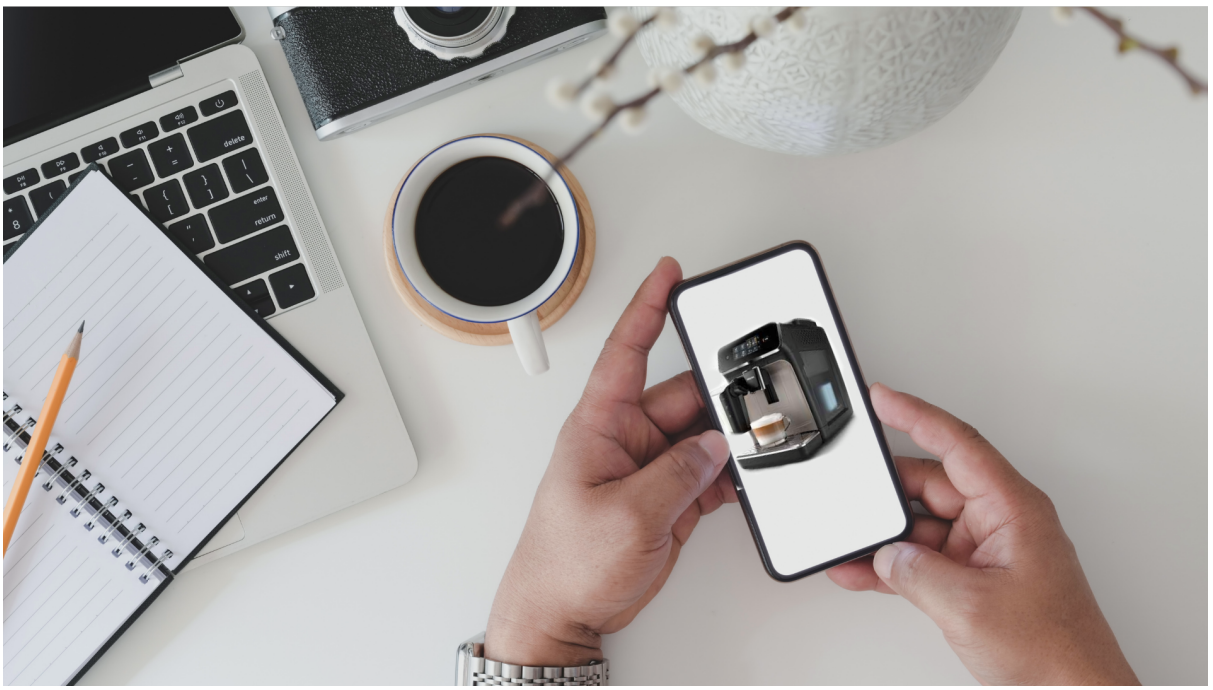
CASE: A COFFEE MACHINE AT KEA



Afleveringsdato: Den 11/09-2021

Uddannelse: BE-IT - KEA

Hold nr.: BE-IT A21 DA1

Gruppe nr.: 10

Udarbejdet af:

Danny Rasmussen

Emil Bystrup Lenschau

Jonas Hvid Nielsen

Magnus Christophersen

Maria Lysdal Buur

Mathias Villa Fonseca

Vejledere:

Jamshid Eftekhari

Muniba Talha

Indledning:

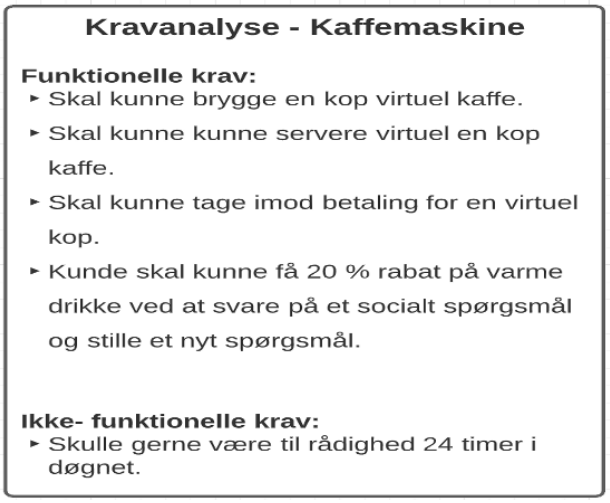
Det menes, at de fleste innovationer i verden skete, efter at folk talte om vanskelige emner ved kaffemaskinen. For en kop kaffe er en god måde at 'bryde isen' på og starte en samtale. Her på KEA har vi dog alligevel ikke sådan et socialt midtpunkt for studerende. Så i stedet for at sidde på vores hænder, udarbejder vi et socialt midtpunkt for KEA, hvilket vil ske via en virtuel kaffemaskine. En der passer til vores behov og kan brygge, servere og opkræve os. Det vil øge moralen, stemningen og samtidig være en real money maker!

I vores rapport kommer vi med vores bud på en virtuel kaffemaskine, og hvordan denne maskine kan bruges til at opføre et socialt midtpunkt og et sted hvor folk lære hinanden bedre at kende. Vi har gjort det sådan, at man med sit KEA-login kan logge på vores virtuelle kaffemaskine og købe sig en drik. Det sociale element består af, at man kan få 20% rabat, hvis man vælger at svare på et spørgsmål fra en tidligere kunde. Derefter skal man selv stille et nyt spørgsmål til den næste kunde, for at få en samtale i gang. Vi tror på at spørgsmålene kan bidrage til det sociale på KEA, da de vil fungere som en ægte "icebreaker". Dette vil også blive gennemgået i vores rapport.

Systems Development

Requirements

Kravanalyse:



Vi har udarbejdet en kravanalyse for vores kaffemaskine, da dette giver et indblik i hvad vores kaffemaskine skal kunne og hvilke krav vi har til den. Herunder har vi udarbejdet kravspecifikationer, med både *funktionelle krav* og *ikke-funktionelle krav.*

Disse bestemte funktionelle/ikke funktionelle krav kan ses her. Kravspecifikationerne revideres gennem hele forløbet, i den første fase har vi blot fastlagt de krav med størst risici (Risici dreven udvikling). Grunden til vi har udarbejdet en kravanalyse, er at det giver os et overblik over vores krav til vores system, og det er med til at kunne udarbejde forskellige use cases.

UML diagrams

Use cases:

Vi har i vores rapport udarbejdet forskellige use cases ud fra vores kravspecifikationer. Vi har haft en iterativ tilgang, hvilket medfører at nogle use cases ikke nødvendigvis er fra vores kravspecifikation.

Use case 1 beskriver handling omkring Login for aktøren, som er studerende og lærer på KEA, til beskrivelsen er der nogen succes scenarie og udvidelser til beskrivelsen af handlingen.

Use case 2 omhandler beskrivelsen af når man skal betale for sin drik, hertil er der forskellige succes scenarier, såsom at aktøren kan vælge om han/hun gerne vil have rabat på sin ordre ved at besvare et spørgsmål fra en tidligere kunde og stille et nyt spørgsmål efter.

Use case 3 beskriver hvordan hele funktionen med at vælge sin drik fungerer. Når de skal vælge deres drik kan de se ingredienser, når denne er blevet valgt, brygger maskinen drikken, og brugere får derefter at vide at drikken er færdig.

Use case 4 beskriver et mere ekstern procedure i hvorledes manageren redigere i sine drinks, eller fjerne drinks fra menu'en. Først vælger manageren om drikken skal redigeres eller slettes, derefter vælges hvilken drik. Ellers kan manageren enten tilføje drik eller fjerne drik fra menu.

Activity:

Activity diagrammet viser hvordan hele processen udfolder sig, når lærer/elev bruger app'en. Man følger brugeren igennem "programmet" ved at starte med login, hvis ens login fejler, starter man fra starten af. Når man succesfuldt har logget ind, kan man vælge om man vil svare på et spørgsmål, eller stille et spørgsmål. Hvis brugeren svarer "ja" skal brugeren først svare på spørgsmålet fra den tidligere bruger, og derefter skrive et nyt til den næste bruger. Ved at svare/spørge, sparer brugeren 20% på deres drik. Alle drikke koster det sammen, dog skal de vælge i mellem en stor eller en lille størrelse. Brugeren betaler før de vælger hvilken slags drik de ønsker. Hvis brugeren svarer "Nej" skal de ikke igennem spørgsmålsrunden, men betaler så fuld pris på deres drik. De vælger størrelse på drik, og betaler også før de

vælger hvilken drik. Hvis betalingen fejler skal brugeren igennem betalingen igen. Når betaling er gået igennem, vælger brugeren hvilken drik, når den er færdig, får kunden besked, og logger derefter ud.

Sequence diagram:
Vi har valgt at gøre brug af et sequence diagram, for at gøre det nemt og forståeligt at give et indblik i hvordan vores virtuelle kaffemaskine fungerer. Forskellen på et sequence diagram og et system sequence diagram og vigtigheden i at vi har inkluderet et i opgaven, er at der ikke er aktør i sequence diagrammet, så frem for at det er et overblik over hvordan aktøren opnår slutresultatet, er det hvordan objekterne interagerer med hinanden for at slutresultatet opnås.

System Sequence diagrams:
Vi har valgt at lave et system sequence diagram til hver af vores use cases. Årsagen til at vi lavede fire diagrammer var for at danne os et overblik over hvad der skulle ske i vores kode, og i hvilken rækkefølge de forskellige sekvenser kommer i. I vores login diagram, vises der de skridt, som vi går i for at logge ind på vores kaffemaskine. Vores system er lavet således at man skal logge ind med sin KEA-mail. Da kunden ikke er oprettet til vores virtuelle kaffemaskine, skal kunden først oprette sig. Derefter skal login verificeres og valideres, hvis dette ikke stemmer overens med den bruger der er oprettet, skal man prøve igen. Denne proces fortsættes indtil man indtaster de korrekte oplysninger.

I næste diagram skal vi betale og vælge størrelse på vores drik. Der er her forskellige muligheder blandt andet, om man vil have rabat, ved at svare på et spørgsmål fra en tidligere bruger. Hvis ja, så stilles der et spørgsmål, og der skal stilles et spørgsmål til den næste kunde. Hvis der vælges nej, går man direkte til valg af størrelse. Her kan man vælge stor eller lille drik, prisen gives så derefter afhængigt af om du har valgt at svare og stille spørgsmål. Herefter betales der for drikken. Drikken skal nu vælges ud fra vores menu, når drikken er valgt, bliver den brygget og er klar til at nydes. Vores sidste use case og system sequence diagram er for manageren(gruppen). Vi kan rette i menuen og tilføje drik til menuen.

Class diagram:

Klassediagrammet giver et omfattende perspektiv af strukturen i vores system. Det gør det ved at opdele vores system i *klasser* og vise forholdet mellem dem. Klassediagrammet er den vigtigste byggesten i objektorienteret modellering, og derfor også grunden til at vi har gjort brug af sådan et. Klassediagrammet synliggøre på en overskuelig måde hvordan systemets klasser er afhængige af hinanden. Under de forskellige klasser findes deres attributter (*attributes*) og metoder (*methods*). En klasses attributter beskriver hvilke typer af data vores kode skal bestå af. Det kan eksempelvis være int (*integer*) eller str (*strings*). En klasses metoder er kort sagt de funktioner som en given klasse forsyner sytemet med.
I vores klassediagram har vi ligeledes vist, hvilke klasser vores system er opbygget af og forholdet mellem hinanden.

Software Construction

Kodevalg

Vi har valgt at kode vores virtuelle kaffemaskine i programmeringssproget python.
Vi gør brug af simple funktioner som print() og input(). **print()** bruger vi for at skrive til brugeren hvad han/hun skal gøre. Se eksempel nedenunder.

```
print('Hej, og velkommen til den virtuelle kaffebar. Før du kan bestille skal du logge ind.\n')
```

input() bruger vi for at brugeren kan svare på de spørgsmål vi ønsker at stille. Spørgsmålet stiller vi som et argument til input. Se eksempel nedenunder, hvor vi ønsker at kende Email.

```
self.brugernavn = input('Kea-Email : ')
```

Vi gør endvidere brug af **classes**. Se eksempel længere nede. En af vores classes er vores Login() class. I vores login class har vi defineret en række forskellige funktioner. Hver funktion har hver sin opgave i login fasen. Brugeren bliver først spurgt om login oplysninger (__init__(self)). Derefter ryger de indtastede oplysninger over i vores login_check(self) funktion. Her tjekkes om login er korrekt. Hvis login er korrekt slutter vores Login() class med et print. Hvis en bruger derimod logger ind første gang, skal han/hun gøre brug af opdater_system(self) funktion/metode. Denne funktion opdaterer brugerens oplysninger således brugeren kan logge direkte ind næste gang.

Til sidst i vores opdater_system(self) funktion startes Login() class på ny og brugeren kan nu logge ind. Vi bruger classes for at have så lidt kode som muligt. Det er nemt at genbruge når al koden er delt op i forskellige funktioner. Vi kan bruge vores login check(self) som eksempel. Her kalder vi på forskellige funktioner for at bruge dem i stedet for at skrive hele koden igen. Vi kalder på en funktion ved at bruge self.(funktion) eller login.(funktion).

```
class Login():
    def __init__(self):
        self.brugernavn = input('Kea-Email : ')
        self.adgangskode = input('Adgangskode : ')

    def login_check(self):
        file = open("Login.txt","r")
        if '@' not in self.brugernavn:
            print('Kea-Email skal indeholde @')
            print('Log venligst ind igen')
            login = Login()
            login.login_check()
        elif self.brugernavn and self.adgangskode in file:
            print('\nVelkommen', self.brugernavn)
        else:
            print("\nlogin mislykkedes")
            spørgsmål = input("\nEr det første gang du logger ind på denne kaffemaskine med din Kea-Email? ja/nej : ")

            if spørgsmål == "ja":
                print('Du skal skrive din Kea-Email og adgangskode første gang du logger ind.')
                print('Dette skal du gøre for at opdatere dine oplysingner i vores system.')
                self.opdater_system()

            if spørgsmål == 'nej':
                print('Prøv at logge ind igen.')
                login = Login()
                login.login_check()
            file.close()
    def opdater_system(self):
        file = open("Login.txt", "a")
        file.write("\n" + input('Kea-Email : '))
        file.write("\n" + input('Adgangskode : '))
        print('Login opdateret i vores system. Prøv venligst at logge ind igen.')
        file.close()
        login = Login()
        login.login_check()


login = Login()
login.login_check()
```

Vi har yderligere brugt en fil i vores kode. Denne fil har vi brugt til at gemme brugerens login oplysninger i. Vi skriver i filen med et simpelt input(). Se eksempel nedenunder.

```
file = open("Login.txt", "a")
file.write("\n" + input('Kea-Email : '))
```

Vi bruger også filen til at hente brugerens oplysninger. Se eksempel nedenunder.

```
file = open("Login.txt","r")
```

```
elif self.brugernavn and self.adgangskode in file:
    print('\nVelkommen', self.brugernavn)
```

Vi har som en ekstra feature, gjort brug af en randomizer. Denne bruges til at vælge et tilfældigt spørgsmål til brugeren fra en liste.

```
import random
```

```
self.question_list = ['Spørgsmålet før dig, blev spurgt af Rebecca. Hvad er din yndlings film?'
self.choose = random.choice(self.question_list)
print(self.choose)
```

Vores menu har vi valgt at sætte op som en class, med den fællesnævner at der indgår i alt er vand. Den er opsat, så kunden kan se de ingredienser der indgår i de forskellige drikke.

```
75  #nu skal kunde vælge hvilken form for drikkelse de ønsker
76  class Drik():
77      def __init__(self, vand):
78          self.vand = vand
79
80      def info(self):
81          return "Vand: "+ self.vand
```

Efterfølgende skal man vælge en drik ud fra menuen, dette gøres ved hjælpe af en liste vi har oprettet, hvor alle drikke indgår.

Mangler i kode

Vores kode er begrænset af vores viden, dette påvirker vores resultat. Vores Login() class ville ikke være sikkert, da alle brugernavne og loginoplysninger ligger i en fil frit tilgængeligt for alle. Derudover kan en bruger skrive hvilket som helst brugernavn så længe adgangskoden er rigtig. Dette prøvede vi at løse, men måtte indse efter en længere kamp, at vi ikke kunne.