beskriver hvad brugerens handlinger på appen.

Aktør: Brugeren
Supporting aktør: Bing Maps og Facebook API

Use Case 1 - Læs Facebook Feed

Brugeren læser Facebook Feeds i højre side af vores GUI, hvor der bliver vist de seneste posts på fra deres Facebook-side. Der kan også læses generelle informationer om Visitroskilde.

Use Case 2 - Vælg kategori

Brugeren vælger en kategori - aktiviteter, restaurant, butikker eller seværdigheder - og får vist en liste af disse. Brugeren kan læse de seneste kommentarer, som er skrevet af andre.

Use Case 3 - Læs om det valgte

Efter valget af en kategori, som sker i Use Case 2, kan brugeren læse om den valgte seværdighed, butik eller oplevelse, samt aflæse destinationen som er vist på et Bing Maps. I højre side kan man aflæse de seneste kommentarer om stedet.

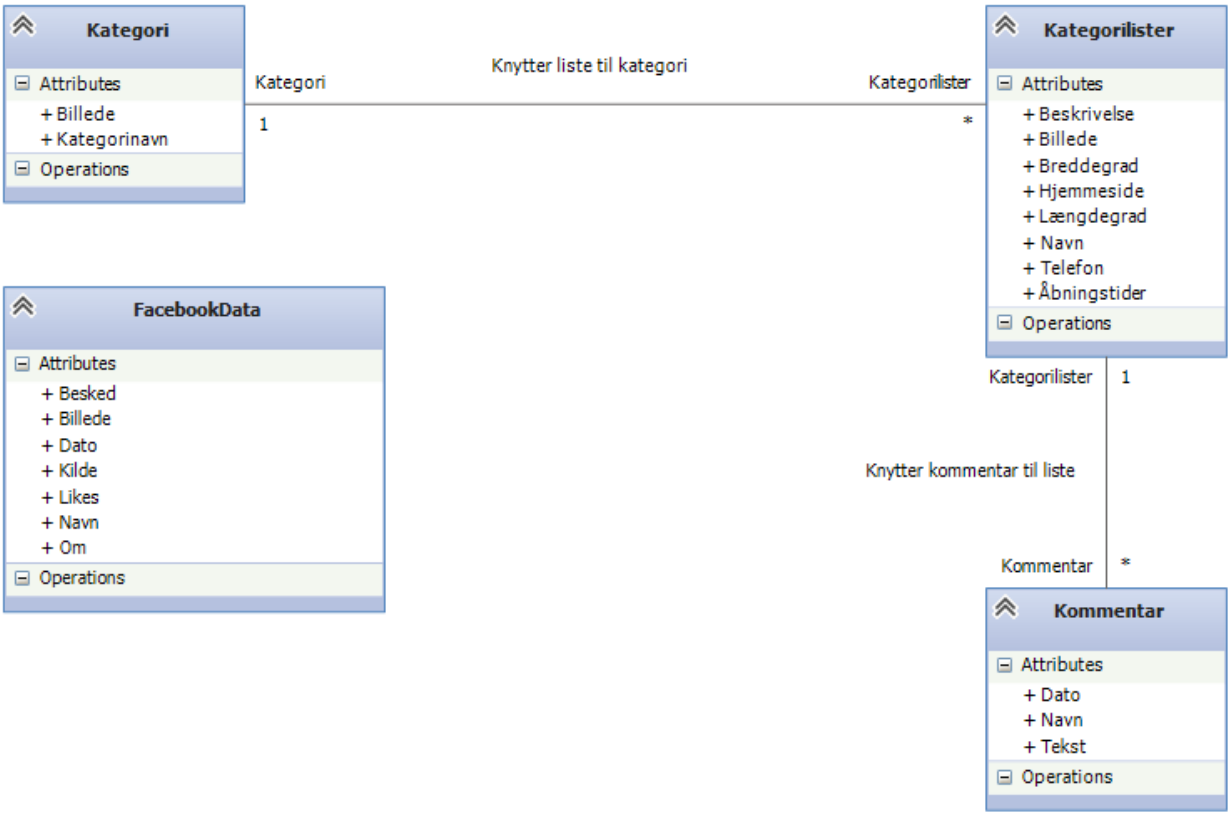
Use Case 4 - Tilføj kommentar

Brugeren kan knytte en kommentar til det valgte punkt, som bliver vist under “Læs om det valgte”.

Use Case 5 - Clear to-do liste

Brugeren kan clearer hele to-do listen, så alle elementer bliver fjernet.

Domain model

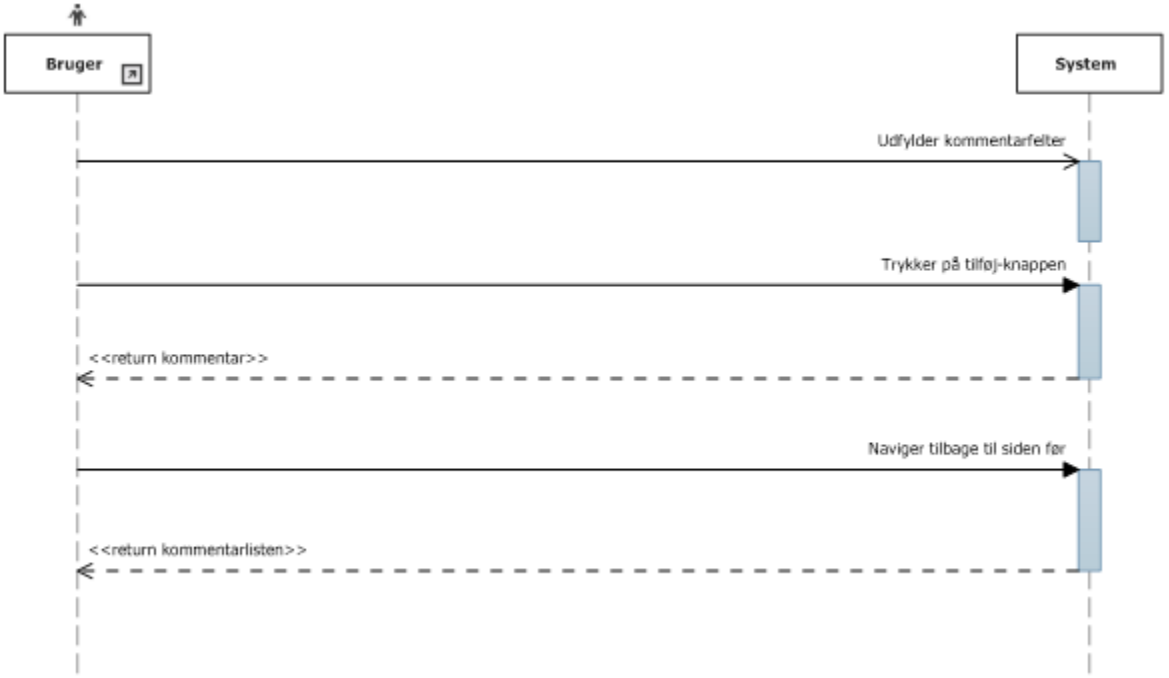


Vi har valgt at lave Kommentar klassen , som består at navn tekst og dato, da det er væsentligt når man skal indtaste en kommentar. Derudover skal der også være tilknyttet kommentar til de forskellige kategorilister, da man skal se kommentar til de forskellige steder, som ligger inden i de forskellige kategorier. De forskellige aktiviteter har en masse attributter og de er; navn da begivenheden skal have et navn, beskrivelse så brugeren har en mulighed for at læse om hvad begivenheden går ud på, da der er også bing map med i vores projekt er der bredde og-længdegrader med som attribute. der er også et billede, telefon og åbningstider som giver brugeren et bredere overblik over aktiviteten. Derudover er der vores kategori klasse som indeholder billede og kategorinavn, som kan vise et navn over en kategori og tilhørende billeder. Det er også med til at give brugeren et overblik hvad brugeren kan vælge, af forskellige aktiviteter.

Det væsentlige denne sammenhæng er, at flere lister kan knyttes til en kategori og ligeledes kan flere kommentarer skrives til kategorlisterne.

Ydermere har vi en FacebookData-klasse, der indeholder oplysninger vi ønsker at have med i vores program, herom, besked, billede, datote, kilde, likes, navn, om.

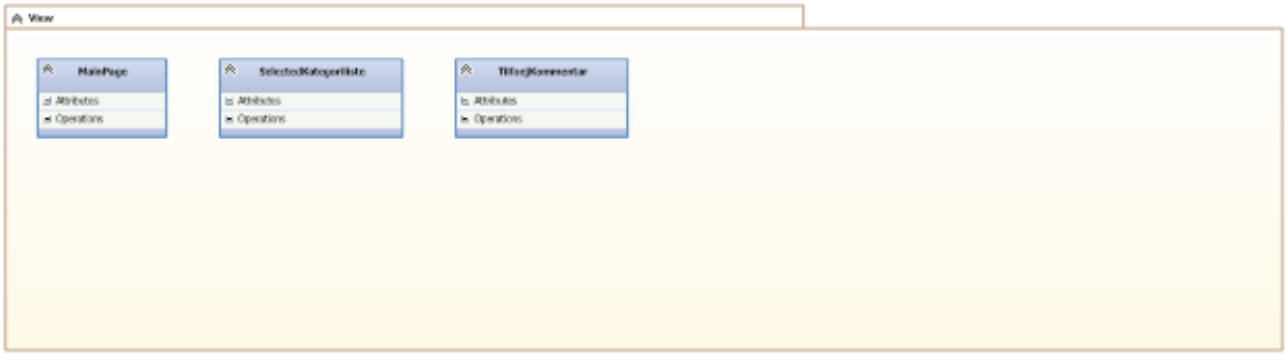
Systemsekvensdiagram



I vores systemsekvensdiagram har vi taget udgangspunkt i at tilføje kommentar og få den vist. Vores bruger indtaster navn, restaurant samt kommentaren til den valgte kategori, hvorefter brugeren tilføjer kommentaren til kommentarlist. Programmet udskriver en kommentar for at synliggøre at kommentaren er tilføjet. For at se kommentaren kan brugen gå tilbage til profilen for begivenheden og se de seneste kommentarer.

Designklassediagram

View



MainPage

Vores view består af 3 page; en MainPage, SelectedkategoriKlasse, TilføjKommentarHandler. Vores MainPage består af en slags form for præsentation af appen samt af Roskilde, hvor der i venstre side er forskellige collection, hvor det der appen primært køre. I højre side er tilføjet facebook info. For at komme videre til det næste view vælger man en kategori og en aktivitet via vores implementeret appbar. Der er en mindre undtagelse ved kategorierne i forside, hvor der er en profil, som fungerer som valgte aktiviteter.

SelecterdKategoriliste

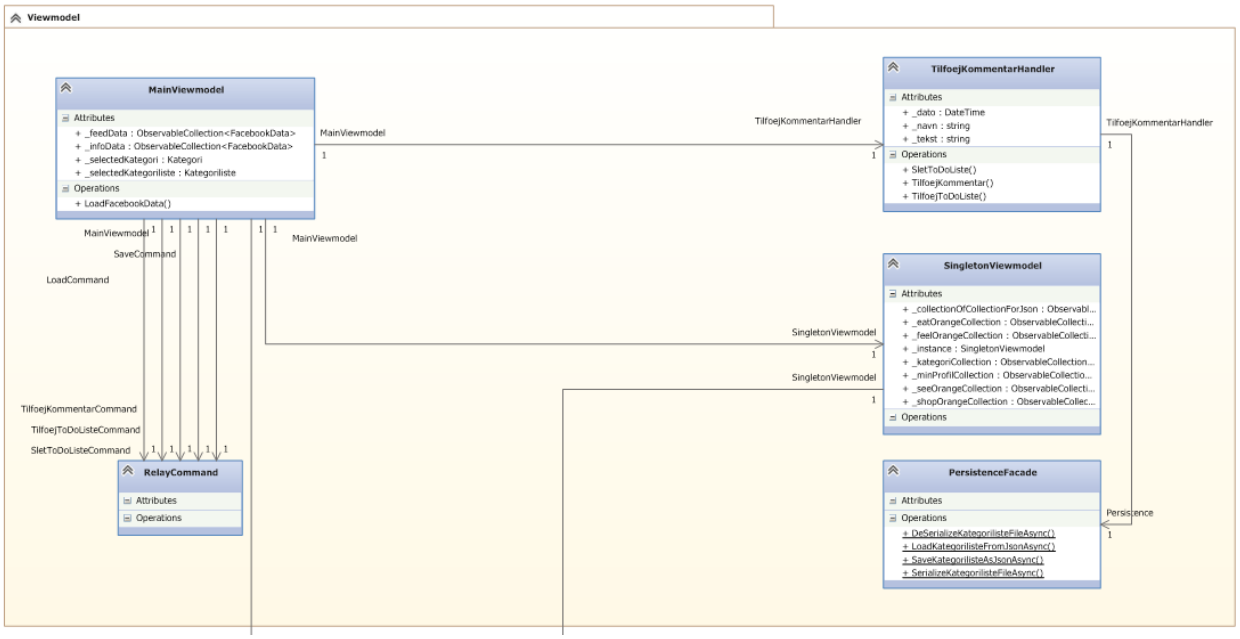
Ved view 2, som er visning af den valgte aktivitet, hvor der frem for information samt mulighed for at besøge hjemmesiden. Der er i det nederste højre hjørne tilføjet Bing Maps, hvor brugeren har mulighed for at se, hvor aktiviteten er og på den måde kan finde vej til aktiviteten via GPS. I højre side af appen vises der kommentarer fra andre gæster, hvor man kan høre deres holdning til stedet.

TilfoejKommentar

Ved valg af det sidste view tilføjer man en kommentar, som indeholder navn, evt. anden aktivitet og beskrivelse af aktiviteten.

Viewmodel

Vores Viewmodel er sat som DataContext i vores Views og gør at vi kan binde de forskellige ting med hinanden.



Mainviewmodel

Der er dannet en association mellem FacebookData og Mainviewmodel, hvor både feed'et og info-teksten bliver lagt ned i 2 ObservableCollections. Desuden er der lavet en association til TIlfoejKommentarHandler samt til vores SingleViewmodel. LoadFacebookData-metoden laver kaldet til Facebooks API med en Access Token og ligger herefter de returnerede værdier ned i en ObserableCollection. Vores load og save command ligger i vores Mainviewmodel, da Relaycommand'en har en objektreference i Mainviewmodel.

TilfoejKommentarHandler

Handleren håndterer de informationer som bliver indtastet af brugeren samt når der bliver tilføjet noget til to-do listen. Handleren har en reference til PersistenceFacade, som indeholder de statiske Load og Save-metoder.

SingletonViewmodel

Grunden til at singleton har en objekt reference fra Kategoriliste, samt at den har en object reference i mainviewmodel er at den skal samle alle informationerne, dels overholder man GRASP-principper om lav kobling.

PersistenceFacade

Klassen indeholder de statiske metoder som gemmer og henter data fra en JSON-fil. Associationen fra Handleren til Facade er lavet for at kunne gemme data i en JSON-fil.

Model

Vores Model er bestående af 4 forskellige klasser:

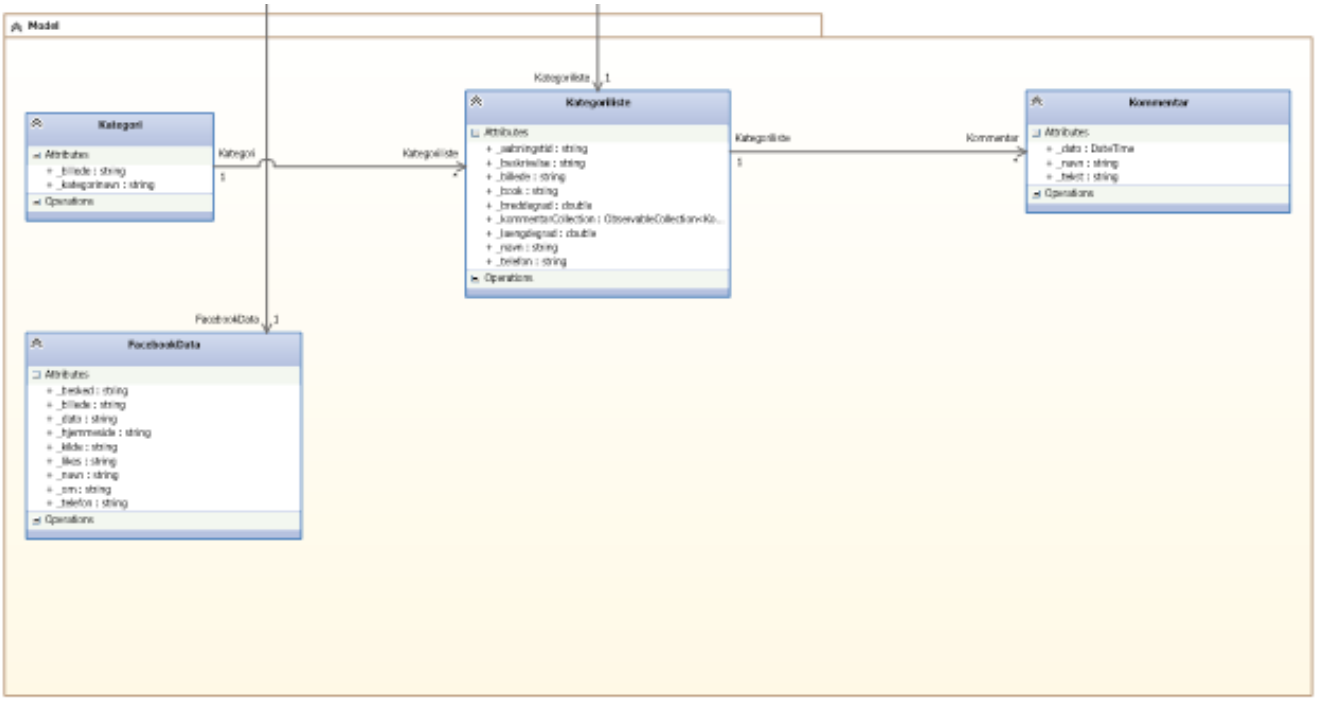
- Kategori
- Kategoriliste
- Kommentar
- FacebookData

Kategori-klassen holder styr på de forskellige overordnede emner så som Eat Orange, Shop Orange, Feel Orange, See Orange. Kort fortalt er det vores kategorier.

Kategorilisten består af selve de aktiviteter, restauranter, butikker og seværdigheder, som er knyttet til en kategori. Heraf kommer associationen fra Kategorliste til Kategori. Diagrammet fortæller, at der kan oprettes flere lister til en kategori.

Kommentar-klassen består af en association mellem kommentaren og en kategoriliste. Der kan oprettes flere kommentarer til en liste.

FacebookData-klassen holder styr på de attributter, der knytter sig til de data, som der bliver vist på MainPage.



Business Model Canvas

UD fra Canvas har vi valgt at betragte Visit Roskilde, som samarbejdspartnere da vi ikke får løn eller sælger appen til dem. Istedet har vi valgte at fokusere på Visit Roskilde og brugt dem som vores udgangspunkt.

Vi har valgt at fokusere på alle punkterne ud over cost structure og revenue stream, hvor de sidste 2 punkter er mere op til VisitRoskilde selv, eftersom att de har budgetter og andet til appen.

Customer segments

Primære målgruppe: Visit Roskildes primære målgruppe er turisterne, hvor de så vidt muligt forsøger at skaffe flere turister til Roskilde kommune. Appen er udviklet til at turisterne skal downloade den og besøge Roskilde. Målgruppen for turisterne er 20-40 år, hvor der er fokus på familieturer. hvor appen har samlet tilbud og aktiviteter for Roskilde. Selve turisterne er både udenlandske og danske turister, hvor appen er på dansk, men kan oversættes til dansk i den færdige udgave. Dette kræver dog at de fra Visitroskildes siddes lægges en ny marketingstrategi for, hvordan de vil udfolde sig på Facebook på engelsk.

Sekundære målgruppe: Den skundære målgruppe er de forskellige aktiviteter, som museer og oplevelser der kan opleves i Roskilde. Visit Roskilde tager et mindre beløb for, hver billet de sælger til de forskellige aktiviteter.

Channels

Appens benytter app store til at komme ud til kunden. Derudover kan der også benyttes det blad, som “Visit Roskilde” laver, hvor man kan distribuere den via QR kode.
Visit Roskilde kan også få de forskellige butikker, restauranter og oplevelser/aktiviteter til at reklamere via banner, små folderer og et lille skilt der også indikerer at de er med i samarbejdet.

Relations

Vores relation til den primære målgruppe er lav, eftersom at kunden henter den på nettet. Kunderne vil opleve at det er en “envejskommunikation” der foregår fra Visit Roskilde.
Den sekundære målgruppe har et tættere samarbejde, hvor Visitroskilde skal forsøge at få dem til at acceptere appen. Relationen er derfor god og vil fortsat blive vedligeholdt med møder, hvor man der er mulighed for at forbedre appen.

Value propositions

Visit Roskilde's app er en service, hvor værdien til tider er sværere at vurdere.
I forhold til den sekundære målgruppe, hvor Visit Roskilde skaber værdi for diverse aktiviteter via reklame for kommunen og på den måde tiltrække flere turister, hvilket er mulige kunder for . Den primære målgruppe, er turisterne og de får en nem tilgængelig oversigt aktiviteterne, som ligger i appen. Appen indeholder derudover også booking system, hvor det skaber en form for værdi ved at man kan købe billetten til den valgte aktivitet man vil se en bestemt dag.

Key resources

For at skabe vores value propositions, skal vi selvfølgelig have udviklet appen, men også have informationen og samarbejdet imellem os og de forskellige virksomheder, hvor de skal overbevises til at deltage i det samarbejde omkring appen.

Key activities

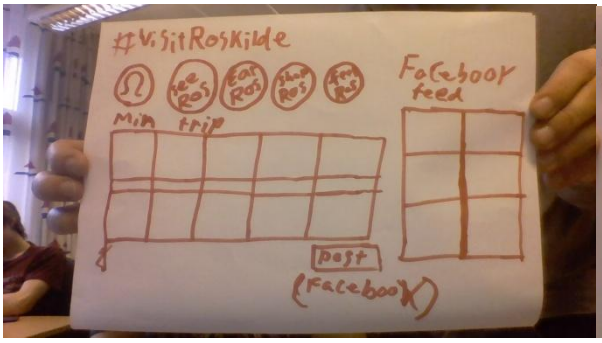
Vores key activities består af vores udvikling af appen, hvor man samtidig skal have forholdet/netværket op til ens partnere, for at skabe det bedst mulige samarbejde.

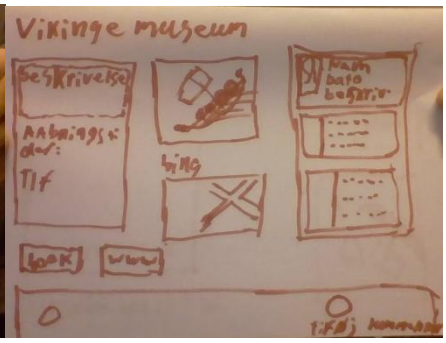
Key partners

Vores key partners er de forskellige turistattraktioner, butikker og oplevelser. Man kan også argumentere for at key partnerne er leverandører og kunder, hvor vi her tager udgangspunkt i at de til dels deltager i et samarbejde, hvor attraktionerne betaler for at man kan booke gennem appen, hvilket gør dem til partnerer og kunder for virksomheden. Vores nøgle ressourcer

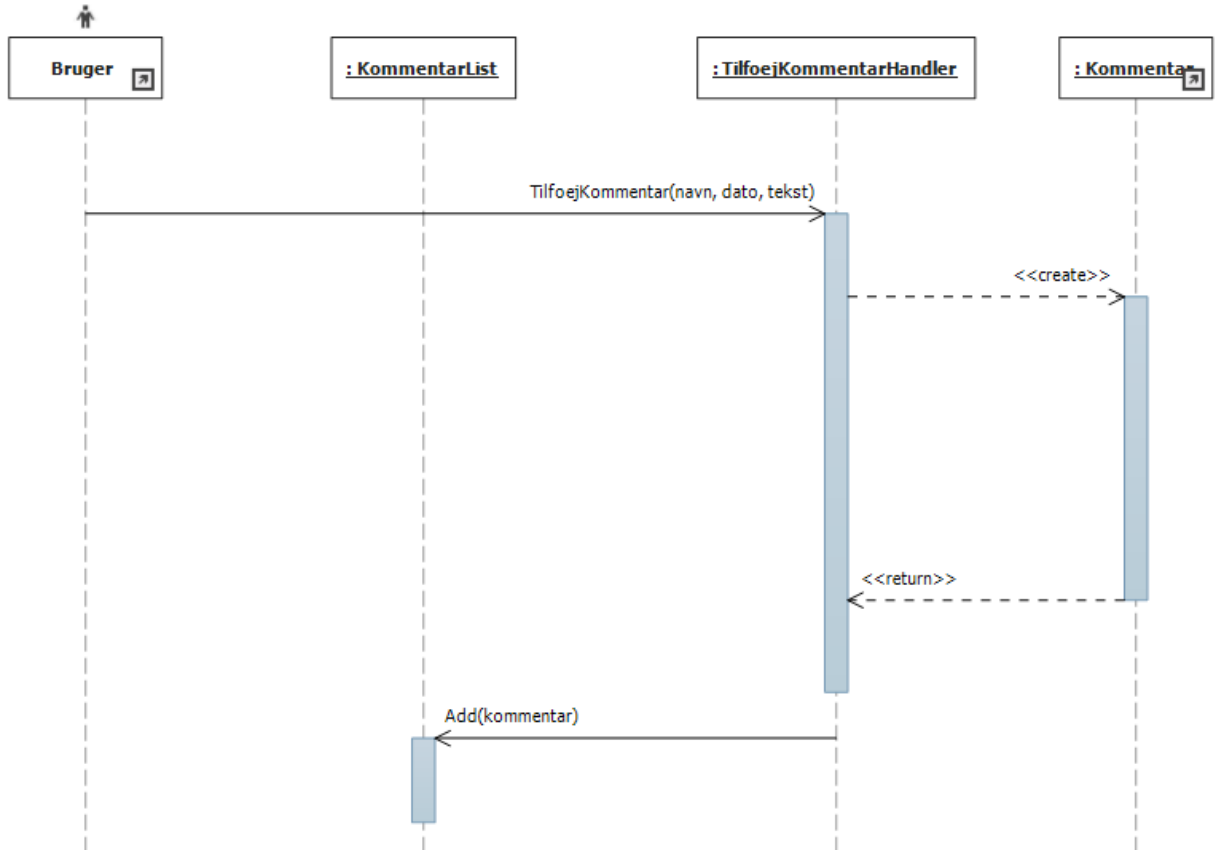
GUI prototype

Vi har anvendt programmet POP, hvilket vi gjorde som noget af det første i vores proces imod den færdige app. Vi startede med at udveksle tanker om design samt programmet kunnen. Derefter lavede vi et par skitser og blev enige om et design, for at få et fælles udgangspunkt så vi kunne gå igang med de forskellige diagrammer. Nedenstående ses vores skitser.





Sekvensdiagrammer



I vores sekvens diagram har vi tre Objekter, kommentarliste, og TilføjKommentarHandler. Brugeren tilføjer en kommentar, til TilføjKommentarHandler, som skaber en objektreference med informationer fra Kommentar klassen, det bliver returneret til Handleren, som tilføjer objektreferencen til Kommentarlist.

Test Case

#	Beskrivelse	Forventet	Resultat
1	Navn er 1	OK	
2	Navn er null	Fejl	
3	Navn er empty	Fejl	
4	Navn er 30	OK	
5	Navn er 31	Fejl	
6	Tekst er 20	OK	
7	Tekst er 19	Fejl	
8	Tekst er 500	OK	
9	Tekst er 501	Fejl	
10	Tekst er null	Fejl	
11	Tekst er empty	Fejl	

I vores projekt har vi valgt at teste navn og tekst. Grænseværdierne er lavet for at teste at vores bruger overholder reglerne for indtastning af antal tegn i vores GUI.
Man kan se på testplanen at tekstmængden er mellem 20 og 500 tegn, samt at navnet skal have en længde som er mellem 1 og 30 tegn. Grænseværdierne er derfor sat til 501 og 19 for test metoderne af kommentarbeskrivelser og grænseværdien er sat til at være 31 for kommentarnavn.Derudover har man testet for om stringen er oprettet og om den er empty.

Software Construction

MVVM arkitektur

Datacontext og binding

For at få MVVM arkitekturen til at virke mellem lagene skal vores Views have Datacontext til vores MainViewModel som sjovt nok ligger i ViewModel laget. Binding gør at der er en del kode, som man slipper for at programmerer, samt skabe et dataflow i programmet, hvor man henter data fra bruger input ved twoway, men hvor man ved normal binding finder kildens kode.

MVVM er opdelt i tre lag:

Model

Model laget indeholder klasserne:

Kategori.cs
Kategoriliste.cs
Kommentar.cs

View

View laget indeholder forskellige Views som er de sider brugeren kommer til at navigere på.

Mainpage.xaml
SelectedKategoriliste.xaml
TilfoejKommentar.xaml

ViewModel

MainViewmodel.cs
PersistenceFacade.cs
RelayCommand.cs
SingletonViewmodel.cs
TilfoejKommentarHandler.cs

ViewModel laget indeholder de klasser som styrer de visuelle elementer der kan interageres med af brugeren fx koden bag en knap. Viewmodel er den der binder i mellem de 2 andre lag Model og View, hvilket også gør den så vigtig.

Relevante datastrukturer

Datastrukturer hjælper os til at kode mere effektivt, ved at have nogle pre-kodet strukturer vi kan benytte os af.
Den mest relevante datastruktur for vores projekt har været ObservableCollection(element). ObservableCollection har en masse pre-kodet metoder som vi benytter os af. Bl.a. Add(element), som tilføjer et element til ObservableCollection.
En datastruktur der minder om ObservableCollection er List(element). List er i sin enkelthed bare en liste, hvor man, ligesom med ObservableCollection, kan tilføje og fjerne elementer. For at tilføje eller fjerne elementer benytter man de pre-kodet metoder Add(element) og Remove(element). Både ObservableCollection og List har mange flere pre-kodet metoder, men få som vi har benyttet os af i vores projekt.

Binding

Bindings er forbindelsen mellem kode og view. Bindings er en vigtig del af MVVM, da bindings forbinder model klasserne og viewmodel med viewet.
Det kan fx være at man har en ObservableCollection i sin viewmodel og man vil gerne vise indholdet på sin mainpage.
Derfor binder man ObservableCollection til evt. et gridview der ligger på mainpage.

```
<GridView Foreground="White" ItemsSource="{Binding KategoriCollection}" SelectedItem="{Binding SelectedKategori, Mode=TwoWay}" Grid.Row="1">
```

Som det kan ses ovenfor er vores GridViews ItemsSource bundet til KategoriCollection. I dette eksempel er KategoriCollection en ObservableCollection der ligger i vores ViewModel. Dvs. alle de items der ligger i KategoriCollection bliver vist i GridViewet.
Hvis vi kigger på GridViews SelectedItem, ser vi at den er bundet til SelectedKategori, Mode=Twoway.
Twoway vil sige at hvis SelectedKategori bliver opdateret i viewet, vil SelectedKategori også blive opdateret i koden. Default er bindings OneWay, hvilket vil sige at koden kan opdatere bindingen, men viewet kan ikke opdatere bindingen.

Persistens

Vi gemmer informationer om kommentarerne samt de elemeter som er tilføjet til to-do listen i en JSON-fil, som ligger lokalt på computeren for så - når programmet åbner - loader elementerne ind i div. collections.

I PersistenceFacade har vi 2 metoder, en load-metode, der læser fra filen samt en save-metode, der gemmer til filen.

Exceptions handling

Ved at bruge try and catch kan man prøve ting af som burde crashe programmet. Det kan fx være:
Brugeren har udfyldt et felt forkert
Brugeren har ikke udfyldt et felt

Brugeren har angivet for lidt eller for mange tegn i fx et telefonnummer
osv.

Det gør man ved at smide noget man har en ide om at det vil crashe programme ind i en try block.
Catch blocken kan så 'catche' med eksempelvis Assert.AreEqual statementet som sammenligner ens specficerede string som skal være den samme som bliver smidt af en throw block.

Try and catch blokkene er noget der foregår inde i Unit Test

Commands

I vores projekt har vi brugt 3 commands; vælg, tilføj kommentar, og tilføj til to do liste. Vores vælg command bruges til at vælge en begivenhed og vise hen til den næste page, den har vi med da det skal være muligt at vælge en begivenhed . Tilføj kommentar command bruges til at tilføje en kommentar til en begivenhed, så brugeren kan læse om kommentaren om begivenheden. og tilføj til todolisten som tilføjer informationer til profilen, derfor har vi valgt den i vores projekt .

RelayCommand

RelayCommand er den klasse, som gør at man kan bruge ICommand funktionen, derfor har vi den med i vores projekt.

Test class

I vores tilfælde tester man på kommentar klassen. Da brugeren skal indtaste Kommentar med navn, og dato skal man teste Kommentar klassen .Vores kommentar klasse består af 3 parameter. navn, beskrivelse og dato. Testen består af navn og beskrivelse, da test af dato er ud over 1 semester pensum..

Unit test

I vores projekt har vi lavet unit test på vores test klasse og to parameter i vores klasse . vi har lavet unit test på kommentarnavn og kommentarbeskrivelse. vi brugte vores testplanen, som udgangspunkt samt try and catch til at lave fejlhåndtering i vores unit test, i den forbindelse brugte vi Argument Exception og Assert are equal til at teste på vores metoder.

Kodeeksempel

Facebook API

Vi har så vidt muligt forsøgt at efterkomme nogle af Mettes krav og tanker til appen. Heraf har vi valgt at anvende Facebook Graph API, til at trække data ud fra Visitroskildes Facebookgruppeside.

Det førse trin i processen har været at få en Access Token, som kan bruges til, at trække public data ud via en GET request. Vores request ser således ud: **visitroskilde/feed?limit=3&fields=message, picture, created_time**. Limit-parameteret begrænser feedet til kun at udskrive max 3 samt fields-parameteret angiver hvilke felter, som vi ønsker at have med.

Inde i vores MainViewmodel opretter vi en async metode, der hedder **LoadFacebookData()**, som bliver kaldt i klassens konstruktør.

I selve metoden laves et nyt objekt af FacebookClient med referencen accessToken, der indeholder værdien af vores Access Token. På accessToken-referencen kaldes så metoden GetTaskAsync, som tager enten et gruppeID eller gruppenavn - heraf “visitroskilde”.

Vi tilføjer data til 2 ObservableCollections; en collectione indeholdende generelle informationer fra Visitroskildes Facebookside, samt en collection, der samler feedsene i et Listview control.

Måden vi udskriver feedsene sker via en foreach-løkke, der looper igennem JSON-dataen og kun tilføjer message, picture samt created_time til collectionen.

Facebook-kodeblokken er omgivet af en Try. Hvis der ikke kan forbindes til Facebook, eller hvis der opstår andre fejl under eksekveringen af koden, bliver der fanget en Exception i Catch, således at programmet ikke crasher.


```
{
  "data": [
    {
      "message": "POP POP POP-UP.....PÅ [SKÄNK]\n\nDer er kun plads til 35 pers. når [Skänk] inviterer til POP-UP restaurant den sidste onsdag i måneden - og aftenerne plejer at være komplet UDSOLGT (y)....\n\nEn perfekt oplevelses-julegave til en du holder af...\n\n3 retter | 325 kr. | sidste onsdag i februar, marts, april og juni måned | book nu: http://www.visitroskilde.dk/booking/4838",
      "picture": "https://scontent-a.xx.fbcdn.net/hphotos-xfp1/v/t1.0-9/p130x130/10665864_317715085074402_6727139357816452316_n.jpg?oh=bd6ffbca638b169f8dc0ba3b3df8e5c6&oe=55040119",
      "created_time": "2014-12-09T18:40:00+0000",
      "id": "163262523852993_317718318407412"
    },
    {
      "message": "Vi julehygger i Roskilde...kom og vær med (y)...",
```

"picture": "https://scontent-a.xx.fbcdn.net/hphotos-xpa1/v/t1.0-9/s130x130/10686645_342607472585163_1247156447080425808_n.jpg?oh=87bddfef7c74d896ae26f0f7a89e853c&oe=55425057",
"created_time": "2014-12-08T18:30:01+0000",
"id": "163262523852993_342607955918448"
},
{
"message": "Julen nærmer sig, og næste uge i Roskilde byder på en masse skønne juleaktiviteter lige fra vikinge-julefest til nisseværksted. I weekenden er der julemarked på Stændertorvet, og Gimle inviterer børn og barnlige sjæle til julebanko. Der er også masser af julemusik at varme sjælen på. Hør fx Roskilde Domkirkes pigekor synge julen ind, kom i godt humør med Roskilde Gospel Choir eller fang Bjældebanden, når de går rundt i byens gader i weekenden og spiller julejazz. \nNext week in Roskilde is all about Christmas! During the weekend, there is a Christmas Market at Stændertovet, the city's main square, and Gimle invites children and young at heart to Christmas bingo. There are also lots of Christmas music to warm the soul. For instance, Roskilde Cathedral Girls' Choir is singing Christmas carols, Roskilde Gospel Choir will put a smile on your face and Bjældebanden are walking around the city's streets on Saturday playing Christmas jazz.",
"picture": "https://scontent-a.xx.fbcdn.net/hphotos-xpa1/v/t1.0-9/s130x130/1469977_341522786026965_4714709485711089921_n.jpg?oh=3834c5ef3c821663c01b211f5c453271&oe=5514D81A",
"created_time": "2014-12-05T09:18:09+0000",
"id": "163262523852993_341533356025908"
}
],
"paging": {
"previous": "https://graph.facebook.com/v2.2/163262523852993/feed?fields=message,picture,created_time&limit=3&since=1418150400",
"next": "https://graph.facebook.com/v2.2/163262523852993/feed?fields=message,picture,created_time&limit=3&until=1417771088"
}
}

```
private async void LoadFacebookData()
{
    try
    {
        FacebookClient accessToken = new FacebookClient("722191401190090|zV8YHfAogIsAsGHsE8TOZWIY_0g");
        dynamic infoData = await accessToken.GetTaskAsync("visitroskilde");
        InfoData.Add(new FacebookData { Navn = infoData["name"], Om = infoData["about"], Kilde = infoData["cover"]["source"], Telefon

        dynamic feedData = await accessToken.GetTaskAsync("visitroskilde/feed?limit=3&fields=message, picture, created_time");
        foreach (dynamic item in feedData["data"])
        {
            FeedData.Add(new FacebookData { Billede = item["picture"], Besked = item["message"], Dato = "Skrevet d. " + DateTime.Parse
        }
    }
    catch (Exception ex)
    {
        //throw new Exception(ex.Message);
        MessageDialog fbError = new MessageDialog("Kunne ikke connecte til Facebook API", "Ups! Der skete en fejl");
        fbError.ShowAsync();
    }
}
```

Bing Maps

Vi har valgt at gøre brug af Bing maps til at vise hvor en given restaurant/attraktion ligger. Samtidigt har vi gjort så man kan se sin egen position.

For at bruge Bing maps i en Windows Store app skal man have en token som bruges når man opretter mappet. Det ser således ud når vi opretter kortet:

<maps:Map x:Name="Map" ZoomLevel="13" Credentials="AhVIFAhaFsdkLTO5W_zBuQRA29ly0ZMwfR1mx_jqnW2QVYH4WQEmtWayB25cA7Aq" Margin="0,0,10,0" Grid.Row="1" Culture="da" Grid.ColumnSpan="2" Loaded="Map_Loaded" >

Vi har så valgt at vores kort er centreret på Roskilde da det nu er Roskilde og det attraktioner vores app handler om. Det ser så sådan her ud i koden

```
<maps:Map.Center>
    <maps:Location Latitude="55.641910" Longitude="12.087845" />
</maps:Map.Center>
```

Vores længde og breddegrad er sat til Roskilde.

For at der ligesom er en mening med vores kort har vi Binded vores længde og breddegrader til den valgte attraktion. Det ser således ud:

```
<maps:Pushpin>
    <maps:MapLayer.Position>
        <maps:Location Latitude="{Binding SelectedKategoriliste.Breddegrad}" Longitude="{Binding SelectedKategoriliste.Laengdegrad}"  />
    </maps:MapLayer.Position>
</maps:Pushpin>
```

Til allersidst har vi valgt at brugeren skal kunne se sin egen position og ud fra det vi har fundet ud af er det ret besværligt at databinde nogle af tingene i Bing maps. Der bliver tilføjet en rød pushpin til mappet som er placeret på ens egen placering. Derfor har vi lavet denne del i code-behind selvom det ikke er vel set.

```
private async void Map_Loaded(object sender, RoutedEventArgs e)
{
    try
    {
        Geoposition pos = await _geolocator.GetGeopositionAsync();
        Location location = new Location(pos.Coordinate.Latitude, pos.Coordinate.Longitude);
        Pushpin mylocationPushpin = new Pushpin();
        mylocationPushpin.Background = new SolidColorBrush(Colors.Red);
        MapLayer.SetPosition(mylocationPushpin, new Location(location));
        Map.Children.Add(mylocationPushpin);
    }
    catch (Exception)
    {
        MessageDialog mapError = new MessageDialog("Ups! Der skete en fejl!", "Kunne ikke finde din placering på kortet");
        mapError.ShowAsync();
    }

}
```