Hovedopgave - Datamatiker

CalFak

Integration mellem kalender og regnskabssystem - et proof of concept

Navn: Johanne Marie Riis-Weitling

Vejleder: Patrick Agergaard

Uddannelsesinstitution: Københavns Erhvervsakademi

Hold: DAT21D 5.semester

Dato: 05.01.2024

Antal tegn i opgaven: 84690

Indhold

Introduktion **4**
Problemformulering **4**
Projektets form **4**
Hvorfor proof of concept? 5
PoC vs MVP 6
Undersøgelse og proof of concept **7**
Målgruppen og deres arbejdsdag/arbejdsgang 7
Hvordan kan applikationen adskille sig fra lignende produkter på markedet? 7
Afgrænsning af eksterne systemer 8
De indledende tanker **9**
Første tanker om arbejdsdeling mellem systemer **9**
Første tanker om app-opførsel **9**
Udviklingsopgaver i proof of concept 11
Proces og metode **12**
Undersøgelsens indledende fase **13**
Valg af eksterne systemer **13**
Valg af regnskabssystem 13
Valg af kalender 14
Valg af platform - Python og Django **15**
Selve undersøgelsen **16**
Forbindelse til regnskabsprogram (Billy) 16
Forbindelse til kalender (Google Calendar) 17
Videre test af Billy 18
Mere test af Google Calendar API 18
Datamodel og import-logik 18
Håndtering af fritekst 20
“Default” og aftaler, der ikke kan matches 21
Import og eksport af aftaler 22
Forbindelse til E-conomic 22
Konklusioner af undersøgelserne indtil nu 23
Import af kunder og produkter 25
Sammenligning af variabler i regnskabsprogrammer 25
Revideret arbejdsdeling mellem systemer **28**
Revideret app-opførsel **28**
Teknologier **29**
REST 29
OAuth 29
Bearer tokens 30
JSON Web Tokens (JWT) 31
HTTPS 31
Hvorledes skal applikationen designes, så den kan overholde GDPR, regnskabslovning, og behandle følsomme data? **32**

Regnskabslovgivning..... 32
GDPR..... 32
Tredjepartsudbyder..... 34
Underdatabehandler.....35
Konklusion på undersøgelsen (PoC)..... 36
Feasibility..... 36
Vurdering af projektet sat i forhold til andre løsningsmuligheder.....36
Økonomisk og Finansiel feasibility..... 36
Teknisk feasibility.....38
Samlet vurdering af projektet.....38
Hvordan kan potentielle brugere inddrages?..... 38
Spørgeskemaundersøgelse..... 38
Spørgsmål til en potentiel fokusgruppe..... 42
Konklusion..... 43
Glossary/ordliste..... 44
Litteraturliste..... 46

Introduktion

Projektet tager inspiration fra min omgangskreds, hvor der er flere enmandsfirmaer, der har svært ved at få lavet deres fakturaer, og dermed sikre deres cash flow. Ofte bliver fakturering en træls pligt, der bliver lagt til side, indtil det ikke kan udskydes længere.

Mange af disse erhvervsdrivende anvender deres online-kalender som aftalebog, som de sidenhen anvender ifbm. fakturering. Dette er et manuelt tastearbejde, som er kedeligt og hvor der let kan blive overset kalenderaftaler, som skulle have været faktureret.

Jeg vil gerne i dette projekt undersøge, om man kan lave en integration mellem et kalendersystem og et regnskabssystem, evt. med en lille brugerflade henvendt til erhvervsdrivende. Denne integration skal kunne tage kalenderaftaler, og lave dem om til fakturaer.

For at undersøge dette, vil jeg gerne lave et proof of concept på denne integration.

Sekundært vil jeg gerne undersøge, hvordan et proof of concept kan tages til næste iteration.

Problemformulering

- Hvordan etablerer man en integration mellem en online kalender og et regnskabssystem?
- Hvordan kan applikationen adskille sig fra lignende produkter på markedet?
- Hvordan skal applikationens brugeroplevelse være?
- Hvorledes skal applikationen designes, så den kan overholde GDPR, regnskabslovgivning, og behandle følsomme data?
- Hvordan kan potentielle brugere inddrages?

Projektets form

Jeg vil i den følgende tekst referere til den applikation, jeg vil udvikle, som CalFak. Dette er en sammenskrivning af Calendar og Faktura.
Jeg vil i dette projekt primært koncentrere mig om analyse- og design-delen i udviklingen af en ny applikation.
Jeg er i en situation, hvor jeg ikke har en bestemt kunde, men har en mulig kundegruppe i tankerne.
Jeg vil gerne undersøge, hvad der kræves af en sådan applikation rent juridisk, og hvad dette betyder for det videre design.

Jeg vil også prøve at kortlægge nogle af de udfordringer, der kan være i selve programmeringen af en sådan applikation.
Projektets form er derfor mere en undersøgelse end et færdigt produkt. Ved projektets indledning er der mange ting, som jeg er usikker på, både teknologisk, lovgivningsmæssigt, og i forhold til markedet og målgruppen. Men det er netop de mange usikkerheder, som er interessante: hvordan kan jeg gribe dem an og vende dem til viden, som et videre arbejde kan bygge på?
Min hypotese er, at jeg bliver meget klogere på usikkerhederne ved først at fokusere på et proof of concept, og derefter anvende den opnåede viden til at give et bud på retningen for videre udvikling. Grundlæggende er jeg usikker på om ideen bag CalFak teknisk set kan lade sig gøre, og dette vil jeg gerne afklare, før jeg går videre med at tænke i feasibility, osv. Derfor er projektet overordnet delt i to: første del beskæftiger sig med at formulere og afprøve et proof of concept, mens anden del bygger videre på de erfaringer, som er opnået i første del, og forsøger at kortlægge, hvordan et proof of concept kan tages videre til næste iteration.

Hvordan teksten er struktureret

Teksten følger den struktur, som er opridset ovenfor. Først beskrives det indledende design og programmering af et proof of concept. Herefter følger en opsamlende og konkluderende del, der giver et bud på retningen for videre udvikling. Dette betyder bl.a. at anden del diskuterer projektets feasibility - noget, man normalt nok ville forvente at læse om i starten af et projekt.
I forbindelse med vurderingen af om et produkt skal udvikles eller ej, bør en en feasibility undersøgelse være noget af det første man gør.
Dette kræver at man får afklaret hvad der kan lade sig gøre rent teknisk, og hvilke krav der er i forhold til jura og andre spillere på markedet.
Men jeg vil gerne dele projektet op således, at jeg først undersøger, om det **kan** lade sig gøre, før jeg undersøger, om projektet **skal** føres videre ud i livet. Dette leder videre til en konklusion, der gerne skal besvare begge spørgsmål - hvorvidt CalFak kan lade sig gøre, og hvorvidt det skal bygges videre.

Hvorfor proof of concept?

Jeg har valgt at lave et proof of concept (PoC) (Gillis, 2023) da der for mig at se er for mange ubekendte faktorer til at kunne starte på et Minimum Viable Product (MVP) (Wikipedia, 2023a) med det samme.

Ved projektets start har jeg meget lidt erfaring med at benytte API'er, der stilles til rådighed af andre produkter, og har ikke nogen fornemmelse af, hvilke data man kan trække ud af en kalender, og hvor svært det er at få de relevante adgange til de forskellige systemer (kalendere såvel som regnskabssystemer.)

Der er flere elementer i programmeringen, hvor jeg har en klar fornemmelse af at det vil kunne lade sig gøre, men har ingen reel viden om hvorvidt og hvordan det konkret skal foregå - og hvis det kan lade og gøre, ligger det så inden for mine evner?
Jeg har heller ikke et klart indtryk af opgavens omfang, og håber jeg vil få et bedre overblik over applikationens omfang ved at undersøge og programmere nogle af grundfunktionerne i CalFak, og finde ud af om de kan lade sig gøre.

Værdien af et proof of concept ligger derfor først og fremmest i at kunne demonstrere over for mig selv, at centrale udfordringer kan løses rent teknisk.
Et PoC kan også være med til at kortlægge hvilke nye, uopdagede opgaver eller problemer, der skal være ekstra fokus på ved udvikling af et senere MVP.

PoC vs MVP

Formålet med et PoC, er som navnet indikerer, at demonstrere, om en ide eller et koncept er mulig at udvikle. (Gillis, 2023)
Et PoC har særlig fokus på at undersøge om det er teknisk muligt at gennemføre ideen.
Det er ikke et komplet produkt, men en demonstration af en funktion eller en ide, der kan bruges til at overbevise stakeholdere om, at ideen er værd at gå videre med.

Når man laver et MVP (Minimum Viable Product), er målet at ende med et produkt, der ikke er færdigt, men det kan udføre kerneopgaver eller løse kernebehov for kunderne, dog i en meget skrabet udgave. (Wikipedia, 2023)
Dette kan være for at tiltrække nye kunder, eller for at indhente feedback fra den kunde man udvikler til, så man kan afgøre om udviklingen er på rette spor.

Det er generelt sådan at et PoC skal benytte færre ressourcer og tid end et MVP.
Man kan fx starte en udviklingsproces med at lave et PoC for at teste noget specifikt teknologi eller en specifik ide.
Dette kan så bruges til at udvikle et MVP, som man kan frigive til test og feedback hos kunderne.

Undersøgelse og proof of concept

Målgruppen og deres arbejdsdag/arbejdsgang

Jeg tænker at CalFak primært skal henvende sig til små/enkeltmandsvirksomheder.
Mere specifikt virksomheder, der leverer en form for service eller ydelse, fremfor at sælge fysiske ting. De tager enten ud til en kunde, eller modtager kunder på deres klinik eller lignende, og har en kalender der (forhåbentligt) er fyldt med aftaler. Det kunne fx være en handyman/rengøringshjælp, en behandler/terapeut, eller en kæledyrs-passer.
Mange i målgruppen er ofte på farten, og anvender formentlig en smartphone til at tjekke og opdatere deres kalender.
Senere skal kalenderaftalerne gerne føres over i et regnskabssystem, fordi det er disse aftaler, virksomhederne lever af. Enten fordi en kunde skal have en faktura tilsendt før betalingen kan komme ind, eller fordi der skal oprettes bilag til regnskabet for indbetalinger, der allerede er modtaget.

Hvordan kan applikationen adskille sig fra lignende produkter på markedet?

Jeg har fundet flere “addons” til regnskabssystemer, der er variationer af kalendere, som integreres med regnskabssystemer. Fælles for dem er at man bruger applikationens kalender, og ikke en allerede eksisterende kalender.
Flere af dem er fokuseret på behandlere og har patientjournaler tilknyttet[1]. Andre har lagerstyring og km-afregning indbygget. Men mange af dem virker til at være rettet mod større virksomheder, der skal koordinere flere medarbejdere.
Det virker til at markedet ikke har så mange tilbud af denne art til den helt lille virksomhed, der ikke er så interesseret i ekstra rapporter og statistik, men bare gerne vil bruge tiden til noget andet end at overføre aftaler fra kalender til regnskabssystem, uden at skulle sætte sig ind i en masse nye funktioner og arbejdsprocesser.

Ideen er at CalFak skal kræve så lidt indtastning og interaktion som muligt fra brugeren, således at der generelt spares tid. Der kan selvfølgelig være nogle scenarier, hvor det kan blive nødvendigt at interagere med CalFak, fx hvis der opstår problemer med at overføre eller matche aftaler til regnskabssystemer.
I første udgave skal en bruger måske selv trykke på en knap for at overføre aftaler til regnskab, men dette skulle gerne blive automatiseret i en senere iteration.
Der skal også ske noget opsætning, første gang brugeren sætter CalFak op.
Man skal vælge det anvendte regnskabssystem, give samtykke til at der må hentes kalenderaftaler fra ens kalender, osv. Dette skal gerne være så enkelt som muligt, og kunne udføres af en bruger med minimum af kendskab til it, og uden at de skal have en lang introduktion til systemet af en supporter/sælger.

[1] Eksempelvis https://terapeutbooking.dk/ og https://www.timelog.com/da/.

Afgrænsning af eksterne systemer

Der findes en række regnskabssystemer, der henvender sig til små virksomheder. Ud fra hvad jeg kan læse mig frem til, er nogle af de mest brugte i Danmark Billy, Dinero og E-conomic[2] (Startupsvar.dk n.d., Nordstrøm n.d., BogholderiService, n.d., Steenvinkel, 2022, Regnskabsprogrammerne.dk n.d.)
Mange regnskabssystemer er bundet til specifikke lande, da lovgivning omkring bogføring, moms og lignende kan variere fra land til land. Derfor vil regnskabssystemer, der er relevante i Danmark, ikke nødvendigvis være det i fx Norge eller Tyskland.
Jeg har i denne opgave valgt at afgrænse mig til at se på det danske marked, og har valgt at fokusere på Billy, Dinero og E-conomic.

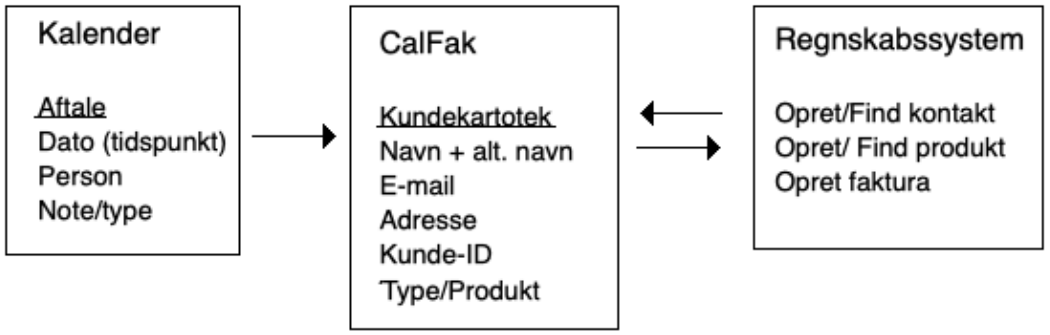
Når det kommer til online-kalendere, er markedet mere globalt. Mit indtryk er at de mest benyttede kalendere leveres af Google (Google Calendar), Microsoft (Office365/Outlook.com) og Apple (iCloud/iCal.)[3]
Det er ofte kalendere, der følger med den mobil eller den mail, som brugeren har (så det er lettere at benytte dem end at finde et alternativ.) Tanken med CalFak er, at brugeren ikke skal lære at anvende en ny kalender, men kunne fortsætte med at bruge den kalender, de allerede kender.
Jeg vender senere i rapporten tilbage til valget af den/de kalendere, som CalFak PoC'et i første omgang skal kunne hente data fra.

[2] https://www.billy.dk, https://dinero.dk/, https://www.e-conomic.dk/.
[3] https://workspace.google.com/products/calendar/?hl=da, https://www.microsoft.com/da-dk/microsoft-365/outlook/email-and-calendar-software-microsoft-outlook?deeplink=%2fowa%2f&sdf=0, https://www.icloud.com/calendar

De indledende tanker

Første tanker om arbejdsdeling mellem systemer

De første tanker om flowet og ansvarsfordelingen mellem systemer ser således ud. Dette er ikke den endelige udgave, men repræsenterer mine tanker ved projektets indledning. Det er på dette tidspunkt fx endnu ikke klart, om en note fra en kalender kan matches med et produkt eller varenummer i regnskabssystemet.



Tanken er at kalenderen ikke skal have input fra CalFak, men kun levere data. CalFak skal kunne læse aftaler i kalenderen, og matche disse med kunder og produkter i regnskabssystemet. Det skal overvejes, hvordan brugeren gøres opmærksom på at en aftale ikke kunne matches, og hvor dette ansvar skal ligge i forhold til programmerne. CalFak skal kunne sende oplysninger til regnskabsprogrammet, og det skal ligeledes kunne modtage oplysninger fra regnskabsprogrammet, så der kan blive registreret et kunde-ID når kunden er blevet oprettet. Også her skal det overvejes, hvor ansvaret for fejlbeskeder skal ligge, hvis der er elementer, der ikke kan matches.

Første tanker om app-opførsel

Dette er de første tanker om hvordan CalFak kunne fungere. Jeg vil vende tilbage til dem efter mine undersøgelser, og vurdere, om de skal ændres eller uddybes.

Der indhentes dagligt kalenderaftaler fra kundens kalender.

Hvis en aftalerække ser ud til at ophøre, skal der dannes en fakturalinje på den rette kunde og med det/de rette produkt(er).

En aftalerække er en periode, hvor der ligger en aftale hver dag, dvs med mindre end 24 timer imellem to aftaler

Der skal kun laves linjer for aftaler, der er overstået.

En aftalerække betragtes som ophørt, når der ikke har været kalenderaftaler på kunden i 24 timer.

Hvis en aftalerække ikke ser ud til at ophøre, skal det gemmes, at der er blevet leveret en ydelse (fx “20 minutters besøg”.)

Der skal sendes nye fakturaer automatisk til regnskabssystemet 1 gang om dagen. Som en start skal det ske, når der trykkes på en knap. Når dette fungerer, skal det være muligt at lave en automatisk overførsel, fx ved midnat hver dag.

Det skal være muligt at synkronisere fakturalinjer uden for det planlagte tidspunkt ved at trykke på en knap, så man kan fakturere “øjeblikkeligt”.

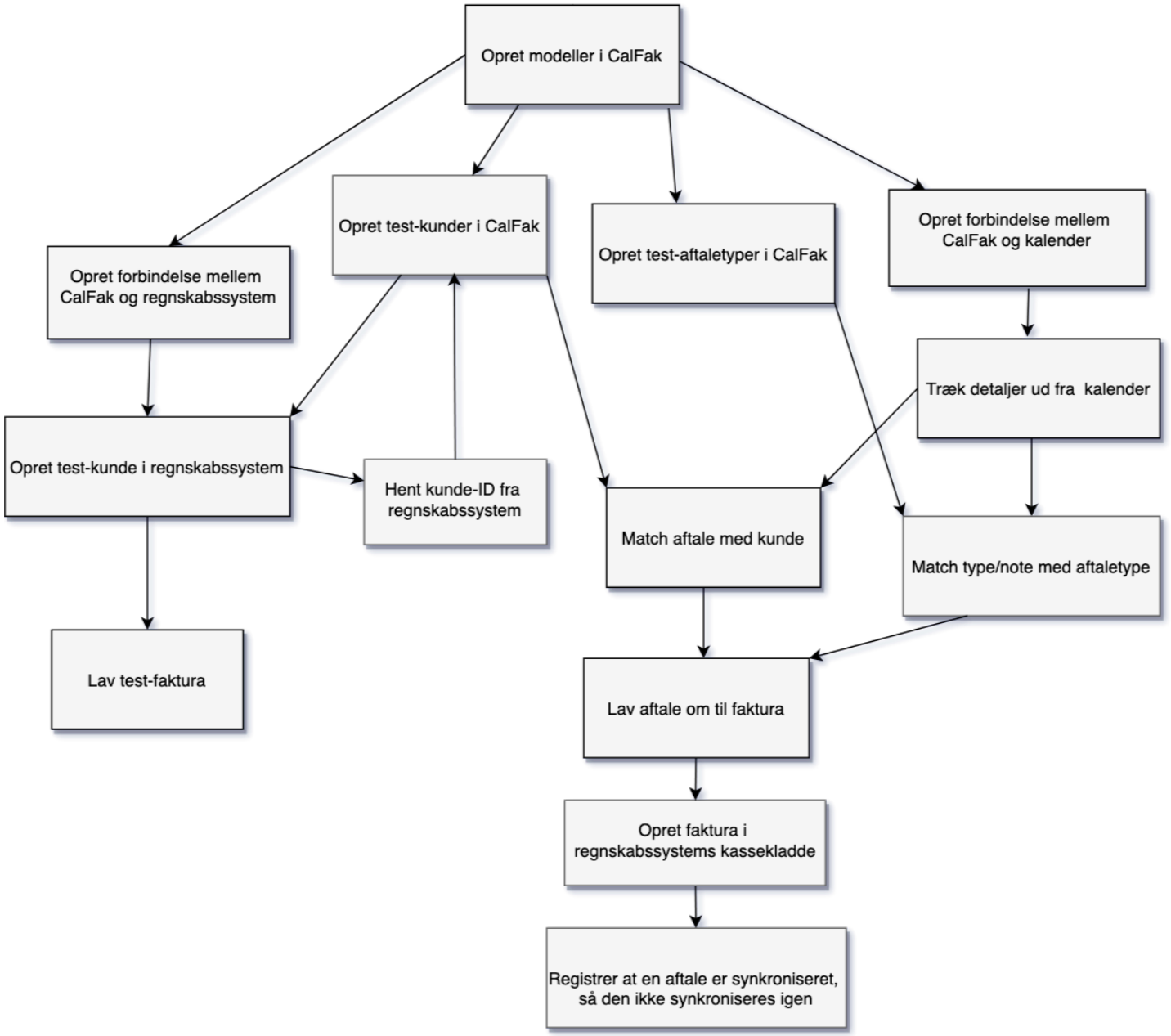
CalFak skal opbevare oplysninger, der kan matche en aftale op mod en kundekonto.

Det skal registreres, at en/flere fakturalinjer allerede er blevet sendt til regnskabssystemet.

CalFak skal kun holde styr på om en linje er sendt til regnskabssystemet eller ej - om den er faktureret, betalt eller ej, er det regnskabssystemet, der håndterer.

Udviklingsopgaver i proof of concept

For at kunne vurdere om ovenstående opførsel kan lade sig gøre, og om der er noget i app opførslen der skal revideres, har jeg identificeret følgende udviklingsopgaver:



Jeg har forsøgt at kortlægge de interne afhængigheder mellem opgaverne, for at få et bedre indtryk af, hvilken rækkefølge der er mest optimal at løse opgaverne i. Det er illustreret ved figuren ovenfor.

Proces og metode

Da der er mange elementer, der ikke er klarlagt ved projektets start, er den oplagte tilgang til udvikling af CalFak den agile tilgang - konstant tilpasning. Projektet er i sagens natur en undersøgelse, af om noget kan lade sig gøre, og hvorvidt forskellige delelementer, der er afgørende for CalFak, kan føres ud i livet på et senere tidspunkt.

I de foregående afsnit har jeg beskrevet lidt nærmere, hvad CalFak gerne skal være, og dermed fået formuleret nogle konkrete udviklingsopgaver. Mange af disse opgaver er typer af udvikling, som jeg ikke har prøvet før, eller kun prøvet i begrænset omfang. Derfor er de svære for mig at estimere på forhånd. Dermed er de også svære at placere i et Scrum-univers - det giver ikke så meget mening at inddele PoC-processen i sprints, og fordele opgaverne på sprints.
Da der er en risiko for at, man bliver nødt til at ændre retning midtvejs, eller en opgave viser sig umuligt. Det ville betyde at sprintet skulle stoppes. Hvorefter et nyt sprint skulle planlægges, dette sprint , ville så have samme risiko for at blive stoppet før tid som det foregående.

Jeg vil i stedet benytte en form for timeboxing (Martins, 2022), når jeg går i gang med en ny opgave. Dette vil jeg gøre for ikke at sidde fast i en opgave for længe uden at komme nogen steder. Når man benytter timeboxing, afsætter man et tidsinterval, hvor man skal have løst en opgave. Dette kan både være en “hård” timebox eller en “soft” timebox.
Dette vil tvinge mig til at evaluere den løsning, jeg er i gang med. Hvis jeg efter en arbejdsdag ikke er kommet nogle steder med koden, eller ikke er tæt på en løsning, skal det overvejes, om jeg skal gå i en anden retning, eller om jeg skal give det et forsøg til.
(Min research i de forskellige emner gælder ikke med i denne timebox.)

Der er også mange uforudsete elementer, der kan dukke op undervejs i forbindelse med udvikling og tests. Jeg har forsøgt at lave en liste over hvilke opgaver, der skal løses for at afgøre om dette PoC er en succes. Men der kan altid være opgaver og elementer, man først bliver opmærksom på undervejs i processen, når man som jeg ikke har prøvet at lave en applikation af denne type før. Der er derfor en del “learning by doing” i dette projekt.

Der er dele der først kan udvikles, når andet er lavet, og der er dele der formentlig skal laves om, hvis en anden funktion alligevel ikke kan lade sig gøre. De indbyrdes afhængigheder dukker også først op undervejs i PoC-processen, og er svære at håndtere på forhånd.

Det positive ved at gå agilt til projektet er også at man kan tage det, der er lavet i forbindelse med PoC, og videreudvikle det til et MVP efterfølgende.
Derefter vil man kunne tage et MVP og teste det blandt en kundegruppe/fokusgruppe, der kan være med til at lave CalFak til et produkt der kan komme ud til flere kunder.
Det ville være oplagt at strukturere evt. udvikling efter PoC i form af Scrum, da usikkerhederne her gerne skulle være markant mindre, og de enkelte opgaver nemmere at estimere. Hvilken udgave af Scrum, der i så fald vil være tale om, kommer an på hvor mange udviklere, der vil være i processen, og hvordan relationen vil være til evt. fokusgruppe eller kunder.
Det vil i hvert fald give mening at fortsætte med den agile tilgang til udviklingen, hvor der gradvist og iterativt kan indarbejdes brugerinput fra en fokusgruppe.

Det vil også give mening at udvikle CalFak til at integrere med flere regnskabssystemer og kalendere. Her er usikkerhederne mindre, og integrationen af den enkelte kalender eller regnskabssystem derfor nemmere at planlægge og estimere.

Undersøgelsens indledende fase

I den indledende fase af undersøgelsen er mit fokus at afgrænse, hvilke eksterne systemer, jeg vil inddrage, samt vælge en platform, som CalFak kan udvikles på.

Valg af eksterne systemer

Valg af regnskabssystem

Da mit fokus er på små/enkeltmandsvirksomheder, er der en del regnskabsprogrammer, der er for store, avancerede og for dyre til denne type virksomheder. De systemer der sætter stort fokus på lagerstyring er heller ikke de mest oplagte.

Ved søgning er der som nævnt primært tre programmer, der dukker op til små virksomheder: E-conomic, Dinero og Billy. Det er alle systemer der har mange år på det danske markedet, og er anerkendte i branchen. Ved at fokusere på disse tre vil man kunne dække en stor del af markedet.

Både E-conomic og Dinero er ejet af Visma[4]. De henvender sig til lidt forskellige markeder. E-conomic er rettet til de små, mellemstore og de større virksomheder. Der er muligheder for at tilpasse produktet, og der er en del integrationer til rådighed for dette. E-conomic er henvendt til brugere, der har lidt erfaring med bogføring, eller allerede har en bogholder. Dinero henvender sig primært til små og mellemstore virksomheder. Dette programs fokus ligger på de virksomheder, der ikke har en bogholder tilknyttet (Visma, n.d.)[5]. Ud fra dette er Dinero nok det regnskabssystem, der henvender sig mest til det segment, jeg prøver at ramme.

Der er dog den udfordring, at for at komme i gang med Dineros API, skal man skrive til Dinero, og indhente tilladelse til at lave en test-integration med en testvirksomhed. Jeg har ikke haft tid og mulighed for at programmere til Dineros API i første omgang, men jeg har i stedet kortlagt kravene til en Dinero-integration ud fra deres offentligt tilgængelige API-dokumentation.

E-conomic er noget lettere at få test-adgang til. De tilbyder både en demo-konto, hvor der allerede er sat fiktive virksomheder og kunder op. Derudover kræver det en udviklerkonto at oprette en test-addon. Dette foregår automatisk efter udfyldelse af en online-formular, så man er hurtigt i gang. Deres API-dokumentation er offentligt tilgængelig (selvom den er lidt svær at få overblik over, da den er meget omfattende.)

[4] https://www.visma.dk/
[5] Opsummering af Vismas egen sammenligning af de to programmer.

Det sidste regnskabssystem, jeg har valgt at undersøge, er Billy, der markedsfører sig til selvstændige og nyopstartede virksomheder. Deres grundplan er gratis. De henvender sig til brugere, der ikke har erfaring med bogføring, og bliver anbefalet af flere sider til de små virksomheder. Deres API er også til frit tilgængeligt,

Jeg har valgt at starte med at teste integrationen til Billy, da deres dokumentation af deres API er ret kort og overskuelig. Der er flere kodeeksempler for forskellige sprog, og de har en gratis udgave, jeg kan teste på.
Dermed kan jeg opbygge noget mere viden om hvordan man kan udvikle resten af CalFak. Forhåbentlig kan det også give mig en bedre fornemmelse af hvordan jeg integrerer med E-conomic og andre regnskabssystemer, hvis dokumentation er sværere at gå til uden erfaring.

Valg af kalender

Jeg har set på de tre mest udbredte kalendere: Outlook/Office365, Google Calendar, og iCal/iCloud.
Både Google Calendar og Outlook/Office365 har åbne API'er, hvor man som udvikler kan læse data fra kalenderen, såfremt ejeren af kalenderen har givet tilladelse til dette. Tilladelsen munder konkret ud i tokens, som skal anvendes for at kunne kommunikere med deres API.

Mht. iCal/iCloud er det lidt mere indviklet, da det ikke stiller et API direkte til rådighed.
Det er noget mere kompliceret at opsætte adgangen, særligt hvis brugeren ikke har en Mac/iPhone/iPad. Jeg undersøgte deres “EventKit” API (Apple, n.d.), men det lader ikke til at tilbyde den type adgang, som CalFak har brug for, da EventKit er lavet til at blive brugt på Apple OS produkter.
Det lader dog til at der er personer, der har lavet deres egne Github-repos[6] til at tilgå iCal.
Der er også firmaer, der tilbyder at man kan forbinde til alle slags kalendere, som fx Cronofy[7], men for nu vil jeg ikke se mere på iCloud/iCal.
Hvis man ønsker at udvide applikationen til iOS senere, kan man overveje de forskellige muligheder på dette tidspunkt.

Jeg har undersøgt hvad der skal til for at anvende Outlooks kalender-API (Microsoft, 2023). I første omgang skal man oprette en udviklerprofil hos Microsoft, for at komme videre. Trinnet efter tilmelding virker dog ret forvirrende. Når jeg kigger på deres API-dokumentation (Microsoft, 2022), ser det dog ud til at minde meget om Google Calendars API (Google, n.d.a.) Så hvis CalFak kan komme til at virke med Google Calendars API, er der formentligt gode muligheder for, at det også kan komme til at virke med Outlooks kalender.

Jeg har i første omgang valgt at teste med Google Calendar, da det er en gratis kalender, og den har et offentligt API, hvor man nemt kan få et access token til at teste API-adgang med. Det er også via Google Calendar muligt at sætte en consent form op, så man kan informere brugeren om, hvad vedkommende giver adgang til.

[6] https://github.com/MauriceConrad/iCloud-API
[7] https://www.cronofy.com/developer

Valg af platform - Python og Django

Jeg har valgt at kode backenden i programmeringssproget Python[8] af flere årsager:
Det er et af de sprog, som jeg har modtaget undervisning i på KEA.
Billys API-dokumentation giver flere eksempler, der er skrevet i Python.
Google Calendar har ligeledes både libraries og API-eksempler, der er skrevet i Python.
Så på den måde kan jeg hurtigt komme i gang med at undersøge adgang til nogle eksterne systemer.

En anden fordel ved at benytte Python er, at det giver mulighed for at anvende Django[9] til at programmere selve CalFak. Django er et open source web framework, der kan anvendes til at udvikle database-baserede webapplikationer.
Django er designet til at bygge projekter hurtigt, og er dermed også oplagt i forbindelse med et PoC, hvor der er mere fokus på funktionalitet end brugerinterface og UI/UX design.
Ved at benytte Django kan jeg desuden hurtigt få et "admin interface", (Django n.d.a) der kan testes i. Her får man en hel CRUD-implementering forærende, når man opretter et projekt.
Når/hvis MVP skal udvikles, vil det være oplagt at lave noget bedre UI/UX, selv om man kan udføre mange af opgaverne direkte i Djangos admin interface.
Man kan vælge at mappe frontend-elementer til database-modeller på mange forskellige måder, og variere designet, når man har fundet ud af, hvad der er obligatorisk for hvert enkelt regnskabssystem.
Django giver derudover mulighed for at tilpasse siderne i admin interfacet (Trudeau n.d.), og derved skabe overblik over databasen, og hvordan data er knyttet sammen.

I Djangos admin interface kan jeg teste om data bliver indhentet korrekt fra en kalender, og sendt korrekt til et regnskabssystem, uden at skulle kode CRUD-interfaces eller egne API-endpoints. Jeg kan derfor eksperimentere med data, og hurtigt tilføje eller fjerne variabler i en formular, uden at skulle rette en masse steder i koden. Det er en fordel at kortlægge dette, inden det egentlige brugerinterface bliver designet, da det kan have stor indflydelse på hvordan det egentlige interface skal se ud, og hvad interfacet skal håndtere.

Med Django følger der også indbygget support for en SQLite-database[10]. SQLite er en relationel database, der er indlejret i en applikation. Det betyder at alle data som udgangspunkt kun er gemt på den maskine/enhed, som applikationen kører på.
I undersøgelsesperioden er SQLite fint, men der skal nok tages stilling til om der skal skiftes database på et senere tidspunkt, hvis CalFak skal videreudvikles, men for nu er det tilstrækkeligt.
SQLite-databasen er let at rette og redigere i undervejs, så man kan ændre sit database-skema uden en masse ekstra arbejde.

Jeg har i et tidligere job arbejdet med data i Django-admin interfacet, dog uden at kode selv. Jeg stiftede senere bekendtskab med at sætte Django-admin op på et kursus hos Django Girls (Django Girls, n.d.a.) Ved at følge deres tutorial kan jeg ret hurtigt sætte et interface med en SQLite database op (Django Girls, n.d.b.) Som jeg efterfølgende kan rette til, i forhold til de behov jeg har i CalFak.

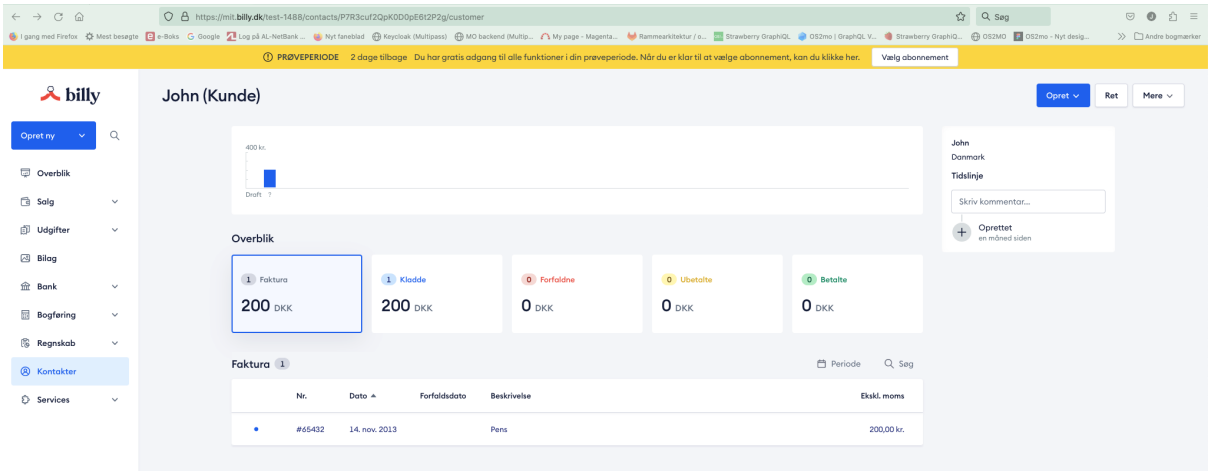
[8] https://www.python.org/
[9] https://www.djangoproject.com/
[10] https://sqlite.org/index.html

Selve undersøgelsen

De to vigtigste ting for at afgøre om CalFak kan lade sig gøre er at skabe forbindelse til hhv. et regnskabssystem, og til en kalender. Om det er kalenderen eller regnskabssystemet, der er vigtigst for at afgøre om CalFak er muligt, kan der argumenteres for, men hvis den ene del ikke kan lade sig gøre, falder den anden del også til jorden, så jeg ville sige at de er lige vigtige.
Jeg begyndte med det ene regnskabsprogram, udelukkende fordi deres dokumentation var lettest at tilgå, så det var en måde at få momentum på i forbindelse med undersøgelserne.

Forbindelse til regnskabsprogram (Billy)

Jeg oprettede en konto hos Billy, og fulgte deres guide til at oprette et access token via deres brugerinterface. Med dette access token kunne jeg - ved at læse Billys dokumentation (Billy n.d.a.), og følge de kodeeksempler (Billy n.d.b.) de har givet - oprette faktura-kladder og kunder i regnskabssystemet.



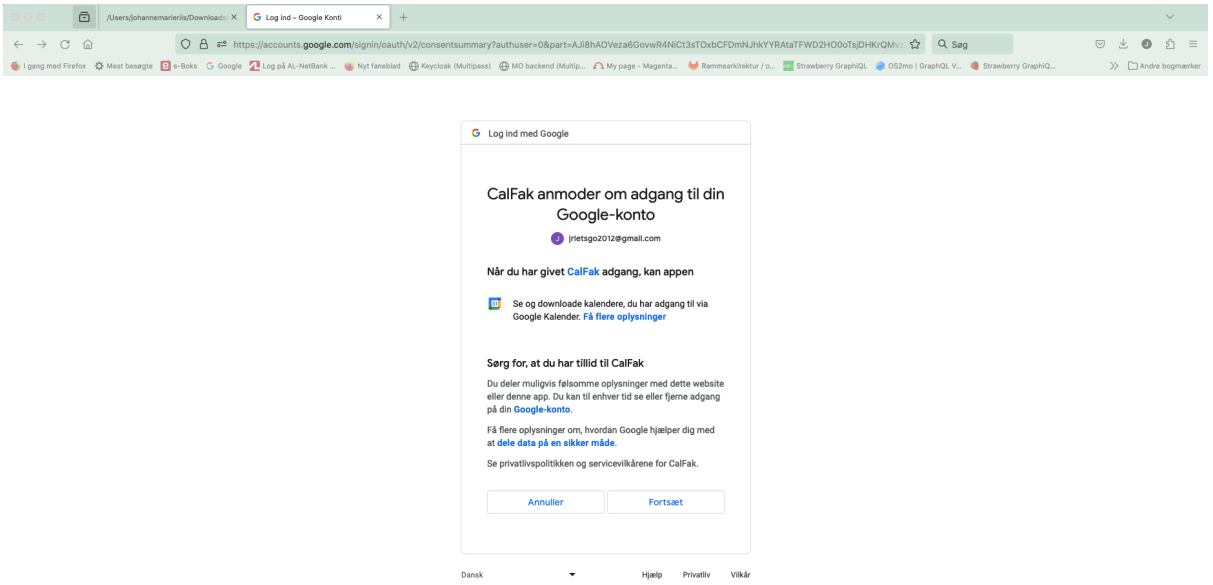
I Billys API oprettes en faktura-kladde ved at foretage en POST-request til endpointet "/invoices". Her er det vigtigt at understrege, at der er forskel på en faktura-kladde, og en endelig faktura. Og det er vigtigt, at CalFak kun opretter faktura-kladder. Hvis CalFak skal oprette fakturaer, kan der opstå problemer i forhold til ansvarsfordelingen mellem systemerne. Det kan også blive forvirrende for brugere, hvis der findes flere "udgaver" af den samme faktura (dvs. en i regnskabssystemet og en i CalFak.)
Hvis fakturaen kun eksisterer i regnskabssystemet er det der, den endegyldige sandhed er i denne sammenhæng.
Hvis der laves fakturaer i CalFak, vil det også blive sværere for brugerne at forstå, at det ikke er et elektronisk bogføringssystem, med alt der dertil hører af regler og lovgivning.
Dette vil også kræve mange flere godkendelser fra forskellige instanser

Det kan være at oprettelse af egentlige fakturaer kunne blive aktuelt på et senere tidspunkt, hvis man vælger at udvide CalFak til at kunne vise kunder deres fakturaer, trække betalinger osv., men det vil være i en betydelig senere udgave. Og langt forbi PoC og MVP. Det vil også være opførsel, der hører til i en meget større applikation.

På dette tidspunkt kan vi i hvert fald konkludere, at vi har forbindelse til regnskabssystemet, hvilket er en god start.

Forbindelse til kalender (Google Calendar)

Næste del af undersøgelsen går ud på at teste, om der kan skabes forbindelse til Google Calendar. Det viser sig at Google har et Python-bibliotek (Google, n.d.b.), der kan hjælpe med dette. Derudover er der en stor mængde dokumentation (Google, n.d.c.)
Ved at følge denne dokumentation kan CalFak oprettes som en applikation i Googles univers. Dette kræver at udvikleren (jeg) tænder for generel API-adgang i Google Cloud, hvorefter CalFaks "OAuth consent screen" kan oprettes i Google Cloud Console. Herefter kan et "OAuth client ID" oprettes - dette gemmes lokalt som "credentials.json" (Google, n.d.b.)
Nu kan Googles eget Python-eksempel[11] startes. Første gang dette eksempel køres, vises "consent screen", hvor man skal acceptere, at CalFak får adgang til kalender-data i Google:



Når/hvis brugeren godkender forespørgslen, bliver der lavet en fil med et "grant" ("token.json"), der tillader API-adgang til kalenderen. Dette "grant" har som udgangspunkt en udløbsdato, og efter den skal der igen indhentes godkendelse fra brugeren. Denne udløbsperiode er dog noget, der kan stilles på.
Efter jeg tilføjede mit grant til koden, kunne jeg hente de kommende ti aftaler fra Google-kalenderen.

Som default er det kun den primære kalender, der er adgang til. Så vidt jeg kan se, er dette noget, der kan ændres på i koden. Men jeg er i tvivl om, hvordan man kan tilbyde en bruger at vælge mellem flere kalendere, hvis de har dem - dette er dog ikke noget jeg har undersøgt nærmere.

Der kan justeres på, hvilke oplysninger der hentes fra kalenderen, så det kan være dato, note, sted osv, hvilket er en lovende start. Som sagt munder kørslen af ovenstående

[11] https://github.com/googleworkspace/python-samples/blob/main/calendar/quickstart/quickstart.py

kodeeksempel fra Google ud i, at de næste 10 kalender-aftaler kan læses via et API. Vi kan derfor konkludere, at vi har opnået læseadgang til Google Calendar.

Videre test af Billy

I forbindelse med min undersøgelse af Billys API var der nogle ting, der overraskede mig. Det er fx. obligatorisk at tilføje en enhedspris på en vare, når man skal oprette en fakturalinje, hvilket virker som dobbeltarbejde, eftersom man alligevel skal oprette en pris i Billy, når man opretter et nyt produkt.
Det er også svært at finde frem til et produkt-id i Billys brugerflade. Man kan give et produkt et varenummer, men det er ikke det, man bruger til at oprette en fakturalinje i API'et. Det egentlige varenummer (eller produkt-id) kunne jeg kun se, hvis jeg sendte en API-forespørgsel og fik printet resultatet i terminalen.

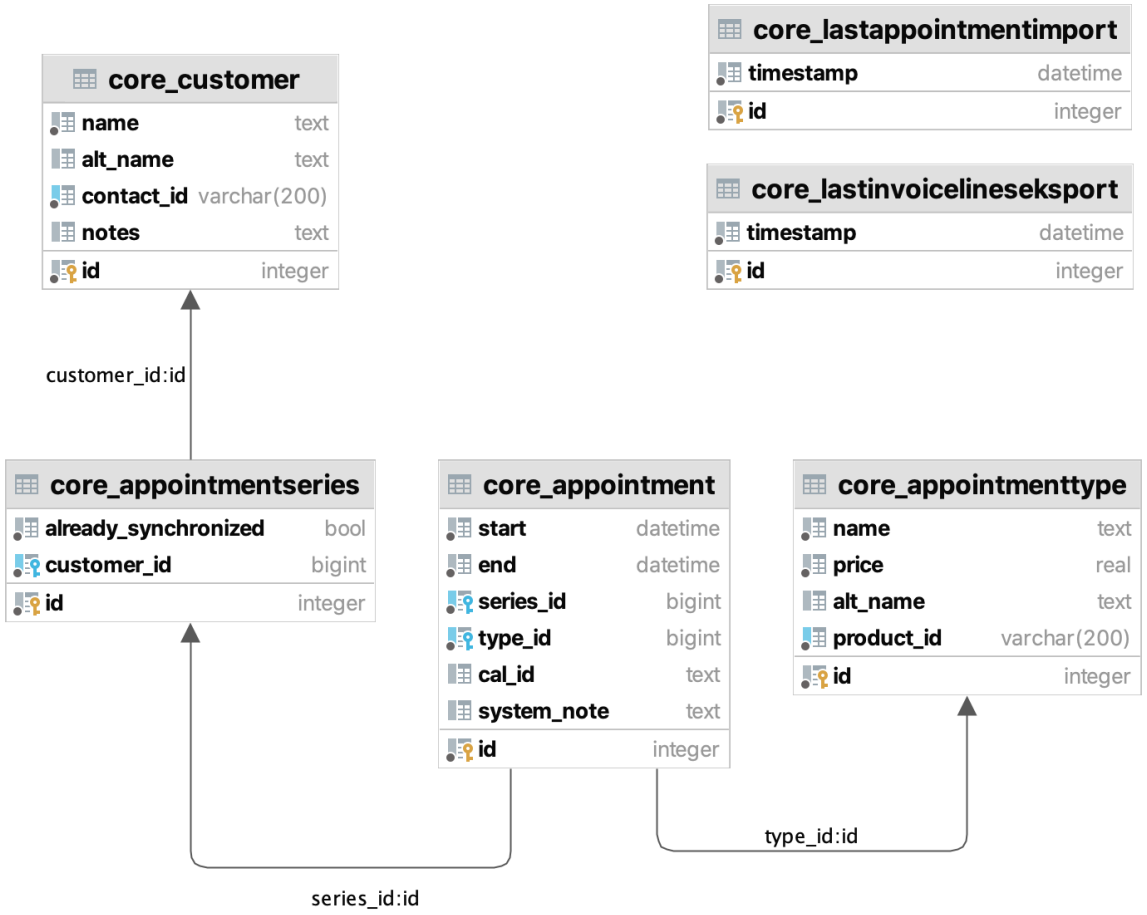
Kunde-ID'er er heller ikke tydelige i brugerfladen, men er dog synlige i adresselinjen på siden. Disse kan også trækkes ud af Billys API, så det vil være muligt at lave en funktion, der matcher navne fra kalenderen op mod kunder i Billy.
Det er muligt at oprette flere kunder med præcis samme navn, så hvis man kun benytter navne kan der opstå problemer med tvetydige matches. Det vil derfor kræve, at der enten bruges et alternativt navn, eller tilføjes andre oplysninger, når brugeren opretter en aftale i sin kalender.

Mere test af Google Calendar API

Det lykkedes mig at ændre på variablerne, osv. i Googles kode-eksempel, så der bliver hentet alle aftaler fra de sidste 24 timer. Jeg har endnu ikke testet, hvordan det fungerer med aftaler, der strækker sig over flere dage, eller heldags-aftaler, da mit fokus i første omgang vil være enkelte aftaler med et bestemt start- og sluttidspunkt på en given dag.
Jeg har fundet ud af at hvert "event" i Google Calendar har et event-ID, der kan trækkes ud. Dette vil kunne bruges til at registrere, om en aftale allerede er hentet ind fra kalenderen.

Datamodel og import-logik

Udover de ovennævnte funktioner, der tager udgangspunkt i hhv. Billys og Googles kode-eksempler, har jeg på nuværende tidspunkt defineret en række "modeller", som det hedder i Django - dvs. jeg har defineret første udgave af de entiteter og relationer, som udgør CalFaks datamodel.
Derudover har jeg skrevet kode, der kan lave kalender-aftaler fra Google Calendar om til Django model-objekter, og kode, der kan omsætte et Django model-objekt til de oplysninger, der kræves for at oprette en faktura i Billy.



Figuren viser CalFaks datamodel (i den endelige udgave.)

Jeg har en idé om at man kan lave det, jeg vil kalde en aftalerække. Det betyder at aftaler, der ligger lige efter hinanden enten på samme dag eller på hinanden følgende dage, kommer ind i samme aftalerække. En aftalerække vil så kunne blive faktureret sammenhængende. Som kunde ville jeg synes, at det var svært at modtage fem adskilte fakturaer, i stedet for een faktura, der dækker en periode på fem dage.
I datamodellen er en aftalerække repræsenteret ved entiteten “core_appointmentseries”, som svarer til Django-modellen “AppointmentSeries”.

Jeg skrev en indledende version af kode, der kan knytte flere kalender-aftaler sammen i en aftalerække. Jeg skrev derefter nogle unittest af import-logikken, der skulle dække disse scenarier, og de så ud til at fungere som de skulle.
Men da jeg i en anden forbindelse ændrede i visningen i Djangos admin interface, opdagede jeg en fejl i den logik, jeg havde sammensat, i forhold til oprettelse af aftalerækker. Det var frustrerende, at det ikke fungerede som tiltænkt (til trods for at unittests indikerede at det gjorde). Men det blev i det mindste hurtigt klart for mig, da jeg tydeligt kunne se i admin-interfacet, at der var noget galt, så det kunne rettes til, inden der var blevet kodet for meget oven på det.

Det viser også at test er gode, men der kan være fejl der ikke bliver fanget, hvis man ikke tænker det med i sine unittest. Man skal have en god blanding af logik og fantasi for at lave

test, der dækker alle scenarier, men mange gange må man nøjes med at have unittest af kernefunktionalitet eller kerne-scenarier. Det understreger måske også vigtigheden af at teste mange forskellige scenarier på mere manuel vis, for at vide om et system altid gør som man forventer.

Faktisk var der flere problemer med første udgave af "aftalerække-sammenknytningen." Første version tog kun højde for de scenarier, der opstår, hvis alle kalenderaftalerne kommer ind i samme import-kørsel, så der var noget, der skulle rettes til her.
Det gik også op for mig, at man nok vil finde det mere naturligt som bruger, hvis aftaler bliver knyttet til samme aftalerække, når de ligger på hinanden følgende dage, uanset om der er 25 eller 23 timer imellem dem. Så jeg ændrede logikken, således at alle aftaler, der ligger på foregående dag, skal regnes med i en aftalerække, så det er datoen der afgør om de skal knyttes sammen og ikke time-afstanden mellem dem.
Jeg måtte også indse, at første version antog, at der altid ville gå 24 timer mellem hver import, hvilket viste sig at være for skrøbeligt. Så det blev ændret, så det nu bliver logget (gemt i databasen), hvornår der sidst blev importeret, og det er så dette tidspunkt, der bliver brugt som nedre grænse ved næste import. Dette vil gøre import-funktionaliteten mere "stabil", da der ikke går noget tabt, hvis der er en dag, hvor der ikke bliver kørt nogen import. Det gør også, at der ikke mistes noget, hvis brugeren manuelt igangsætter en import (og ikke altid gør det på samme tidspunkt på dagen.)

Håndtering af fritekst

For at CalFak kan matche aftaler fra kalenderen med aftale-typer, skal der kodes en funktion, der kan omsætte en tekst fra kalenderaftalens notefelt og/eller overskrift til en aftale-type.

I første omgang tænkte jeg på at brugere nok gerne ville have at fx "Larsen" matcher med "larsen". Derfor prøvede jeg at finde en Python-funktion, der svarer til "equalsIgnoreCase" i Java. Jeg fandt frem til at Python har en funktion der hedder "casefold", der giver en ikke-case-sensitiv sammenligning af strenge (Ramuglia, 2023.)
Men det gik også hurtigt op for mig at det ville kræve nogle meget mere komplicerede match-funktioner, hvis der skulle tages højde for alle de forskellige fejl, der kan være i et brugerinput, da man også bliver nødt til at tage højde for slåfejl, stavefejl eller alternative stavemåder.
Normalt ville jeg foretrække at lave en form for dropdown til brugerinput, eller et søgefelt hvor man kan søge efter match i databasen, da dette ville eliminere tastefejl og lignende. Men i dette tilfælde kan jeg definere brugerinteraktionen eller interfacet, da det jo er brugerens egen kalender, hvor indtastningen sker. Det ville kræve en kalender der er bygget ind i CalFak, og det er ikke det målet er her.
I stedet skal CalFak selv kunne foretage et match, uden at involvere brugeren.

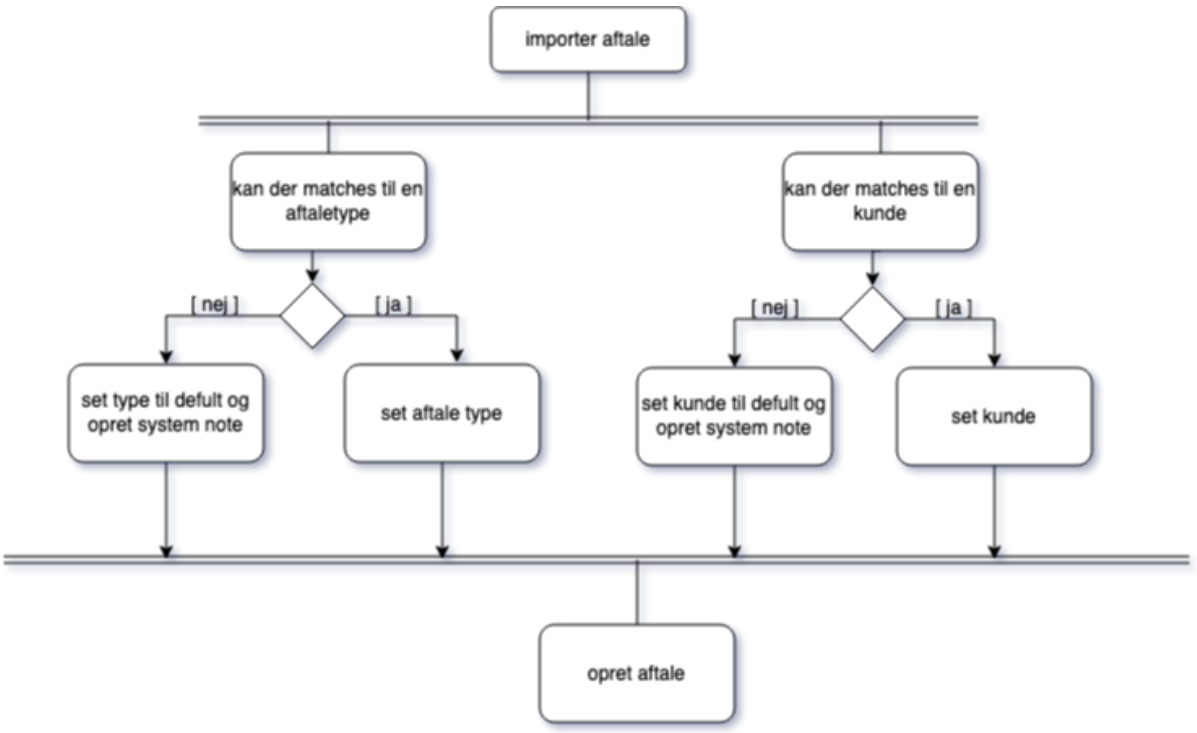
I mine undersøgelser af metoder til at matche brugerinput med aftaletyper, er jeg stødt på nogle forskellige biblioteker, der kan hjælpe med at lave det, der hedder "fuzzy matching" (Silva, 2022). Det virker lovende i forhold til mit matching-problem.
Når man fuzzy-matcher, får man ikke et "true" eller "false"-svar på et match, men derimod en procentværdi, der indikerer, hvor tæt to strenge er på at være ens.

Dette betyder, at en stavefejl eller et ekstra ord i en sætning ikke er en hindring for et match. Der er forskellige algoritmer til at lave disse match, og forskellige Python-biblioteker, der implementerer nogle af disse algoritmer. Jeg har valgt at benytte biblioteket RapidFuzz[12], der kombinerer flere metoder til at matche strenge. (Ali, 2022)
Når man fuzzy matcher, vil det være den streng, der har den højeste match-procent, der bliver "vinderen". For at man ikke skal få en "vinder", der kun matcher med få procent, kan man sætte et minimum for en match-procent. Kommer værdien ikke over dette, vil strengen ikke blive betragtet som et match.
Det er meget muligt at denne tærskelværdi skal ændres op og ned i forbindelse med evt. videreudvikling. Det er lettest at finde en god værdi for det specifikke program ved at teste frem og tilbage med forskellige minimum-procenter.

"Default" og aftaler, der ikke kan matches

Jeg har spekuleret over, hvad jeg skal stille op med de kalenderaftaler, der ikke kan matches - enten til en kunde, en aftale-type, eller begge dele.
Det er vigtigt at de ikke går tabt, da det er en potentiel indtægt eller bilag, der så går tabt.
Jeg har besluttet at CalFak indeholder en default-kunde og en default-aftaletype, der kan bruges til at samle disse op. Samtidig kan CalFak skrive en note, der bl.a. indeholder den tekst, der ikke kunne matches. Det er så tanken, at man som bruger af CalFak, kan matche disse manuelt. Om denne redigering skal ske i CalFak eller i regnskabssystemet, må besluttes på et senere tidspunkt.
Der er også mulighed for at der løbende skal forsøges at matche aftaler, der er markeret med default-værdier, op mod databasen, så hvis det manglende match skyldes at typen eller kunden ikke var oprettet i første omgang, vil matchet ske, når de er tilføjet.
Det er under alle omstændigheder vigtigt, at disse aftaler ikke går tabt, og at de ikke forhindrer andre aftaler i at blive overført til regnskabssystemet.

[12] https://rapidfuzz.github.io/RapidFuzz/



Figur, der viser forløbet i matching-logikken i form af et aktivitetsdiagram (Larman, C., 2004.)

Import og eksport af aftaler

For at give en bruger et indtryk af, hvad der er ved at blive hentet fra kalenderen, og hvad der er ved at blive sendt til regnskabssystemet, har jeg lavet to meget basale views (uden styling osv.)
De to sider viser et kort resume af hvad de har fundet, og som er klar til at komme ind fra kalenderen, dvs. alle aftaler der er afsluttet, siden der sidst blev hentet aftaler. Her vil det også være muligt at se hvor mange aftaler, der vil blive markeret med default-kunden eller default-aftaletypen. Det vil måske være en løsning at brugeren kan rette aftalerne allerede her, men det forudsætter så, at brugeren selv skal ind og hente aftalerne, i stedet for at der kører et script ved fx. midnat, som afvikler processen uden brugerinput.
Når man eksporterer til regnskabssystemet, vil alle aftalerækker, der er klar til fakturering, blive listet. En aftalerække er klar til fakturering, når der ikke har været aftaler inden for det seneste døgn, og dermed aftalerækken derfor betragtes som afsluttet. Når aftalerækkerne bliver overført til regnskabssystemet, vil de blive markeret som synkroniseret.

Forbindelse til E-conomic

For at CalFak kan være relevant for flere brugere, og for at få en bredere viden om hvordan regnskabssystemer og deres API'er adskiller sig fra hinanden, valgte jeg at afprøve forbindelse til et par forskellige regnskabssystemer.
Dette vil også give et klarere billede af, hvilke dele af koden, der kan genbruges umiddelbart, og hvilke dele, der skal rettes til.
For at kunne tilknytte en "hjemmelavet" app til en E-conomic konto, skal man lave en udviklerkonto, så man i alt har to konti - en til regnskabet, og en til udvikling. Når man har

dette, kan man få genereret et “app token” og en URL, som indehaveren af regnskabskontoen så skal trykke på for at forbinde de to.
Når man har gjort dette, kan man via API tilgå og oprette kunder, fakturaer, varer og meget andet. Der er forventeligt nok forskelle i navngivningen af variabler/attributter. I E-conomic hedder det ofte “number”, hvor det hedder “id” i Billy. E-conomic kundenumre og varenumre består af tal (integers), mens Billys kundenumre og varenumre er tilfældige strenge bestående af tal og bogstaver.
I E-conomic kan API-brugeren også selv bestemme id’er på kunder og varer - i modsætning til Billy, hvor det er autogenereret.
Der er også forskelle på hvordan de forskellige API-requests skal struktureres. Der er fx forskelle på, hvad der er obligatorisk, når der oprettes kunder og varer. Her kræver E-conomic en del flere obligatoriske oplysninger end Billys tilsvarende API.

Det er dog ikke større forskelle, end at det kan håndteres, og datamodellen i CalFak kan forblive uændret og genbruges. Men det kræver, at der kodes et API-modul i CalFak til hhv. Billy og E-conomic. Dette vil formentlig være gældende for hvert ekstra regnskabssystem.

Konklusioner af undersøgelserne indtil nu

Efter at have arbejdet med integrationerne til de to forskellige regnskabssystemer, er jeg nået frem til, at det er bedst at hente vare/produkt-navne såvel som kundeoplysninger ind fra regnskabssystemet, og vedligeholde CalFaks lister over aftaletyper og kunder ud fra dette.
Min første tanke var at CalFak selv skulle kunne oprette varer og kunder i regnskabssystemet, hvis de ikke allerede var til stede der. Men dette medfører en del nye udfordringer, i forhold til hvordan man undgår at dobbeltoprette en kunde eller en vare. En tanke var, at man kunne afgøre om noget skulle synkroniseres, ved at se om der allerede var et eksternt ID registreret i CalFak, men det ville kræve, at brugeren første gang skulle taste id’et ind i hånden.
Efter at have set Billys id’er (bestående af tilfældige bogstaver og tal) er min konklusion også, at det ikke er en mulighed for en bruger at taste disse id’er ind i CalFak. Det er for langt og for indviklet til at taste ind i hånden, for ikke at tale om, hvor svært det er at finde frem til i Billys brugerflade. Der vil simpelthen være for stor risiko for indtastningsfejl.
Der vil også være den udfordring, at oprettelsen af vare/aftaletyper kræver flere oplysninger, end der kan trækkes ud af en kalender. Det vil betyde, at en bruger skal tilføje disse oplysninger til CalFak for at oprette en vare, for at den så bliver oprettet i regnskabssystemet, som så skal sende et ID tilbage til CalFak. Dette betyder at der vil være en del unødvendig kommunikation frem og tilbage, sammenlignet med en arbejdsgang, hvor brugeren opretter varer/produkter i regnskabssystemet, og henter dem ind i CalFak, med alle de nødvendige oplysninger.
Derfor vil jeg mene at det bedste er at CalFak importerer kunder og varer fra regnskabssystemet, så det er der, brugeren opretter nye elementer og retter dem til. På denne måde vil der også kun være én kilde til disse data, og ét system, der har dette ansvar, hvorved risikoen for at systemerne ikke har ensartede oplysninger, minimeres.
Det betyder også at CalFak ikke behøver at opbevare nær så mange oplysninger om en kunde, som tidligere antaget.
Det skal så være muligt at importere ændringer og tilføjelser løbende, uden at en kunde eller vare blive duplikeret i CalFak.

Der vil være en udfordring med at brugeren måske ikke skriver det fulde navn på et produkt eller på en kunde i hver enkelt kalender-aftale.
Her vil brugeren så kunne benytte “alt_name”-feltet i databasen, som kan indeholde et “alternativt” kundenavn, der minder mere om det kundenavn, brugeren plejer at anvende på kalenderaftaler.
Hvis et lignende “alt_name”-felt bliver tilføjet på aftaletype (produkt) vil det også gøre matching af dette lettere ved import fra kalender.
Det vil så kræve at CalFak får et brugerinterface, hvor brugeren kan tilføje dette.
Der skal også findes en løsning på, hvordan man kan holde oplysningerne løbende opdaterede fra regnskabssystemerne til CalFak, uden at slette evt. alternative navne.

Det vil være en nødvendighed at tilføje et modul for hvert regnskabssystem, der skal integreres med i CalFak, både grundet forskelle i variabler, men først og fremmest på grund af forskellen i navngivning.
Lige nu håndteres forskellen på Billy og E-conomic med en streng og en if/else sætning. Om dette er den endelige version, eller om der skal skiftes til en form for enum, kommer an på hvor stort man vil lave CalFak. I øjeblikket, hvor der kun er to muligheder, er if/else fint.
Hvis man fortsætter med at tilføje moduler og gør importen af de forskellige funktioner fleksibel via if/else eller et enum, vil det også være lettere at tilføje og fjerne programmer hvis man beslutter sig for at udvide med flere kalendere eller regnskabssystem, eller vælger at fjerne nogle af systemerne. Det vigtige er, at alle regnskabssystem-moduler har samme "interface", dvs. har de samme funktioner, som kræver de samme argumenter, osv. Eksempelvis "export_invoice(appointmentseries)", som hedder det samme og tager samme argument i både Billy- og Economic-udgaven. Ligeledes med "get_products" og "get_customers".

Det er tydeligt undervejs i mine undersøgelser, at der skal testes mange forskellige scenarier i forbindelse med videreudviklingen, da der desværre er huller af varierende størrelse i de forskellige regnskabssystemers dokumentation.
Billy nævner fx ikke noget om deres id’er i deres dokumentation, men man kan se ud fra deres kodeeksempler, at de skal bruges - og nogle af id’erne kan man kun se, når man henter dem ud med API’et.
E-conomic har en meget omfattende dokumentation, men det er af og til svært at finde de detaljer, der er brug for. Fx hvilke felter der er obligatoriske. De nævner følgende obligatoriske variabler når man opretter fakturakladder: “currency, customer, date, layout, paymentTerms, recipient, recipient.name, recipient.vatZone[13]” Men når man opretter en fakturakladde via API’et, opdager man at der er flere obligatoriske felter, hvis man skal have fakturalinjer på fakturakladden.

En interessant detalje er at man i API’et godt kan oprette en linje på en faktura, uden at give linjen et navn/beskrivelse, men man kan ikke sende eller redigere linjen i E-conomics brugerflade, hvis der ikke er et navn/beskrivelse - da det er obligatorisk, ifølge formularen i brugerfladen.

[13] https://restdocs.e-conomic.com/#get-invoices-drafts-draftinvoicenumber

Import af kunder og produkter

Ved at ændre CalFaks design til at importere kunder og produkter, løser man en del problemer, men indfører nogle nye. Hvordan importerer og opdaterer man oplysninger i databasen, uden at overskrive eller dublere elementer?

For at løse dette, benytter jeg Djangos “bulk_create” metode (Django n.d.b) Metoden kan som udgangspunkt bruges til at oprette en liste af objekter i databasen, så længe de er af samme type.

Det viser sig, at hvis man benytter version 4.2 eller derover af Django, er der sket en tilføjelse til funktionaliteten i “bulk_create”, hvor udvikleren nu kan definere, hvad der skal ske ved konflikter. Dvs. hvis nogle af de objekter, man opretter, allerede findes i databasen, kan man opdatere dem, i stedet for at oprette dubletter.

Det ser således ud:

```
AppointmentType.objects.bulk_create(
    objs,
    update_conflicts=True,
    unique_fields=["product_id"],
    update_fields=["name", "price"],
)
```

Dette betyder, at to objekter med samme “product_id” betragtes som en konflikt, hvor der skal opdateres, i stedet for at oprette et nyt objekt. Når der opdateres, så må kun felterne “name” og “price” ændres. På den måde undgås det, at produkt-importen overskriver evt. alternative navne (“alt_name”) som brugeren evt. har sat på et “AppointmentType” objekt.

En lille udfordring i forbindelse med import af priser, var at det viste sig, at et produkts pris skulle hentes fra et andet endpoint, end selve listen af produkter. Så for hvert produkt er CalFak nødt til at “spørge” regnskabssystemet igen, for også at få prisen med. Dette viste at gælde både Billy og E-conomic.

Sammenligning af variabler i regnskabsprogrammer

Tanken med CalFak er, at det skal kunne integrere med forskellige regnskabssystemer. Ved at gøre det muligt at vælge mellem forskellige systemer, vil man kunne ramme en større kundegruppe.
I PoC’ets minimale brugerflade vælges regnskabssystemet via en "radio button”, der skal vælges, hver gang man vil importere, og to forskellige endpoint, når der skal eksporteres.
I forbindelse med et videre design skal der nok være en menu, hvor brugerens valgte regnskabssystem bliver gemt. Og det vil så være dette valg, der afgør, præcis hvordan CalFak sender og modtager data til/fra regnskabssystemet.

Jeg har sammenlignet variabler på faktura, fakturalinjer og kunder i hhv. Billy, E-conomic og Dinero. Dette bygger på konkret brug af API'erne i Billy og E-conomic, mens Dinero er kortlagt ud fra deres API-dokumentation. Dette har jeg gjort for at få et overblik over, hvordan de tre API'er adskiller sig fra hinanden, og hvor de er ens.
Der vil være forskellige navne for de samme variabler i de forskellige systemer, men hvis de reelt dækker over samme oplysning, vil de stå i samme kolonne. Fx hedder det "price" i det ene system, og "unit_price" i et andet - men begge variabler dækker over enhedsprisen uden moms for et produkt.
Denne sammenligning kortlægger også, hvilke data der er skrivbare, og hvilke der er obligatoriske, i forbindelse med oprettelse af fakturakladder og kunder i de tre systemer.

Symbolforklaring til variabel-skemaer

symbol	betydning
x	Obligatorisk
(x)	Valgfri, ellers default/autogenereret
(-)	Valgfri, udfyldes ikke
o	Kan ikke sættes, men autogenereres

Sammenligning af variabler i kundeoprettelse

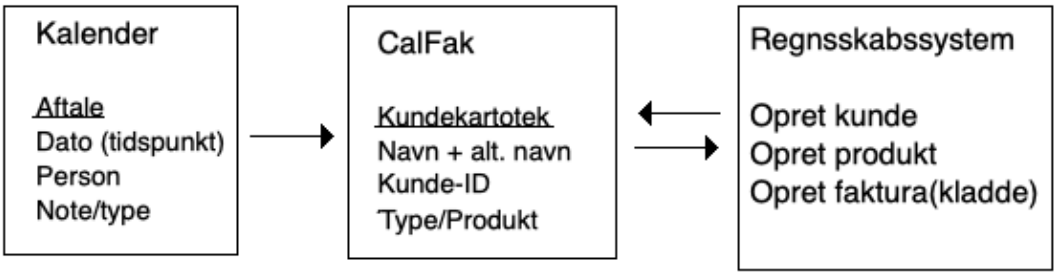
Felt	Billy	E-conomic	Dinero
Organisation-ID	x		x
Kunde-ID	o	x	
Navn	x	x	x
Gruppe		x	
Land	x	(-)	x
Email	(-)	(-)	
Telefon	(-)	(-)	
Adresse	(-)	(-)	
Betalingsbetingelse		(x)	
isPerson			x
isMember			x
useCvr			x

Sammenligning af variabler i fakturakladde-oprettelse

Felt	Billy	E-conomic	Dinero
Organisation-ID	x		x
Kunde-ID	x	x	(-)
Produkt-ID	x	x	*
Antal	(x)	x	x
ProduktPris	x	x	x
Dato	x	x	(x)
Valuta		x	(x)
Layout		x	(x)
Produktnavn/beskrivelse	(-)	x	*
Modtagernavn		x	(x)
Modtager momszone		x	(x)
Betalingsbetingelser		x	(x)
Fakturanummer	(x)	o	x
Produktlinje			x
Modkonto			x
Rabat			x
Linjetype			x
Unittype			x

* Disse to afhænger af hinanden, så hvis den ene angives, må den anden ikke udfyldes

Revideret arbejdsdeling mellem systemer



Kalenderen leverer kun læseadgang, og får derfor ikke input fra CalFak, men leverer data til CalFak.
CalFak kan læse aftaler i kalenderen, og matche disse op mod kunder og produkter/aftale typer i regnskabssystemet.
CalFak kan importere grundoplysninger om varer/produkter og kunder (ID og navn.) Der kan tilføjes et alternativt navn på disse. Alternativ-navnet gør det lettere at matche kalenderaftaler op mod kunder og varer. CalFak skal sende fakturakladder til regnskabsprogrammet, og CalFak registrerer, hvilke aftalerækker, der allerede er sendt til regnskabssystemet som fakturakladder.

Revideret app-opførsel

- Der kan løbende importeres kalenderaftaler. Aftaler kan først hentes, når deres sluttidspunkt er passeret. Der vil blive indhentet afsluttede aftaler for tidsrummet mellem seneste import og nu. (Med tiden skal importen formentlig automatiseres.)

- Hvis en aftale allerede er importeret i CalFak, vil den ikke blive importeret igen.

- CalFak skal opbevare oplysninger, der kan matche en aftale op mod en kundekonto.

- En aftale vil blive tilknyttet til en aftalerække og en kunde. Der kan være en eller flere aftaler i en aftalerække, men kun én kunde.

- En aftalerække er en periode, hvor der ligger en eller flere aftaler på hinanden følgende dage. Dvs. at en hvis en aftale ligger kl 13.00 den ene dag, og 14.00 dagen efter, vil disse blive knyttet sammen, selvom der er mere end 24 timer imellem dem.

- Når en aftalerække slutter, skal der oprettes en fakturakladde i regnskabssystemet, på den rette kunde og med det/de rette produkt(er).

- En aftalerække betragtes som ophørt, når der ikke har været kalenderaftaler på kunden i mere end et døgn.

- Der kan kun laves fakturaer for aftalerækker, der er overstået.

- Der kan løbende eksporteres aftalerækker til regnskabssystemet. En aftalerække kan eksporteres, når der er gået et døgn siden sidste aftale i rækken.

- Det er regnskabssystemet, der opretter, udsender og opbevarer fakturaer, CalFak skal kun sende de fornødne detaljer til videre til regnskabssystemet i form af en fakturakladde.

- CalFak registrerer, om en aftalerække er overført til regnskabssystemet, så den ikke overføres igen.

- CalFak registrerer at en aftalerække er sendt til regnskabssystemet eller ej, men ikke om den er faktureret.

- Der logges timestamp for både seneste kalender-import og regnskabs-eksport fra CalFak.

Teknologier

Jeg vil i den følgende tekst lave en kort gennemgang af nogle af de teknologier, jeg er stødt på i forbindelse med mine undersøgelser.

REST

Alle de systemer jeg har set på, benytter REST API'er.

REST (eller RESTful) (Wikipedia, 2023b) er en måde, hvor forskellige it-systemer kan udveksle information over HTTP (Hypertext Transfer Protocol) eller HTTPS (Hypertext Transfer Protocol Secure.) Det sker ved at klient-systemer kan anmode server-systemer om data, eller foretage ændringer i data, vha. HTTP-verberne GET, POST, PUT, PATCH, osv. REST er "stateless", hvilket betyder at serveren ikke opbevarer sessions-information. Derfor skal en klient sende al den information der skal bruges af serveren med, hver gang der sendes forespørgsler.
Selve dataudvekslingen sker oftest via JSON (Json.org, n.d.) som er et dataformat, der er understøttet i en lang række forskellige programmeringssprog og frameworks.

OAuth

Alle systemerne benytter også OAuth (OAuth.net, n.d.), der er en protokol-standard for autentifikation. Den benyttes til at give websteder og applikationer adgang til at dele

information, uden at dele adgangskode. Brugere kan dermed tillade, at et it-system får lov at dele data med et andet på deres vegne, uden at det sekundære system kender til brugerens adgangskode i det primære system. Det betyder også, at brugeren kan fjerne/trække et token tilbage, uden at skulle ændre adgangskoder. OAuth gør det muligt at definere, hvilke data der skal udstilles, så brugeren ikke ikke giver adgang til al sit data, men fx kun til en kalender, kontakter. Eller kun give læseadgang frem for fuld adgang.

OAuth giver også mulighed for at en bruger bliver informeret om hvilke data, en applikation beder om adgang til, fx placering eller kamera.

Det giver også mulighed for at sætte et udløbstidspunkt på et access token, så det skal fornyes med jævnlige mellemrum, for at øge sikkerheden.

Der er i OAuth flere forskellige former for access token, men alle systemerne, jeg undersøger, benytter "Bearer token" formen. (Det ser dog ud til at man kan opsætte Dinero til at benytte "JSON Web Tokens"[14].)

Mit ønske er, at det skal være let for en bruger at sætte CalFak op, uden at brugeren skal have forstand på it. Første gang, en bruger starter CalFak, skal de kunne kopiere et access token fra deres regnskabssystem eller kalender ind i en formular, og på den måde give CalFak adgang på deres vegne. Det betyder, at det skal gemmes på en sikker måde, hvor det ikke skal kunne tilgås af andre, da dette vil udgøre en sikkerhedsrisiko.

Bearer tokens

Dette er en type af access token, der tillader at alle med dette token kan forbinde til serveren, uden at skulle bekræfte deres identitet yderligere. Det er let at implementere, men er også mindre sikkert i sig selv.

Risikoen ved denne form for token er at det kan blive "aflyttet/stjålet", hvis det ikke bliver beskyttet korrekt.

Sikkerheden udgøres derfor primært af HTTPS, og andre sikkerhedsforanstaltninger, såsom kort levetid for tokens, begrænsninger af hvilke domæner, der kan bruge tokenet, mv. Man kan også lave begrænsninger af hvilke IP-adresser der kan benytte tokenet, eller indføre en to-faktor validering.

Det er vigtigt at huske på, at sikkerheden ikke er garanteret af OAuth i sig selv, da det i høj grad er implementationen, der afgør niveauet af sikkerhed.

[14] Forkortes JWT. Wikipedia, 2023c.

JSON Web Tokens (JWT)

Denne form for token kan fungere både som “bearer token” eller et “sender-constrained token.” Det tillader både enkle og mere avancerede authetfications-metoder, afhængigt af hvordan det anvendes.

JWT'er sendes som en del af en HTTP Authorization header, og fordi de er digitalt signeret, kan serveren verificere, at tokenet er ægte.

Et JWT indeholder tre dele: Header, Payload og Signature. Headeren indeholder typisk information om hvilken form for token der er tale om, samt hvilken hashing-algoritme der er brugt. “Payload” indeholder oplysninger om brugeren/enheden. Dette kaldes “claims” og kan omfatte forskellige brugerdefinerede data. Og indeholder ofte bruger-ID og oplysninger om, hvor længe et token er gyldigt, hvem der har udstedt det og lignende. “Signature” dannes ved at tage header og payload, omdanne dem via Base64, og derefter signere dem via en hemmelig nøgle eller et nøglepar. Dette sikrer, at en tredjepart ikke kan ændre i tokenet, uden at det vil blive opdaget.(Wikipedia, 2023c.)

HTTPS

HTTPS, eller Hypertext Transfer Protocol Secure, er en udvidelse af HTTP, der har til formål at sikre data under overførsel. Sikkerheden består i en kryptering af data, hvilket besværliggør aflytning af de data, der overføres mellem klient og server.(Wikipedia, 2023d.)
HTTPS kræver at en server er autentificeret med et SSL/TLS-certifikat, hvilket reducerer risikoen for at klienten kommunikerer med andre end den legitime server (et "man-in-the-middle"-angreb.)
Krypteringen sørger også for, at data ikke kan ændres under overførsel, uden at det opdages.

Kryptering i HTTPS sker via Transport Layer Security (TLS). Tidligere anvendte man Secure Sockets Layer (SSL). Men de er begge protokoller, der er designet til at sikre kommunikation over netværk. SSL er betragtet som forældet i dag, da TLS har en stærkere kryptering, har forbedret nøgleudveksling og hele “håndtrykket” er krypteret.
Men grundprincipperne for de to protokoller er ens, og TLS bliver stadig ofte omtalt som SSL. (cio.gov, n.d,)

Hvorledes skal applikationen designes, så den kan overholde GDPR, regnskabslovning, og behandle følsomme data?

Regnskabslovgivning

I disse år sker der en række ændringer i den danke regnskabslovgivning, som også kan påvirke CalFak.
Jeg har i første omgang valgt at CalFak ikke skal lave egentlige fakturaer, og ikke opbevare faktura og betalingsoplysninger. Det skal udelukkende sørge for, at de relevante aftaler bliver sendt fra kalender til det anvendte regnskabssystem.
Såvidt jeg kan læse mig frem til, betyder dette at CalFak er fritaget fra selv at skulle godkendes af Erhvervsstyrelsen som et digitalt regnskabssystem, da det må opfattes som et tredjepartssystem, så længe at al bogføring foregår i et samlet regnskabssystem (Billy, E-conomic, osv.):

> *"Anvender en virksomhed en tredjepartsapplikation til dele af sin bogføring, det kan fx være bogføring af salgstransaktioner i en webshop eller bogføring af projekt- eller medarbejderudgifter, kan virksomheden vælge at samle hele sin bogføring i et registreret digitalt standard bogføringssystem. I en sådan løsning skal virksomheder, der anvender tredjepartsapplikationer til dele af bogføringen, sikre at såvel bogførte transaktioner som bilag overføres fra tredjepartsapplikationen til det digitale standard bogføringssystem."*
> (Erhvervsstyrelsen, 2023b.)

Det er desuden et spørgsmål, om CalFak står for regnskabsgrundlaget. Hvis det er tilfældet, skal der kunne laves en form for bilag til de fakturaer der oprettes. Hvis CalFak betragtes som regnskabsgrundlag, skal det afklares, hvad der kræves af et bilag, og hvordan man knytter dette til en faktura.
For mig at se kan man argumentere både for og mod at CalFak udgør et regnskabsgrundlag. Derfor vil det nok være en god ide at få grundig vejledning omkring dette, fra en person, der bedre kan gennemskue juraen, hvis man beslutter at føre CalFak ud i livet.
Der er som nævnt ved at ske en del ændringer i bogføringsloven, der kommer til at påvirke små- og enkeltmandsvirksomheder i de kommende år - så noget rådgivning i forhold til hvilke krav, der skal leves op til, og hvilke godkendelser man skal have, vil være nødvendig.

GDPR

Der er til gengæld ingen tvivl om at der skal leves op til GDPR (General Data Protection Regulation).

GDPR har til formål at give borgere kontrol over deres personlige data, og omhandler derfor regler om hvad og hvordan en borgers data skal opbevares og må benyttes. (GDPR.dk, n.d.a. , Digitaliseringsstyrelsen, 2018)

GDPR foreskriver, at en virksomhed skal opbevare data sikkert, og ikke længere end nødvendigt for det formål, som data er blevet indsamlet til. Der skal også være et lovligt grundlag for at indsamle data.

Tanken med CalFak er, at det skal kunne matche kalenderaftaler med kundekonti. Dette vil kræve, at der opbevares kunde-oplysninger, såsom navn og kundenummer, i CalFak. Det vil ikke være svært at argumentere for, at disse oplysninger er nødvendige, for at systemet kan fungere.

Oplysninger som navn vil falde ind under kategorien “almindelige” personoplysninger, der ikke betragtes som særligt følsomme data. Men de skal stadig beskyttes, og CalFak skal leve op til standardkravene til databeskyttelse. Der skal også være et lovligt grundlag for at indsamle data, men at afregne kunder for tjenester, de har modtaget, må anses som lovligt grundlag.

Det skal dog stadig gøres klart over for brugerne, hvilke oplysninger der bliver indsamlet, og hvad oplysningerne bliver brugt til. Dette gøres typisk i en privatlivspolitik.

GDPR omfatter også “retten til at blive glemt”, dvs. at en bruger kan anmode om at deres data bliver slettet, hvis der ikke er en "lovmæssig" grund til at opbevare dem. (Der kan være tilfælde, hvor anden lovgivning bestemmer, hvor længe noget skal opbevares, fx i forbindelse med regnskab, færdselsloven eller noget helt tredje.)

Dette betyder at data ikke må opbevares længere, end der er nødvendigt, i forhold til det formål, som data er blevet indsamlet til. Det vil i denne sammenhæng betyde, at en kunde skal fjernes fra CalFak, når de ikke længere er aktive, eller når de specifikt beder om det.

Jeg kan ikke helt gennemskue, om dette kan foregå automatisk, da der kan gå kort eller lang tid mellem en kundes aftaler.

Man kan implementere en manuel sletning, men man vil så skulle finde en måde, hvor man sikrer at en kunde ikke bare bliver oprettet igen, når der importeres fra regnskabssystemet. Det er muligt at sætte kunder til at være inaktive/spærrede i regnskabssystemerne; man vil muligvis kunne tilføje et tjek ved importen, så det kun er aktive kunder, der bliver importeret. Men det kræver så, at en bruger husker at holde disse spærringer af kunder opdateret.

Når det skal være en mulighed at slette kundeoplysninger i databasen, er det vigtigt at den bliver sat korrekt op. Både, så det er muligt at slette en kunde, selvom der er oplysninger eller fakturalinjer tilknyttet til denne. Men også på en sådan måde, at det ikke har indflydelse på resten af databasen, hvis der bliver slettet elementer fra den.

I forhold til GDPR kan der opstå nogle problemer i forbindelse med noter på kalenderaftaler, i og med at det ikke at styre deres indhold. Hvis noterne henviser til fx sygdom, foreningsmedlemskaber, religion eller lignende, bevæger vi os ind i “særlige følsomme personoplysninger”. Hvor stor risikoen er for dette, kommer nok an på hvilken form for virksomhed, der benytter CalFak.

Denne type oplysninger er underlagt yderlig beskyttelse, og der skal være klare og specifikke grunde til at indsamle disse oplysninger. Der skal indhentes særlig tilladelse hos den registrerede, eller det skal være af vigtig almen interesse.

Det er måske et fortænkt eksempel, at der skulle kunne være den slags noter på aftaler. Men så skal det som minimum præciseres i databehandleraftalen, at sådanne noter ikke må optræde, så det ikke er hos CalFak, problemet ligger, hvis det skulle komme en sag ud af det.

Endnu et krav fra GDPR er, at det skal være muligt for en kunde at modtage en kopi af de oplysninger, som CalFak har gemt om dem. Dette skal også tænkes i et endeligt produkt.

Det er også et GDPR-krav, at en kunde kan få rettet deres oplysninger, hvis de ikke er korrekte. I dette tilfælde vil det være deres navn. Hvis et navn bliver opdateret i regnskabssystemet, vil det blive opdateret i CalFak. Så det er nok den nemmeste måde at imødekomme dette krav, men det skal dokumenteres overfor brugeren.

Hvis CalFak udvider sig væsentligt i funktionalitet i kommende iterationer, kan der være flere GDPR-paragraffer, der er relevante. Men i sin nuværende udgave er det de ovennævnte krav, der skal tænkes ind i udviklingen.

Tredjepartsudbyder

CalFak vil fungere som det, der hedder en tredjepartsudbyder eller en databehandler. (Datatilsynet, 2017)

Det vil sige at det er en tjeneste/applikation, der behandler data på vegne af den dataansvarlige, som vil være den virksomhed, som benytter CalFak. Dette fratager på ingen måde CalFak fra at leve op til reglerne omkring GDPR.

Der skal stadig være sikkerhed omkring data, og det skal rapporteres, hvis der sker sikkerhedsbrud. Og data må kun behandles i overensstemmelse med den dataansvarliges instruktioner. Dvs. CalFak må ikke bruge de data, virksomhederne stiller til rådighed, til andet end at lave fakturaer.

Det vil også være nødvendigt at lave en databehandleraftale for hver af de virksomheder, der benytter CalFak. Her vil det nok være en standardformular, der skal godkendes af virksomheden, når de første gang logger ind. Det vil være en meget enslydende aftale, hvor det er brugerens/virksomhedens navn, der ændrer sig. Dette medfører også, at der ikke skal sidde nogle manuelt og oprette disse aftaler, når en ny bruger kommer på.

En databehandleraftale skal opdateres løbende i forbindelse med videreudvikling, og det skal være muligt for brugeren/virksomheden at trække sit samtykke tilbage.

En af de ting, man skal være opmærksom på, når man laver en databehandleraftale er, at ansvarsområder skal defineres klart, og at der bliver rådgivet om brugen af systemet.

En databehandler har forpligtigelse til at implementere sikkerhedsforanstaltninger, og skal gøre det muligt for en dataansvarlig at overholde GDPR. Hvis applikationen ikke gør dette, vil ansvaret for overtrædelsen af reglerne lande hos databehandleren.

Har databehandler derimod levet op til sit ansvar, sikret mod overtrædelse og rådgivet omkring brugen af applikationen, vil ansvaret primært ligge hos dem, der har misbrugt systemet (lagt oplysninger ind, der overskrider GDPR.) Men det er vigtigt, at de forskellige ansvarsområder bliver overholdt.

Jeg er lidt overrasket over at flere regnskabssystemer benytter en simpel bearer token som access token, for det vil sige at alle med denne streng kan tilgå et system, oprette kunder og fakturaer.

Det vil derfor være vigtigt at beskytte disse oplysninger så sikkert som muligt. Det betyder at det er vigtigt, at HTTPS bliver sat korrekt op, da det i sidste ende er dette, der komme til at beskytte de data, der bliver sendt til og fra CalFak.

Det er vigtigt, at data bliver opbevaret sikkert, fordi det potentielt indeholder oplysninger omkring folks adresser, og alt efter service: personlige oplysninger, hvornår kunder er på ferie, og lignende.

Underdatabehandler

Der skal formentlig også laves en underdatabehandler-aftale. (DDPG.dk, n.d.b) Det vil være mellem CalFak og de servere, hvor data hostes og opbevares.
Hvis dette trin skal undgås, skal jeg selv hoste serveren, men det er urealistisk for mig at gøre dette, da det ligger uden for mine kompetencer. Det vil derfor være en klar sikkerhedsrisiko, da jeg ikke ved, hvordan jeg skal sætte en sådan server sikkert op, sørge for at den bliver vedligeholdt og lignende. Derfor ville jeg overlade denne del til eksperterne, der lever af at udbyde cloud-servere og hosting.
Men det vil så kræve en underdatabehandler-aftale, hvor de forskellige ansvarsområder også bliver klart defineret. Det vil formentlig også primært foregå via standardformularer, men det er vigtigt at holde sig for øje. Det kan have påvirkning på, hvem der har hvilket ansvar, hvis det skulle gå galt i forhold til lovgivningen.

Det kan også blive et problem, hvis man uvidende har sendt persondata videre til nogle, der sælger data, eller bruger oplysningerne til ikke-intenderede formål, uden at man er opmærksom på det. Det gælder for alle virksomheder, der levere tjenester eller produkter, der kan bruges af borgere i EU, at de skal overholde GDPR, uanset hvor i verden de er placeret. Det er under alle omstændigheder godt at have styr på, hvad ens underdatabehandler/underleverandør gør med de data, man lagrer, så det ikke er dig, der ender med ansvaret, hvis det viser sig at en underdatabehandler ikke lever op til sin del.

Konklusion på undersøgelsen (PoC)

Feasibility

Her vil jeg se på de elementer af en feasibility-analyse, som ikke allerede er behandlet i undersøgelsen ovenfor.

Vurdering af projektet sat i forhold til andre løsningsmuligheder

Jeg har kigget på, hvad der findes af eksisterende løsninger, der integrerer kalendere med regnskabssystemer (dvs. Billy, Dinero og E-conomic.)

Der findes flere lignende addons eller plugins til disse regnskabssystemer. De ligger typisk op til at kunden køber licens (der findes muligvis et par addons, der er gratis) og at kunden benytter dette programs brugerflade til at lave kundeaftaler, og som kalender. Disse systemer kategoriserer ofte sig selv om kundestyringssystemer eller timeregistreringssystemer.

CalFak skal ikke konkurrere med disse andre programmer direkte, men der er produkter der har lignende funktionalitet indbygget i deres programmer. Mange af de eksisterende addons vil nok også kunne opleves som “overkill” for mange små virksomheder, der måske ikke har brug for journaler, lagerstyring og lignende - i hvert fald ikke til at begynde med.

Spørgsmålet er, om CalFak skal tilbydes som et addon til de regnskabssystemer, som det kan integrere, eller om det skal sælges uden om disse. Fordelen ved at udbyde et addon, er at applikationen vil blive præsenteret på regnskabssystemernes egne sider, og dermed optræde i det forum, som den skal benyttes i. Programmet vil også blive verificeret af regnskabssystemet, og vil derfor fremstå som mere troværdigt for potentielle kunder. Det er heller ikke sikkert at det falder potentielle kunder ind, at søge efter løsninger uden for addon-universet.

Det lader til, at der ikke er så mange programmer, der er rettet mod de helt små virksomheder. Om det så er fordi, det ikke er et marked der kan betale sig at sælge til, eller fordi der ikke har været opmærksomhed på det, er svært for mig at sige.

Økonomisk og Finansiel feasibility

Man skal starte med at gøre sig klart, at en applikation som denne skal vedligeholdes løbende, selvom den er “færdigudviklet” og taget i brug.

Der sker hele tiden opdateringer i platform og frameworks, og der sker ændringer i lovgivningen. Sikkerhedskrav bliver ændret, og der kan ske ændringer i de systemer, der integreres med. Jo flere forskellige kalendere og regnskabssystemer, der implementeres i CalFak, desto flere API'er skal der holdes øje med.

Man skal også sikre at opdateringer af browsere og styresystemer ikke kommer til at skabe problemer for applikationen, ligesom en server kræver vedligeholdelse. Der skal indregnes udgifter til domæner og servere. Brug af Googles, Microsofts eller Apples API'er kan også være forbundet med udgifter.

Der vil formentlig også være en forventning fra kunder til en form for support til CalFak. Dette behøver ikke varetages af en udvikler, men det vil stadig være forbundet med en udgift.

Det vil derfor kræve en del kunder, før det vil kunne løbe rundt. Samtidig vil flere kunder også give flere udgifter. Dermed ikke sagt at man slet ikke kan komme hen til et sted, hvor det kan løbe rundt. Men det vil formentlig også kræve, at man investerer i promovering af produktet.

Man skal nok regne med et underskud i starten, og en evt. investor/samarbejdspartner skal være indforstået med, at der ikke kommer sorte tal på bundlinjen med det samme.

Et alternativ kunne være at indgå en samarbejdsaftale med en anden virksomhed, hvor CalFak kunne tilbydes som en tilføjelse af det de allerede har, fx en udvidelse til et af de regnskabssystemer, der ellers ville have CalFak som addon.

Da tanken med CalFak er, at brugere helst ikke skal bruge ret meget tid i brugerfladen, er reklamevejen nok ikke den mest oplagte måde at finansiere CalFak på. Specielt ikke, hvis hverken kalender eller regnskabssystem indeholder reklamer.

Det er også svært at argumentere for at CalFak skal koste et månedligt beløb, hvis dette ikke gælder det regnskabssystem, som brugeren anvender. Dette lader dog til at ændre sig, da en ny regnskabslov træder i kraft ved årsskiftet 2023/2024. Ændringerne betyder bl.a., at små virksomheder *skal* have et digitalt regnskabssystem. Det samme gælder personligt ejede virksomheder fra 2026. (Erhvervsstyrelsen, 2023a.)

Dette medfører at flere gratis programmer vil begynde at koste penge. Der vil stadig være gratis regnskabsprogrammer, men kun meget små planer vil være helt gratis. (Schjoldager, 2023a; Schjoldager, 2023b; Billy, n.d.) Dette åbner for muligheden at opkræve en form for betaling for CalFak. Man kan enten lave et abonnement med et fast månedligt beløb, uanset hvor meget man benytter CalFak. Det er også en mulighed at lave en forbrugsbaseret aftale, hvor kunden betaler for hvor mange gange, der overføres linjer til regnskabssystemet, eller hvor mange aftaler eller kontakter, som kunden har.

Ændringen i regnskabsloven kan til gengæld også have en betydning for markedet, da der kan komme mere fokus på de små virksomheder. Så det bliver vigtigt for CalFak at skille sig ud på markedet. Det skal dog også overvejes, at der umiddelbart ved årsskiftet nok vil være en del frustration i de små virksomheder over at deres tidligere gratis regnskabssystem nu koster penge (Schjoldager, 2022.) Så det er nok ikke der, man skal lave en lancering af CalFak - men det vil heller ikke være muligt i forhold til udvikling, da CalFak ikke ville kunne blive færdigt og have de relevante aftaler i hus, før tidligst i løbet af det kommende år.

Det vil formentlig godt kunne betale sig for en lille virksomhed i målgruppen at benytte et program som CalFak, hvis de har mere end en håndfuld aftaler med deres kunder, da ideen er at automatisere en manuel

arbejdsgang, der ellers forsinker cash flowet, hvilket er særligt kritisk for små virksomheder. Men det kan være, at den oprindelige tanke om CalFaks funktionalitet skal tilpasses lidt, så det ikke kun fokuserer på fakturering, men også på de andre ting, der kan være praktiske for en lille virksomhed at få ind i regnskabet, såsom ret til kørselsfradrag eller lignende.

Det er en mulighed at CalFak med tiden (med flere iterationer) begynder at bevæge sig væk fra den basale del, og have flere funktionaliteter, så det bliver et større program med flere funktioner, og dermed også mere "værd."

Jeg ville gerne kunne tilbyde produktet gratis til at hjælpe små virksomheder med at gøre deres hverdag lettere. Men at køre CalFak som et "hobbyprojekt" er kun holdbart med en lille håndfuld kunder, man kender godt, og som er indforståede med, at man ikke kommer til at smide alt man har i hænderne for at lave fejlrettelser eller opdateringer.

Teknisk feasibility

Der er dele af den krævede teknologi, som jeg ikke har forstand på - og enten skal sætte mig ind i, eller finde en der kan hjælpe mig med at sætte op. Der er fx nogle script der vil være nødvendige at sætte op hvis der skal køres automatisk synkronisering ved midnat, men dette er ikke en hindring i første omgang, da jeg ved at det er noget der er muligt at udvikle, måske ikke af mig, men så af en anden.

Der er flere emner, jeg skal bruge en andens assistance til, hvis CalFak bliver til noget. Dette er både indenfor kode, jura, og hosting. Men det er alle ting jeg ved kan lade sig gøre - jeg har bare ikke den nødvendige viden til at kunne gøre det selv.

Samlet vurdering af projektet

Min vurdering er at CalFak godt kan lade sig gøre rent teknisk, og også rent juridisk.

CalFak kan komme hen til et sted, hvor det er rentabelt, men der vil gå nogle år. Min vurdering er også, at hvis man vælger at gå videre med CalFak, vil kundegrundlaget vokse i de kommende år, da det bliver et krav for alle virksomheder at have et digitalt regnskabssystem, hvorved det potentielle marked bliver større.

Hvordan kan potentielle brugere inddrages?

Spørgeskemaundersøgelse

Jeg ville som nævnt gerne afklare en række usikkerheder, *før* jeg begyndte at inddrage potentielle brugere. Hen mod slutningen af undersøgelsen har jeg dog lavet en

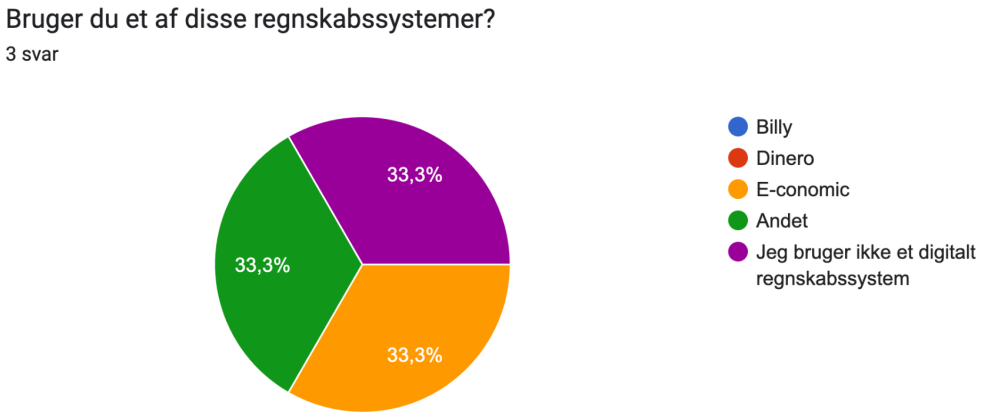
spørgeskemaundersøgelse, for at undersøge om CalFak overhovedet er relevant for den målgruppe, jeg har i tankerne.

Spørgeskemaet er ret kort, og indeholder disse spørgsmål:

1. Bruger du et af disse regnskabssystemer?
    - Billy
    - Dinero
    - E-conomic
    - Andet
    - Jeg bruger ikke et digitalt regnskabssystem
2. Bruger du en af disse online-kalendere?
    - Google Calendar
    - Office365/Outlook.com
    - iCloud/iCal (Apple)
    - Andet
    - Jeg bruger ikke en online-kalender
3. Bruger du i dag et særligt program til at holde styr på kundeaftaler, eller tid brugt på en kunde? Hvis ja, hvilket?
4. Er du enig eller uenig i flg.:
   "Jeg bruger meget tid på at fakturere kunder ud fra de kundeaftaler, jeg har i min kalender"? (1 er meget enig, og 5 er meget uenig.)
5. Ville det være en hjælp, hvis aftalerne automatisk blev overført fra din kalender til dit regnskabssystem? (1 = det vil klart være en hjælp, 5 = det gør ingen forskel.)

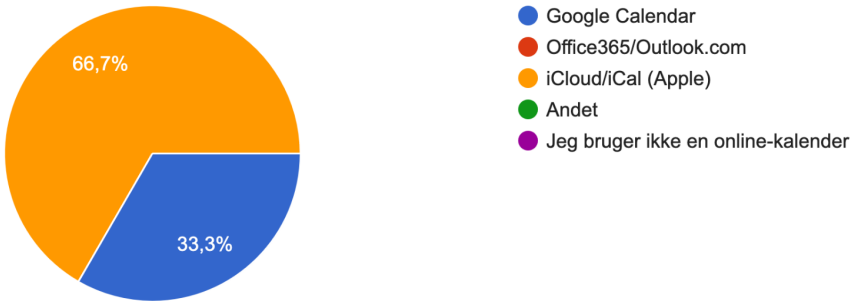
Dvs. jeg har forsøgt at afgøre, om personer i målgruppen anvender nogle af de systemer, som CalFak på sigt kan integrere med, samt om de genkender udfordringen med at overføre aftaler til et regnskabssystem.

Jeg har delt spørgeskemaet med forskellige små erhvervsdrivende i min omgangskreds, i alt ca. 10 personer. Desværre har kun 3 personer svaret:



Bruger du en af disse online-kalendere?

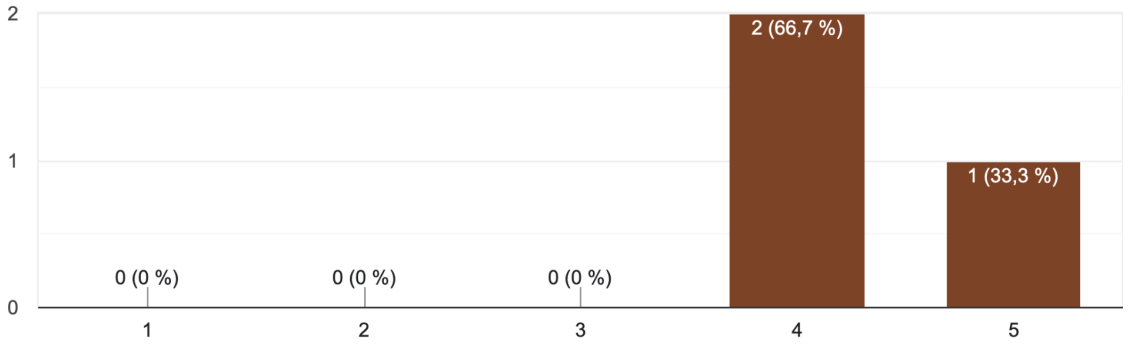
3 svar



Blandt de 3 besvarelser er der desværre kun 1, der bruger Google Calendar, og 1, der bruger E-conomic.

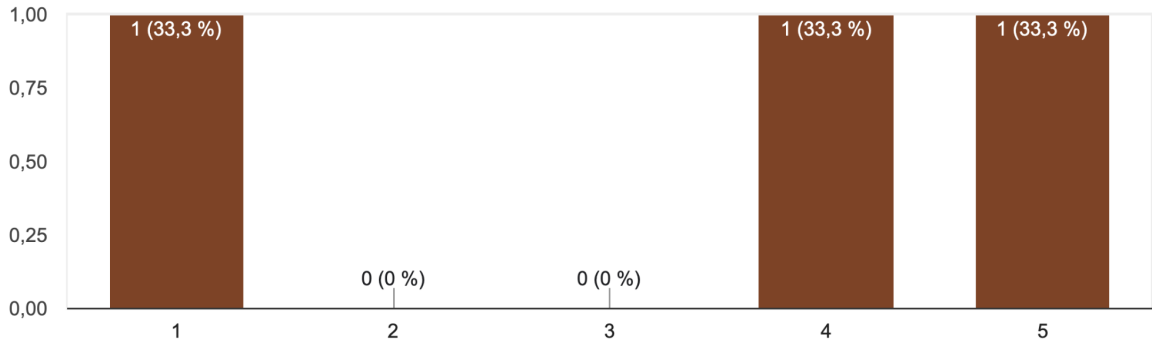
Er du enig eller uenig i flg.: "Jeg bruger meget tid på at fakturere kunder ud fra de kundeaftaler, jeg har i min kalender"? (1 er meget enig, og 5 er meget uenig.)

3 svar



Ville det være en hjælp, hvis aftalerne automatisk blev overført fra din kalender til dit regnskabssystem? (1 = det vil klart være en hjælp, 5 = det gør ingen forskel.)

3 svar



Dvs. svarene på de to sidste spørgsmål er ikke udelt positive, selvom der dog er 1 besvarelse, der tyder på at det klart ville være en hjælp, hvis aftaler automatisk blev overført til regnskabssystemet.

Tre besvarelser er dog meget lidt at konkludere på, og der er også muligheden for at spørgsmål er blevet misforstået. Når jeg har talt med respondenterne, har de været mere positive overfor ideen, samt interesserede i generelt at spare tid i forbindelse med regnskabsarbejde.

Spørgsmål til en potentiel fokusgruppe

Hvis valget er at gå videre med udviklingen af CalFak, vil det være en fordel, hvis man kan lave en fokusgruppe, eller finde et par virksomheder, der vil være interesserede i et samarbejde med henblik på udvikling af et MVP.
Ved at lave en fokusgruppe vil det være muligt at diskutere forskellige løsninger i forbindelse med design og opførsel af CalFak.
Det bedste vil være, hvis fokusgruppen kan afdække flere forskellige brugsscenarier eller virksomhedsmodeller, da dette vil kunne give det bedste grundlag for en applikation, der vil kunne benyttes af mange virksomheder.

Da CalFak i høj grad henvender sig til brugere uden særlig kendskab til it, vil det efter min opfattelse være bedre, når der er lavet nogle mockup og test, der kan diskuteres ud fra.

Hvis der findes en investor eller samarbejdspartner, skal denne også tages med på råd.

Undersøgelsen og arbejdet med proof-of-concept har været meget nyttigt, da det har afdækket en række spørgsmål til CalFaks adfærd, som ville være oplagte at diskutere med en eventuel fokusgruppe:

- Hvad skal der ske, hvis en aftale bliver rykket frem eller tilbage i en kalender?
- Hvad skal der ske når en aftale bliver aflyst? Skal der opkræves noget eller skal disse aftaler blot ignoreres?
- Hvordan skal aftaler, der ikke at matches med en kunde, håndteres? Skal der være et interface, hvor de kan rettes til, eller skal aftalerne overføres til regnskabssystemet så de kan rettes til her? Hvis disse aftaler skal rettes til i CalFak, hvornår skal dette så ske - ved import af aftaler, eller senere - og hvordan skal en bruger gøres opmærksom på, at der er aftaler der kræver behandling?
- Vil aftaler altid ligge i en separat kalender, og vil det være et problem, hvis det er et krav for at benytte CalFak?
- Skal det være muligt for en virksomhed at have flere kalendere tilknyttet? Hvis man nu er to eller tre ansatte i virksomheden?
- Hvordan skal man vælge, hvilke programmer man benytter, og hvordan skal/kan en bruger tilføje access tokens, uden at brugeren har det store kendskab til teknologi?
- Hvordan undgår man, at en kunde, der er blevet slettet i CalFak, ikke bliver genoprettet i CalFak ved en efterfølgende import af kunder? Markeres inaktive kunder på en bestemt måde i regnskabssystemet, eller vil det være en mulighed at gøre det? Eller er det noget der bliver nødt til at blive fundet en løsning på i CalFak?
- Skal det være muligt at sætte en dato for første dag for fakturering? Skal det være muligt at bestemme, hvor langt tilbage i tiden første import skal “kigge”, eller skal det være fra den dag brugeren tilføjer sine oplysninger til CalFak?
- Er der oplysninger, der er vigtige at få med fra en kalenderaftale, som ikke blive indhentet, som det er nu? Dette kan være adresser på aftaler, i hvilken kalender aftalen er oprettet i, eller lignende.

Der vil uundgåeligt komme flere ting der skal diskuteres, ligesom forskellige design også kan blive diskuteret.

På denne måde skulle man gerne kunne dække flest mulige behov.

Når man er nået til et produkt, der kan testes, vil det også være en fordel at have nogle brugere der kan beta-teste CalFak i deres dagligdag. På denne måde kan man få erfaringer i forhold til om der er opførsel der skal rettes til, ligesom der altid er ting der dukker op i daglig brug, som man ikke har forestillet sig ville ske, og derfor ikke har testet eller taget højde for.

Konklusion

Undersøgelsen og proof-of-concept har vist, at CalFak rent teknologisk godt kan lade sig gøre. Det er muligt at integrere med flere af de kalendere og regnskabssystemer, som vil være relevante. Og det er lykkedes at designe og programmere en matching-logik, som kan håndtere udfordringen med at forbinde kalenderaftaler med kunder og ydelser.
Jeg har undersøgt de juridiske vilkår for CalFak, og ikke identificeret problemer, der direkte sætter en stopper for projektet. Der er dog nogle spørgsmål i forhold til regnskabslovgivningen - er CalFak regnskabsgrundlag eller ej? - som jeg skal have en jurist eller revisor til at hjælpe med at svare på.
De ændringer, der sker i regnskabslovgivningen i disse år - hvor små virksomheder i løbet af de kommende år skal tage et digitalt regnskabssystem i brug - kan give CalFak en mulighed for at ramme et marked, som er ved at opstå netop nu.
Økonomisk vil det formentligt tage noget tid, før CalFak vil kunne løbe rundt, hvis det nogensinde rigtigt gør det. Det kræver måske at produktet vinkles anderledes, og at der kommunikeres til målgruppen på en anden måde. Alternativt kunne CalFak måske blive købt af et af de firmaer, der står bag regnskabssystemerne.

Glossary/ordliste

Ord	Aliases	I koden	Beskrivelse
Adresse			Kundens adresse.
Aftale		Appointment	En aftale i en brugers kalender. Indeholder aftale oplysninger.
Aftaleoplysninger			Personoplysninger Type Dato
Aftalerække		AppointmentSeries	Flere på hinanden følgende aftaler, enten samme dag eller flere dage i træk. Når der ikke har været en aftale på kunden i et døgn eller mere, anses aftalerækken for at være slut. En aftalerække kan bestå af en eller flere aftaler.
Alternativt navn	Alt. navn	“alt_name”	Hvis der er mere end et navn, der kan fremgå af en aftale, fx et sted, et kæledyr eller lignende - så kan et “alternativt navn” bruges til at matche de andre anvendte varianter.
Bogføringssystem	Regnskabssystem		Det system som brugeren har til at lave bogføring og oprette fakturaer, registrere betalinger osv
Bruger			Personen, der benytter CalFak. Har adgang til CalFak og interagerer med applikationen.
CalFak			Applikationen, der bliver undersøgt i denne opgave
Dato			Dato start (tidspunkt frivilligt) Dato slut (tidspunkt frivilligt)
Faktura			Den regning, der bliver lavet, når en aftalerække er afsluttet. Laves i regnskabssystemet.
Fakturakladde			En faktura, der er oprettet, men endnu ikke bogført, og heller ikke udsendt, betalt, osv. Oprettes af CalFak.
Fakturalinje			En eller flere linjer på en faktura eller -kladde, der bliver oprettet af CalFak, når en aftalerække er afsluttet.
Kalender			En brugers kalender, med kundeaftaler
Kunde		Customer	Kunder hos en bruger.

			En kunde har ikke adgang til CalFak, og interagerer ikke med applikationen, men er kunder hos dem, der gør det.
Kunde-ID			Det nummer, der identificerer en kunde i en brugers regnskabssystem.
Navn		“name”	Kundes navn
Note			
Person			Kunde
Personoplysninger			Navn alt. navn e-mail adresse
Produkt	aftaletype type vare	AppointmentType	Den service, der er udført, og som skal faktureres.
Regnskabssystem	Bogføringssystem		Det system som brugeren har til at lave bogføring og oprette fakturaer, registrere betalinger osv
Tidspunkt	Dato		

Litteraturliste

Ali, M., 2022. *Best Libraries for Fuzzy Matching In Python.* https://medium.com/codex/best-libraries-for-fuzzy-matching-in-python-cbb3e0ef87dd [Lokaliseret d. 14.12.2023]

Apple, n.d., *EventKit*. https://developer.apple.com/documentation/eventkit [Lokaliseret d. 11.10.2013]

Billy n.d.a, *Billy API v2 Documentation* https://www.billy.dk/api/ [Lokaliseret d. 8.11.2023]

Billy, n.d.b, *Code examples https://www.billy.dk/api/#code-examples* [Lokaliseret d. 8.11.2023]

Billy, n.d.c, *Priser. https://www.billy.dk/pris/* [Lokaliseret d. 8.11.2023]

BogholderiService, n.d. *Regnskabsprogram: Hvilket skal din virksomhed vælge?* https://bogholderiservice.dk/regnskabsprogram/ [Lokaliseret d. 8.11.2023]

Datatilsynet, Justitsministeret. 2017. *Vejledning om dataansvarlige og databehandlere* *https://www.datatilsynet.dk/Media/7/6/Dataansvarlige%20og%20databehandlere.pdf* [Lokaliseret d. 18.12.2023]

Digitaliseringsstyrelsen. 2018. *Hvad betyder databeskyttelses-forordningen for dig som borger? https://digst.dk/media/20233/gdpr-folder.pdf* [Lokaliseret d. 18.12.2023]

Django, n.d.a. *The Django admin site.* https://docs.djangoproject.com/en/5.0/ref/contrib/admin/ [Lokaliseret d. 13.11.2023]

Django, n.d.b, *bulk_create(),* https://docs.djangoproject.com/en/4.2/ref/models/querysets/#django.db.models.query.QuerySet.bulk_create [Lokaliseret d. 28.12.2023]

Django Girls, n.d.a. *What is Django Girls?* https://djangogirls.org/en/ [Lokaliseret d. 2.1.2024]

Django Girls, n.d.b. *Django installation.* https://tutorial.djangogirls.org/en/django_installation/ [Lokaliseret d. 13.11.2023]

Erhvervsstyrelsen, 2023a. *Digital bogføring: Tidsplan for virksomheder og systemudbydere.* https://erhvervsstyrelsen.dk/digital-bogfoering-tidsplan-virksomheder-og-systemudbydere [Lokaliseret d. 19.12.2023]

Erhvervsstyrelsen, 2023b. *Vejledning om bogføring.* https://erhvervsstyrelsen.dk/vejledning-om-bogforing [Lokaliseret d. 19.12.2023]

GDPR.dk,n.d.a. *De 7 principper* *https://gdpr.dk/persondataforordningen/de-7-principper/* [Lokaliseret d. 18.12.2023]

GDPR.dk, n.d.b. *Vejledning til databehandlere* https://gdpr.dk/persondataforordningen/vejledning-til-databehandlere/ [Lokaliseret d. 20.12.2023]

Gillis, A., Pratt, M., 2023. *Proof of concept (POC).* https://www.techtarget.com/searchcio/definition/proof-of-concept-POC [Lokaliseret d. 20.11.2023]

Google, n.d.a. *Events: list.* https://developers.google.com/calendar/api/v3/reference/events/list [Lokaliseret d. 13.11.2013]

Google, n.d.b. *Python quickstart.* https://developers.google.com/calendar/api/quickstart/python [Lokaliseret d. 13.11.2023]

Google, n.d.c. *Google Calendar API overview.* https://developers.google.com/calendar/api/guides/overview [Lokaliseret d. 9.11.2023]

Json.org, n.d. *Introducing JSON.* https://www.json.org/json-en.html [Lokaliseret d. 20.11.2023]

Larman, C., 2004. *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development*. 3. udgave. Upper Saddle River, NJ: Pearson Education, Inc.

Martins, J., 2022. *Timeboxing: The goal-oriented time management strategy.* https://asana.com/resources/what-is-timeboxing [Lokaliseret d. 18.11.2023]

Microsoft, 2022. *Calendar resource type*. https://learn.microsoft.com/en-us/graph/api/resources/calendar?view=graph-rest-1.0&preserve-view=true [Lokaliseret d. 10.11.2023]

Microsoft, 2023. *One Outlook REST API - your favorite platform - 400+ million users.* https://learn.microsoft.com/en-us/outlook/rest/#outlook-rest-api-via-microsoft-graph [Lokaliseret d. 10.11.2023]

Nordstrøm, T., Lønne, N., n.d. *12 Bedste Danske Regnskabsprogrammer i 2023* https://www.theme.dk/regnskabsprogram/ [Lokaliseret d. 8.11.2023]

OAuth.net, n.d. *OAuth 2.0*. https://oauth.net/2/ [Lokaliseret d. 9.11.2023]

Ramuglia, G., 2023. *Using Python to Compare Strings: Methods and Tips* https://ioflood.com/blog/using-python-to-compare-strings-methods-and-tips/ [Lokaliseret d. 4.12.2023]

Regnskabsprogrammerne.dk n.d. *Billigt / Gratis Regnskabsprogram til små virksomheder (enkeltmandsvirksomheder, foreninger og lignende)* https://regnskabsprogrammerne.dk/gratis-regnskabsprogram/ [Lokaliseret d. 9.11.2023]

Schjoldager, J. 2022. *Ny lov sender millionstor it-regning til tusindvis af små danske virksomheder.* https://www.version2.dk/artikel/ny-lov-sender-millionstor-it-regning-til-tusindvis-af-smaa-danske-virksomheder [Lokaliseret d. 28.11.2023]

Schjoldager, J., 2023a. *Gratis software forsvinder for tusindvis af virksomheder: Ny lov får leverandør til at hæve prisen.* https://www.version2.dk/artikel/gratis-software-forsvinder-tusindvis-af-virksomheder-ny-lov-faar-leverandoer-til-haeve-prisen [Lokaliseret d. 28.11.2023]

Schjoldager, J., 2023b. *Visma skruer prisen op på lovpligtigt it-system.* https://www.version2.dk/artikel/visma-skruer-prisen-op-paa-lovpligtigt-it-system [Lokaliseret d. 28.11.2023]

Silva, E., 2022. *What is Fuzzy Matching?* https://redis.com/blog/what-is-fuzzy-matching/ [Lokaliseret d. 14.12.2023]

Startupsvar.dk, n.d. *Regnskabsprogram.* https://startupsvar.dk/efter-start/regnskab-og-bogfoering/regnskabsprogram [Lokaliseret d. 9.11.2023]

Steenvinkel, C., 2022. *Regnskabsprogram: Hvilket er Danmarks bedste?* https://www.ageras.dk/blog/regnskabsprogram-hvilket-er-bedst [Lokaliseret d. 9.11.2023]

cio.gov, n.d, *Technical Guidelines* https://https.cio.gov/technical-guidelines/ [Lokaliseret d. 12.11.2023]

Trudeau, C. n.d. *Customize the Django Admin With Python.* https://realpython.com/customize-django-admin-python/ [Lokaliseret d. 13.11.2023]

Visma, n.d. *e-conomic vs Dinero.* https://www.e-conomic.dk/regnskabsprogram/sammenlign-regnskabsprogram/e-conomic-vs-dinero [Lokaliseret d. 9.11.2023]

Wikipedia, 2023a. *Minimum viable product.* https://en.wikipedia.org/wiki/Minimum_viable_product [Lokaliseret d. 20.11.2023]

Wikipedia, 2023b. *REST*. https://en.wikipedia.org/wiki/REST [Lokaliseret d. 19.11.2023]

Wikipedia, 2023c. *JSON Web Token.* https://en.wikipedia.org/wiki/JSON_Web_Token [Lokaliseret d. 12.11.2023]

Wikipedia, 2023d. *HTTPS* https://en.wikipedia.org/wiki/HTTPS [Lokaliseret d. 12.11.2023]