Indholdsfortegnelse

[Projekt opstart 4](#_Toc500747764)

[Sprint 1 5](#_Toc500747765)

[Sprintforløbet 5](#_Toc500747766)

[Sprint 1 Review 6](#_Toc500747767)

[Sprint 1 Retrospective 6](#_Toc500747768)

[Mål 6](#_Toc500747769)

[1 Problemstilling 7](#_Toc500747770)

[Perspektivering af problemstilling 7](#_Toc500747771)

[Problemformulering 7](#_Toc500747772)

[Krav til løsning 8](#_Toc500747773)

[SWOT 8](#_Toc500747774)

[Stærke sider: 8](#_Toc500747775)

[Svage sider: 9](#_Toc500747776)

[Trusler: 9](#_Toc500747777)

[Muligheder: 9](#_Toc500747778)

[SWOT modellen 9](#_Toc500747779)

[Metode 10](#_Toc500747780)

[Unified Process and SCRUM 11](#_Toc500747781)

[Unified Process 11](#_Toc500747782)

[Inception 11](#_Toc500747783)

[Eleboration, construction og Transition 11](#_Toc500747784)

[SCRUM 12](#_Toc500747785)

[Sprints 12](#_Toc500747786)

[Roller 12](#_Toc500747787)

[Scrumboard 12](#_Toc500747788)

[Møder 14](#_Toc500747789)

[Agile Manifesto 14](#_Toc500747790)

[Individer og samarbejde frem for processor og værktøjer. 15](#_Toc500747791)

[Velfungerende software frem for omfattende dokumentation 15](#_Toc500747792)

[Samarbejde med kunden frem for kontraktforhandling 15](#_Toc500747793)

[Håndtering af forandringer frem for fastholdelse af en plan 15](#_Toc500747794)

[Cynefin framework 16](#_Toc500747795)

[Domænemodel 16](#_Toc500747796)

[User stories 18](#_Toc500747797)

[Login 18](#_Toc500747798)

[Oprette 18](#_Toc500747799)

[Redigere 18](#_Toc500747800)

[Slette 19](#_Toc500747801)

[Sammenligne 19](#_Toc500747802)

[Projektets opstart 19](#_Toc500747803)

[Perspektivere 19](#_Toc500747804)

[System sekvens diagram 19](#_Toc500747805)

[SD for login 20](#_Toc500747806)

[ERD 22](#_Toc500747807)

[Budget 22](#_Toc500747808)

[Finansgrupper 23](#_Toc500747809)

[Finanskonti 23](#_Toc500747810)

[Periode 23](#_Toc500747811)

[Mapping 23](#_Toc500747812)

[Normaliseringsregler 24](#_Toc500747813)

[Første normalform 24](#_Toc500747814)

[Anden normalform 25](#_Toc500747815)

[Tredje normalform 25](#_Toc500747816)

[Client-Server arkitektur 26](#_Toc500747817)

[Fordele og Ulemper 27](#_Toc500747818)

[Fordele 27](#_Toc500747819)

[Ulemper 27](#_Toc500747820)

[Client-Server med Budgetmanager 28](#_Toc500747821)

[Microservice arkitektur 28](#_Toc500747822)

[Hvad er microservice? 28](#_Toc500747823)

[Hvem bruger microservice? 29](#_Toc500747824)

[Fordele ved at bruge microservice: 29](#_Toc500747825)

[Ulemper ved at bruge microservice: 30](#_Toc500747826)

[Microservice med Budgetmanager 30](#_Toc500747827)

[Prototyping 30](#_Toc500747828)

[Presentation prototype 30](#_Toc500747829)

[Prototype proper 30](#_Toc500747830)

[Breadboard prototype 31](#_Toc500747831)

[Pilot system 31](#_Toc500747832)

[Vores valg 31](#_Toc500747833)

[Sketching 31](#_Toc500747834)

[Hvordan har vi benyttet sketching? 31](#_Toc500747835)

[Design interfaces 32](#_Toc500747836)

[Jennifer Tidwell 32](#_Toc500747837)

[Knapper 32](#_Toc500747838)

[Escape hatch 33](#_Toc500747839)

[Tekst 33](#_Toc500747840)

[Gestalt og principper 33](#_Toc500747841)

[1. Similarity 33](#_Toc500747842)

[2. Continuation 33](#_Toc500747843)

[3. Closure 34](#_Toc500747844)

[4. Proximity/Grouping 34](#_Toc500747845)

[5. Figure/ground 34](#_Toc500747846)

[6. Symmetry & order 34](#_Toc500747847)

[Gestalt i budgetmanager 34](#_Toc500747848)

[Sprint 2 34](#_Toc500747849)

[Sprintforløbet 34](#_Toc500747850)

[Sprint 2 Review 35](#_Toc500747851)

[Sprint 2 Retrospective 35](#_Toc500747852)

[Sikkerhed 35](#_Toc500747853)

[Oauth2 35](#_Toc500747854)

[Bruger 35](#_Toc500747855)

[Klienten 36](#_Toc500747856)

[Ressource / Autentificering server 36](#_Toc500747857)

[Oauth flow 36](#_Toc500747858)

[Applikation registrering 37](#_Toc500747859)

[Vores valg 37](#_Toc500747860)

[Andre muligheder 38](#_Toc500747861)

[GitHub – versionskontrol 38](#_Toc500747862)

[Test 39](#_Toc500747863)

[Whitebox 39](#_Toc500747864)

[Blackbox 39](#_Toc500747865)

[Destructive 39](#_Toc500747866)

[Usability 39](#_Toc500747867)

[Database – valg af data 39](#_Toc500747868)

[Sprint 2 40](#_Toc500747869)

[Sprintforløbet 40](#_Toc500747870)

[Sprint 2 Review 40](#_Toc500747871)

[Sprint 2 Retrospective 40](#_Toc500747872)

[Sprint 3 41](#_Toc500747873)

[Sprintforløbet 41](#_Toc500747874)

[Sprint 3 Review 41](#_Toc500747875)

[Sprint 3 Retrospective 41](#_Toc500747876)

# Projekt opstart

Af: Anders

Vi startede vores projektperiode mandag den 27-11 klokken 09 hvor vores ’Product owner’ Klaus Nørregaard holdte oplæg om hvad vores produkt skulle indeholde samt hvilke retningslinjer vi skulle holde os indenfor. Der var mulighed for at stille spørgsmål til både Klaus Nørregaard og Christian Clausen.

Efter oplægget satte vores gruppe sig ned sammen og læste hele opgaveformuleringen igennem en ekstra gang. Vi satte os hver især ned og skrev hvad vi mente der var vigtigt at fokusere på, derefter diskutere vi hvad alle i gruppen havde skrevet ned.

Vi prøvede at finde et lokale, som vi kunne have for os selv med et stille miljø. Dette var desværre ikke muligt på dag 1. vi var alle enige om at vi ville møde op på studiet hver dag, da vi mente vi ville arbejde bedst der og have muligheden for ’ping pong’ mellem hinanden, uden at forstyrre andre eller blive forstyrret. Vi ville også have muligheden for at kunne kontakte vores ’product owner’ når vi har noget vi vil præsentere for ham.   
Vi lavede derefter, samlet hele gruppen, vores domæne model for at sikre at alle var enige i det endelige resultat af modellen.

Da vi havde et endeligt resultat begyndte vi at skrive user stories. Vi valgte at bruge ’planning poker’ til at estimere hvor lang tid der skulle bruges til hver user story. Efter det fik vi lavet task til vores user stories, hvor vi igen brugte ’planning poker’ til at estimere tiden på hver task og se om det gik op i hvor meget vi havde estimeret til vores user stories.

Vi sluttede første dag af med at finde nogle tasks vi var sikker på vi kunne blive færdig med og så sørgede vi for at få dem sat over i ’done’ på vores scrum board før vi måtte tage hjem.

Vi mødte alle op på dag 2. hvor vi havde fået et stille lokale for os selv, hvor der var projektor og tavle som vi kunne bruge til at lave skitser på samt hvis vi havde noget der skulle diskuteres på gruppen så ville det være optimale værktøjer at have hver dag. Vi startede dagen ud med vores ’standing scrum meeting’ hvor vi fik forklaret for hinanden hvad vi lavede den foregående dag og hvad planen var for den pågående dag. Da mødet var slut satte vi os igen i vores lokale hvor rapportskrivningen begyndte.

Vi sørger for at inddrage vores PO når vi mener der er noget vigtigt at diskutere med ham om produktet så vi undgår, at vi pludselig kunne blive stoppet i processen fordi der har været misforståelser fra nogens side om hvad der menes.

# Sprint 1

Af: Anders

## Sprintforløbet

Vi har i første sprint valgt at tage vores 2 userstories, ”Som udvikler vil jeg definere/dokumentere projektets opstart” og ”Som bruger vil jeg gerne kunne logge ind på budgetmanager med Oauth via Xena”, fra vores backlog. Vi havde estimeret 90 timer til første sprint. I dette sprint var der udelukkende fokus på at vi fik lavet alle vores modeller rigtigt for at undgå konflikter i fremtiden når vi skulle lave vores database og begynde at kode.  
Hvis der skulle være blevet underestimeret angående vores tid, så vil der i den resterende tid blive arbejdet på at få udviklet en god database, som skal være fundamentet for at programmet køre ordentligt og vi kan arbejde med de data vi ønsker.

## Sprint 1 Review

Vi havde valgt af taget udgangspunkt i 2 af vores userstories der primært bestod af rapport skrivning. Der blev lavet diverse diagrammer, modeller osv., nogle af dem er ”SD[[1]](#footnote-1), SSD[[2]](#footnote-2), ERD[[3]](#footnote-3), mapping” som alle er med til at danne fundamentet for vores program, samt hjælpe med at sørge for vi får alt tilføjet i vores tabeller i databasen i første forsøg, så vi vil slippe for at skulle slette databasen flere gange for at kunne oprette en ny. Vores database ligger på ”Azure[[4]](#footnote-4)” i skyen. Da vi havde underestimeret tiden i sprintet havde vi tid til at sørge for vores database blev lavet færdig allerede i første sprint.

## Sprint 1 Retrospective

Ved at vi har valgt at have alle vores arbejdsdage i første sprint har vi undgået hverdagens overspringshandlinger, som kunne forekomme hjemme. De risikoer forbliver mindsket ved at vi bliver ved med at mødes på studiets område.  
Der har været rigtig god mulighed for diskussion med begge vejledere/PO[[5]](#footnote-5), som sørger for vi ikke sidder for længe fast i et hul eller graver det dybere end nødvendigt.  
scrumboarded er godt til at sikre alle har noget at lave, og holder styr på hvad der mangler test osv. som gør at hvis man lige har behov for en break fra det man laver, så kan der testes eller læses korrektur på det de andre scrum team medlemmer har lavet.

# Mål

Af: Anders

Vores mål er at skabe en Budgetmanager, som skal være i stand til at synkronisere med Xena’s budget og på den måde hente de eksisterende finansgrupper og finanskontoer ud fra Xena budgettet.

Det skal derefter være muligt ud fra de synkronisere grupper/kontoer at inddatere sit budget som ønsket og derefter kunne sammenligne sit budgetmanager budget med det budget man som kunde har liggende på Xena.

Det skal være muligt for kunden selv at kunne skabe nye ”finanskontoer” så der vil være mulighed for at skabe de kontoer der er nødvendig for den enkelte bruger. Samt slette eller redigere diverse kontoer hvis nødvendigt.

Vi vil ved hjælp af ”SCRUM” administrere vores opgaver/task ud fra de sprints vi har defineret på gruppen.

Vi har valgt at der bruges Oauth via Xena for at man skal kunne logge på budgetmanager, det har vi valgt for at sikre sikkerhed i vores budgetmanager.

Der bruges udelukkende Github som versionskontrol for at sikre alt er opdateret, og al data altid vil være tilgængelig for alle gruppens medlemmer.

# Problemstilling

Xena.biz har ikke et dedikeret budgetværktøj til sine kunder. Kunder der ønsker at arbejde med budgetprocessen og budgetplanlægning, er nødsaget til at finde alternative værktøjer. Kontoplanen i Xena viser kun de tal, som man har indrapporteret i Xena i forbindelse med varekøb, forbrug, salg m.v. Kontoplanen er således et billede af historiske aktiviteter som virksomheden har udført. At Xena ikke har et budgetværktøj er utilstrækkeligt for de kunder der ønsker at handle på baggrund af en sammenstilling af budgetter og regnskab. En sammenstilling af budgetter og regnskaber vil kunne give brugerne af regnskabet et overblik over realiseret aktiviteter og de budgetterede – løbende og med en minimal forsinkelse.

## Perspektivering af problemstilling

Af: Nikolaj

Der uddrages fra oplægget at Xenas brugere ikke har et budgetværktøj, og at vi som udviklere skal udarbejde en tredjeparts app (Budgetmanager) som kan implementeres på Xena.

Det er blevet beskrevet at brugerne skal kunne få et realistisk overblik over deres budget, ved hjælp af en sammenligning af de reelle tal fra deres regnskaber på Xena.

# Problemformulering

Af: Nikolaj

* Hvordan sikre vi at vores brugers data er beskyttet?
* Hvilke udfordringer opstår der ved at hente data ud fra Xenas API?
* På hvilken måde vil vi sammenligne data fra Xenas budget og de oprettede budgetter i vores Budgetmanager?
* Der kan opstå nogle udfordringer hvis dataene i de oprettede budgetters finanskonti ikke stemmer overens med de data der er i Xenas, på hvilken måde vil man løse dette?
* Hvordan kan vi give mulighed for at oprette et budget på en intuitiv måde for brugerne?

# Krav til løsning

Af: Patrick

Hvilke krav stilles der til vores løsningsforslag? Hvad skal vores løsningsforslag kunne?

Kunden har opstillet nogle krav til hvad vores løsningsforslag skal indeholde. Kunden har nogle specifikke funktioner, som vores budgetmanager skal kunne for at opfylde kundens ønske til produktet. Kundens krav er:

* Oprette, redigere og slette budgetter
* Koble budgetter til regnskaber i Xena
* Overvåge vareforbrug for en periode
* Se hvordan aktuelle konti forholder sig til de budgetterede konti

Ud fra kundens krav til budgetmanageren, er det for kunden at kunne koble budgetter sammen med regnskaber på Xena. Dette giver mulighed for at kunne se uforudsete udgifter og hvilket vareforbrug man har haft i en given periode.

# SWOT

Af: Anders

## Stærke sider:

Vi skal ikke ud og publisher vores app på samme måde som andre udviklere, når vores app er godkendt af Xena vil den være tilgængelig inde på deres hjemmeside, hvor andre vil lave mange forskellige reklamer på diverse hjemmeside, hvor man som alm. bruger kommer forbi når man browser på nettet.

Budgetmanager er en simpel og optimeret app, som fremstår meget brugervenlig så enhver person der har sat sig en lille smugle ind i budgetter vil kunne bruge appen og let inddatere data.

Du vil som bruger af budgetmanager have mulighed for at sammenligne dit budget med det budget du har på Xena, det vil være med til at kunne give et bedre og større overblik over ens budgetter og se hvor der måske skal optimeres eller hvor der måske er mulighed for at skære lidt ned og på den måde spare penge.

## Svage sider:

Hvis vores budgetmanager ikke bliver reklameret andre steder end Xena, så er det kun folk med kendskab til Xena der vil have mulighed for at finde den og blive kunde.

I budgetmanager vil der i første version ikke være mulighed for at lave balancer for at kunne holde dine aktiver op mod passiver i din virksomhed.

## Trusler:

I dag er det meget let ved hjælp af værktøjer som f.eks. Excel at lave et simpelt budget fremfor at ville ud og investere i et program til det. Det ikke sikkert man som ny iværksætter på markedet tænker det noget man har lyst til at smide penge efter, da det er de færreste iværksættere der har stor egenkapital.

Et krav til at kunne bruge budgetmanager er, at man som bruger bare har lidt kendskab til budgetter, altså at man som min. ved hvad et resultat budget er.

## Muligheder:

Der er stor mulighed for at større virksomheder, der måske ikke har et optimeret system til deres budget/budgetter ville tage brug af en simpel optimeret app som budgetmanager, og på den måde måske kunne holde bedre overblik over deres indtægter kontra deres omkostninger.

En mulighed for fremtiden kunne være at få optimeret appen nok til at kunne lave aftaler med revisorer som så kunne tage det videre til deres kunder og forklare om det når de alligevel skal snakke om deres årsregnskab.

Nye kunder af Xena vil kunne købe appen budgetmanager og på den måde få en lettere start i kampen om at opbygge deres budget fra starten, fremfor hvis de stod på bar bund. Det vil være muligt via Xena at tage et kursus på 2 timer om deres system og budgetter, som kan være med til at fremme forståelsen af vores app på samme tid.[[6]](#footnote-6)

# SWOT modellen

Nedenunder er vist en model af SWOT som er med til at give et kort og simpelt overblik over budgetmanagers interne samt eksterne situation.

Interne situationer er handlinger vi selv kan påvirke, hvor eksterne er omverdenen.

SWOT giver et overblik over produktet på et givent tidspunkt, og er med til at evaluere og drage konklusioner, som kan bruges som grundlag for fremtidige handlinger.

|  |  |
| --- | --- |
| **Interne situation** | |
| **Stærke sider** | **Svage sider** |
| * Mindre publishering * Brugervenlig app * Godt overblik over budget  kontra det hos Xena | * Eventuelt mangel på reklame * Mangel på f.eks. balance |
| **Eksterne situation** | |
| **Muligheder** | **Trusler** |
| * Folk der mangler simpelt værktøj til deres budgetter * Xena’s kunder * Aftale med revisore | * Værktøjer som ”Excel” * For lidt viden om budgetter |

# Metode

Af: Lasse

Vi havde i gruppen en diskussion om hvilket framework der ville passe bedst, til den opgivet projektopgave. Vi har på vores 3. semester arbejdet meget med SCRUM, og det ville være et naturligt valg at bruge dette. Vi har udover SCRUM også kigget på et andet framework, extreme programming, dette fandt vi dog ikke attraktivt, da det store fokus her besidder på pair-programming, og det er et krav i opgaven at alle skal deltage aktivt, og der skal navn på hvad hver person har bidraget med. Ved XP ville der komme to navne på alt hvad vi har lavet, og det vil derfor ikke give et indblik i, hvad hver person har bidraget med til gruppen individuelt. Vi har derfor valgt at benytte SCRUM, da det giver os en masse brugbare værktøjer.

Udover SCRUM vil vi også benytte os af Unified Process metoden, til at hjælpe med at skabe en rød tråd igennem hele arbejdsprocessen. Vi vil også benytte UML notations diagrammer i form af fx Domæne model, SSD, SD, m.v.

## Unified Process and SCRUM

Vi vil som tidligere nævnt benytte SCRUM og Unified Process sammen, for at hjælpe med at danne overblik over hele projektet. Vi vil benytte de fire faser Inception, Eleboration, Construction og transition. Grunden til vi vil benytte både SCRUM og UP er, at vi i vores projekt, har brug for at få en bredere forståelse af domænet, samt hvilke krav der er til løsningen af opgaven, det kan business modeling og requriments faserne hjælpe os med. I SCRUM er det dog også meningen, at man planlægger i starten af projektet, vi vil blot benytte disse redskaber fra UP til og hjælpe os med at danne et samlet overblik, da SCRUM ikke har en decideret metode til dette.  
 Vi vil benytte os af udtrykket sprints og ikke iterationer. Disse sprints kan hjælpe os med at danne et tidsoverblik, samt give os nogle krav der skal overholdes, i form af fx en definition of done, der skal overholdes før en given tasks kan defineres som færdig lavet.

## Unified Process

### Inception

Vi vil starte ud med inception fasen for at få en bedre forståelse af Domænet, samt finde frem til alle de krav der er opsat i opgaven. Vi vil derfor starte med at fokusere på de User stories der har fokus på disse opgaver, og andre relevante opgaver der måtte være, som fx at få skrevet en problemformulering og få opsat nogle mål.

### Eleboration, construction og Transition

Da vi bruger sprints frem for iterationer, giver det god mening at disse tre er forbundet. Vores User stories bliver lavet på den måde, at analyse og design er forarbejdet på en user story, der skal laves inden implementation kan finde sted. Så i et sprint vil der både være analyse og design samt implementation før en user story, kan blive betragtet som udført. Derudover skal vores definition of done overholdes, som indeholder krav til testning. Deploy af software vil foregå løbende, som en user story bliver færdig, dette gør vi ikke, fordi vi har krav fra en kunde omkring agil udvikling, men for at hjælpe os med at få et overblik over hvor langt vi er kommet i projektet.

## SCRUM

En af de ting der er lagt op til i opgaven og af product owner er, at der er mulighed for at der kan komme ekstra krav til features løbende i projektet. Hvis det skulle være tilfældet, kan vi ligge disse features ind som user stories i vores backlog, og derefter finde frem til, hvilket sprint de skal med i, alt efter hvor stor en prioritet de har.

### Sprints

SCRUM er som tidligere nævnt delt op i sprints, vores projekt er på 3 uger, og vi har derfor delt det op i tre sprints af en uge af gangen. Vores sprints er tidfast og har en klar definition på hvad der skal være færdig ved udgangen af sprintet. I starten af vores sprints vil vi vælge de opgaver, som har højest prioritet for at få lavet projektet, i en rækkefølge der kan være med til at skabe en rød tråd igennem hele projektet.

### Roller

Vi har tre roller opfyldt i vores scrum team, vi har en Product owner som er Klaus Nørregaard, han har stillet opgaven med de krav der skal opfyldes, og det er ham vi kan gå til ved eventuelle spørgsmål til projektet. Vi har en Scrum master som er Lasse Meldgaard, han står for indkaldelse af møder, samt hjælpe med at udbedre eventuelle impediments der måtte være for Development teamet. Development teamet er den sidste rolle der skal udfyldes, og denne består af alle fire medlemmer i gruppen. Dette team står for at færdiggøre de enkelte sprints, indenfor de fastsatte rammer.

### Scrumboard

Vi bruger [www.trello.com](http://www.trello.com), til håndtering af vores Scrumboard, hvor vi har listet alle vores User stories op.  
En af de store fordele ved et scrumboard er at det hjælper os med at holde styr på vores user stories samt task, så vi ved hvad der skal laves i et gældende sprint, og hvem der laver hvad. Det sikrer også at vi hele tiden holder alle i gruppen i gang, da de kan påbegynde en ny opgave, ved blot at se hvilke tasks der er tilbage på boardet.

Vi vil desuden bruge planing poker til at tidsangive vores stories, det gør vi for at få et så præcist estimat som muligt, og så alle kan klare en opgave inden for den planlagte estimering. Vores scrumboard består af 6 kolonner, som er Backlog, To Do, Work in progress, Test af kode, dokumentation korrektur og en Done.

Backloggen indeholder alle de User stories vi har fundet frem til, de ligger i en prioriteret rækkefølge efter product owners ønske. I vores tilfælde skal vi aflevere et færdigt projekt, og ikke med løbende afleveringer efter hvert sprint, til en kunde. Vi har derfor valgt at prioritere dem efter hvordan rapporten skal opbygges, samt hvordan applikationen skal bygges op. Derudover vil vi tage fat i nogle af de store problemstillinger først, fx hvordan vi kan benytte Xenas api. Det er ikke alle vores stories, der tager udgangspunkt i brugeren, da vi har en rapport der skal laves og vi vil gerne have alle vores opgaver på scrumboarded. Vi vil derfor have nogle User stories der tager udgangspunkt i os som udvikler, der sikrer vi får dokumenteret alt i rapporten.  
To Do er vores sprint backlog, hvor vi har opdelt vores user stories i tasks, som vi igen har tidsangivet ved hjælp af planning poker. De tasks vi har i denne kolonne skal være færdige efter sprintet er ovre, hvis de ikke er blevet lavet, vil dette blive taget op i vores sprint retrospective møde.  
Work in Progress, er de tasks som er under udvikling af en fra teamet, vi sætter navne på tasks, når de bliver flyttet WIP, for at holde styr på hvem der er i gang med hvad.  
Test af kode, er hvor vi flytter en task hen når den er klar til at blive testet, vi har et krav om der som minimum skal være whitebox testing af alle tasks, før de kan flyttet over i Done. Det samme gør sig gældende på dokumentation, hvor der i kolonnen ”dokumentation korrektur”, hvor der skal læses korrektur, på den skrevne dokumentation. På den måde får vi spottet eventuelle fejl og mangler inden, vores arbejde bliver erklæret Done, som er den kolonne der det bliver flyttet i når alle krav er overholdt. De krav der skal overholdes har i defineret i vores Definition of done er:

1. Dokumentation er skrevet

2. Kode er testet (White box)

3. Dokumentation læst igennem

4. Alt er i versionskontrol

### Møder

Vi kommer i vores team til at holde 3 forskellige slags møder daily scrum meeting, sprint planning meeting og sprint retrospective. Vi holder hver morgen vores daily scrum meeting, det foregår udenfor arbejdsrummet, hvor computere kan være en distraktion. Til mødet er det scrum master og development teamet der deltager, der er sat 10 minutter af til mødet. Der vil hver morgen blive stillet tre spørgsmål til alle deltagere til mødet:

* Hvad lavede du i går?
* Hvad skal du lave i dag?
* Er der nogen hindringer i vejen for arbejdet?

På denne måde får alle et indblik i hvor langt man er kommet i det nuværende sprint. Skulle der være nogle hindringer for arbejdet, skal disse løses.  
Vi holder vores sprint planning meeting hver mandag morgen klokken 8.15. Her finder vi de User stories der skal i vores sprint backlog for det kommende sprint. Vi laver ikke et sprint goal, da vores product owner, ikke vil deltage på disse møder. Det sidste møde, sprint retrospective, holder vi hver fredag klokken 13.30, her vil vi igen fokusere på 3 spørgsmål, disse vil blive stille i åbent forum og ikke til hver enkelt deltager.

* Hvad skal vi begynde at gøre?
* Hvad skal vi stoppe med at gøre?
* Hvad skal vi fortsætte med at gøre?

Efter der er kommet ideer/svar til disse spørgsmål, vil vi i samarbejde finde til enighed om, hvilket af disse ting der skal ændres eller bibeholdes. På næste retrospektive vil, der blive kigget på om det er lykkes at ændre på de pågældende ting.

## Agile Manifesto

I og med SCRUM er et agilt framework følger det værdierne fra Agile Manifesto. Det består af 4 sætninger, med 2 påstande i hver. Disse sætninger er med til at definere agil softwareudvikling, og sikrer at man bruger agil softwareudvikling. Sådan som definitioner af sætninger er skrevet, er der lagt op til følgende udsagn: ”Der er værdi i punkterne til højre, men vi værdsætter punkterne til venstre mere”. Dette vil ikke nødvendigvis passe 100 procent til vores projekt, men dette vil vi komme nærmere ind på under hver værdi.   
De fire sætninger er følgende:

* **Individer og samarbejde** frem for processor og værktøjer.
* **Velfungerende software** frem for omfattende dokumentation.
* **Samarbejde med kunden** frem for kontraktforhandling.
* **Håndtering af forandringer** frem for fastholdelse af en plan

### **Individer og samarbejde** frem for processor og værktøjer.

SCRUM er et holdbaseret framework, hvor samarbejder er en nødvendighed, for at bruge det effektivt. Der er mange møder, der kræver åbenhed og ærlighed fra alle medlemmerne i grupper, og derfor er vi nødt til at finde et fælles ståsted, der kan fungere for alle i gruppen, for at opnå et godt samarbejde. Vi vil dog stadig benytte processor og værktøjer, til at hjælpe med vores planlægning og udførelse. Et værktøj vi bruger, som også er en del af SCRUM, er vores Scrumboard, det vil vi have et aktivt fokus på at benytte, da det er med til og sikre vi når alle vores opsatte krav.

### **Velfungerende software** frem for omfattende dokumentation

SCRUM handler om af få afleveret et velfungerende stykke software ved hvert sprint. I mange tilfælde vil der, derfor være stor fokus på at få lavet netop denne software færdig, da der er en kunde på den anden side, som forventer at det er klar til aflevering efter sprintet. I vores tilfælde er det ikke sagen, vores produkt, rapport og software, skal afleveres samlet. Det er også en systemudviklingseksamen, vi vil derfor have lige så stor fokus på dokumentationen såvel som en velfungerende software, da vi bliver vurderet på vores rapport.

### **Samarbejde med kunden** frem for kontraktforhandling

Vi har fået listet en række krav til vores projekt, hvilket fungere som vores kontrakt. Der er som tidligere nævnt mulighed for, at der vil komme nogle ekstra opgaver ind over projekt fra product owner. Skulle dette være tilfældet, er der nogle nye krav vi skal tage stilling med i samarbejde med kunden/product owneren, for at komme frem til det bedst mulige resultat, på trods af det ikke var nedskrevet i opgaven fra projektopstarten. Vi er derfor også nødsaget til at samarbejde med kunde frem for at kontraktforhandle.

### **Håndtering af forandringer** frem for fastholdelse af en plan

Dette udsagn passer godt med SCRUM, vi har mulighed for i hver sprint opstart at håndtere og prioritere diverse ændringer, der måtte opstå i vores projekt, i og med vi ikke har planlagt de næste tre uger, men kun en uge frem. Ud fra vores oprindelig tidsangivelse på vores user stories, har vi ekstra tid i projektperioden, hvis der skulle opstå nogle uforudsete problemer, der gør vi bruger længere tid på en task, end først angivet, eller hvis der måtte komme nogle ekstra krav fra product owner.

## Cynefin framework

Vi kan bruge cynefin til at hjælpe med at finde ud af oplysninger omkring vores domæne, samt hvordan dette skal anskues. Om vi har brug for hjælp fra eksperter, eller om opgaverne er nogle vi selv kan håndtere. Dette kan vi gøre efter vi har placeret vores domæne i en af de 5 segmenter i modellen, simple, complicated, complex, chaotic og disorder.

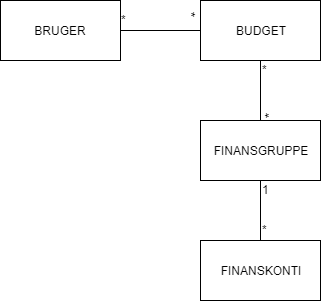
Efter gennemlæsning af det opstillede projekt, havde vi alle en god forståelse af domænet og de krav der er til opgaven. Vi har samlet i gruppen et fælles kendskab til budgetter og regnskaber, da det er noget vi tidligere har arbejdet, med under andre forhold. Derudover var der en god beskrivelse af regnskaber og budgetter i opgaveoplægget. Ud fra disse oplysninger befinder vi os i det simple segment, i det ordnede univers. Her er et ”cause- and-effect” forhold, hvilket passer godt med vores viden omkring domænet, vi kan se hvilke resultater vi får, ud fra de handlinger vi laver, inden vi foretager os dem. Dette gør også at vi basere de fleste af vores beslutninger på facts, og den viden vi allerede har omkring domænet, eller en viden der vil være os lettilgængelig.

Man kan argumentere for, at vi kunne bevæge os over i det kompliceret segment, hvis vi tog Xena med i vores vurderinger omkring domænet. Vi skal benytte Xenas API’er, og vi har ikke meget viden omkring dette. Her er vi afhængig af den dokumentation de levere omkring, hvilke data vi skal sende med, samt hvilke data, vi kan forvente at modtage. Vi skal her bruge ekstra tid på analyse af hvilke API vi skal bruge, derudover skal vi have oprettet nogle test data, på deres applikation, som vi kan bruge til at teste vores applikation. På dette område kan det også blive aktuelt, at vi møder udfordringer, der kræver at vi tager kontakt til Xena, for at kunne udbedre dem.

# Domænemodel

Af: Lasse

Efter gennemlæsningen af den udleverede projektopgave, var en af vores første opgaver at få en bedre forståelse af det domæne, som vi skal beskæftige os med. Vi har valgt at fokusere på vores eget domæne Budgetmanager, og ikke Xena, da vores løsning bliver en ekstern applikation udenfor Xena, der skal kunne snakke sammen med Xenas api, og hente data fra et firmas regnskab.  
Vi har dog brugt elementer fra Xenas domæne, som finansgrupper og finanskonti, da det er de konceptuelle klasser, vi skal sammenligne data med.

Vi havde en kort snak med vores product owner Klaus, omkring hvad hans forventninger til løsningen indebærer mht. til hvor specifik et budget skulle være i forhold til et regnskab. Vi kom frem til der skulle være mulighed for, at inddatere tal for både finansgrupperne og finanskonti i budgettet. Alternativet ville være kun at sammenligne tallene på finansgrupper, men på denne måde får vi en mere realistisk, komplet og dybdegående sammenligning med Xena regnskaberne.

Vores domæne model består af 4 konceptuelle klasser, Brugere, Budget, Finansgrupper og Finanskonti.

En bruger kan have flere budgetter, da der skal være mulighed for at oprette flere budgetter for fx flere virksomheder, eller budgetter for flere år.  
Et budget kan også have flere brugere, da man som bruger i Xena bliver tilføjet til en virksomhed, og da der kan være flere bruge på samme regnskab i Xena, skal dette også være en mulighed for vores løsning.  
Et budget kan have flere finansgrupper. Der vil som standard ved oprettelse af et projekt være tilkoblet de nødvendige finansgrupper der skal til, for at oprettet et fyldestgørende budget.  
En finansgruppe kan også eksistere på flere budgetter.  
En finansgruppe kan have mange finanskonti. Der skal være mulighed for at tilkoble de finanskonti, en virksomhed måtte finde nødvendige for at lave et budget der passer til deres behov.  
En finanskonto kan kun være tilkoble til en finansgruppe, da en postering som fx ”salg af cykler”, kun skal være posteret under omsætning, og ikke samtidig også kunne fremkomme under omkostninger.

De finanskonti der vil være mulighed for at oprette, vil blive hentet fra det respektive regnskab inde fra Xena. På denne måde får vi alle de konti der er relevante for et firmas regnskab med i budgettet. Dette resulterer i at regnskabet skal have oprettet finanskonti i Xena, før det er muligt at hente dem over i Budgetmanager.

# User stories

Af: Patrick

## Projektets opstart

Som udvikler vil jeg definere/dokumentere projektets opstart, så vi kan følge vores arbejdes proces i forløbet for hver uge. Vi vil her definere hvordan vi har tænkt os at starte projektet, sådan vi efterfølgende får det bedst mulige grundlag for resten af projektet. Vi vil også dokumentere hvordan vi igennem forløbet, har arbejdet med de forskellige dele af projektet.

## Login

Som bruger vil jeg gerne kunne logge ind på budgetmanager med Oauth via Xena, så jeg som bruger er sikker på, at det kun er personer med gyldigt login til Xena som kan bruge budgetmanager. Men selvom andre brugere har et gyldigt login til Xena, betyder det ikke at de har mulighed for at kunne benytte en anden brugers budget. Brugeren som har oprettet budgettet skal stadig give tilladelse til hvem der har adgang til det pågældende budget gennem budgetmanageren. Hvorfor have login? Dette giver brugeren en sikkerhed, fordi personer der ikke har adgang heller ikke kommer til at kunne se budgettet i budgetmanageren.

## Oprette

Som bruger vil jeg gerne kunne oprette et nyt budget i budgetmanageren. Dette vil give brugeren mulighed for at kunne inddatere, hvordan brugeren forventer sin indkomst og udgifter vil fordele sig for en given periode. Dette giver brugeren mulighed for nemt at få et overblik over, hvor mange penge man har tilbage efter alle faste udgifter er betalt. Derfor kan brugeren nemt få et overblik over hvordan en forventet periode vil se ud. Brugeren får ud fra det oprettede budget, et værktøj til hvordan det forventes at udgifter og indkomster passer sammen.

## Redigere

Som bruger vil jeg gerne kunne redigere et budget i budgetmanager, fordi der kan forekomme uforudsete udgifter som har en stor betydning for andre punkter i budgettet. Hvorfor bygge dette? Dette vil give brugeren mulighed for at kunne tilføje eller ændre et budget for en given periode, hvis der skulle være sket en tastefejl i navnet eller beløbet. Men brugeren har også mulighed for at kunne tilføje en uforudset udgift, som der kan have en større indflydelse på resten af budgettet. På den måde kan han nå at tilpasse nogle andre udgifter til, sådan at der for en periode ikke går i minus i forhold til indtægter og udgifter.

## Slette

Som bruger vil jeg gerne kunne slette et budget i budgetmanager. Som bruger er det en god mulighed at kunne slette et helt budget ad gangen hvis det skulle være nødvendigt.

## Inddatere

Som bruger vil jeg kunne se inddaterede data for et specifikt budget, sådan at brugeren vil kunne inddatere i et oprette budget. Dette giver brugeren muligheden for at kunne inddatere indtægter og udgifter for et specifikt budget, sådan at brugen kan inddatere oplysningerne for eks. Januar.

## Sammenligne

Som bruger vil jeg gerne kunne sammenligne mit budget op imod Xena. På Xena vil et resultatbudgettet blive præsenteret for en given periode. Som bruger giver dette mulighed for løbende at kunne inddatere i Xena, omkring hvilket salg og omkostninger man som virksomhed eller bruger har foretaget sig. Her vil vores bruger kunne sammenligne sit budget med resultatbudgettet fra Xena. Her vil brugeren kunne få et overblik over om det forventede budget er blevet overholdt eller om udgifterne er højere end indtægterne.

## Design af budgetmanager

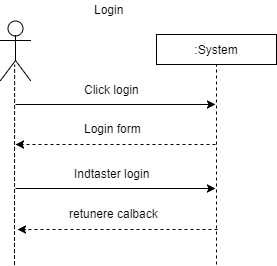
Som udvikler vil jeg gerne kunne dokumentere designet af budgetmanager. En god dokumentation sikre at brugeren nemt og hurtigt kan få besvaret eventuelle spørgsmål til budgetmanageren ved at læse i dokumentationen. Men for andre udviklere kan også ved at læse dokumentationen hurtigere sætte sig ind i eventuel koden bagved designet. Ved at have en god dokumentation sikre vi os at udvikleren af produktet ikke skal kunne kontaktes for små spørgsmål eksempel omkring, hvorfor knapperne er placeret hvor de er.

## Perspektivere

Som udvikler vil jeg konkludere/perspektivere på projektet, som det sidste inden projektet afsluttes. Her tager man som udvikler udgangspunkt i hvordan projektet har forløbet og giver en opsummering af hvad man har undersøgt i forbindelse med projektet. Hertil kommer man også ind på hvordan opgaven er løst i forhold til hvilken teori og metoden man har brugt gennem udviklingen. Dette giver udviklerne for et projekt mulighed for at opsummere perioden, metoder og den teori man har undersøgt og benyttet.

## System sekvens diagram

Af: Patrick

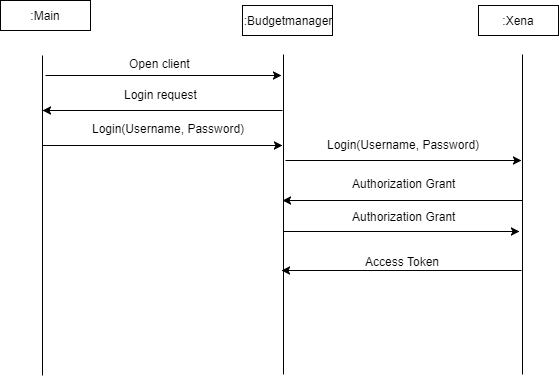


Vi har valgt at lave en SSD[[7]](#footnote-7) for loginprocessen. Dette diagram skal være med til at give et overblik over, hvilke handlinger der sker mellem systemet og brugeren. Dette sikre at man ikke overser nogen af de handlinger der sker mellem brugeren og systemet.

Vores SSD for ”login” viser hvordan processen ser ud, for vores brugers handlinger og hvordan systemet håndtere brugerens handlinger. I dette tilfældefor vores SSD ønsker brugeren at logge ind på vores budgetmanager, dette gøres ved at systemet returnere 2 tekstbokse hvor man skal indtaste brugernavn og adgangskode. Når brugeren har indtastet oplysningerne og trykker login, bliver brugeren af systemet dirigeret ind på en ny side, hvor brugeren nu kan benytte funktionaliteten af budgetmanageren.

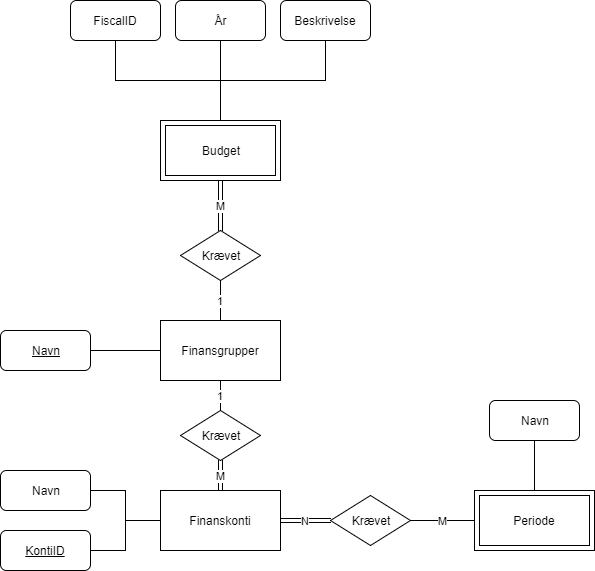
# SD for login

Vores SD1 viser interaktionen mellem de forskellige klasser. Dette giver et visuelt billede, af hvilken rækkefølge interaktionerne sker i. Interaktionerne bliver arrangeret i den rækkefølge som de bliver sendt i.

****

Ud fra vores SD[[8]](#footnote-8) kan vi se at når vores budgetmanager bliver åbnet, bliver brugeren bedt om at logge ind, for at kunne benytte vores budgetmanager. Processen for login bliver brugeren verificeret ved brug af Oauth2, hvor brugeren giver adgang til brugerinformationer fra brugerens Xena konto. På den måde er man helt sikker på at personen der vil logge ind, er ham han udgiver sig for. Når brugeren har indtastet sine loginoplysninger til Xena bliver de sendt til Xena, for at blive valideret for om man har en konto. Når brugeren er bliver godkendt bliver en ’authorization grant’ sendt tilbage til budgetmanageren, som sender den tilbage til Xena. Dette returnere en ’Access token’ som så kan bruges fremad rettet i samme session, sådan at man ikke skal logge ind hver gang.

# ERD



Efter vi havde analyseret domænet har vi udarbejdet et ERD. På baggrund af de informationer vi har fået fra projektoplægget og PO, har vi fundet frem til dette resultat.

## Budget

Vores Budgetentitet har 3 attributter: FiscalID, År og Beskrivelse.  
FiscalID’et er et ID som er givet af Xena når en virksomhed oprettes i deres system.   
År er til at definere hvilket år budgettet er for.  
Beskrivelse er til eventuelle kommentar omkring budgettet, da der kan være flere brugere der arbejder med det samme budget.

Budget er en svag entitet da der ikke er en åbenlys primær nøgle.

Budget har en relation til Finansgrupper som er krævet, da der ikke kan eksistere et budget uden finansgruppe.  
Denne relation er en, én til mange relation. Da et budget kun kan have én finansgruppe, men én finans gruppe kan have mange budgetter.

## Finansgrupper

Finansgruppeentiteten har kun én attribut og dette er Navn. Navn er på samme tid primærnøglen for denne entitet, da vi har besluttet at gruppe navnet er unikt etc. Der kan ikke være 2 finansgrupper der hedder Omsætning.

Finansgrupper har en relation til Finanskonti, denne relation er en, én til mange relation, da én finansgruppe kan have mange finanskonti og én finanskonti kan kun have én finansgruppe.

## Finanskonti

Finanskontientiteten har 2 attributter: Navn og KontiID.

Navn er navnet for finanskontien som bliver fremvist for brugeren på klient siden.  
KontiID er et ID som er givet fra Xena ved oprettelse af Finanskontiere, da denne er unik vil den være en perfekt primær nøgle.

Finanskontientiteten har en mange til mange relation med Periodeentiteten, det er i denne relation vi vil oprette en relations tabel, hvori vi kan inddatere vores estimater. Det er en mange til mange relation, da der fx skal kunne oprettes 12 finanskonti der hedder det samme, men er koblet på hver deres måned.

## Periode

Periodeentiteten har 1 attribut, som er Navn. Det er her vi kan gemme informationer omkring fx interval, halvår, hele år, måneder og dage.   
Dette er en svag entitet da der ikke er nogen åbenlys primærnøgle

# Mapping

Af: Lasse

Vi har været fælles i vores gruppe omkring, at lave mapping til vores database. Der er mange faktorer vi skulle tage højde for, og målet var at ramme det rigtige resultat første gang, så vi ikke senere skal ind og foretage ændringer i databasen. Grunden til vi vælger at bruge mapping er, at den giver os et overblik over designet, alternativet var at lave vores queries direkte. Med mapping får vi overblik over alle vores tabeller, primarykey og foreignkeys, og hvordan vores queries skal oprettes. Vi kan fx ikke oprette Finanskonti tabellen, før vi har oprettet Finansgruppe tabellen, da der er en foreign key i Finanskonti som peger på Finansgrupper.

I vores mapping har vi 5 tabeller, hvoraf 1 er en samlingstabel mellem finanskonti og perioder.   
Vi har valgt at bruge en surrogatnøgle i Budget tabellen, da vi som tidligere nævnt, ikke har nogle entiteter vi kan bruge som primary key.   
Finanskonti har en foreign key fra Navn på finansgrupper, dette er nødvendigt for at få koblet de to tabeller sammen, så vi har mulighed for at se hvilken gruppe en konto hører under. Det samme gør sig gældende for id fra Budget, hvor vi også har en foreign key. Derudover har vi en composite key, der er sat sammen af KontoId og BudgetId. Vi har valgt dette, da KontoId i sig selv ikke er unik, da det er et tal vi henter ud fra Xena, og flere forskellige regnskaber kan indeholde det samme KontoId. På denne måde får vi skabt en unik nøgle der samtidig er koblet på et budget.  
Vores Perioder tabel indeholder som vores budget tabel også en surrogatnøgle, da vi ikke har en værdi vi ellers kan bruge som primary key.  
Den sidste tabel er vores FinansKontiPeriode, som er en samlingstabel mellem Finanskonti og Perioder. Denne tabel bruger vi til at opbevare vores estimater til budgetterne. Det estimat bliver så koblet sammen med en periode og en finanskonto via 2 foreign keys. Derudover er der en foreign key fra Finanskonti på BudgetId, dette er påkrævet da vi har en composite key i vores finanskonti tabel, og derfor skal vi have begge keys med i samlingstabellen. Vi har været nødt til at lave en samlingstabel, da det skal være muligt, at give estimater på en Finanskonti på flere forskellige perioder, dette kun være for måneder, halv årligt, hel årligt osv.

## Normaliseringsregler

Da vi lavede vores mapping, har vi samtidig haft fokus på de tre første normaliseringsregler, dette har vi gjort for at gøre databasen mere fleksibel og fjerne overflødige data. Det kunne fx være at have den samme data gemt i forskellige kolonner, ved at følge reglerne har vi kun data gemt et sted, og skal ikke rette data i flere forskellige tabeller.

### Første normalform

* Fjern gentagende grupper i individuelle tabeller.
* Opret en særskilt tabel for hver sæt af relateret data.
* Identificer hver sæt af relateret data med en primærnøgle

Vi har opfyldt de tre punkter i den første normalform, vi mener at den data der er relevant for en tabel, er placeret i den pågældende tabel. Vi har ingen gentagende grupper i individuelle tabeller, alt vores data skal kun rettes et sted. Vi har også en primær nøgle til hver sæt af relateret data i de tabeller, hvor det er relevant.

### Anden normalform

* Opret særskilte tabeller til værdisæt, som kan anvendes til flere poster.
* Relater disse med en fremmednøgle.

Vi har oprettet vores mapping efter, at der skal være særskilte værdisæt i alle tabeller, som kan anvendes til flere poster. Vi har igen ingen data der er duplikeret og som skal ændres i flere tabeller ved hver rettelse.

### Tredje normalform

* Fjern de felter der ikke afhænger af nøglen

Vi har også fjernet de felter, som ikke er nødvendige for definition af en tabel, et eksempel på dette er Finansgruppe navnet hvor vi har oprettet en særskilt tabel, da det skal være muligt at trække en komplet liste over alle finansgrupper, selvom de ikke nødvendigvis er blevet brugt af en finanskonto. Man kunne også argumentere for, at år kolonnen i Budget tabellen, kunne flyttes ud i sin egen tabel. I vores tilfælde mener vi dog, at år er relevant for at definere et budget, og på den baggrund hører til Budget tabellen.

Vi har valgt kun at holde fokus på de første tre normaliseringsregler, da det er i vores design er nok til at have en database, hvor funktionaliteten stadig er intakt.

# Client-Server arkitektur

Client-Server er en arkitektur af et computernetværk, hvori mange klienter laver request til en central server(host).   
Klient siden indeholder et interface / gui, også kaldet front-end, hvorpå der kan laves request til serveren, og et interface til at fremvise den returnerede data.  
Serveren venter på requests fra klienten og svare på dem. Klienten har nødvendigvis ingen kendskab til hvordan serveren fungere, dette kaldes også back-end.  
Klienter er for det meste en arbejdsstation eller en personlig computer, hvorimod en server er en kraftigere computer der kan håndtere en masse trafik.  
Client-Server ariktekturen er effektiv i situationer hvor klienten og serveren har forskellige arbejdes fordelinger, fx På en hospitals computer, kan klient computeren køre en applikation hvori en bruger kan indtaste patient informationer, og server køre et andet program der står for at inddatere disse informationer i en database[[9]](#footnote-9).

## Fordele og Ulemper[[10]](#footnote-10)

### Fordele

Med Client-Server arkitekturen er der mange fordele som både brugerne og virksomheder kan udnytte.

* **Data deling**

Det er muligt for flere klienter at til gå den data der ligger på serveren på samme tid. Der kan tilføjes og læses data på samme tid.

* **Tilgængelighed**

Det er muligt for alle klienter at tilgå den data de ønsker, uden at tilslutte sig en specifik server igennem en terminal eller programmer som fx Teamviewer.  
Klienten kan også tilgå data fra alle steder, så længe de er tilsluttet internettet, hvilket giver en fordel for firmaer der har rejsende medarbejder, eller hjemme arbejde.

* **Forskellige systemer**

Om du er på en mac, pc eller andet. Er det fri mulighed for at tilgå serveren. Du skal blot have internetadgang, så kan man tilgå serveren. Dette gør at man ikke skal tage forbehold for hvilke systemer der skal bruge serveren.

* **Vedligeholdelse**

Med denne arkitektur kan man nemt opgradere, reparere, eller udskifte en server uden at det påvirker klient siden. Det er dog nødvendigt at have flere servere for at systemet skal fungere hvis man deaktivere én eller flere servere.

* **Sikkerhed**

Serveren har bedre kontrol over dataene, og at det er autoriserede bruger der kun har adgang til dataene.

### Ulemper

Selvom der er mange fordele ved Client-Server arkitekturen, har den også sine ulemper.

* **Server overload**

Hvis der er meget trafik på netværket, kan serveren blive overloaded. Hvilket betyder at den ikke kan følge med til alle de request der bliver kaldt fra klient siden, og derved får klienten en dårligere/langsommere oplevelse. Som når Skat, udgiver forskudsopgørelsen og hele den danske befolkning på samme tid vil tilgå deres data.

* **Downtime**

Hvis en vigtig server brænder sammen eller lign. Vil klientens request ikke blive behandlet og systemet går offline. (Dette kan løses ved at have backup servere)

## Client-Server med Budgetmanager

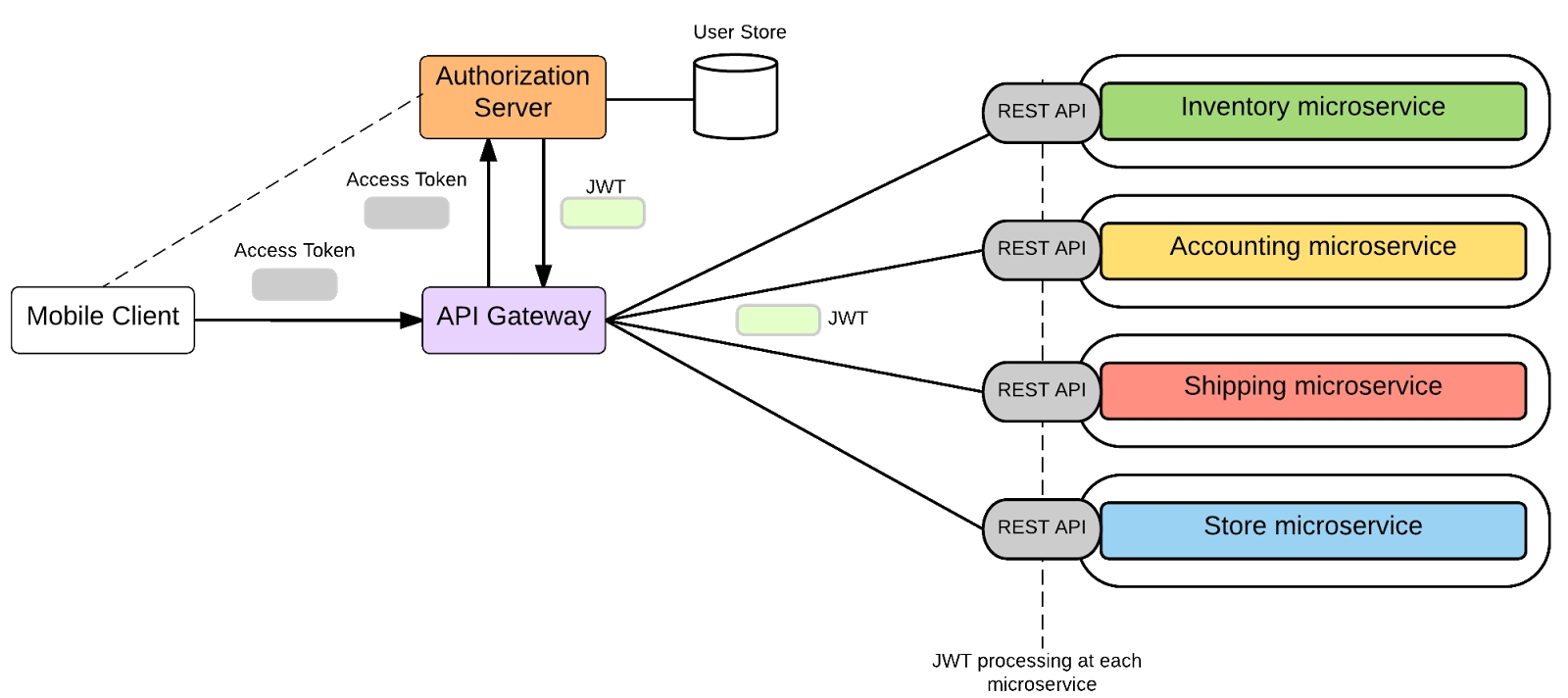
I vores system vil det give god mening at bruger client-server arkitekturen, da klienter skal kunne inddatere deres budgetter og hente informationer ud på samme tid.  
Derved har vi en front-end, klient side, og et back-end, server side. Som kan håndtere de nødvendige opgaver, der skal til for at systemet fungere bedst muligt.  
Ved at vi laver vores Budgetmanger i MVC, har vi udfyldt client-server arkitekturen, ved at brugeren ser deres ønskede views, og laver nogle requests der bliver behandlet af vores controllers. Dataene vil blive inddateret i en SQL database ved hjælp af nogle kald fra vores controller, her ved har vi en client-server arkitektur.

# Microservice arkitektur

Af: Anders

## Hvad er microservice?

Microservice er en software arkitektur som består af en eller flere små applikationer som for det meste vil være implementeret i skyen som eksempelvis en virtuel maskine. Ved at de køre på en virtuel maskine giver det bedre mulighed for at administrere hvor meget CPU samt hukommelse der bruges af servicen. Microservice arkitektur bruges til at lave enkelte applikationer som passer til andre servicer. Hver microservice vil normalt have deres egen database frem for en stor samlet.  
For at der kan kommunikeres mellem de andre services bruges der API’er.



## Hvem bruger microservice?

Hvis du har en stor applikation der har behov for at være hurtig, kan det være en fordel at bruge microservice, da der lettere kan arbejdes på flere områder indenfor applikationen, samt mindre risiko for at applikationen crasher da den er delt op i mange små uafhængige services i stedet for en stor. Det giver bedre mulighed for test og opdatering uden behov for at lukke resten af applikationen ned.

Fx Netflix benytter sig af microservices for at sikre at de holder hastigheden i top. I stedet for der kun er en service der skal håndtere de flere millioner request fra flere forskellige enheder hver dag. Sørger microservices for at dele arbejdet op, så de forskellige microservices specialisere sig inden for et bestemt område som så til sidst for sendt filmen frem til brugeren.

## Fordele ved at bruge microservice:

Ved brugen af microservice vil du som udvikler arbejde i mindre fokuseret grupper da der kun skal fokuseres på en service. Det kan også have effekt på koden da der er betydeligt mindre end hvis du stod med en almindelig service, der er mindre kode at sætte sig ind i når det skal implementeres.

Hvis en service går ned betyder det at det kun er den ene service der ikke virker mere, da en service er uafhængig af alle andre services.

Det giver mulighed for at opdatere en service uden at man er nød til at ligge hele applikationen ned, og hvis der under opdateringen skulle ske nogle bugs eller andre mindre fejl, vil det betyde at det stadig kun er den ene del der ikke er aktiv og lettere at fokusere på at få det fikset.

## Ulemper ved at bruge microservice:

Det skal sikres at den microservice du laver ikke kan være skyld i crash fordi din microservice er afhængig af andre services. Hvis f.eks. en anden microservice er blevet opdateret kunne det ske at din service ikke er kompatibelt mere, så det kræver et godt design for at undgå crash.

Du kan risikere at hver microservice er skrevet i et nyt sprog eller med nyt framework, hvilket kan være med til at gøre det svært at opretholde servicen. Derfor kan det være en fordel at der er sat nogle få regler op som man skal holde sig indenfor, det vil dog være i strid mod at hvert udviklingsteam selv bestemmer deres arbejdsmetode.

## Microservice med Budgetmanager

Det vil ikke give meget mening at bruger microservices i vores budgetmanager da der ikke er behov for splitte arbejdet ud over flere services og processere. Det vil sagtens kunne laves som microservice, men vi har valgt at Client-Server arkitekturen passer bedre med budgetmanager. Der er heller ikke behov for flere forskellige databaser, som vi ville få med microservices hvor vi med Client-Server også bare vil have en database.

# Prototyping[[11]](#footnote-11)

Af: Nikolaj

Prototyping bruges til at repræsentere løsningen af et problem. I vores tilfælde brugte vi det på vores Oauth.

Der er fire typer af prototyping: presentation prototypes, prototype proper, breadboard prototypes and pilot system.

## Presentation prototype

Denne prototype er en, udviklere udleverer til en evt. fremtidige kunde, der skal overbevises om at produktet er værd at investere i.

## Prototype proper

Denne prototype bliver udarbejdet og testet, for at forstå brugernes krav. Imens det reelle produkt stadig er under udvikling.

## Breadboard prototype

Breadboard bruges af udvikler til at teste tekniske opgaver krævet af projektet.

## Pilot system

Pilot prototype er en prototype der indeholder alle de nødvendige aspekter af et produkt, og er tæt på at være det færdige produkt. Man kan kalde denne type prototype for Beta, hvis man skal sammenligne med testning.

## Vores valg

Vi valgte at bruge Breadboard prototyping, da det passer godt ind i vores projekt som udviklere, og vi havde nogle spørgsmål omkring vores Oauth implantation.  
Vi lavede en test metode til at hente data ud fra Xena igennem vores Oauth applikation. Dette gjorde vi for at sikre os at vi havde hul igennem. Dette ville spare os en masse tid i sidste ende, da vi ikke skulle bekymre os om at evt. problemer med data ville skyldes vores Oauth.  
Protypen er efterfølgende blevet implementeret i der færdige produkt, da den levede op til de krav vi som udviklere havde og som var stillet af projektoplægget.

## Sketching

Af: Patrick

Sketching er en måde at finde designet til forskellige interfaces i et program. Denne proces skal hvert gruppemedlem tegne hvordan han forestiller sig designets interface. Ved brug af denne metode, giver det som udviklingsteam god mulighed for, at komme omkring alle de forskellige punkter et interface kan designes ud fra. Dette forekommer eks ved at en person fra gruppen, har ideer omkring hvordan et punkt i designet skal se ud, som resten af gruppen ikke har tænkt på. Så derfor ved at lave skitser af designet, giver det mulighed for hele gruppen at være enige og finde frem til punkter der ikke vil komme til at fungerer i designet.

### Hvordan har vi benyttet sketching?

Af: Patrick

Vi har benyttet sketching til at finde frem til et design af vores interfaces. Her fandt vores gruppe et stille rum hvor vi kunne sidde uforstyrret. Vi fik hver især et papir og et skriveredskab. Vi startede så en timer på 3 min, på denne tid skulle vi så sidde og tegne vores ideer til hvordan et design kunne se ud. Efter tidtagningen satte vi os ned og forklarede hvorfor vi havde tegnet designet som vi havde gjort. Efter vi hver især havde gennemgået vores tegning fandt vi punkter på de forskellige tegninger som vi ikke synes skulle være med i det endelige design. Efter vi var kommet frem til nogle punkter vi skulle have med i designet, indstillede vi timeren igen og begyndte at tegne nye skitser af designet. Denne proces gentog vi lige så mange gange som var nødvendigt til vi kom frem til et design vi synes så ordentlig ud og som ville give brugeren et godt overblik over hvad der skulle udfyldes.

# Design interfaces

Af: Anders

## Jennifer Tidwell

I vores valg af design har vi taget faktorer, som Jennifer Tidwell beskriver i sin bog ”Designinig interfaces”, i brug til at skabe interfacet for vores ”Budgetmanager”.   
Hun fortæller at folk har tildens til at være hurtige og gerne vil frem i en fart, så folk laver en hurtig ”scan” af siden de er på og tager derefter beslutningen om hvilken handling de vil foretage, selvom det måske er den forkerte handling.

### Knapper

Derfor har vi valgt at lave få men simple knapper deriblandt har vi ved hver afslutning en ”prominent done button” som er en knap der uden beskrivelse forklare brugeren at det er her du færdiggøre dit arbejde. Det vises tit med at den er større end andre knapper og nær højre hjørne bund.  
Inde på vores side hvor man skal inddatere data har vi valgt at placere 1 knap ”Prominent done button” til at gemme, den er placeret i bunden. Dette vil sørge for man husker at trykke på gem når alle ens ønskede data er inddateret og derved undgå fejl i opretning af sit budget. Vi har valgt som en ekstra feature at når du klikker væk fra tekstboksen, så er der et ”onclick event” der gør at når man trykker ud af boksen gemmes de data man lige har indtastet, og for at ramme alle brugernes behov beholdte vi gem knappen, for at sikre trygheden om handlingen for brugeren Det er muligt at skifte mellem de forskellige måneder ved hjælp af en dropdown boks i toppen af siden. Ved at den er placeret der vil det være første handling der bliver foretaget, og ved klik på en specifik måned vil måneden skifte i toppen af inddateringsformularen.

### Escape hatch

I menu baren er der placeret en såkaldt ”Escape hatch” knap. En sådan knap bruger vi for at hjælpe brugeren med let at navigere tilbage til start. Ved at vi bruger denne knap gør at brugeren kan føle en vis tryghed i at hoppe rundt og tjekke forskellige features, samt have muligheden for at kunne starte forfra med et klik. Knappen hjælper samtidigt med at promovere navnet på vores applikation ved at det er en statisk knap som forbliver i menu baren, lige meget hvilken side man er på. Menu baren er statisk og vil derfor forblive det samme, lige meget hvilken siden du befinder dig på. Det hjælper på at brugeren ikke bliver forvirret og altid have muligheden for let at gå tilbage og starte forfra. Derudover vil menu baren været markeret med en anden baggrund end resten af siden, på den måde undgår vi at den ikke går i et med hele siden, det hjælper med at gøre det simpelt og brugervenligt.

### Tekst

Alle finansgrupper er markeret med **fed** skrift for at indikere at en ny gruppe starter, samt baggrunden skifter for hver anden konto for at hjælpe med at skille dem fra hinanden i forhold til hvis hele siden bare var hvid.

## Gestalt og principper

Gestalt er en design metode der har 6 principper. Disse principper beskriver hvordan man vil se det fulde billede frem for de individuelle dele når man kigger på en gruppe af objekter.  
Ved at vælge et princip hjælper det dit design til at blive mere sammenhængende og komplet.

### Similarity

Her holdes designet simpelt. F.eks. hvis dit logo for virksomheden er lavet ud af trekanter, så vil du sørge for hele logoet er lavet af identiske trekanter, dog må de godt være forskellige størrelser så længe de ellers er ens.

### Continuation

Dette illustreres oftest ved at der er en streg/linje eller lign der krydser et objekt, kan f.eks. være en virksomheds navn hvor der går en curve fra første bogstav ned gennem navnet og slutte i toppen af sidste bogstav. Så det hjælper med at fange øjets opmærksomhed så man som bruger ikke vil kunne undgå at se navnet.

### Closure

Dette princip går ud på at man snyder øjet til at se et bestemt objekt, som er lavet ud fra figurerne der danner et logo evt. WWF. Her ses en panda ud fra en masse forskellige sammensatte figurere.

### Proximity/Grouping

Der dannes et element ud fra en masse ens figurere som kan have forskellige farver, størrelser eller lign.

### Figure/ground

Ud fra dette princip kan man ofte se 2 billeder i et billede. Disse laves oftest i sort/hvid for at skabe en kontrast der gør øjet opfanger en figur der er hvid og en der er sort. På billedet ses både en lysestage (hvid) og 2 ansigter (sort).

### Symmetry & order

Her gælder det om at få designet noget systematisk og simpelt som gør at brugeren ikke ender med at bruge tid på at prøve at finde en sammenhæng.

## Gestalt i budgetmanager

I budgetmanager har vi taget nogle af Gestalts principper i brug bl.a. har vi taget ”Proximity/Grouping” i brug på vores oversigt side. I budgettet er der lavet bokse til at vise dit reelle budget op mod dit eget budget samt differencen. Disse bokse er lavet som 1 stor boks hvor der er 3 firkanter indeni, de 2 øverste viser posteringer og den nederste viser differencen i procent.   
På siderne oversigt, inddatere og redigere har vi taget princippet ”Symmetry & order” i brug, da vi har sørget for at alle vores tabeller er sorteret i rigtig rækkefølge i vores sql script. Så man som bruger ikke vil komme til at skulle bruge tid på at hoppe frem og tilbage når tal indtastes, da budgetter starter med at vise omsætningen hvor man derefter postere sine udgifter i rigtig rækkefølge.

# Sikkerhed

Af: Anders

# Oauth2

Af: Nikolaj

Oauth er et verificeringsframework, der giver en applikation begrænset adgang til bruger data fra en HTTP service, som fx Facebook, Github osv.   
Frameworket virker ved at overføre bruger verifikationen, til den service hvorpå bruger informationerne ligger. Herefter skal brugeren give applikationen tilladelse til at tilgå deres information.  
  
Der er 4 roller i Oauth:

1. Bruger
2. Klienten
3. Autentificering server
4. Ressource server

## Bruger

Brugeren også kaldet ressource ejer, er den bruger som giver en applikation tilladelse til at tilgå deres information.

## Klienten

Klienten er den applikation der gerne vil tilgå brugerens informationer. Før den kan dette skal den have tilladelse fra brugeren, og valideres af HTTP servicen.

## Ressource / Autentificering server

Ressourceserveren er der hvor brugerens information er lageret, og Autentificering serveren verificere identiteten af brugeren og returnere en access token til klienten.

# Oauth flow



1. Applikationen laver et login request til brugeren
2. Hvis brugeren giver sine logininformationer til applikationen sender han en godkendelse.
3. Applikationen requester en access token fra autentificering serveren, ved at sende sin egen identitet med og godkendelsen fra brugeren.
4. Hvis applikationens identitet verificeres og bruger godkendelsen er valid, sender autentificeringsserveren en access token til applikationen.
5. Applikationen sender et request til ressourceserveren, og vedhæfter sin access token.
6. Hvis det er en valid access token returnere ressourceserveren de informationer applikationen requester.

# Applikation registrering

Før man kan bruge Oauth på sin applikation skal den registreres på den service hvor man vil hente data fra fx Xena.

Servicen skal bruge nogle informationer om applikationen:

* Applikations navn
* Applikations website
* Callback URL

Callback URL er der hvor man bliver overført til når autentificeringen er godkendt.

# Vores valg

Vi har valgt at bruge Oauth på vores applikation, da det giver en høj sikkerhed og giver os adgang til alle de data vi skal bruge fra Xenas ressource server.  
Vi har i et tidligere projekt arbejdet direkte med Xena omkring Oauth, og derfor er det et klart valg at vi udnytter vores viden omkring denne verificerings service i dette projekt.

## Andre muligheder

Vi havde en anden mulighed for verificering ved Xena. Dette var API-Keys, API-Keys er mindre sikkert end Oauth, da det er krævet at der bliver sendt en key med i hvert request til Xena APIen, Xena og vores applikation kan derved heller ikke være sikker på hvem der sender disse requests, da det kan falsificeres hvis man får fingrene i api-keyen. Hvorimod med Oauth tildeles man en access token, der er unik for den enkelte bruger så man altid kan identificere hvem der laver hvilke requests. Oauth giver også brugeren mere tryghed, da det er noget man kan genkende fra andre applikationer.

## GitHub – versionskontrol

Vi har valgt på gruppen at tage GitHub i brug som vores versionskontrol værktøj. Det et værktøj vi har arbejdet meget med det seneste semester og er et meget mere fleksibelt værktøj end hvad vi lærte om på første semester ”Team Foundation”. Ved brug af TF[[12]](#footnote-12) lå al vores data på en lokal server som krævede vi tilgik den via. en vpn[[13]](#footnote-13) forbindelse, hvis vi skulle have mulighed for at tilgå den fra andre steder end hvis vi var på samme IP-adresse som serveren. Hvilket som sådan ikke ville være noget problem medmindre IP’en som serveren kørte på var nede, risikoen ved at det kunne ske ved en lokal server er større end hvis vi ligger det på en af GitHubs serveren. Få eksempler på at TF ikke vil være tilgængeligt kunne være pga. internetudbyder fejl, vejarbejde der rammer et kabel eller noget helt tredje.

Derfor har vi valgt denne gang at bruge GitHub da omverdenen på samme måde ikke ville kunne påvirke os. Går nettet ned hos udbyderen vil det altid være muligt at koble sig på via. fx mobilt netværk, hvor vi så kan pull seneste ændringer og så arbejder videre med seneste version af projekt og rapport. Det gir os også fordelen at vores data altid vil være let tilgængeligt lige meget om vi er hjemme, på studiet eller et helt andet sted.

Det er dog en mulighed for andre folk at clone hele vores GitHub, dog vil de ikke kunne push til vores GitHub. Det vil kræve at de bliver tilføjet som collaborators til vores repository. Muligheden for at gøre vores repository private vil altid være der, hvilket gør at andre folk end os der skal arbejde på det ikke har nogen form for adgang. På den måde kan vi sikre at vores soruce code ikke bliver brugt af andre. GitHub er et meget oplagt valg som vores versionskontrol grundet sikkerheden i at bruge det og fleksibiliteten i at vi ikke er bundet af lokale server.

## Test

### Whitebox

En af vores test består af ”whitebox” test. Ved at køre whitebox test giver vi source code med til personen som skal teste softwaren. Det en lidt dyr måde at teste på, da det kræver at en anden sætter sit eget arbejdet på pause for at kunne teste. Fordele ved at lave whitebox test er at der måske vil blive opfanget ”dead code” eller andre fejl, da det en person som forstår koden, men ikke har arbejdet med det specifikke stykke kode.

### Blackbox

Ved at køre blackbox test sikre vi at en person uden viden indenfor vores kode tester app’en[[14]](#footnote-14). Det gør at det ikke er en mulighed at finde eventuelle crash ved hjælp af koden. Vores primære blackbox tester vil være vores PO[[15]](#footnote-15), da han er en erfaren tester, og vi på samme tid kan få respons på mulige ønsker angående tilføjelser til app’en eller hvis der noget vi har misforstået ud fra forklaring.

### Destructive

Vores destructive test vil blive udført på samme tid som whitebox og blackbox test, formålet ved denne test form er at prøve alle tænkelig metode at få vores app til at crashe, så vi kan finde ud af hvor der skal sættes tid af for at sikre at når app’en bliver deployed at kunderne ikke vil kunne få programmet til at crashe

### Usability

Denne test vil primært blive udført som blackbox test for at vi kan sikre at vores interface er let at forstå for andre end os der har arbejdet på det i en længere periode.

## Database – valg af data

Vi har valgt i vores database, at vi ikke vil gemme alle folks oplysninger fra Xena når de vil sammenligne deres budgetmanager med Xena’s budget. Dog har vi valgt at det vil være en mulighed at kunne lave en identisk kopi af deres finanskonto navne, finansgruppe navne og kontonumre, da det vil være nødvendigt at de er helt identiske for at kunne sammenligne de 2 budgetter. Dette vil gøre det lettere for den budgetansvarlige at inddatere al nødvendige data, idet alle finaskontoer, finansgrupper og kontonumre er tilføjet, så der kun mangler diverse omkostninger og indtægter.   
Da det ikke er personfølsomme data måtte vi i teorien godt gemme alt deres data så vi kunne implementere det hele i deres nye budget som de kunne arbejde videre i. Men vi mener selv på gruppen det ikke vil være en nødvendighed. På den måde sikre vi også de ikke bare har 2 ens kopier af et budget, men at de får 2 forskellige budgetter de kan arbejde ud fra, og på den måde kan de få hjælp til at finde ud af hvor de skal optimere eller skære ned.

Ved at vi vælge ikke at gemme kontaktoplysninger eller lignende, sikre vi også at vi på ingen måde vil kunne miste oplysninger vi mener andre folk ikke bør få fat i, i tilfælde af hacking.

# Sprint 2

Af: Anders

## Sprintforløbet

Vi valgte i andet sprint at hive 3 userstories over i vores ”To do” kolonne i scrumboarded. De stories vi hev over hed ”Som bruger vil jeg gerne kunne oprette/inddatere et budget i Budgetmanager”, ”Som bruger vil jeg gerne kunne redigere et budget i Budgetmanager” og ”Som bruger vil jeg gerne kunne slettet et budget i Budgetmanager”. Dette kunne vi gå direkte i gang med da vi havde underestimeret vores tid i sprint 1 og derfor nåede at få lavet vores database på sidste dag i sprint 1.  
Der vil hovedsageligt blive fokuseret på at skrive kode i dette sprint.   
Nogle af vores task i dette sprint er måske blevet over estimeret, da vi før har haft en del problemer med Xena og tilgå deres data, hvis det skulle ske vi blev hurtigere færdig vil vi bare påbegynde et nyt sprint i denne periode og således beholde vores gode position i sprintet.

## Sprint 2 Review

Fredag d. 8/12 havde vi fået booket et møde med vores PO Klaus for at holde ham ”up to date” og sikre at vi havde opfyldt hans ønsker samt have muligheden for at høre ham ad om han havde yderlige ønsker til programmet.

Vi fik vist PO vores product som det så ud pt. og der startede en dialog på tværs hvor han spurgte ind til funktionalitet og vi besvarede så godt vi kunne og tog ønsker om eventuelle forbedringer til os. Vi spurgte ind til hvordan PO ønskede oversigten af budgetter samt sammenligningen skulle se ud. Vi mente det var hovedparten af opgaven, så var meget opsat på det blev lavet som han ønskede det.  
PO fortalte på en tavle hvordan han ønskede designet skulle se ud, han ville have sat det selv lavet budget op mod det reelle i en firkant hvor den procentvise difference vil blive vist under for at hjælpe til at give et hurtigt overblik.

## Sprint 2 Retrospective

Vores møde med PO havde været meget nødvendigt for at sikre vi ville komme alle PO’s ønsker til bund.   
I andet sprint blev alle userstories igen sat over i ”Done” vi fik klaret 5 userstories hvilket var meget positivt. Det var inddatering/redigering af budget, se specifikt budget, dokumentation af designet for budgetmanager, muligheden for at slette budgetter, oprette budgetter.

Ved at vise PO produktet som det ser ud nu, har vi noget ”continuous delivery” indover som er med til at skabe en hvis sikkerhed, da vi får bekræftet om vi er på rette vej, viser kunden at vi er inden for tidsrammen, og at vi har mulighed for at implementere nye features som blev ønsket af PO under vores møde. Vi får vist vores PO et reelt produkt som er funktionelt for at vise kunden at deadline nok skal blive overholdt samt som kunde er det lettere at forholde sig til noget man har set, hvor processen går fremad, fremfor man får et billede og en beskrivende tekst via mail eller snakket over telefon.

# Sprint 3

Af: Anders

## Sprintforløbet

Mandag morgen den 11/12 blev vores sidste ’sprint planning meeting’ holdt. Resten af vores userstories blev smidt over i ’To do’ så vores backlog var tom og målet var sat for at vi skulle ramme vores deadline med de forudsætninger vi har. Vores 2 userstories der blev hevet over var ”Som bruger vil jeg gerne kunne sammenligne mit budget op imod Xena” og ”Som udvikler vil jeg konkludere/perspektivere på projektet”.

I første del af sprint 3 vil der blive fokuseret på at kunne sammenligne budgetter. Når det er ordnet vil resten af sprintet stå på at skulle konkludere/perspektivere hele forløbet og derefter få læst korrektur og sat rapporten rigtigt sammen, så den vil kunne blive afleveret med et funktionelt produkt inden søndag den 17/12.

## Sprint 3 Review

## Sprint 3 Retrospective

1. Sequence diagram [↑](#footnote-ref-1)
2. System sequence diagram [↑](#footnote-ref-2)
3. Entity relationship diagram [↑](#footnote-ref-3)
4. Microsoft cloud service [↑](#footnote-ref-4)
5. Product owner [↑](#footnote-ref-5)
6. <https://xena.biz/da/support/kursus/> [↑](#footnote-ref-6)
7. System sekvens diagram [↑](#footnote-ref-7)
8. Sekvens diagram [↑](#footnote-ref-8)
9. https://www.britannica.com/technology/client-server-architecture [↑](#footnote-ref-9)
10. http://clientserverarch.blogspot.dk/2013/03/advantages-and-disadvantages-of-client.html [↑](#footnote-ref-10)
11. Prototyping and Software Development Approaches – Mahil Carr & Dr. June Verner [↑](#footnote-ref-11)
12. Team Foundation [↑](#footnote-ref-12)
13. Virtuel private network [↑](#footnote-ref-13)
14. Applikation [↑](#footnote-ref-14)
15. Product owner [↑](#footnote-ref-15)