

2019

Energy drink Programmering - B



Max Hansen

Odense tekniske Gymnasium

21. september 2019

▶ Databaser & Gui

Denne synopsis indeholder metoder fra programmeringsfaget til at lave et program som har et Gui fra tkinter, og en database fra sqlite 3

Indhold

Indledning.....	1
Planlægning af programmet.....	1
Krav / Brugerhistorier	1
Iterationer	2
Udførelse af programmet.....	2
Oprettelse af databasen	2
Oprettelse af klasserne til behandling af data.....	3
Brugerhistorie 1 – tilføj ny energidrik	3
Brugerhistorie 2 – fjern energidrik.....	4
Brugerhistorie 3 – tilføj producent	5
Brugerhistorie 4 – fjern producent	6
Test af programmet	6
Test 1	6
Test 2	6
Test 3	7
Bilag.....	8
Bilag 1 – Brugerhistorier og iterationer.....	8
Brugerhistorie 1: Tilføj energidrik	8
Brugerhistorie 2: Fjern energidrik	8
Brugerhistorie 3: Tilføj producent.....	9
Brugerhistorie 4: Fjern producent	9
Brugerhistorie 5: Ændre energidrik.....	10
Brugerhistorie 6: Ændre producent	10
Iteration 1:	10
Iteration 2:	10
Iteration 3:	11
Bilag 2 – Energy_drinks_data.py.....	11
Bilag 3 – Energy_drinks_gui.py	13

Indledning

Et program som har en grafisk brugerflade og kan huske de data som man indtaster, selv efter programmet er lukket, er meget almindeligt i vores tid. Derfor arbejder jeg med at lave et program som har en grafisk brugerflade, som kan arbejde sammen med en database. Programmet vil komme til at omhandle energidrikke. Jeg vil i denne synopsis dokumentere min arbejdsgang med lave det her program, og komme ind på de forskellige programmeringsmetoder til planlægning og udførelse af programmet.

Planlægning af programmet

Krav / Brugerhistorier

For at kunne lave mit program skulle jeg først planlægge mit arbejde med programmet. Jeg startede med at tænke over de funktioner som mit program skulle have for at fungere. Jeg fandt ud af at jeg gerne ville have seks funktioner i mit program. I programmet skulle brugeren af programmet kunne: Tilføje en energidrik, fjerne en energidrik, tilføje en ny producent af energidrikke, fjerne en nuværende producent af energidrikke, ændre en nuværende energidrik og ændre en nuværende producent.

Nu, hvor jeg havde de funktioner som jeg vil have mit program til at have, så gik jeg i gang med at beskrive brugerhistorierne for funktionerne. Brugerhistorierne skulle jeg bruge til at kunne "simulere", hvordan brugeren ville kunne bruge de forskellige funktioner i programmet korrekt. Det var også for at kunne se hvordan programmet skulle sættes op i opbygningen, for at kunne give den bedst mulige oplevelse. Eftersom jeg havde seks funktioner, så ville jeg også skulle have seks brugerhistorier, da hver funktion skulle beskrives korrekt.

Brugerhistorierne fungerer også som krav til programmet, hvilket var en anden grund til at lave dem så detaljeret som muligt.

Hvis vi ser på den første brugerhistorie (Brugerhistorie 1 under bilag 1), så kan brugeren af programmet starte den her brugerhistorie uanset, hvor i programmet brugeren befinder sig. Brugeren skal derefter navigere hen til området hvor man tilføjer en ny energidrik, indtaste oplysningerne som programmet kræver til at lave en ny energidrik, og derefter trykke på "Tilføj energidrik". Programmet skal derefter tage brugerens indtastede oplysninger, lave til en energidrik og tilføje den energidrik til brugerens database over energidrikke. Programmet skal derefter opdatere nogle tekstfelter kaldet Labels så informationerne på brugerfladen passer. De forskellige brugerhistorier skal jeg så få indsat i nogle iterationer, hvilket er forskellige planer for programmet.

Iterationer

Til mit program har jeg valgt at have tre iterationer. Den første iteration er de mest basale brugerhistorier, såsom Brugerhistorie 1, 2, 3 og 4, se bilag 1. Jeg har dog valgt at opdele de fire brugerhistorier op i to, en til iteration 1 og en til iteration 2, dette kommer jeg ind på senere. Når man arbejder med iterationer, så starter man med at planlægge iterationen, her har jeg planlagt at have de fire første brugerhistorier med i min første iteration. Jeg kan derefter kode programmet så det har de ønskede brugerhistorier, teste at programmet virker som det skal, og til sidst finde ud af om der er noget der skal laves om, eller om programmet fungerer som det skal, som så har klaret den første iteration, mere om det senere.

Udførelse af programmet

Jeg var nu nået til at komme i gang med selve udførelsen af mit program. Jeg starter med at se på min første iterations plan, iteration 1, se bilag 1. Den iteration skulle have brugerhistorie 1, 2, 3 og 4 med.

Jeg valgt at udføre de forskellige brugerhistorier i den rækkefølge som de er lavet i, for at det kommer til at fungere bedst muligt i mit program. Jeg startede derfor med brugerhistorie 1.

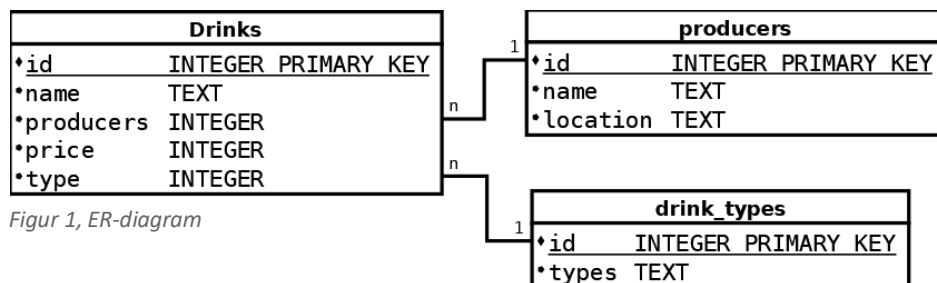
Oprettelse af databasen

I programmering arbejder man med en model der kaldes "Trelagsmodellen", det er en model, hvor et program bliver opdelt i tre lag. Det nederste lag kaldes datalag, det midterste for applikationslaget, og det øverste for brugerfladen. Databasen vil altså være det nederste lag i trelagsmodellen, da det er det lag der har alt med data at gøre. Koden til databasen, ses på bilag 2.

Til at arbejde med databaser i Python bruger jeg et bibliotek der hedder sqlite3, det er et bibliotek, som gør at jeg kan lave en database og faktisk bruge den i Python.

I min database har jeg tre tabeller. På figur 4 har jeg et såkaldt ER-diagram over min database. Her er der en oversigt over hvad der er i de forskellige tabeller i databasen. I min hovedtabel der hedder drinks, gemmer jeg alle energidrikkene. Her vil producers indeholde et tal, som passer med det id, som producer har i

tabellen producers, som er en tabel over alle producenter i databasen. Type i drinks, fungerer på samme måde. Dette kaldes en, "en til mange relation", som går ud på at forhindre at skrive



navnet flere gange i en tabel, men derimod have en tabel over navnet på producenten. Det gør så man kun skal tilføje det navn 1 gang, som kan bruges senere hen til alle andre energidrikke der skal have den producent. Det er smart til, hvis man ændrer navnet på den ene producent, så vil det blive ændret for alle de energidrikke der har den tidligere producent til den nye producent.

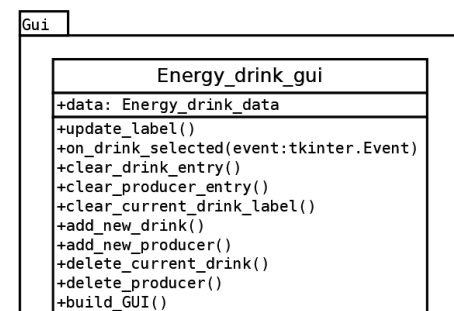
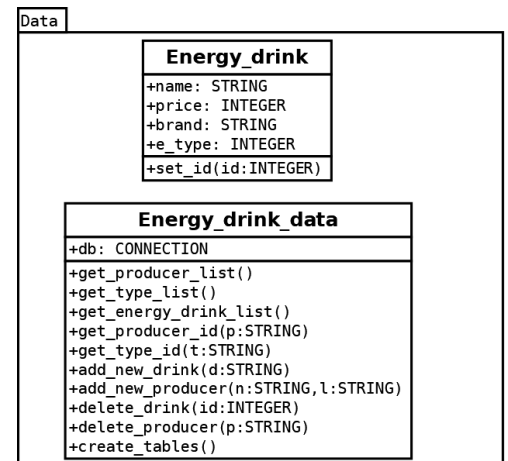
Oprettelse af klasserne til behandling af data

For at kunne bruge dataen fra datalaget i trelagsmodellen, så skal der laves nogle funktioner, som kan hente og indsende data til databasen. Her er jeg altså i det midterste lag af trelagsmodellen.

For at komme det problem til livs, har jeg valgt at lave to klasser som skal arbejde med datalaget. Se figur 2, Data.

Den første klasse, `Energy_drink`, er den klasse som bruges til at lave en energidrik som et slags objekt. Den bruges altså til at kunne indsætte en energidrik i databasen med alle de informationer som brugeren har indtastet i brugerfladen.

Energidrik objektet bliver så brugt i klassen `Energy_drink_data`, hvor alle funktionerne til at arbejde sammen med datalaget eksisterer. Det er funktioner lige fra at fjerne en producent til at tilføje en ny energidrik. De her to klasser er derfor vigtige for at programmet virker som det skal.



Figur 2, klassediagram

Brugerhistorie 1 – tilføj ny energidrik

Min første brugerhistorie går ud på at brugeren skulle kunne tilføje en ny energidrik til databasen over energidrikke. Det krævede tre ting af mit program, nemlig at programmet havde en database, noget kode som kunne håndtere den her data, og så en brugerflade så brugeren kunne kommunikere med databasen.

```
1 def add_new_drink(self, d):
2     p = self.get_producer_id(d.brand)
3     t = self.get_type_id(d.e_type)
4     c = self.db.cursor()
5     c.execute('INSERT INTO drinks (name, price, producers, type) VALUES (?, ?, ?, ?);', (d.name, d.price, p,
6     t))
7     self.db.commit()
```

Figur 3, tilføj energidrik funktionen

Brugerhistorie 1 gik jo ud på at kunne tilføje en energidrik. Til det har jeg lavet en funktion til dette formål. Se figur 3.

Denne funktion, har en parameter `d`, som modtager en energidrik, der kommer fra klassen `Energy_drink`, se figur 2. Her skal jeg så finde id'et på den producent som brugeren har valgt, og id'et på den type energidrik, som brugeren har valgt på figur 4.

De data vil så blive indsat i databasen i en tabel kaldet `drinks`.

Navn

Pris

Producenter

Type

Tilføj energidrik

Figur 4, brugerfladen af tilføj energidrik

Nu er vi nået til det øverste lag i trelagsmodellen nemlig brugerfladen. Brugerfladen til brugerhistorie 1, kan ses på figur 4. Her er der 2 tekstfelter, 2 dropdown-mener og 1 knap. For at kunne lave det, bruger jeg et bibliotek i Python der hedder tkinter. Tkinter bruges til at lave en brugerflade til et program, som så arbejder sammen med applikationslaget, som arbejder sammen med datalaget, og omvendt. Alt der tegner og viser noget på brugerfladen, er lavet i den klasse der hedder `Energy_drinks_gui`, se figur 2, Gui.

Den her menu er forholdsvis simpel. Hvis man ser på figur 5, så er de fire linjer, det der laver et mærkat også kendt som label, hvor der står "Navn", og lavet et tekstfelt også kendt som en entry, ved siden af. De to objekter bliver placeret ved noget som kaldes et grid, det fungerer ved at man har nogle kolonner og rækker, og på den måde placere ting i forhold til hinanden. På figur 6, ser vi så den kode som der er brugt til at lave dropdown-menerne, som også er kendt som en combobox, den her combobox, indeholder de producenter som brugeren har i sin database.

```
1 self.label_name = ttk.Label(self.button_panel, text = 'Navn')
2 self.label_name.grid(row = 0, column = 1)
3 self.entry_name = ttk.Entry(self.button_panel, width = 23)
4 self.entry_name.grid(row = 0, column = 2, pady = (0,5))
```

Figur 6, kode til et tekstfelt med et tekst mærkat

```
1 self.cb_producers = ttk.Combobox(self.button_panel, values = producers, state = 'readonly')
2 self.cb_producers.grid(row = 2, column = 2, pady = (0,5))
```

Figur 5, koden til en dropdown-menu

Brugerhistorie 2 – fjern energidrik

Min anden brugerhistorie går ud på at brugeren skulle kunne fjerne en energidrik, som brugeren har tilføjet til databasen. Det kræver her at brugeren kan vælge energidrikken på brugerfladen. Det er blevet lavet ved et såkaldt treeview. Det er sådanset en tabel, der viser alle

Navn: Monster Energy
Producent: Moner_Energy
Pris: 10
Type: Standard
Fjern energidrik

Navn	Producent	Pris	Type
Monster Energy	Moner_Energy	10	Standard

Figur 8, billede af fjern energidrik

```
1 self.db_view = ttk.Treeview(self.data_panel, column = ('column1', 'column2', 'column3', 'column4', 'column5'), show
= 'headings', height = 49)
2 self.db_view.bind("<ButtonRelease-1>", self.on_drink_selected)
3 self.db_view.heading('#1', text = 'Navn')
4 self.db_view.heading('#2', text = 'Producent')
5 self.db_view.heading('#3', text = 'Pris')
6 self.db_view.heading('#4', text = 'Type')
7 self.db_view.heading('#5', text = 'id')
8 self.db_view['displaycolumns'] = ('column1', 'column2', 'column3', 'column4')
```

Figur 7, koden til treeview (tabellen)

de energidrikke som brugeren har i sin database, med alle de nødvendige informationer, se figur 7.

På figur 8, kan koden til treeviewet ses. På linje 1 startes treeviewet, hvor vi bestemmer hvor den skal placeres, hvor mange kolonner der skal være, hvor høj den skal være, og hvad der skal vises. På linje 2 registres brugerens mus, og laver et event når musen har klikket på en række i tabellen. Linje 3-7 navngiver de forskellige kolonner, og linje 8, bestemmer hvilke kolonner der skal vises.

Energidrikkene bliver så indsat i treeviewet ved at få alle energidrikkene der er i databasen, se figur 9 linje 1, derefter kører programmet et loop igennem, som kører så mange gange, som der er af energidrikke, se linje 5.

Herefter indsætter programmet så energidrikkens data ind i treeviewet i den rigtige rækkefølge, se linje 6.

Når brugeren så vælger en energidrik, køres en funktion kaldet

"on_drink_selected", se

Figur 9, indsættelse af energidrikke

```
1 def update_label(self):
2     l = self.data.get_energy_drink_list()
3     self.drinks_label.config(text = 'Der er {} registrerede energidrikke i databasen'.format(len(l)))
4     self.db_view.delete(*self.db_view.get_children())
5     for e in l:
6         self.db_view.insert("", tk.END, values = (e.name, e.brand, e.price, e.e_type, e.id))
```

Figur 10, koden til en knap

```
1 self.button_delete = ttk.Button(self.button_panel, text = 'Fjern energidrik', command = self.delete_current_drink)
2 self.button_delete.grid(row = 4, column = 3)
```

bilag 3. Den finder alle informationerne om den energidrik der er valgt, og laver nogle mærkater med informationerne. Unde de mærkater, kan man se på figur 7, at der er en knap til at fjerne en energidrik. Knappen bliver lavet på figur 10. Knappen kører så en funktion kaldet "delete_current_drink", som fjerner den valgte energidrik fra databasen.

Brugerhistorie 3 – tilføj producent

Min tredje brugerhistorie, går ud på at kunne tilføje en ny producent af en energidrik. Det er meget simpelt at lave, da det eneste det kræver, er 2 tekstfelter og 1 knap på brugerfladen. Brugeren indtaster navnet på producenten, og hvor producenten er fra, og trykker på knappen "Tilføj producent", for at tilføje dem. Se figur 11. Knappen

aktivere funktionen

"add_new_producer", se figur 12.

Den funktion modtager så de

informationer brugeren har

indtastet, og kører en anden

funktion fra data programmet. Se

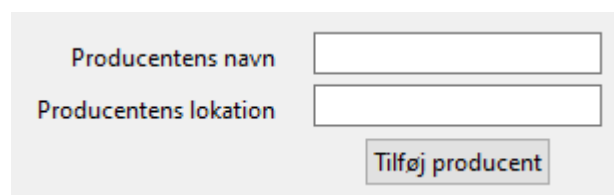
figur 12 linje 2 og figur 13. Resten

af programmet opdatere bare de forskellige

producenter dropdown-menuer, så de har

den nyeste producent med.

På figur 13 er den funktion som den anden funktion kører, og den tilføjer bare producenten til databasen.



Figur 11, tilføj producent

```
1 def add_new_producer(self):
2     self.data.add_new_producer(self.entry_producer_name.get(), self.entry_producer_location.get())
3     producers = self.data.get_producer_list()
4     self.cb_producers = ttk.Combobox(self.button_panel, values = producers, state = 'readonly')
5     self.cb_producers.grid(row = 2, column = 2, pady = (0,5))
6     self.cb_delete_producers = ttk.Combobox(self.button_panel, values = producers, state = 'readonly')
7     self.cb_delete_producers.grid(row = 9, column = 2)
8     self.clear_producer_entry()
```

Figur 12, funktionen ny producent

```
1 def add_new_producer(self, n, l):
2     c = self.db.cursor()
3     c.execute('INSERT INTO producers (name, location) VALUES (?, ?);', (n, l))
4     self.db.commit()
```

Figur 13, ny producent

Brugerhistorie 4 – fjern producent

Min fjerde og sidste brugerhistorie til den første iteration, går ud på at kunne fjerne en nuværende producent fra databasen, hvis nu producenten går konkurs eller lignende.

Igen var det rimelig simpelt at lave, da det krævede en dropdown-menu på

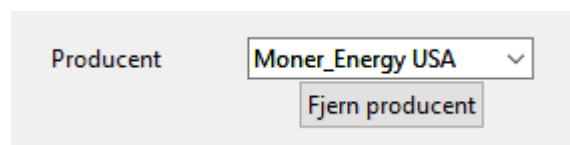
brugerfladen sammen med en knap til at fjerne producenten.

Brugeren skal altså vælge en producent fra dropdown-menuen, og trykke på knappen "Fjern producent", se figur 14.

Knappen kører en funktion

kaldet "delete_producer", se figur 15. Denne funktion fjerner den producent som brugeren har valgt ved at køre en funktion fra "energy_drink_data" som også er kaldet "delete_producer", se figur 16. Funktionen på figur 15 fra linje 4 – 6 opdaterer alle dropdown-menuer som har med producenter at gøre, så man ikke kan vælge dem mere.

Funktionen på figur 16, er den del af koden som fjerner producenten fra databasen. Her fjerner den også alle de energidrikke der har den producent for at forhindre fejl i programmet. Se figur 16 linje 4.



Figur 14, fjern producent

```
1 def delete_producer(self):
2     self.data.delete_producer(self.cb_delete_producers.get())
3     self.update_label()
4     producers = self.data.get_producer_list()
5     self.cb_producers = ttk.Combobox(self.button_panel, values = producers, state = 'readonly')
6     self.cb_producers.grid(row = 2, column = 2, pady = (0,5))
7     self.cb_delete_producers = ttk.Combobox(self.button_panel, values = producers, state = 'readonly')
8     self.cb_delete_producers.grid(row = 9, column = 2)
9     self.cb_delete_producers.set('')
```

Figur 16, kode til fjern producent

```
1 def delete_producer(self, p):
2     d = self.get_producer_id(p)
3     c = self.db.cursor()
4     c.execute('DELETE FROM drinks WHERE producers = ?;', (d,))
5     c.execute('DELETE FROM producers WHERE id = ?;', (d,))
6     self.db.commit()
```

Figur 15, kode til fjern producent

Test af programmet

Nu hvor jeg har fået udarbejdet alle de forskellige brugerhistorier til min første iteration, kan jeg nu teste, og se at de fungerer korrekt.

Test 1

Jeg testede programmet den første gang, og fandt hurtigt ud af at jeg ikke kunne fjerne eller tilføje noget som helt. Det viste sig dog at være en simpel lille fejl i min kode der styrer min database. Efter jeg fik ændret det, kunne jeg nu test programmet igen.

Test 2

Jeg testede programmet for anden gang, og fandt ud af at det hele faktisk fungerede nogenlunde som de skulle. Jeg havde dog et problem med at min producent i dropdown-menuen havde nogle bølgede parenteser ({}), rundt om det første navn, som var "Monster Energy". Jeg brugte utroligt lang tid på at finde ud af hvad der lavede denne fejl, da den fejl gjorde at man ikke kunne tilføje en

ny energidrik med den producent. Det viste sig så at være fordi der var mellemrum i navnet, som tkinter så ikke kunne finde ud af. Jeg har endnu ikke tænkt på en løsning, så lige nu må man ikke have mellemrum i navnet på producenten.

Test 3

Tredje og sidste test jeg lavede af programmet, virkede det hele som det skulle og mit program har nu været igennem den første iteration, og er nu klar til iteration 2.

Figur 17 viser billede af brugerfladen som programmet har i det færdige stadige.

The screenshot shows a web application titled "Energidrikke". On the left, there is a form for adding a new energy drink. It includes fields for "Navn" (Name), "Pris" (Price), "Producenter" (Producer), and "Type", each with a "Tilføj" (Add) button. Below these are fields for "Producentens navn" (Producer's name) and "Producentens lokation" (Producer's location), also with "Tilføj" buttons. At the bottom, there is a "Producent" dropdown menu with "Moner_Energy USA" selected and a "Fjern producent" (Remove producer) button. On the right, there is a table with the following data:

Navn	Producent	Pris	Type
Monster Energy	Moner_Energy	10	Standard

Figur 17, hele brugerfladen

Bilag

Bilag 1 – Brugerhistorier og iterationer

Brugerhistorie 1: Tilføj energidrik

Plan til iteration 1

Denne brugerhistorie kan startes af brugeren, uanset hvilken side brugeren er på.

1. Brugeren vælger at tilføje en ny energidrik.
2. Brugeren indtaster oplysninger til programmet:
 - a. Energidrikkens navn
 - b. Energidrikkens producent
 - c. Energidrikkens pris
 - d. Energidrikkens type
3. Brugeren trykker på "Tilføj energidrik" for at tilføje en energidrik med de oplysninger som brugeren har indtastet.
4. Programmet tilføjer energidrikken i databasen.
5. Programmet opdaterer den viste liste med energidrikke.
6. Programmet lægger 1 til antallet af energidrikke i databasen.
7. Programmet fjerner alle brugerens input, så det er klar til noget nyt.

Plan til iteration 2

1. Brugeren vælger at tilføje en ny energidrik.
2. Brugeren indtaster oplysninger til programmet:
 - a. Energidrikkens navn
 - b. Energidrikkens producent
 - c. Energidrikkens pris
 - d. Energidrikkens type
3. Brugeren trykker på "Tilføj energidrik" for at tilføje en energidrik med de oplysninger som brugeren har indtastet.
 - a. Programmet tjekker om brugeren har indtastet et tal som pris.
 - i. Hvis pris ikke er et tal, programmet laver en fejl og der kommer en rød firkant på skærmen med beskeden "Prisen skal være et tal!".
4. Programmet tilføjer energidrikken i databasen.
5. Programmet opdaterer den viste liste med energidrikke.
6. Programmet lægger 1 til antallet af energidrikke i databasen.
7. Programmet fjerner alle brugerens input, så det er klar til noget nyt.
8. Programmet viser en grøn firkant på skærmen med beskeden "Energidrikken blev tilføjet til databasen!".

Brugerhistorie 2: Fjern energidrik

Plan til iteration 1

Denne brugerhistorie kan startes af brugeren, uanset hvilken side brugeren er på.

1. Brugeren vælger den energidrik der skal slettes fra listen over energidrikke.
2. Brugeren trykker på "Slet energidrik".
3. Programmet sletter den valgte energidrik fra databasen.
4. Programmet trækker 1 fra antallet af energidrikke i databasen.

Plan til iteration 2

1. Brugeren vælger den energidrik der skal slettes fra listen over energidrikke.
2. Brugeren trykker på "Slet energidrik".
3. Programmet sletter den valgte energidrik fra databasen.
4. Programmet trækker 1 fra antallet af energidrikke i databasen.
5. Programmet laver en grøn firkant på skærmen med beskeden "Energidrikken blev fjernet fra databasen!".

Brugerhistorie 3: Tilføj producent

Plan til iteration 1

Denne brugerhistorie kan startes af brugeren, uanset hvilken side brugeren er på.

1. Brugeren vælger at tilføje en ny producent
2. Brugeren indtaster oplysninger til programmet:
 - a. Producentens navn
 - b. Producentens lokation
3. Brugeren trykker på "Tilføj producent" for at tilføje en producent med de oplysninger som brugeren har indtastet.
4. Programmet tilføjer producenten til databasen over producenter.
5. Programmet fjerner alle brugerens input, så det er klar til noget nyt.

Plan til iteration 2

1. Brugeren vælger at tilføje en ny producent
2. Brugeren indtaster oplysninger til programmet:
 - a. Producentens navn
 - b. Producentens lokation
3. Brugeren trykker på "Tilføj producent" for at tilføje en producent med de oplysninger som brugeren har indtastet.
4. Programmet tilføjer producenten til databasen over producenter.
5. Programmet fjerner alle brugerens input, så det er klar til noget nyt.
6. Programmet laver en grøn firkant på skærmen med beskeden "Producenten blev tilføjet til databasen!".

Brugerhistorie 4: Fjern producent

Plan til iteration 1

Denne brugerhistorie kan startes af brugeren, uanset hvilken side brugeren er på.

1. Brugeren vælger en producent fra en dropdown-menu som brugeren vil slette.
2. Brugeren trykker derefter på "Slet producent".

3. Programmet sletter den valgte producent fra databasen over producenter.
4. Programmet sletter også energidrikke med den valgte producent, fra databasen over energidrikke.
5. Programmet fjerner brugerens input fra dropdown-menuen, så det er klar til noget nyt.

Plan til iteration 2

1. Brugeren vælger en producent fra en dropdown menu som brugeren vil slette.
2. Brugeren trykker derefter på "Slet producent".
3. Programmet sletter den valgte producent fra databasen over producenter.
4. Programmet sletter også energidrikke med den valgte producent, fra databasen over energidrikke.
5. Programmet fjerner brugerens input fra dropdown-menuen, så det er klar til noget nyt.
6. Programmet laver en grøn firkant på skærmen med beskeden "Producenten blev fjernet fra databasen!".

Brugerhistorie 5: Ændre energidrik

Denne brugerhistorie kan startes af brugeren, uanset hvilken side brugeren er på.

1. Brugeren vælger en energidrik fra listen over energidrikke.
2. Brugeren trykker på "Ændre".
3. Brugeren ændrer de oplysninger som skal ændres.
4. Brugeren trykker på "Ændre energidrik".
5. Programmet opdaterer energidrikken i databasen.
6. Programmet opdaterer listen med energidrikke, så oplysningerne er korrekte.
7. Programmet fjerner inputstederne til at ændre energidrikke, så det er klar til nyt input.

Brugerhistorie 6: Ændre producent

Denne brugerhistorie kan startes af brugeren, uanset hvilken side brugeren er på.

1. Brugeren vælger en producent fra en dropdown-menu som brugeren vil ændre.
2. Brugeren trykker på "Ændre".
3. Brugeren ændrer de oplysninger som skal ændres.
4. Brugeren trykker på "Ændre producent".
5. Programmet opdaterer producenten i databasen over producenter.
6. Programmet opdaterer listen med energidrikke, så oplysningerne er korrekte.
7. Programmet fjerner inputstederne til at ændre producenten, samt nulstiller dropdown-menuen, så det er klar til noget nyt.

Iteration 1:

Brugerhistorier:

- Brugerhistorie 1, Brugerhistorie 2, Brugerhistorie 3 og Brugerhistorie 4.

Iteration 2:

Brugerhistorier:

- Brugerhistorie 1, Brugerhistorie 2, Brugerhistorie 3 og Brugerhistorie 4.

Iteration 3:

Brugerhistorier:

- Brugerhistorie 1, Brugerhistorie 2, Brugerhistorie 3, Brugerhistorie 4, Brugerhistorie 5 og Brugerhistorie 6.

Bilag 2 – Energy_drinks_data.py

```
import sqlite3

class Energy_drink():
    def __init__(self, name, price, brand, e_type):
        self.name = name
        self.price = price
        self.brand = brand
        self.e_type = e_type

    def set_id(self, id):
        self.id = id

class Energy_drink_data():
    def __init__(self):
        self.db = sqlite3.connect('energy_drinks.db')

    def get_producer_list(self):
        c = self.db.cursor()
        c.execute('SELECT name, location FROM producers;')
        p_list = []
        for p in c:
            p_list.append((p[0], p[1]))
        return p_list

    def get_type_list(self):
        c = self.db.cursor()
        c.execute('SELECT types FROM drink_types;')
        t_list = []
        for t in c:
            t_list.append(t[0])

        return t_list

    def get_energy_drink_list(self):
        c = self.db.cursor()
        c.execute('SELECT d.name, p.name, d.price, dd.types, d.id FROM drinks d INNER JOIN producers p ON d.producers = p.id INNER JOIN drink_types dd ON d.type = dd.id;')
```

```
d_list = []
for d in c:
    drink = Energy_drink(d[0], d[2], d[1], d[3])
    drink.set_id(d[4])
    d_list.append(drink)
return d_list

def get_producer_id(self, p):
    c = self.db.cursor()
    c.execute('SELECT id FROM producers WHERE name = ?;', (p.split(' ')[0],))
    p = c.fetchone()
    return p[0]

def get_type_id(self, t):
    c = self.db.cursor()
    c.execute('SELECT id FROM drink_types WHERE types = ?;', (t,))
    e = c.fetchone()
    return e[0]

def add_new_drink(self, d):
    p = self.get_producer_id(d.brand)
    t = self.get_type_id(d.e_type)
    c = self.db.cursor()
    c.execute('INSERT INTO drinks (name, price, producers, type) VALUES (?, ?, ?, ?);', (d.name, d.price, p, t))
    self.db.commit()

def add_new_producer(self, n, l):
    c = self.db.cursor()
    c.execute('INSERT INTO producers (name, location) VALUES (?, ?);', (n, l))
    self.db.commit()

def delete_drink(self, id):
    c = self.db.cursor()
    c.execute('DELETE FROM drinks WHERE id = ?;', (id,))
    self.db.commit()

def delete_producer(self, p):
    d = self.get_producer_id(p)
    c = self.db.cursor()
    c.execute('DELETE FROM drinks WHERE producers = ?;', (d,))
    c.execute('DELETE FROM producers WHERE id = ?;', (d,))
    self.db.commit()

def create_tables(self):
    try:
        self.db.execute("""DROP TABLE IF EXISTS drinks;""")
```

```
self.db.execute("""DROP TABLE IF EXISTS producers;""")
self.db.execute("""DROP TABLE IF EXISTS drink_types;""")

print('Table deleted')
except Exception as e:
    print('ERROR while deleting table!')
    print(e)

try:
    self.db.execute("""CREATE TABLE IF NOT EXISTS drinks (id INTEGER PRIMARY KEY, name TEXT, producers INTEGER, price INTEGER, type INTEGER);""")
    self.db.execute("""CREATE TABLE IF NOT EXISTS producers (id INTEGER PRIMARY KEY, name TEXT, location TEXT);""")
    self.db.execute("""CREATE TABLE IF NOT EXISTS drink_types (id INTEGER PRIMARY KEY, types TEXT);""")

    print('Tables created succesfully')
except Exception as e:
    print('ERROR while creating tables!')
    print(e)

self.db.execute("""INSERT INTO drinks (name, producers, price, type) VALUES (?, ?, ?, ?);""", ('Monster Energy', 1, 20, 1))

self.db.execute("""INSERT INTO producers (name, location) VALUES (?, ?);""", ('Monster_Energy', 'USA'))

self.db.execute("""INSERT INTO drink_types (types) VALUES (?);""", ('Standard',))
self.db.execute("""INSERT INTO drink_types (types) VALUES (?);""", ('Sukkerfri',))

self.db.commit()
```

Bilag 3 – Energy_drinks_gui.py

```
from Energy_drinks_data import Energy_drink, Energy_drink_data
import tkinter as tk
import tkinter.ttk as ttk
class Energy_drink_gui(ttk.Frame):
    def __init__(self, master = None):
        ttk.Frame.__init__(self, master)
        self.data = Energy_drink_data()
        self.build_GUI()
```

```
self.update_label()

def update_label(self):
    l = self.data.get_energy_drink_list()
    self.drinks_label.config(text = 'Der er {} registrerede energidrikke i databa
sen'.format(len(l)))
    self.db_view.delete(*self.db_view.get_children())
    for e in l:
        self.db_view.insert("", tk.END, values = (e.name, e.brand, e.price, e.e
_type, e.id))

def on_drink_selected(self, event):
    cur_item = self.db_view.item(self.db_view.focus())['values']
    if len(cur_item) > 0:
        self.label_current_name.config(text = 'Navn: {}'.format(cur_item[0]))
        self.label_current_producer.config(text = 'Producent: {}'.format(cur_it
em[1]))
        self.label_current_price.config(text = 'Pris: {}'.format(cur_item[2]))
        self.label_current_type.config(text = 'Type: {}'.format(cur_item[3]))

def clear_drink_entry(self):
    self.entry_name.delete(0, tk.END)
    self.entry_price.delete(0, tk.END)
    self.cb_producers.set('')
    self.cb_type.set('')

def clear_producer_entry(self):
    self.entry_producer_name.delete(0, tk.END)
    self.entry_producer_location.delete(0, tk.END)

def clear_current_drink_label(self):
    self.label_current_name.configure(text = 'Navn: ')
    self.label_current_producer.configure(text = 'Producent: ')
    self.label_current_price.configure(text = 'Price: ')
    self.label_current_type.configure(text = 'Type: ')

def add_new_drink(self):
    e = Energy_drink(self.entry_name.get(), int(self.entry_price.get()), self.c
b_producers.get(), self.cb_type.get())
    self.data.add_new_drink(e)
    self.update_label()
    self.clear_drink_entry()

def add_new_producer(self):
    self.data.add_new_producer(self.entry_producer_name.get(), self.entry_produ
cer_location.get())
    producers = self.data.get_producer_list()
```



```
        self.cb_producers = ttk.Combobox(self.button_panel, values = producers, state = 'readonly')
        self.cb_producers.grid(row = 2, column = 2, pady = (0,5))
        self.cb_delete_producers = ttk.Combobox(self.button_panel, values = producers, state = 'readonly')
        self.cb_delete_producers.grid(row = 9, column = 2)
        self.clear_producer_entry()

    def delete_current_drink(self):
        cur_item = self.db_view.focus()
        if len(self.db_view.item(cur_item)['values']) >= 4:
            self.data.delete_drink(self.db_view.item(cur_item)['values'][4])
        self.update_label()
        self.clear_current_drink_label()

    def delete_producer(self):
        self.data.delete_producer(self.cb_delete_producers.get())
        self.update_label()
        producers = self.data.get_producer_list()
        self.cb_producers = ttk.Combobox(self.button_panel, values = producers, state = 'readonly')
        self.cb_producers.grid(row = 2, column = 2, pady = (0,5))
        self.cb_delete_producers = ttk.Combobox(self.button_panel, values = producers, state = 'readonly')
        self.cb_delete_producers.grid(row = 9, column = 2)
        self.cb_delete_producers.set('')

    def build_GUI(self):
        self.data_panel = ttk.Frame(self)
        self.button_panel = ttk.Frame(self)

        self.button_panel.grid_columnconfigure(3, minsize = 200)
        self.drinks_label = ttk.Label(self.button_panel, text = 'Der er {} registrede energidrikke i databasen'.format(None))
        self.drinks_label.grid(row = 0, column = 0)
        self.button_update = ttk.Button(self.button_panel, text = 'Opdater', command = self.update_label)
        self.button_update.grid(row = 1, column = 0)

        self.label_name = ttk.Label(self.button_panel, text = 'Navn')
        self.label_name.grid(row = 0, column = 1)
        self.entry_name = ttk.Entry(self.button_panel, width = 23)
        self.entry_name.grid(row = 0, column = 2, pady = (0,5))
        self.label_price = ttk.Label(self.button_panel, text = 'Pris')
        self.label_price.grid(row = 1, column = 1)
        self.entry_price = ttk.Entry(self.button_panel, width = 23)
        self.entry_price.grid(row = 1, column = 2, pady = (0,5))
```

```
producers = self.data.get_producer_list()
label_producers = ttk.Label(self.button_panel, text = 'Producenter')
label_producers.grid(row = 2, column = 1)
self.cb_producers = ttk.Combobox(self.button_panel, values = producers, state = 'readonly')
self.cb_producers.grid(row = 2, column = 2, pady = (0,5))
types = self.data.get_type_list()
label_type = ttk.Label(self.button_panel, text = 'Type')
label_type.grid(row = 3, column = 1)
self.cb_type = ttk.Combobox(self.button_panel, values = types, state = 'readonly')
self.cb_type.grid(row = 3, column = 2, pady = (0,5))
self.button_add = ttk.Button(self.button_panel, text = 'Tilføj energidrik', command = self.add_new_drink)
self.button_add.grid(row = 4, column = 2, pady = (0,10))

self.label_producer_name = ttk.Label(self.button_panel, text = 'Producentens navn')
self.label_producer_name.grid(row = 6, column = 1)
self.entry_producer_name = ttk.Entry(self.button_panel, width = 23)
self.entry_producer_name.grid(row = 6, column = 2, pady = (0,5))
self.label_producer_location = ttk.Label(self.button_panel, text = 'Producentens lokation')
self.label_producer_location.grid(row = 7, column = 1, padx = (0,17))
self.entry_producer_location = ttk.Entry(self.button_panel, width = 23)
self.entry_producer_location.grid(row = 7, column = 2, pady = (0,5))
self.button_add_producer = ttk.Button(self.button_panel, text = 'Tilføj producent', command = self.add_new_producer)
self.button_add_producer.grid(row = 8, column = 2, pady = (0,20))

self.label_current_name = ttk.Label(self.button_panel, text = 'Navn: ')
self.label_current_name.grid(row = 0, column = 3)
self.label_current_producer = ttk.Label(self.button_panel, text = 'Producent: ')
self.label_current_producer.grid(row = 1, column = 3)
self.label_current_price = ttk.Label(self.button_panel, text = 'Pris: ')
self.label_current_price.grid(row = 2, column = 3)
self.label_current_type = ttk.Label(self.button_panel, text = 'Type: ')
self.label_current_type.grid(row = 3, column = 3)
self.button_delete = ttk.Button(self.button_panel, text = 'Fjern energidrik', command = self.delete_current_drink)
self.button_delete.grid(row = 4, column = 3)

self.label_producer = ttk.Label(self.button_panel, text = 'Producent')
self.label_producer.grid(row = 9, column = 1)
self.cb_delete_producers = ttk.Combobox(self.button_panel, values = producers, state = 'readonly')
```

```
self.cb_delete_producers.grid(row = 9, column = 2)
self.button_delete_producer = ttk.Button(self.button_panel, text = 'Fjern p
roducent', command = self.delete_producer)
self.button_delete_producer.grid(row = 10, column = 2)

self.db_view = ttk.Treeview(self.data_panel, column = ('column1', 'column2'
, 'column3', 'column4', 'column5'), show = 'headings', height = 49)
self.db_view.bind("<ButtonRelease-1>", self.on_drink_selected)
self.db_view.heading('#1', text = 'Navn')
self.db_view.heading('#2', text = 'Producent')
self.db_view.heading('#3', text = 'Pris')
self.db_view.heading('#4', text = 'Type')
self.db_view.heading('#5', text = 'id')
self.db_view['displaycolumns'] = ('column1', 'column2', 'column3', 'column4
')

scroll_bar_y = ttk.Scrollbar(self.data_panel, command = self.db_view.yview,
orient = tk.VERTICAL)
self.db_view.configure(yscrollcommand = scroll_bar_y.set)
self.db_view.pack(side = tk.RIGHT)
self.data_panel.pack(side = tk.RIGHT)
self.button_panel.pack(side = tk.TOP)
self.pack()

root = tk.Tk()
root.iconbitmap('./Icon/icon_AQK_icon.ico')
root.geometry('1280x720')
root.state('zoomed')

app = Energy_drink_gui(root)
app.master.title('Energidrikke')
app.mainloop()
```