# Projektopgave forår 2016
# 02363 Videregående programmering

Gruppe nr: 1
Afleveringsfrist: Tirsdag d. 10/05-2016 kl. 23.59.

Denne rapport indeholder 15 sider inkl. denne side, med bilag.

s133781, Akkermans, Sandie, _____

s141653, Blumensaadt, Rasmus, _____

s145091, Javaid, Arbab, _____

s145095, Petersen, Simon, _____

# Indholdsfortegnelse

# 1: Forord og indledning

I dette projekt har vi fået til opgave, at udvikle en opdateret version af DTU's rejseafregningssystem TEM. Dette skal vi gøre vha. værktøjet GWT (Google Web Toolkit), som bruger asynkron kommunikation. Kravene bliver i denne omgang, at gøre designet af systemet mere intuitivt og selve systemet mere automatiseret. Desuden skal applikationen ikke indeholde overflødige knapper for at opdatere i backend systemet, da dette ikke er hensigtsmæssigt for brugeren. Projektet er testet igennem, for at teste for eventuelle fejlkilder, og skabe et stykke software der virker efter hensigterne.

# 2: Problemstilling og baggrund

Baggrunden for projektet er det nuværende system der er i brug på DTU til, at få godtgørelse for sin rejse, som brugerne mener er meget omstændigt og meget lidt intuitivt at benytte. Derfor får vi muligheden for, at lave en forbedret udgave på baggrunden af de ansattes ønsker.
Funktionaliteten i systemet skal stadig være den samme, og det vil stadig være bygget op omkring et workflow, hvor en ansat der har været på rejse i arbejdstiden, her opretter medarbejderen/brugeren en rejseafregnings sag med informationer omkring hvornår rejsen har fundet sted, hvor turen gik hen og hvorfor vedkommende har været afsted, hertil skal der være bilag for rejsen. Herefter skal en ansat med ledelsesansvar ind og tjekke oplysningerne og godkende, at den ansatte har været afsted og er berettiget til, at rejsen skal betales. Skulle sagen bliver godkendt skal en ansat fra økonomiafdelingen anvise beløbet til udbetaling.
Dette vil være det samme grundlag for vores system, som et krav for projektet vil vi dog bygge systemet op, som en moderne webapplikation med asynkrone kald til backend for at skabe en responsiv frontend, i stedet for den nuværende server pull-arkitektur, som brugerne også ønsker at få lavet om på.

## 2.1: Begrebsafklaring

**JDK**

Java Development Kit er en samling af forskellige værktøjer, der tillader udviklere, at dokumentere, skrive og teste Java-programmer.

**API**

Application Programming Interface er et sæt af protokoller, værktøjer og rutiner, der er til for, at kunne bygge software applikationer. Et API er et software komponent i form af dennes underliggende typer, outputs, inputs og operationer. Et API definerer funktionaliteter, som er uafhængige af deres respektive implementeringer, hvilket tillader definitioner og implementeringer, at variere uden at gå på kompromis med hinanden. API'et tillader altså forskellige software, at snakke med hinanden sådan så man kan benytte sig af forskellige funktioner i flere programmer.

**GUI**

Dette er et grafisk user interface, som præsenterer en brugervenlig mekanisme til at interagere med en applikation. Dette giver en applikation et specielt "look" og "feel". Denne består af GUI komponenter. Disse er objekter hvor en bruger interagerer med programmet med en form for input, såsom knapper eller tryk fra mus.

**Frame**

Et frame er komponenten som danner grundlag for GUIen. Det er selve vinduet hvori hele GUIen vil blive indsat.

**Panel**

Paneler er der til for at dele framet op. Det er for at skabe såkaldte større komponenter i framet, for at kunne have dynamiske så vidt som statiske komponenter i framet.

**Button**

En knap er en komponent som brugeren klikker på for at aktivere en specifik handling. Der er forskellige knapper, såsom toggle buttons og command buttons.

### Label

Dette er en etiket, som enten kan bruges som tekst eller et ikon.

### TextField

Dette er et tekstfelt der benyttes til at skrive informationer såsom dit brugernavn.

### PasswordField

Dette er en form for textfield, men i dette tilfælde ændrer den de indtastede informationer til "*" for at man ikke kan læse ens kode/password.

### Deck Layout

Dette er en layout metode der fungerer ligesom et sæt kort. Kun det øverste komponent, altså sidste aktuelle komponent, vises ad gangen.

### Dock Layout

Dette er en form for layout, som identificerer sig selv ud fra 5 punkter på framet. Enten north, south, east, west eller center. Dette gør det muligt at opsætte en GUI på en pæn og overskuelig måde.

### Event Handler

Disse benyttes til at operere modtagede inputs til programmet (events). I denne kontekst er events (begivenheder) et vigtigt element inden for applikation information fra et underliggende framework, som regel fra et grafisk user interface (GUI) input. På GUI siden omhandler events flere forskellige funktioner som påbegyndes ved f.eks.: tastetryk, mus aktivitet, handlings tryk , eller ved timer.

### EventBus

EventBus er et værktøj i GWT der gør det muligt at håndtere diverse brugergenererede events imellem de forskellige klasser i applikationen. Den giver en løsere kobling mellem klasserne, da det i dette tilfælde ikke er nødvendigt at have referencer til andre klasser når der skal ageres efter events. Så længe alle klasserne deler den samme EventBus, kan alle klasser sættes til at lytte efter events, som bliver sendt på EventBus'en.

**ERMaster**

ERMaster er en GUI editor til E/R-diagrammer. Det er et eclipse plug-in og benyttes til grafisk at lave E/R-diagrammer samt generere SQL-kode til oprettelsen af databasen.

I vores tilfælde har vi benyttet ERMaster til at Konstruere databasen for at have et overblik over relationer mellem databasens tabeller og hvilke entiteter de forskellige tabeller indeholder. Dette giver os en afklaring i forhold til om vores databasen er sat op korrekt.

## 2.2: Kravspecifikation

Til dette projekt har vi fået rimelig frie hænder af kunden. Der er dog specifikke ønsker fra brugernes side, som vi så vidt som muligt har fået dækket. Helt generelt har designet været noget som vi selv har bestemt, den ældre version af afregningssystemet har dog ligget som inspiration til dette. Funktionerne til vores færdigudviklede program har dog stadig været ret specifikke så disse, har vi så vidt som muligt fået implementeret.

### 2.2.1: Funktionelle krav

- En medarbejder skal kunne oprette en rejseafregning.
- Man skal kunne se de rejseafregninger man har i cirkulation.
- Man skal kunne se de rejseafregninger man selv skal godkende.
- Man skal kunne se de rejseafregninger man selv skal anvise.
- Man skal kunne søge i gamle dokumenter.

### 2.2.2: Non-funktionelle krav

- Applikationen skal udvikles i GWT.
- Applikationen skal være en AJAX-applikation med asynkrone kald til backend.
- Der skal designes en database i MySQL til at holde data.

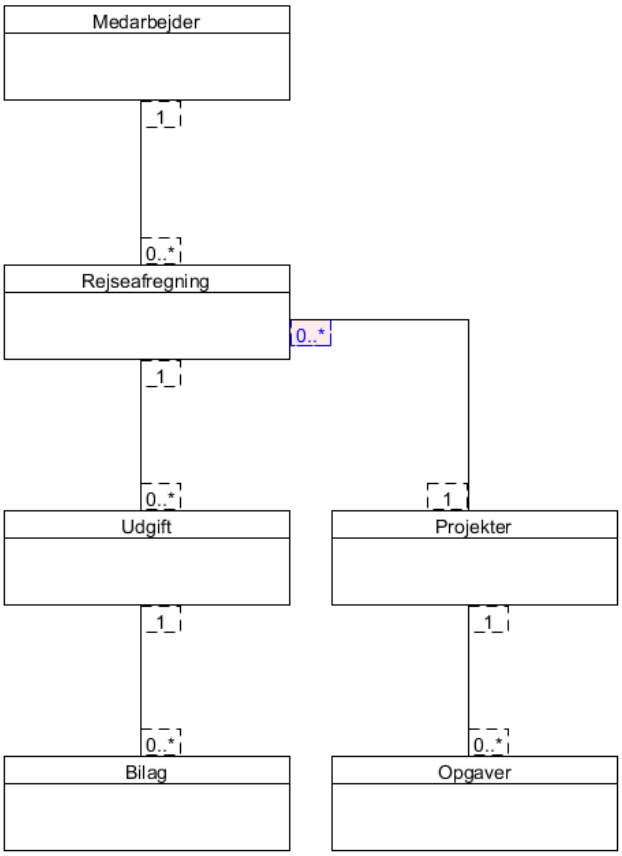
# 3: Analyse

Analyse er det første skridt imod udviklingen af det færdige produkt. Der laves herunder en masse analyseringer udfra den vision som vi har fået udleveret fra kunden. Ud fra disse analyseres der for krav og eventuelt andre ting man får udleveret. På denne måde kan vi komme frem til en kravspecifikation, aktører, og use cases. Dette gør os herefter i stand til at opstille en domænemodel, som vi kan bruge til at videreudvikle softwaren. Da vi allerede har fået udleveret en liste over aktører, use cases og ønsker fra brugerne blev udviklingen af de forskellige diagrammer en hel del mere overskueligt.
Vi har udviklet en prototype af DTUs rejseafregningssystem, TEM. Grundet dette har vi valgt ikke at tage alle funktioner med, fordi det simpelthen ville blive for stort et projekt. Derfor har vi valgt kun, at fokusere på den del, hvor man opretter en ny rejseafregning, hvori man skriver hvilket projekt man har været afsted med og hvor man har været henne, samt hvilke udgifter man har haft på sin rejse. Her har vi valgt ikke at lave en opdeling i udgifter, da dette ville blive mere omstændigt. Her har vi for så vidt muligt lagt vægt på, at brugervenligheden skulle være i fokus, samtidig med at vi har strøget nogle funktioner, da disse var mere omstændige, end vi havde tid til.

## 3.1: Use cases

For at opbygge et grundlag for hvilke funktioner vi ønsker at vores projekt skal indeholde er der nogle vigtige ting der spiller ind. Vi skal først og fremmest kunne holde fast på en del forskellige grundfunktioner i den ældre version af rejseafregningssystemet og vi skal desuden tage højde for hvad kundens og DTU's ønsker til det nye program er.
Basis for det forhenværende afregningssystem har haft funktioner der danner grundlag for hele vores projekt, og disse bliver naturligvis stadigvæk implementeret. De ønsker som brugerne har stillet bliver også implementeret.
Da vi allerede har fået udleveret Use Cases som vi har skulle tage i brug, har vi valgt ikke at lave flere.

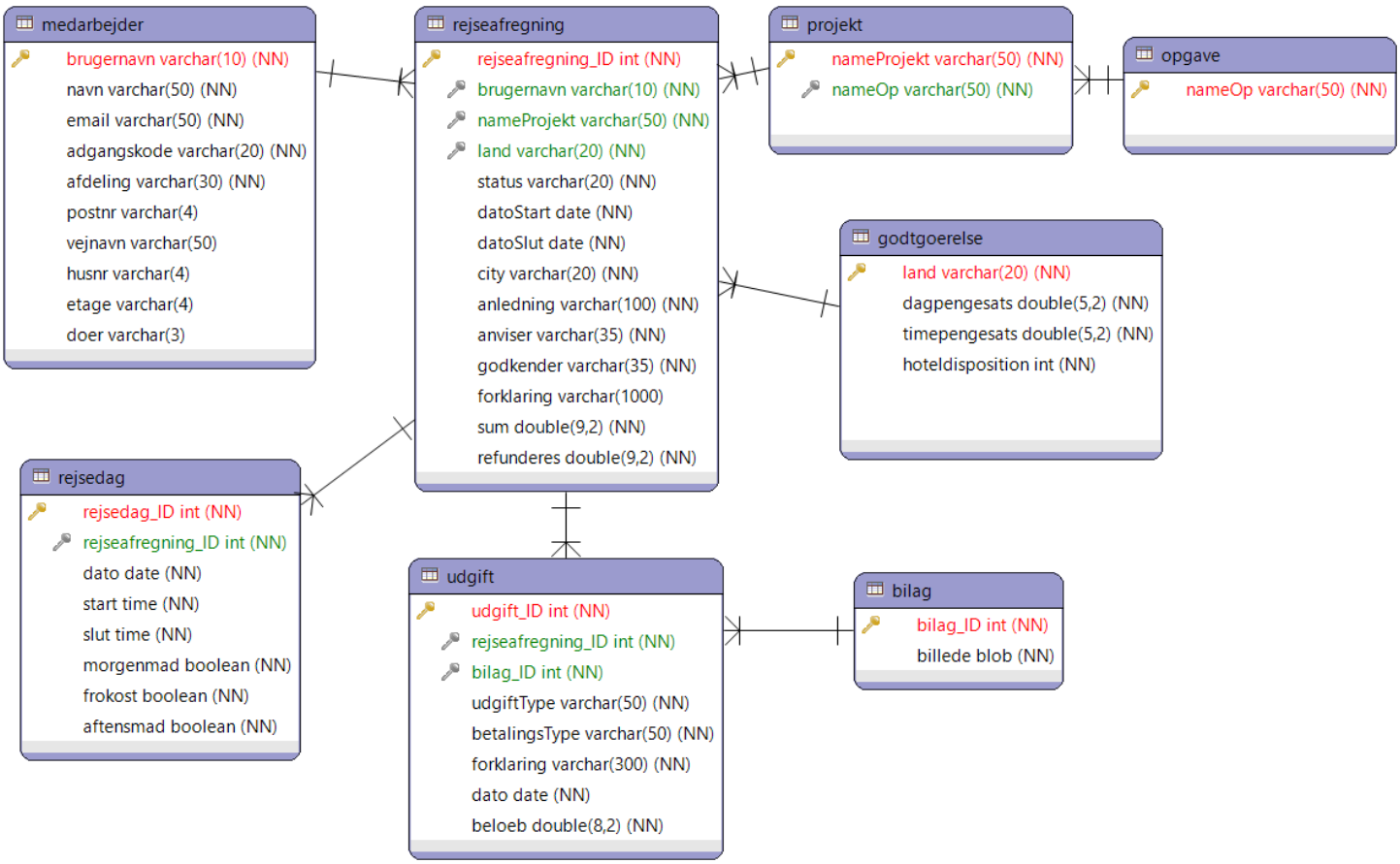
## 3.2: Domænemodel




*Figur 1 - Domænemodel*

Figur 1 viser vores domænemodel. Domænemodellen viser de forskellige aspekter af rejseafregningssystemet. Her kan vi se, hvordan de forskellige dele hænger sammen i systemet. I den forstand, at der er én medarbejder, som kan have 0 eller flere rejseafregninger og lave flere hvis nødvendigt. Ligeledes kan der være 0 til flere udgifter til én rejseafregning osv.

## 3.3: E/R-Diagram




*Figur 2 - E/R-Diagram*

Ved at opstille et E/R-diagram (Entity Relation Diagram), får vi et overblik over hvordan databasen ser ud, hvilke primær- og fremmednøgler vi har og hvordan databasen hænger sammen i forhold til tabellernes entiteter. Helt generelt viser diagrammet strukturen bag vores program. Her ser vi også at vores database er på 3. normalform.

Figur 2 viser, at der er 8 entiteter: Medarbejder, Rejsedag, Rejseafregning, Udgift, Projekt, Opgave, Bilag og Godtgoerelse. Disse entitteter er forbundet af forskellige relationer.
Her har vi  vores primærnøgler, som er den røde tekst i diagrammet: brugernavn, rejsedag_ID, rejseafregning_ID, nameProjekt, land, udgift_ID og bilag_ID. Hver entitet har en unik primærnøgle, som henviser tilbage til den bestemte entitet.
Ud over primærnøglerne har vi også fremmednøgler, disse kommer fra de andre entiteter i databasen, et eksempel kunne være ‘brugernavn’ under Rejseafregning, fremmed nøglerne bliver brugt til at se hvor nogle bestemt informationer kommer fra, som eksempelvis

informationerne fra Medarbejder tabellen bliver fundet via nøglen brugernavn, som så er en fremmednøgle i Rejseafregning.
en. Her ser vi eksempelvis at en medarbejder 'laver' en rejseafregning. Her kan vi også se, at en medarbejder kan lave flere rejseafregninger, men en rejseafregning skal have én medarbejder for at kunne være til. Ligeledes kan vi se, at enhver rejseafregning 'tilhører' et bestemt projekt og uden et projekt kunne det ikke være til og et projekt kan godt hører til flere rejseafregninger osv.

# 4: Design og Implementering

Under implementering er der lagt særligt vægt på, at vi følger 3-lags modellens principper. Hele vores kode er blevet delt op i pakker, for overskuelighedens skyld, og de er som følger:

- dtu.rejseafregning
- dtu.rejseafregning.client
- dtu.rejseafregning.client.events
- dtu.rejseafregning.client.logic
- dtu.rejseafregning.client.services
- dtu.rejseafregning.client.ui
- dtu.rejseafregning.client.ui.celltables
- dtu.rejseafregning.server.dal
- dtu.rejseafregning.shared

## 4.1: 3-lags modellen

Tre-lags-arkitektur bruges til, at inddele et program i tre lag der så vidt som muligt bliver holdt adskilt, så hele programmet bliver mere overskueligt.
De tre forskellige lag er delt op således:

- Grænsefladelaget
- Funktionalitetslaget
- Datalaget

### 4.1.1: Grænsefladelaget

Grænsefladelaget er det lag der er kendt for at være meget tæt på brugeren da det er det øverste lag der håndterer modtagelse og præsentation af data. Denne del er klientdelen i vores GWT-program.

### 4.1.2: Funktionalitetslaget

Funktionalitetslaget er det midterste lag, der håndterer udveksling af data mellem præsentationslaget og datalaget. Dette lag er afkoblet fra klientdelen så den asynkrone kommunikation kan finde sted.

### 4.1.3: Datalaget

Datalaget er det nederste lag er modsat det øverste lag kendt for at være tæt på computeren da dette lag opbevarer og håndterer data.

Vi har, som allerede nævnt, brugt 3-lags arkitekturen. Da dette gør det lettere for os og fremtidige læsere, at læse og forstå vores kode, da alle vores lag så vidt som muligt bliver holdt adskilt. Da dette projekt kun er en prototype til det endelige afregningssystem, vil arkitekturen i vores kode gøre det lettere at videreudvikle og vedligeholde vores system. Da vi har med et større projekt at gøre, har vi fået rykket hele logikken ind i vores Controller-klasser. Dette gør vores grænseflade klasser mere generelle og uafhængige, da de bliver reduceret til blot input og output enheder.

## 4.2: Klassediagram

Da klassediagrammet er for stort til, at kunne være læsbart i rapporten er den vedlagt i doc-mappen i vores projekt under navnet ‘Klassediagram.gif’. Hvis det er for småt på billedet, ligger diagrammet også inde i koden, under classdiagram.ucls.

I klassediagrammet fremgår opbygningen af vores Rejseafregningssystem. Det viser som sædvanlig associationer mellem klasserne samt der metoder og attributter. Alle UI-, logik-, DAO- og DTO-klasser fremgår af klassediagrammet. Det gør Event-klasser samt interfaces dog ikke. Disse er udeladt for at gøre diagrammet en smule simplere, da interfaces er afspejlet i DAO-klasser. Brugen af interfaces er beskrevet i afsnit 4.3.

## 4.3: Implementering

Implementeringen af vores Rejseafregningssystem er bygget ved brug af de generiske GWT-elementer. Brugergrænsefladen er så vidt muligt udviklet ved brug af UiBinder-klasser, da vi her har muligheden for at implementere brugergrænseflade ved brug af den grafiske design editor. Samtidig gør dette også vores View-klasser meget simplere da der ikke skal opsættes hele grænsefladen i Java-kode.
Som nævnt har vi opbygget vores applikation over 3-lags arkitekturen og al logik i applikationen er lagt i Controller-klasserne i logic-pakken. Her sker al kommunikation med backend. Kommunikationen mellem alle UI- og Controller-klasserne sker ved brug af EventBus. Ved brug af EventBus slipper vi for at skulle have referencer til alle de andre klasser, hvilket gør koden en del kortere. I stedet får vi dog en masse Event-klasser som vi selv har defineret for de events der er specifikke for vores applikation. En liste over Event-klasserne og deres formål kan ses i bilag 1.
Kommunikation til backend sker via en række Servlets som er bygget op omkring GWT's standard for Remote Procedure Calls, hvor der laves to interfaces. Det ene definerer de metoder som Servlet-klassen skal have og det andet definerer de samme metoder men med et ekstra argument, et AsyncCallback-objekt. Herefter implementeres metoder til at tilgå tabeller i databasen i en Servlet-klasse. Når Servlet-klasserne så skal benyttes på klient-siden skal der instantieres et AsyncCallback-objekt, som indeholder to metoder til hvad der skal ske når det asynkrone backend-kald er afsluttet, enten med en fejl eller med et succesfuldt resultat.

## 4.4: Synkron og Asynkron kommunikation

Synkron kommunikation foregår mellem to parter der kommunikerer på samme tid. Et 'en-til-en' eksempel på dette kunne være en telefonsamtale mellem to individer.  Tv og radio er eksempler på 'en-til-mange' mens online-chatrum er eksemplet på den synkrone 'mange-til-mange' kommunikation.

Modsat ovenstående er asynkron kommunikation nemlig ikke noget hvor begge parter skal være til stede. Sms'er kunne i dette tilfælde være et eksempel på 'en-til-en', hjemmesider er 'en-til-mange' mens eksemplet på 'mange-til-mange' kunne være vores projekt i dette fag.

Forskellen på synkron og asynkron kommunikation handler i sidste ende om tid. Med dette menes, at den ene form for kommunikation sker i realtid mens den anden er forskudt.

Når forskellen mellem den synkrone og asynkrone kommunikation skal findes, handler det ikke om hvilken måde der er bedst. Det drejer sig om, at begge måder er tilegnet forskellige ting. Det vil som eksempel være meget nyttigt, at kunne finde gamle rejseafregninger frem hvis man får behov for dette, mens det også vil være praktisk, at kunne spille et online spil der var turbaseret.

Vores projekt i dette fag skal gøre brug af asynkron kommunikation da vi har med en online tjeneste at gøre, hvor i der kan gemmes og hentes ældre rejseafregninger. Den asynkrone kommunikation er derfor tilegnet et projekt som vores der bunder ud i fildeling.

## 5: Idriftsættelse

Vi har udarbejdet Rejseafregningssystem i forbindelse med kurset 02342 Distribuerede Systemer. I dette kursus har der været fokus på at skabe et system med en central server og som er bygget op omkring en række forskellige webservices. Tilgangen til disse webservices er ikke færdiggjort ved deadline i dette kursus. Derimod har vi en MySQL-database på Amazon Web Services som kontaktes fra den lokale enhed.

Til databasen: rejseafregningdb.cdkqdugmixv4.us-west-2.rds.amazonaws.com

## 6: Test

| Testrapport | |
|---|---|
| Test case 1: Opret Ny Rejseafregning | Dato: 9/5-2016 |
| Precondition:<br>1. Test brugeren skal være logget ind i systemet. | Ok |
| Test:<br>1. Trykker på opret ny rejseafregning.<br>2. Udfylder felterne på første side.<br>3. Trykker på gem og næste.<br>4. Udfylder felterne på anden side.<br>5. Trykker på afslut.<br>6. Ser på tredje side at rejseafregningen er gemt. | Ok |
| Postcondition:<br>1. Rejseafregningen er gemt i databasen og man kan se den. | Ok |

Her ser vi en test case over ‘opret en ny rejseafregning’. Test rapporten beskriver hvordan forløbet foregår og om det går som vi forventer. Vi har nogle preconditions, som beskriver hvad brugeren skal opfylde inden han/hun kan oprette en ny rejseafregning.
Herefter skal brugeren trin for trin følge testen. Til sidst er der nogle postcondition, som gerne skulle stemme overens med det forventede svar, som er, at rejseafregningen skal være gemt i databasen.

## 7: Brugervejledning

For at kunne kører vores program, skal man compile vores kode i Eclipse. Dette sker ved, at man compiler koden som en Web Application (GWT Super Dev Mode). Herefter vil man få anvist en URL der fører til en webside hvor på man får adgang til vores rejseafregningssystem.
Der skal være installeret JDK 1.6 og GWT 2.6.1 plugin for, at kunne compile vores kode.
Desuden skal der være adgang til internet da forbindelsen til serveren skal kunne oprettes før man kan komme i gang med vores rejseafregningssystem.
Det er muligt at logge ind med brugernavnet ‘MANY’ og adgangskode ‘test’.

## 8: Konklusion

Da dette projekt gik ud på, at få lavet en prototype til det endelige afregningssystem, kan vi som gruppe konkludere, at vores projekt er vellykket.
Hvis der havde været mere tid, kunne søgefunktioner til projekter og opgaver have været en god ting, at have med. Hertil ville det have været rart, hvis vi kunne få oprettet og gemt/hentet bilag til rejseafregningen. Det var dog mere omstændigt at gemme sådan en fil (enten et billede eller en pdf), end regnet med. Det ville også have været rart, hvis vi kunne have delt udgifterne op (Opdeling af kontostreng), som var et ønske fra brugerne.
Af andre mindre tilføjelser kunne være at under oprettelse af rejseafregning kan man kun vælge dage der ligger indenfor det interval man har angivet som at have været afsted. Under oprettelsen af rejseafregning bliver der heller ikke taget højde for halve rejsedage. En side hvor man kan se detaljer for den enkelte rejseafregning, kunne også være en god ting, men dette var der ikke tid til.
Af cirkulæret vedrørende dagpenge- og hoteldispositionssatser fremgår det ikke tydeligt hvad hoteldispositionssatsen er for andre lande end de få angivne, derfor har vi valgt at benytte satsen i Danmark til disse.

En yderligere udvidelse kunne være en funktion der udfylder checkbokse for gratis måltider på samtlige rejsedage.

Vi har fået udviklet vores AJAX-applikation i GWT og fået oprettet en database i MySQL der holder vores data. På samme tid er systemet udviklet således, at der er asynkrone kald til backend.

# 9: Bilag

Bilag 1: Events

Spreadsheet med events kan findes i følgende Google Docs dokument:

https://docs.google.com/spreadsheets/d/16riUjyh1C5csg7fqd-4MIzcVIX728Sew0LMLBvyHelA/edit

# 10: Litteraturliste


- Lewis, Loftus, 2014, *Java Software Solutions*, Pearson, 7. udgave
- http://www.tutorialspoint.com/gwt/


# 11: Klon repository

Vores repository kan findes på: https://github.com/simonpetersen/Rejseafregning