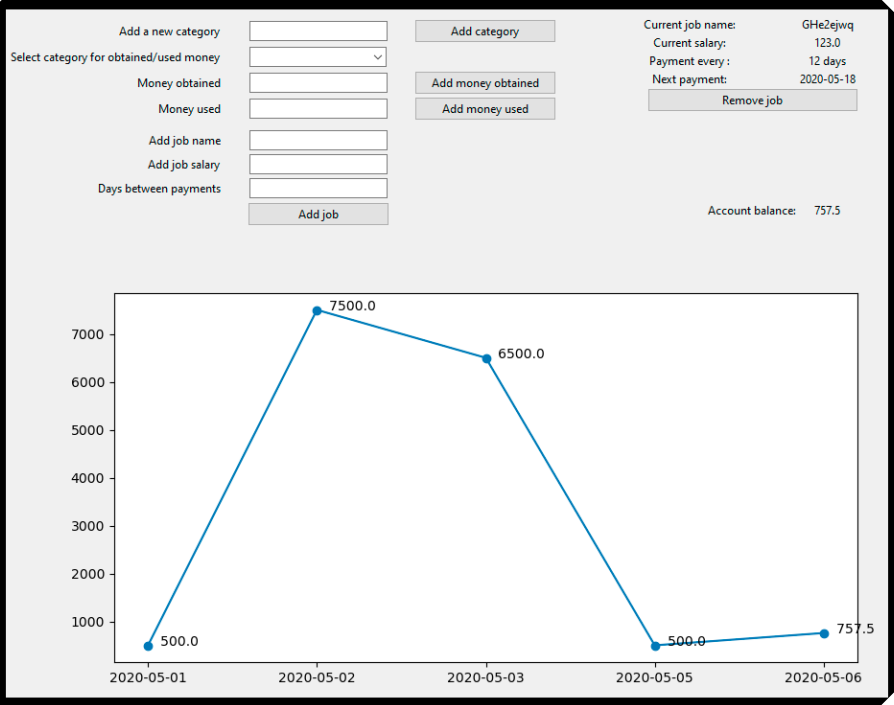
2020

Eksamensprojekt

Programmering - B



Max Hansen

Odense Tekniske Gymnasium

7. maj 2020

Økonomi

Denne synopsis indeholder udførelsen af et program der skal hjælpe med at give et overblik over brugerens økonomi.

TITELBLAD

VEJLEDER

Søren Præstegaard - SPR

FAG

Programmering B

KLASSE OG SKOLE

3.D - Odense Tekniske Gymnasium

ANTAL SIDER

38 A4-sider

7,5 sider à 2400 tegn, brugerhistorier og lignede dokumentation er ikke talt med.

DATO FOR UDLEVERING

11-03-2020

DATO FOR AFLEVERING

07-05-2020

INDHOLDSFORTEGNELSE

INDLEDNING

Økonomi er et vigtigt begreb i dagligdagen, hvilket vi alle skal holde styr på. Det kan til tider være svært at have et overblik over, hvor mange penge man har haft på kontoen på forskellige datoer. Jeg arbejder derfor med at lave et program, som har en grafisk brugerflade, der ved hjælp af en database kan hjælpe brugeren med at holde lidt styr på brugerens økonomi. I denne synopsis vil jeg dokumentere min arbejdsgang med udviklingen af programmet. Her vil jeg komme ind på nogle af de forskellige metoder fra programmeringsfaget, som jeg har brugt til planlægningen og udførelsen af programmet.

Til login systemet, for at enkryptere og dekryptere adgangskoderne, har jeg brugt to funktioner som jeg har fundet og brugt direkte.[1]

Koden til grafen er en kode jeg har fundet. Koden virkede dog ikke til at starte med, så jeg har ændret den, så den virker.[2]

OPGAVEFORMULERING

Jeg skal lave et program der kan:

- arbejde med økonomi. Heriblandt beregne kontosaldoer og vise den tilhørende kontosaldo, på et Graphical User Interface (GUI).
- Programmet skal kunne gemme informationer indtil programmet åbnes igen.

[1] Funktionerne: hash_password() og verify_password() i filen: econodata.py: (Molina, 2018)
[2] Linje 327 - 336: (pythonprogrammring, u.d.)

PLANLÆGNING AF PROGRAMMET

KRAV

For at kunne lave et program så kræver det, at man sætter sig nogle krav for, hvad der skal være med i programmet. Kravene skal også være med til at sætte nogle rammer for, hvordan programmet skal fungere når programmet bruges af en bruger. Jeg har derfor lavet 3 krav til mit program.

KRAV 1 - GUI

Programmet skal have en GUI. På programmets GUI skal brugeren blandt andet kunne indtaste: hvor ofte brugeren får løn, hvor meget brugeren får i løn og hvor mange penge brugeren har brugt/fået siden sit sidste besøg. Brugeren skal til det sidste have mulighed for at kunne lave sine egne kategorier, såsom ”Brændstof”, som kan opdele brugerens økonomi i kategorier. De forskellige informationer skal være vist på GUIen, og brugerens kontosaldo skal vises med en graf.

Programmets GUI skal starte med at bede brugeren om at logge ind, eller lave en konto. Derefter vil brugeren kunne komme ind og ændre sin økonomi i programmet.

KRAV 2 - ADMINISTRERING AF ØKONOMI

Programmet skal kunne administrere økonomien, ved at udregne hvor mange penge der er tilbage på kontoen hvis brugeren har brugt penge osv. Programmet skal også kunne holde styr på løn fra brugerens arbejde, hvis brugeren har tilføjet sit arbejde.

KRAV 3 - VISNING AF ØKONOMI

Programmet skal via programmets GUI vise brugerens nuværende kontosaldo. Programmet skal også via programmets GUI kunne vise forløbet for brugerens kontosaldo, for de datoer, hvor der er sket ændringer i brugerens kontosaldo, med en graf. Visningen af disse saldoer skal kunne give brugeren et hurtigt overblik, over sin kontosaldo igennem dagene.

BRUGERHISTORIER

Jeg har til mit program valgt at lave 3 brugerhistorier. Brugerhistorierne vil jeg bruge til at beskrive nogle mere detaljerede krav og en beskrivelse af hvordan brugeren skal interagere med programmet. Jeg har her valgt at beskrive de 2 brugerhistorier, som står for hoveddelen af mit program. I Bilag 1 har jeg den sidste brugerhistorie, som beskriver, hvordan brugeren tilføjer og fjerner sit job. I afsnittet ”Udførelse af iterationer” vil jeg beskrive, hvordan jeg har udført de forskellige brugerhistorier igennem forskellige iterationer.

BRUGERHISTORIE 1 - LOGIN-SYSTEM

Denne brugerhistorie kan kun startes når programmet bliver åbnet af brugeren.

- Brugeren åbner programmet
- Programmet registrerer at brugeren ikke er logget ind, og viser skærmen for login
- Hvis brugeren har en konto
    - Brugeren indtaster sit brugernavn og adgangskode i de tilhørende felter
    - Brugeren trykker på ”login”
    - Programmet logger brugeren ind og viser hovedsiden for programmet
- Hvis brugeren ikke har en konto
    - Brugeren trykker på knappen ”Don’t have an account? Sign up here”
    - Programmet viser ”Sign up” skærmen
    - Brugeren indtaster sit brugernavn, fornavn, efternavn, adgangskode, bekræftelse af adgangskode og e-mail ind i de tilhørende felter
    - Brugeren trykker på ”Create account”
    - Programmet tjekker om brugernavnet er ledigt
    - Hvis brugernavnet ikke er ledigt
        - Programmet laver en fejl, hvor der bliver beskrevet hvad fejlen er. Teksten ”username” bliver vist med rødt. Brugeren kan ændre brugernavn og prøve igen
    - Hvis brugernavnet er ledigt
        - Programmet tjekker om felterne ”password” og ”confirm password” er ens

        - Hvis felterne er ens
            - Programmet opretter brugerens konto og viser login siden
        - Hvis felterne ikke er ens
            - Programmet laver en fejl, hvor der bliver beskrevet hvad fejlen er, og felterne der er forkerte, bliver vist med rødt. Brugeren kan rette felterne og prøve igen

BRUGERHISTORIE 2 - HVOR MANGE PENGE HAR BRUGEREN BRUGT ELLER FÅET

Denne brugerhistorie kan kun startes når brugeren er logget ind på programmet, og befinder sig på programmets hovedside.

- Hvis brugeren vil tilføje/fjerne penge
    - Brugeren vælger en kategori i dropdown-menuen ved ”Select category for obtained/used money”
    - Brugeren finder inputfeltet ved enten ”Money obtained” / ”Money used”, og indtaster hvor mange penge brugeren har fået / brugt
    - Brugeren trykker på knappen ”Add money obtained” eller ”Add money used”
    - Programmet tjekker om inputfeltet kun består af tal
    - Hvis feltet ikke kun består af tal
        - Programmet laver en fejl og skriver på skærmen ”Please make sure you only used numbers!”, og teksten ved inputfeltet bliver vist med rødt. Brugeren kan rette og prøve igen.
    - Hvis feltet kun består af tal
        - Programmet tjekker om brugeren har valgt en kategori
        - Hvis brugeren ikke har valgt en kategori
            - Programmet laver en fejl og skriver på skærmen ”Please select a category!”, og teksten ved dropdown-menuen bliver vist med rødt. Brugeren kan rette og prøve igen.
        - Hvis brugeren har valgt en kategori
            - Programmet modtager tallet fra inputfeltet og kategorien, og indsætter i databasen for brugeren.

- Programmet opdaterer programmets GUI, og grafen for brugerens saldo ved den nuværende dato er opdateret og viser den nye kontosaldo.

ITERATIONER

Til programmet har jeg valgt at lave 3 iterationer. Iterationerne skal hjælpe mig med at lave programmet i en rækkefølge, som giver mening. Iterationerne hjælper også med at tjekke programmets kvalitet, da hver iteration skal testes og godkendes af programmøren, før man starter på den næste iteration. Iteration 1, som er login-system findes i Bilag 2.

ITERATION 2 - HOVEDMENU

Den anden iteration kommer til at omhandle ”Brugerhistorie 2 - Hvor mange penge har brugeren brugt eller fået” og ”Brugerhistorie 3 - Add job / Remove job”. Disse to brugerhistorier er med til at danne hovedmenuen for mit program, så de er helt essentielle.

ITERATION 3 - GRAF

Den tredje og sidste iteration kommer til at omhandle ”Krav 3 - Visning af økonomi”. Dette er det sidste krav, og er et krav, som ikke er inddækket ordentligt af brugerhistorierne, da det er noget programmet skal gøre automatisk.

UDFØRELSE AF PROGRAMMET

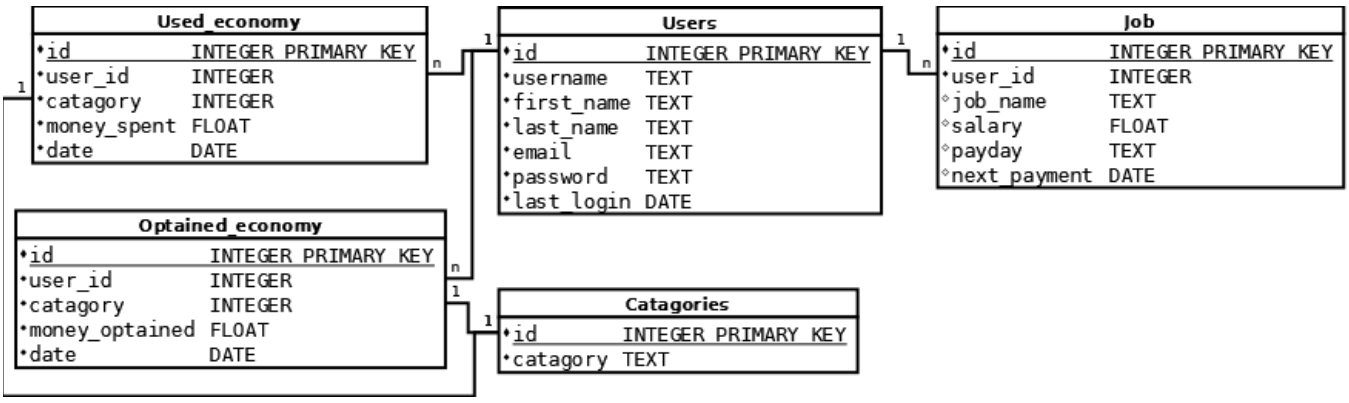
I dette afsnit kommer jeg ind på, hvordan jeg har fået udført mit program. Dette kommer til at omhandle både, hvordan jeg har opsat min database, men også hvordan de forskellige iterationer er blevet udført.

TRE-LAGS MODELLEN

Mit program kan opbygges efter ”Tre-lags modellen”, se Figur 1. Mit program har blandt andet en brugerflade, applikations-lag og data-lag. Min brugerflade i programmet bliver lavet med biblioteket ”Tkinter”, som er et bibliotek til at lave GUI i Python programmer. Et billede af programmets GUI kan ses på Figur 7. Mit programs applikations-lag er lavet i forskellige klasser, som har med hver del af programmet at gøre, gennemgang af klasserne kommer i afsnittet ”Oprettelse af klasserne”. Det sidste lag, som er data-laget, er også lavet i nogle klasser, som kan læses i samme afsnit. Derudover har jeg beskrevet opsætningen af min database, som er her alt dataen bliver gemt, i afsnittet ”Oprettelse af databasen”.

Figur 1, tre-lags modellen

OPRETTELSE AF DATABASEN



Figur 2, ER-diagram, figurern er vedhæftet under "Database_diagram.png" i "Images" mappen

Databasen er en helt essentiel del i mit program. Uden databasen ville hele mit program ikke komme til at virke, og heller ikke opfylde kravene til opgaveformuleringen.

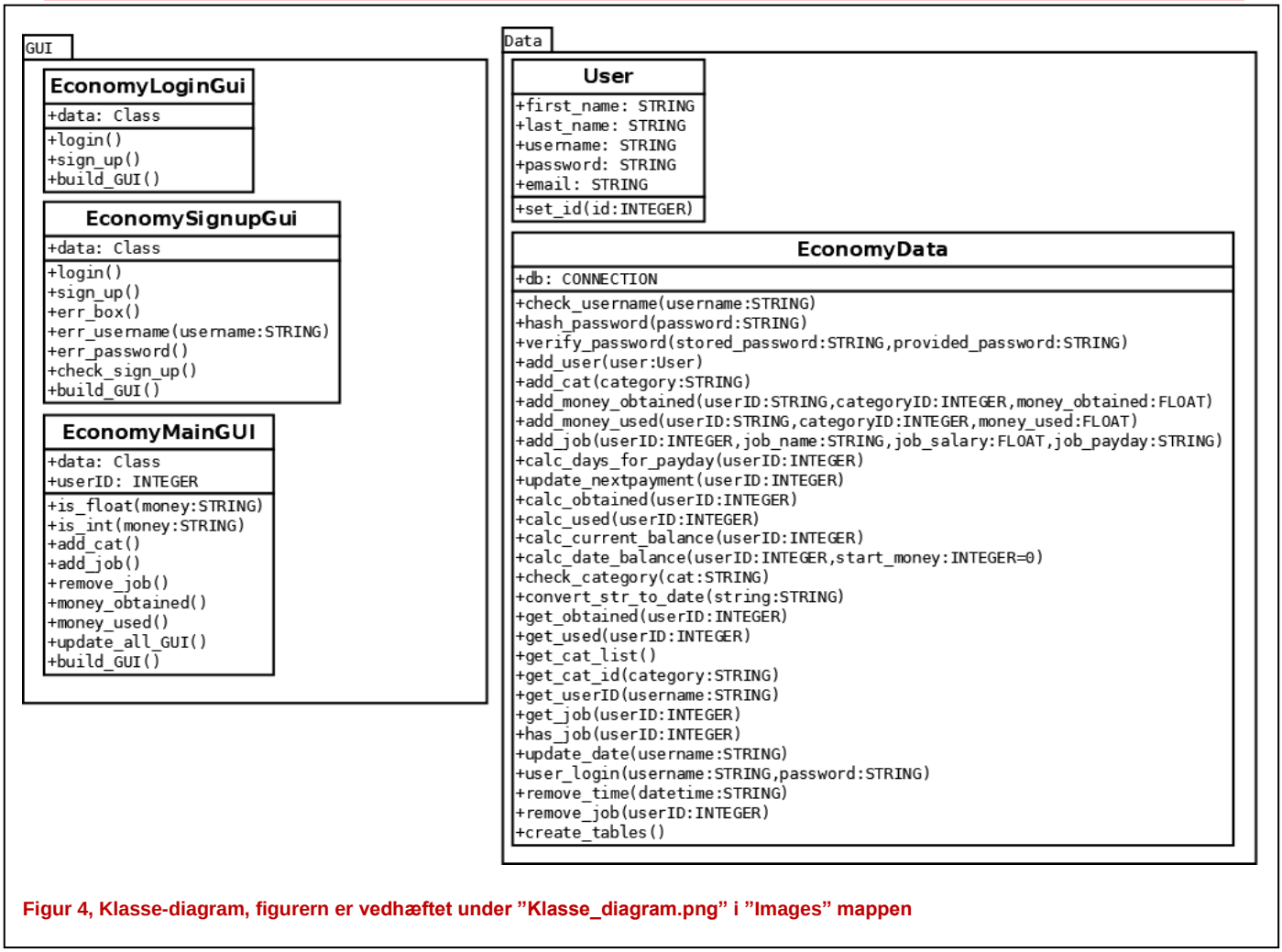
```python
def create_tables(self):
        c = self.db.cursor()

        try:
            c.execute("""DROP TABLE IF EXISTS users;""")
            c.execute("""DROP TABLE IF EXISTS used_economy;""")
            c.execute("""DROP TABLE IF EXISTS obtained_economy;""")
            c.execute("""DROP TABLE IF EXISTS category;""")
            c.execute("""DROP TABLE IF EXISTS job;""")
        except Exception as e:
            print(f'Error while deleting tables: {e}')

        try:
            c.execute("""CREATE TABLE IF NOT EXISTS users (
                id INTEGER PRIMARY KEY,
                username TEXT,
                first_name TEXT,
                last_name TEXT,
                email TEXT,
                password TEXT,
                last_login DATETIME NOT NULL DEFAULT CURRENT_DATE);""")
```

Figur 3, Eksempel på oprettelse af tabeller

For at lave min database, så har jeg lavet et ER-diagram, som kan ses på Figur 2. I mit program er der altså 5 tabeller, der hver især opbevarer forskellige data.

Til at lave databasen bruger jeg biblioteket ”SQLite3”, hvilket er et bibliotek som gør at man kan skrive SQL i Python, hvilket er det programmeringssprog databaser arbejder med. SQLite3 gør det muligt at lave en lokal database til ens program, så informationer kan gemmes. For at lave tabellerne så køres funktionen ”create_tables()”, se linje 1 på Figur 3. Denne funktion starter med at slette alle nuværende tabeller i databasen i linje 4-9. Herefter i linje 14 - 21, vil programmet prøve at lave tabellen ”Users”, som skal indeholde brugeroplysninger til login-systemet. Det samme gør sig gældende for alle de andre tabeller i databasen.

OPRETTELSE AF KLASSERNE



Figur 4, Klasse-diagram, figurern er vedhæftet under ”Klasse_diagram.png” i ”Images” mappen

Klasser er en måde at opdele programmet i flere enkeltstående dele. For nemt at kunne arbejde med dataen fra databasen, så har jeg lavet klasser, som kun arbejder med dataen. Disse klasser kan ses på Figur 4 under ”Data”. Her er klassen ”User”, som skal lave brugeren af programmet til et objekt. Den anden klasse er ”EconomyData”, som

arbejder sammen med databasen. Det er blandt andet i denne klasse, at alle funktionerne til de forskellige brugerhistorier befinder sig. På Figur 4 under ”GUI” er der 3 forskellige klasser. Hver klasse er et ”vindue” i programmet. Det vil sige at vinduet, hvor brugeren kan logge ind er fra klassen ”EconomyLoginGui”, hvor selve hovedprogrammet bliver lavet af klassen ”EconomyMainGui”. Hver klasse har her også nogle specifikke funktioner, der har med deres område at gøre. Klassediagrammet, som er på Figur 4, bruges til at give et indblik over, hvordan programmet skal sættes op så programmet har den ønskede virkning.

UDFØRELSE AF ITERATIONER

ITERATION 2 - HOVEDMENU

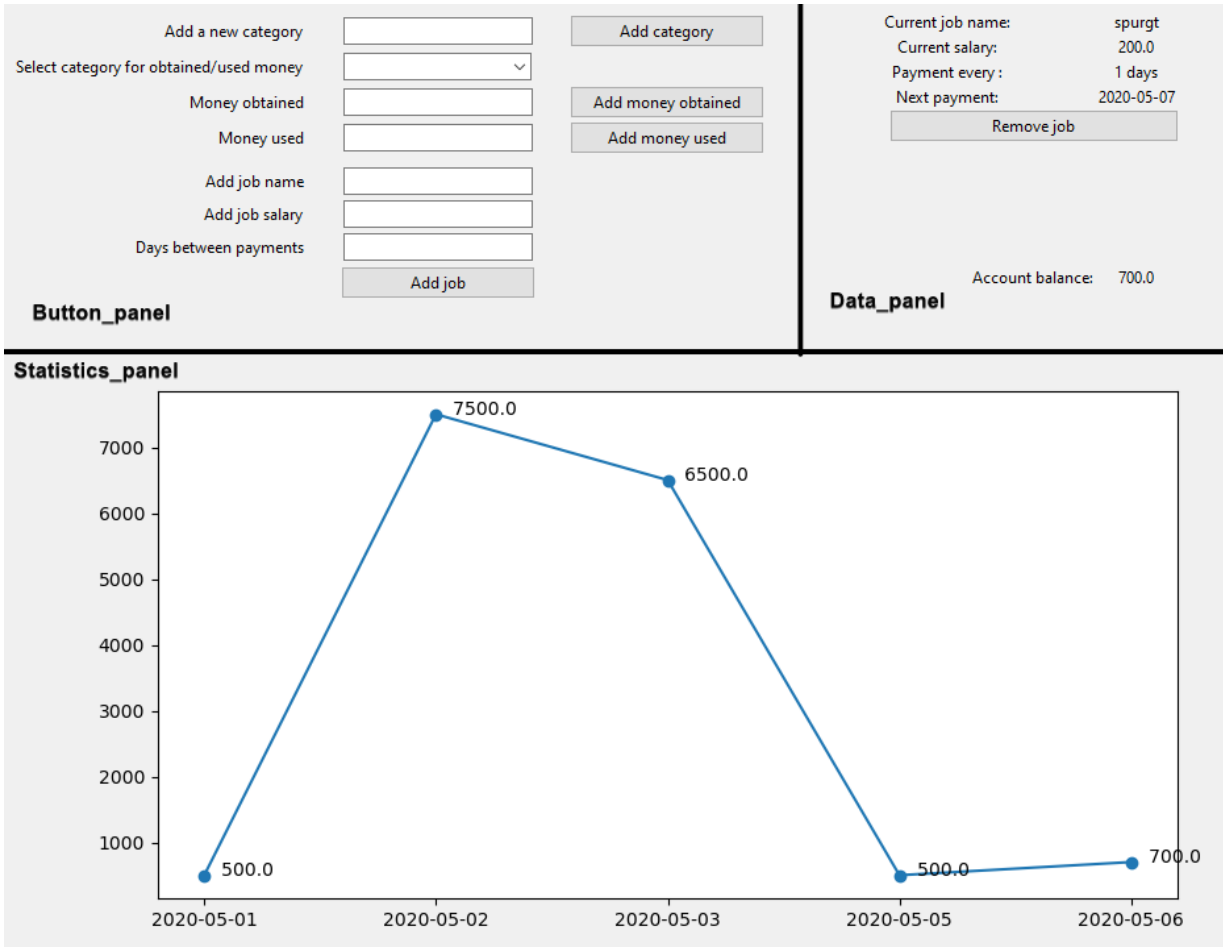
```
1 def mainGui(userID: str):
2     root = tk.Tk()
3     root.geometry('1920x1080')
4     root.state('zoomed')
5     app = EconomyMainGUI(userID, root)
6     app.master.title('Economy logged in')
7     app.mainloop()
```

Figur 5, funktionen mainGui(). Fil: EconomyGui.py

Hovedmenuen i mit program er lavet af biblioteket Tkinter. Jeg starter med funktionen ”mainGui()”, som kan ses på Figur 5. Denne funktion definerer hele vinduet som alt grafikken laves i. Jeg sætter størrelsen af vinduet til maksimal størrelse af min skærm, og så gør jeg den maksimeret så vinduet fylder hele min skærm, dette sker i linje 2-4. Herefter defineres hvilken klasse, som vinduet skal indlæse og vise på skærmen, som her er ”EconomyMainGui()”.

```python
def build_GUI(self):
    #Different variables etc
    self.button_panel = ttk.Frame(self)
    self.label_add_cat = ttk.Label(self.button_panel,
                           text = 'Add a new category')
    self.label_add_cat.grid(row = 1, column = 0,
                              padx = (113,0), pady = 2)
    self.button_panel.pack(side = tk.TOP)
    self.pack()
```

Figur 6, funktionen build_GUI(). Fil: EconomyGui.py



Figur 7, billede af hovedmenuen

I klassen findes funktionen ”build_GUI()”, det er denne funktion, som tegner alt hvad brugeren kan se på skærmen. På Figur 6 er der et lille eksempel på funktionen, som her laver en tekst i toppen af skærmen. Hovedmenuen af mit program er delt op i tre paneler.

Panelerne er henholdsvis ”button_panel”, ”data_panel” og ”statistics_panel”. Disse tre paneler indeholder de forskellige elementer der skal vises på skærmen. Button_panel, er det panel, der indeholder alle de inputfelter som brugeren har, samt tilhørende knapper så brugeren kan tilføje, job, kategorier, indtjente penge og brugte penge. Data_panel er det panel, der viser forskellige informationer. I programmet viser data_panel informationerne om brugerens job, samt brugerens kontosaldo. Statistics_panel viser grafen over brugerens kontosaldo, som bliver gennemgået i den næste iteration ”Iteration 3 - graf”. På Figur 7 ses et billede af hovedmenuen opdelt i de tre forskellige paneler.

BUTTON_PANEL

```
1 def money_obtained(self):
2         money_obtained = self.entry_money_obtained.get()
3         money_obtained = self.is_float(money_obtained)
4         if money_obtained == False:
5             self.label_error.config(text = 'Please make sure you only used numbers!')
6             self.label_money_obtained.config(foreground = 'red')
7             self.label_sel_cat.config(foreground = 'black')
8         else:
9             category = self.combo_sel_cat.get()
10            if category == "":
11                self.label_error.config(text = 'Please select a category!')
12                self.label_sel_cat.config(foreground = 'red')
13                self.label_money_obtained.config(foreground = 'black')
14            else:
15                categoryID = self.data.get_cat_id(category)
16                if self.data.add_money_obtained(self.userID, categoryID, money_obtained):
17                    self.entry_money_obtained.delete(0, tk.END)
18                    self.combo_sel_cat.set('')
19                    self.label_money_obtained.config(foreground = 'black')
20                    self.label_sel_cat.config(foreground = 'black')
21                    self.update_all_GUI()
22
```

Figur 8, funktionen money_obtained(). Fil: EconomyGui.py

Button_panel er det panel, som brugeren kommer til at interagerer mest med. Det er her brugeren skal tilføje og fjerne alle de forskellige informationer, som programmet skal modtage. Brugeren kan på dette panel tilføje forskellige kategorier, som brugeren vil bruge når der skal tilføjes eller fjernes penge fra kontoen. Det er et krav et brugeren vælger sådan en kategori, hvilket er grundlaget for, at de funktioner er øverst på skærmen. Herefter kan brugeren tilføje hvor mange penge brugeren har fået eller brugt. Dette sker igennem funktionen ”money_obtained()” og funktionen ”money_used()”. Begge funktioner

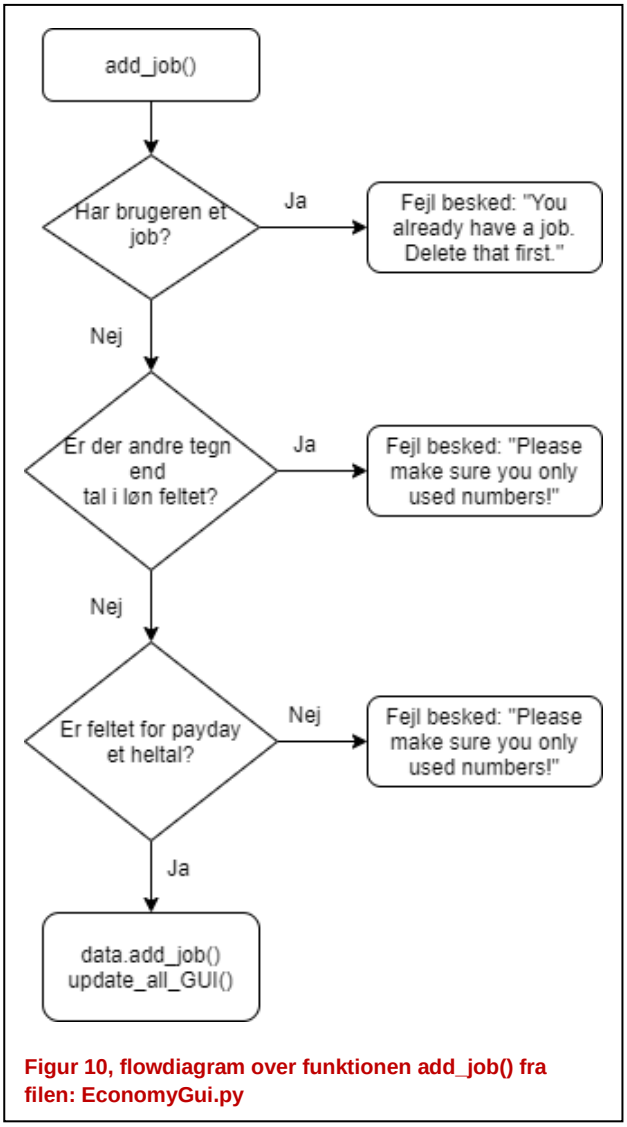
er 100% ens, de indsætter bare forskellige steder i databasen. På Figur 8 ses funktionen for money_obtained(). Denne funktion starter med at modtage det beløb brugeren har skrevet ind. For at være sikker på at brugeren kun har skrevet tal ind i inputboxen så køres funktionen ”is_float()”, i linje 3. Denne funktion tjekker bare om tallet kan laves til et komma tal. Hvis der er bogstaver, vil programmet lave en fejl på skærmen. Ellers så vil programmet modtage den kategori som brugeren har valgt, hvis ingen er valgt melder programmet en fejl. Se linje 9-13.

```
1 def add_money_obtained(self, userID: str, categoryID: int, money_obtained: float):
2         userID = userID
3         categoryID = categoryID
4         money_obtained = money_obtained
5         c = self.db.cursor()
6         c.execute("""INSERT INTO obtained_economy (user_id, category, money_obtained)
7         VALUES (?, ?, ?);""", (userID, categoryID, money_obtained))
8         self.db.commit()
9         return True
```

Figur 9, funktionen add_money_obtained(). Fil: econodata.py

Hvis det hele er som det skal være, så køres funktionen ”data.add_money_obtained()”, der sætter pengene ind i databasen og herefter genindlæser programmets GUI, for at vise ændringerne. Se linje 15-21. Funktionen ”add_money_obtained()” fra linje 16, kan ses på Figur 9. Funktionen modtager brugerens id, kategorien og pengene, som så bliver indsat i databasen i linje 6-7.

Brugeren kan på hovedmenuen også tilføje sit job. På Figur 10 ses et flowdiagram over funktionen add_job(). Flowdiagrammet viser, hvordan funktionen tjekker om alle de oplysninger som brugeren har indtastet, lever op til programmets krav. Hvis kravene ikke er opfyldt, kommer der fejlbeskeder på skærmen. Er kravene opfyldt så køres funktionerne ”data.add_job()” og ”update_all_GUI()”, som skal opdatere hele programmets brugerflade. På Figur 11 ses funktionen ”add_job()” (data.add_job()), der er i filen ”econodata.py”. Funktionen modtager alle de forskellige informationer om jobbet i linje 2-5. Herefter modtager funktionen den nuværende dato, som computeren kører med. Dette sker i linje 6 med biblioteket datetime. Denne dato bruges så i linje 7, hvor programmet udregner hvornår brugeren skal have løn næste gang. Alle disse informationer bliver så tilføjet til databasen i linje 9-10.



Figur 10, flowdiagram over funktionen add_job() fra filen: EconomyGui.py

```
1    def add_job(self, userID: int, job_name: str, job_salary: float, job_payday: str):
2        userID = userID
3        job_name = job_name
4        job_salary = job_salary
5        job_payday = job_payday
6        current_date = datetime.date.today()
7        next_payment = current_date + datetime.timedelta(days = job_payday)
8        c = self.db.cursor()
9        c.execute("""INSERT INTO job (user_id, job_name, salary, payday, next_payment)
10       VALUES (?, ?, ?, ?, ?);""", (userID, job_name, job_salary, job_payday, next_payment))
11       self.db.commit()
12       return True
```

Figur 11, funktionen add_job(). Fil: econodata.py

```
1 def calc_days_for_payday(self, userID: int):
2         c = self.db.cursor()
3         c.execute("""SELECT last_login FROM users WHERE id = ?;""", (userID,))
4         l = c.fetchone()
5         ll = self.convert_str_to_date(l[0])
6         c.execute("""SELECT salary, next_payment, payday FROM job WHERE
7         user_id = ?;""", (userID,))
8         k = c.fetchone()
9         kk = self.convert_str_to_date(k[1])
10        if kk <= ll:
11            self.add_money_obtained(userID, 3, k[0])
12            if self.update_nextpayment(userID) == False:
13                return
```

Figur 12, funktionen calc_days_for_payday(). Fil: econodata.py

Programmet kan ud fra brugerens job automatisk tilføje den løn, som brugeren får af jobbet, hvis den dato brugeren logger ind med, er den samme dato som den nuværende dato, eller er efter datoen for næste lønmodtagelse. Dette sker i funktionen ”calc_days_for_payday()”, som kan ses på Figur 12. Funktionen tjekker her, hvornår brugeren sidst loggede ind, se linje 3. Eftersom datoen i databasen er en STRING, så skal den laves om til et date objekt. Dette sker i funktionen ”convert_str_to_date()”, i linje 5. Herefter modtager programmet informationerne om brugerens job. Her bliver datoen ”next_payment” også lavet om til et date objekt. Programmet tjekker herefter om login datoen er mindre end eller lig med datoen fra next_payment. Hvis dette er sandt, så bliver brugerens løn tilføjet til databasen, hvor programmet så opdatere datoen for hvornår brugeren skal have løn. Dette sker i funktionen ”update_nextpayment()”, se linje 12.

```
1  def update_nextpayment(self, userID: int):
2          c = self.db.cursor()
3          c.execute("""SELECT next_payment, payday FROM job WHERE user_id = ?;""",
4          (userID,))
5          p = c.fetchone()
6          current_date = datetime.date.today()
7          oldpayment = self.convert_str_to_date(p[0])
8          next_payment = oldpayment + datetime.timedelta(days = int(p[1]))
9          c.execute("""UPDATE job SET next_payment = ? WHERE user_id = ?;""",
10         (next_payment, userID))
11         self.db.commit()
12         if next_payment < current_date:
13             self.calc_days_for_payday(userID)
14         else:
15             return False
```

Figur 13, funktionen update_nextpayment(). Fil: econodata.py

På Figur 13, ses funktionen ”update_nextpayment()”. Funktionen modtager her alle informationerne om brugerens job, og udregner så, hvornår brugeren skal have løn igen, dette sker i linje 3-8. Herefter bliver den nye dato indsat i databasen. Programmet tjekker her om den nye dato er mindre end login-datoen for brugeren. Hvis den er det, så skal brugeren have løn igen, hvilket er grunden til at funktionen calc_days_for_payday() køres igen i linje 13. Hvis datoen er over login-datoen, så modtager brugeren ikke løn længere.

ITERATION 3 - GRAF

```
1     def calc_date_balance(self, userID: int, start_money: int = 0):
2         #Får alt indtjent økonomi med datoer som et dict
3         obtained = self.get_obtained(userID)
4         #Får alt brugt økonomi med datoer som et dict
5         used = self.get_used(userID)
6         #Laver et dict med obtained dict
7         balance_dict = obtained
8         #For hver dato (key) i used
9         for use in used:
10            # Hvis datoen ikke er i modtaget dict (balance_dict)
11            if not use in balance_dict:
12                # Tilføj værdien (value) til den dato (key) til balance_dict, da det er den
13                # brugte mængde for den dato
14                balance_dict[use] = -used[use]
15            else:
16                # Træk den brugte mængde(value) fra, hvad der ellers var tilføjet på den
17                # dato(key)
18                balance_dict[use] -= used[use]
19        # https://stackoverflow.com/questions/34129391/sort-python-dictionary-by-date-key
20        # Sorter dict, så den første dato kommer først. For at kunne beregne ændringen i
21        # saldo, for dagene der går
22        balance_dict = OrderedDict(sorted(balance_dict.items(),
23        key = lambda x:dt.strptime(x[0], "%Y-%m-%d")))
24        # Løber igennem alle dagene(key), hvor der er sket ændringer
25        for balance in balance_dict:
26            # Tilføjer den forrige mængde penge til dagen. Dette beregner den totale mængde
27            # penge på kontoen for datoen
28            balance_dict[balance] += start_money
29            # Opdater saldoen for dagen, for at kunne regne videre for næste dato
30            start_money = balance_dict[balance]
31        x = []
32        y = []
33        for bal in balance_dict:
34            x.append(bal)
35            y.append(balance_dict[bal])
36        return x,y
```

Figur 14 funktionen calc_date_balance(). Fil: econodate.py

På Figur 7 ved ”statistics_panel”, kan man se grafen, som bliver vist over brugerens konto saldo. Grafen er lavet med biblioteket ”matplotlib”. Matplotlib er et bibliotek, der kan bruges til at vise alle mulige funktioner, og forskellige former for statistik i Python. Jeg har her lavet et linjediagram, som har datoen på x-aksen og kontosaldoen på y-aksen. Dataen, som vises på grafen, kommer fra funktionen ”calc_date_balance()”, som kan ses på Figur 14. Funktionen modtager i linje 3 og 5, alle udgifter og indtjeninger for hver dag. Programmet modtager disse informationer i et dict, hvor datoen er en ”key”, hvor saldoen for den dato så er en ”value”. Funktionen udregner så for hver enkel dato i hvert dict, en samlet kontosaldo, som også er baseret fra saldoen dagen før, hvor det så bliver lagt

sammen til ét dict. Det dict med alle informationerne bliver så sorteret efter datoen, af funktionen ”OrderedDict()”, som kommer fra biblioteket ”Collect”. Se linje 9-30. Herefter laver jeg to lister, hvor den ene skal være x-aksens koordinater, og den anden er så y-aksens koordinater. Disse koordinater kommer så ud fra det dict, der blev lavet i funktionen.

TEST AF PROGRAMMET

For at være sikker på at mit program opfylder de stillede krav og brugerhistorier, har jeg lavet nogle tests. Jeg har lavet en test med mig selv, og derefter har jeg fået to personer til at teste mit program.

TEST 1 - MIN EGEN TEST

Da jeg testede mit program, var der nogle fejl. Jeg fandt ud af at min funktion til at tjekke om brugernavnet allerede eksisterede, ikke virkede korrekt. Jeg havde lavet en fejl i funktionen, som gjorde at programmet kun tjekkede om brugernavnet passede med det første brugernavn i databasen, og dermed ikke tjekkede alle brugernavnene igennem. Dette fik jeg hurtigt ændret. Jeg skulle også ændre det i funktionen til at tjekke om en kategori allerede eksisterede, da det er den helt samme funktion, der bare arbejder med to forskellige tabeller i min database. Ud over dette, fandt jeg ud af at programmet ikke opdaterede grafen korrekt, og lavede bare en ny graf oven i den anden så der var to grafer på skærmen, på samme tid. Jeg fik løst dette, ved at få programmet til at opdatere alt GUIen, dette sikrer også, at det hele bliver opdateret.

TEST 2 - TEST PERSON 1

Den første person, som jeg fik til at teste mit program, kan virkelig se ideen bag programmet, og se nogle af de muligheder, som programmet kunne give. Personen havde dog svært ved at finde ud af, hvordan programmet fungere, eftersom der ikke er nogen forklarende tekster, ud over det man kan se på Figur 7. Personen foreslog her at man kunne lave et infoikon, som ville vise en forklarende tekst når man holder musen hen over. Personen fandt også ud af, at grafen ikke viste noget den allerførste dag man brugte programmet, eftersom programmet ikke havde nok punkter til at kunne lave en linje. Dette

har jeg ændret ved at grafen nu viser at man havde 0 kr. på sin konto for dagen før man begyndte at bruge programmet.

TEST 3 - TEST PERSON 2

Den anden person, som jeg fik til at teste mit program, synes at der et eller andet sted skulle stå et navn for programmet, på hovedmenuen. Personen sagde at han ikke vidste, hvad man kunne med programmet. Personen synes også at det var irriterende at personen skulle trykke på ”add catagory”, for bagefter at kunne bruge kategorien til at tilføje indtjeninger / udgifter. Personen foreslog her, at man kunne få programmet til at tilføje kategorien når man trykker på knappen ”add money obtained” eller ”add money used”, ud over knappen ”add category”.

TEST OPSUMMERING

Programmet efter de 3 tests, opfylder de 3 stillede krav til programmet.

Programmet opfylder ”Krav 1 - GUI” ved at programmet har en GUI, hvor brugeren har fuld interaktion med programmet. Programmet opfylder også ”Krav 2 - Administrering af økonomi” og ”Krav 3 - Visning af økonomi” da programmet kan udregne brugerens kontosaldoer for de forskellige dage, ud fra indtjeninger og udgifter, hvor dette bliver vist på grafen, samt så kan programmet også arbejde sammen med et job.

PROGRAMMETS FREMTID

Programmets fremtid kan være meget bredt. Programmet kunne komme til at vise de kategorier, som brugerne har valgt når de har tilføjet indtjeninger og udgifter på en graf. Dette vil gøre, at der faktisk er en mening med at have denne funktionalitet i mit program. Herudover kunne man arbejde på at lave GUIen pænere end hvad den er nu, samt lave nogle andre muligheder for at kunne opdatere GUIen, så hele GUIen ikke forsvinder og så kommer igen. Til sidst kunne man arbejde med at implementere de funktionaliteter, som ”test person 2” foreslog med at kategorien automatisk blev tilføjet, og så kunne man også lave forklarende tekster til programmet.

BESKRIVELSE AF ARBEJDSPROCESSEN

TRELLO

Jeg har i mit projekt, haft brug for at have et overblik over de delopgaver, som jeg skulle arbejde med. Jeg har derfor brugt hjemmesiden trello.com, til at hjælpe mig med at holde et overblik over mit projekt, hvilket har sikret mig en let tilgang til at holde styr på, hvor meget programmet mangler osv. Man kan på de forskellige elementer angive hvornår de skal være færdige, så man ender med at bruge tiden fornuftigt.

TIDSPLAN

Jeg har lavet en tidsplan over udviklingen af programmet, som kan ses herunder.

Moduler (12 moduler i alt)	**2**	**4**	**2**	**1**	**3**
Arbejdsopgaver	Krav specifikation til programmet	Program opsætning	Database opsætning	Test af programmet	Synopsis skrivning

Jeg har her opdelt min tidsplan i forskellige arbejdsopgaver, hvor jeg så har angivet hvor mange moduler jeg vil bruge på hver arbejdsopgave.

LITTERATURLISTE

Molina, A. (2018, september 20). *Hashing Passwords in Python*. Retrieved maj 06, 2020, from vitoshacademy: https://www.vitoshacademy.com/hashing-passwords-in-python/

pythonprogrammring. (n.d.). *How to embed a Matplotlib graph to your Tkinter GUI*. Retrieved from pythonprogrammring: https://pythonprogramming.net/how-to-embed-matplotlib-graph-tkinter-gui/

BILAG

BILAG 1

BRUGERHISTORIE 3 - ADD JOB / REMOVE JOB

Denne brugerhistorie kan kun startes når brugeren er logget ind på programmet, og befinder sig på programmets hovedside.

- Hvis brugeren vil tilføje et job
    - Programmet tjekker om brugeren allerede har et job
    - Hvis brugeren ikke har et job
        - Brugeren skriver navnet på brugerens job i inputfeltet ved ”Add job name”
        - Brugeren skriver lønnen som brugeren får i inputfeltet ved ”Add job salary”
        - Brugeren skriver dagene der er mellem hver gang brugeren får løn i inputfeltet ved ”Days between payments”
        - Brugeren trykker på knappen ”Add job”
        - Programmet tjekker om alle 3 felter er udfyldte
            - Hvis der er én eller flere felter der ikke er udfyldte
                - Programmet laver en fejl og skriver en passende fejltekst alt efter hvilke felter der mangler. Teksten ved inputfeltet bliver markeret med rødt, og brugeren kan rette og prøve igen.
            - Hvis alle felter er udfyldte
                - Programmet tjekker om der kun er tal i inputfelterne ved ”Add job salary” og ”Days between payments” (”Days between payments må kun være heltal”).
                - Hvis felterne ikke kun består af tal
                    - Programmet laver en fejl og skriver ”Please make sure you only used numbers!”, og teksten ved inputfeltet bliver markeret med rødt. Brugeren kan rette oplysningerne og prøve igen.

        - Hvis felterne kun består af tal
            - Programmet modtager de forskellige informationer fra inputfelterne og tilføjer dem til databasen.
            - Programmet opdaterer programmets GUI og viser nu oplysningerne for brugerens job.
            - Programmet udregner hvornår brugeren får løn næste gang og viser datoen for næste løn på skærmen.
    - Hvis brugeren har et job
        - Programmet laver en fejl og skriver ”You already have a job. Remove your job before adding a new job.”
        - Brugeren kan fjerne sit job og prøve igen.
- Hvis brugeren vil fjerne sit job
    - Brugeren trykker på knappen ”Remove job”.
    - Programmet fjerner brugerens job oplysninger fra databasen op opdaterer programmets GUI så informationer om brugerens job ikke vises længere.

BILAG 2

ITERATION 1 - LOGIN SYSTEM

Den første iteration kommer til at omhandle ”Brugerhistorie 1 - Login-system”, hvor selve login systemet laves. Der er altså kun én brugerhistorie med i denne iteration.

BILAG 3 - ECONOMYGUI.PY

```
1.  from econodata import EconomyData, User
2.  import matplotlib
3.  matplotlib.use("TkAgg")
4.  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
5.  from matplotlib.figure import Figure
6.  import tkinter as tk
7.  import tkinter.ttk as ttk
8.
9.  class EconomyLoginGui(ttk.Frame):
10.     def __init__(self, master = None):
11.         ttk.Frame.__init__(self, master)
12.         self.data = EconomyData()
13.         # self.data.create_tables()
14.         self.build_GUI()
```

```
15.
16.     def login(self):
17.         username = self.entry_username.get()
18.         password = self.entry_password.get()
19.         if self.data.check_username(username):
20.             self.label_error.config(text = 'Wrong username')
21.         else:
22.             if self.data.user_login(username, password):
23.                 userID = self.data.get_userID(username)
24.                 self.label_error.config(text = '')
25.                 self.master.destroy()
26.                 mainGui(userID)
27.             else:
28.                 self.label_error.config(text = 'Wrong password')
29.
30.     def sign_up(self):
31.         self.master.destroy()
32.         signUpGui()
33.
34.     def build_GUI(self):
35.         self.label_username = ttk.Label(self, text = 'Username:')
36.         self.label_password = ttk.Label(self, text = 'Password:')
37.         self.entry_username = ttk.Entry(self)
38.         self.entry_password = ttk.Entry(self, show = "*")
39.         self.button_login = ttk.Button(self, text = 'Login', command = self.login)
40.         self.button_create = ttk.Button(self, text = '''''Don't have an account?\nSign up
    here''', command = self.sign_up)
41.         self.label_error = ttk.Label(self, text = '', foreground = 'red')
42.
43.         self.label_username.grid(row = 1, column = 0)
44.         self.entry_username.grid(row = 2, column = 0)
45.         self.label_password.grid(row = 3, column = 0)
46.         self.entry_password.grid(row = 4, column = 0)
47.         self.button_login.grid(row = 6, column = 0, pady = 10)
48.         self.label_error.grid(row = 0, column = 0)
49.         self.button_create.grid(row = 10, column = 0)
50.
51.         self.pack()
52.
53.
54. class EconomySignupGui(ttk.Frame):
55.     def __init__(self, master = None):
56.         ttk.Frame.__init__(self, master)
57.         self.data = EconomyData()
58.         self.build_GUI()
59.
60.     def login(self):
61.         self.master.destroy()
62.         loginGui()
63.
64.     def sign_up(self):
65.         self.username = self.entry_username.get()
66.         self.first_name = self.entry_first_name.get()
67.         self.last_name = self.entry_last_name.get()
68.         self.email = self.entry_email.get()
69.         self.password = self.entry_password.get()
70.         if self.username == "" or self.first_name == "" or self.last_name == "" or self.em
    ail == "" or self.password == "":
71.             self.err_box()
72.         else:
73.             self.user = User(self.first_name, self.last_name, self.username, self.email, s
    elf.password)
74.             self.data.add_user(self.user)
```

```
75.             self.login()
76.
77.     def err_box(self):
78.         self.label_error.config(text = "Please fill out all entries!", foreground = "red")
79.         self.label_password.config(foreground = "black")
80.         self.label_conf_password.config(foreground = "black")
81.         self.label_username.config(foreground = "black")
82.
83.     def err_username(self, username):
84.         self.label_error.config(text = "Username already taken!", foreground = "red")
85.         self.label_username.config(foreground = "red")
86.
87.     def err_password(self):
88.         self.label_error.config(text = "Passwords aren't the same!", foreground = "red")
89.         self.label_password.config(foreground = "red")
90.         self.label_conf_password.config(foreground = "red")
91.         self.label_username.config(foreground = "black")
92.
93.     def check_sign_up(self):
94.         self.username = self.entry_username.get()
95.         if not self.data.check_username(self.username):
96.             self.err_username(self.username)
97.         else:
98.             self.password = self.entry_password.get()
99.             self.conf_password = self.entry_conf_password.get()
100.                 if self.password == self.conf_password:
101.                     self.sign_up()
102.                 else:
103.                     self.err_password()
104.
105.         def build_GUI(self):
106.             self.label_username = ttk.Label(self, text = 'Username:')
107.             self.label_first_name = ttk.Label(self, text = 'First name:')
108.             self.label_last_name = ttk.Label(self, text = 'Last name:')
109.             self.label_email = ttk.Label(self, text = 'Email:')
110.             self.label_password = ttk.Label(self, text = 'Password:')
111.             self.label_conf_password = ttk.Label(self, text = 'Confirm Password:')
112.             self.label_error = ttk.Label(self, text = "")
113.             self.entry_username = ttk.Entry(self)
114.             self.entry_first_name = ttk.Entry(self)
115.             self.entry_last_name = ttk.Entry(self)
116.             self.entry_email = ttk.Entry(self)
117.             self.entry_password = ttk.Entry(self, show = "*")
118.             self.entry_conf_password = ttk.Entry(self, show = "*")
119.             self.button_create_acc = ttk.Button(self, text = 'Create account', command =
    self.check_sign_up)
120.             self.button_login = ttk.Button(self, text = 'Have an account?\nLogin', comma
   nd = self.login)
121.             self.label_error.grid(row = 0, column = 0)
122.             self.label_username.grid(row = 1, column = 0)
123.             self.entry_username.grid(row = 2, column = 0)
124.             self.label_first_name.grid(row = 3, column = 0)
125.             self.entry_first_name.grid(row = 4, column = 0)
126.             self.label_last_name.grid(row = 5, column = 0)
127.             self.entry_last_name.grid(row = 6, column = 0)
128.             self.label_email.grid(row = 7, column = 0)
129.             self.entry_email.grid(row = 8, column = 0)
130.             self.label_password.grid(row = 9, column = 0)
131.             self.entry_password.grid(row = 10, column = 0,)
132.             self.label_conf_password.grid(row = 11, column = 0)
133.             self.entry_conf_password.grid(row = 12, column = 0)
134.
```

```
135.                self.button_create_acc.grid(row = 13, column = 0, pady = 10)
136.                self.button_login.grid(row = 14, column = 0)
137.                self.pack()
138.
139.        class EconomyMainGUI(ttk.Frame):
140.            def __init__(self, userID, master = None):
141.                ttk.Frame.__init__(self, master)
142.                self.data = EconomyData()
143.                self.userID = userID
144.                self.build_GUI()
145.
146.            def is_float(self, money):
147.                try:
148.                    money = float(money)
149.                    return money
150.                except ValueError:
151.                    return False
152.
153.            def is_int(self, money):
154.                try:
155.                    money = int(money)
156.                    return money
157.                except ValueError:
158.                    return False
159.
160.            def add_cat(self):
161.                self.category = self.entry_add_cat.get()
162.                if not self.data.check_category(self.category):
163.                    self.label_error.config(text = 'This category already exists!')
164.                else:
165.                    self.data.add_cat(self.category)
166.                    self.update_all_GUI()
167.
168.            def add_job(self):
169.                if self.data.has_job(self.userID):
170.                    self.label_error.config(text = 'You already have a job. Delete that firs
    t.')
171.                    self.entry_job_name.delete(0, tk.END)
172.                    self.entry_job_salary.delete(0, tk.END)
173.                    self.entry_job_payday.delete(0, tk.END)
174.                else:
175.                    job_name = self.entry_job_name.get()
176.                    job_salary = self.entry_job_salary.get()
177.                    job_payday = self.entry_job_payday.get()
178.                    job_payday = self.is_int(job_payday)
179.                    job_salary = self.is_float(job_salary)
180.                    if job_salary == False:
181.                        self.label_error.config(text = 'Please make sure you only used numbe
    rs!')
182.                        self.label_job_salary.config(foreground = 'red')
183.                    elif job_payday == False:
184.                        self.label_error.config(text = 'Please make sure you only used numbe
    rs!')
185.                        self.label_job_payday.config(foreground = 'red')
186.                        self.label_job_salary.config(foreground = 'black')
187.                    else:
188.                        if self.data.add_job(self.userID, job_name, job_salary, job_payday):

189.                            self.update_all_GUI()
190.
191.            def remove_job(self):
192.                self.data.remove_job(self.userID)
193.                self.update_all_GUI()
```

```
194.
195.         def money_obtained(self):
196.             money_obtained = self.entry_money_obtained.get()
197.             money_obtained = self.is_float(money_obtained)
198.             if money_obtained == False:
199.                 self.label_error.config(text = 'Please make sure you only used numbers!'
    )
200.                 self.label_money_obtained.config(foreground = 'red')
201.                 self.label_sel_cat.config(foreground = 'black')
202.             else:
203.                 category = self.combo_sel_cat.get()
204.                 if category == "":
205.                     self.label_error.config(text = 'Please select a category!')
206.                     self.label_sel_cat.config(foreground = 'red')
207.                     self.label_money_obtained.config(foreground = 'black')
208.                 else:
209.                     categoryID = self.data.get_cat_id(category)
210.                     if self.data.add_money_obtained(self.userID, categoryID, money_obtai
    ned):
211.                         self.entry_money_obtained.delete(0, tk.END)
212.                         self.combo_sel_cat.set('')
213.                         self.label_money_obtained.config(foreground = 'black')
214.                         self.label_sel_cat.config(foreground = 'black')
215.                         self.update_all_GUI()
216.
217.         def money_used(self):
218.             money_used = self.entry_money_used.get()
219.             money_used = self.is_float(money_used)
220.             if money_used == False:
221.                 self.label_error.config(text = 'Please make sure you only used numbers!'
    )
222.                 self.label_money_used.config(foreground = 'red')
223.                 self.label_sel_cat.config(foreground = 'black')
224.             else:
225.                 category = self.combo_sel_cat.get()
226.                 if category == "":
227.                     self.label_error.config(text = 'Please select a category!')
228.                     self.label_sel_cat.config(foreground = 'red')
229.                     self.label_money_used.config(foreground = 'black')
230.                 else:
231.                     categoryID = self.data.get_cat_id(category)
232.                     if self.data.add_money_used(self.userID, categoryID, money_used):
233.                         self.entry_money_used.delete(0, tk.END)
234.                         self.combo_sel_cat.set('')
235.                         self.label_money_used.config(foreground = 'black')
236.                         self.label_sel_cat.config(foreground = 'black')
237.                         self.update_all_GUI()
238.
239.         def update_all_GUI(self):
240.             self.statistics_panel.destroy()
241.             self.button_panel.destroy()
242.             self.data_panel.destroy()
243.             self.build_GUI()
244.
245.         def build_GUI(self):
246.             #Different variables etc
247.             self.button_panel = ttk.Frame(self)
248.             self.data_panel = ttk.Frame(self)
249.             self.statistics_panel = ttk.Frame(self)
250.             catagories = self.data.get_cat_list()
251.             self.button_panel.grid_columnconfigure(0, minsize = 200)
252.             self.button_panel.grid_columnconfigure(1, minsize = 200)
253.
```

```
254.                #Button_panel
255.                self.label_add_cat = ttk.Label(self.button_panel, text = 'Add a new category
    ')
256.                self.entry_add_cat = ttk.Entry(self.button_panel, width = 23)
257.                self.button_add_cat = ttk.Button(self.button_panel, text = 'Add category', c
    ommand = self.add_cat, width = 23)
258.                self.label_sel_cat = ttk.Label(self.button_panel, text = 'Select category fo
    r obtained/used money')
259.                self.combo_sel_cat = ttk.Combobox(self.button_panel, values = catagories, st
    ate = 'readonly', width = 20)
260.                self.label_money_obtained = ttk.Label(self.button_panel, text = 'Money obtai
    ned')
261.                self.entry_money_obtained = ttk.Entry(self.button_panel, width = 23)
262.                self.button_money_obtained = ttk.Button(self.button_panel, text = 'Add money
     obtained', command = self.money_obtained, width = 23)
263.                self.label_money_used = ttk.Label(self.button_panel, text = 'Money used')
264.                self.entry_money_used = ttk.Entry(self.button_panel, width = 23)
265.                self.button_money_used = ttk.Button(self.button_panel, text = 'Add money use
    d', command = self.money_used, width = 23)
266.                self.label_error = ttk.Label(self.button_panel, text = "", foreground = "red
    ")
267.                self.label_job_name = ttk.Label(self.button_panel, text = "Add job name")
268.                self.entry_job_name = ttk.Entry(self.button_panel, width = 23)
269.                self.label_job_salary = ttk.Label(self.button_panel, text = "Add job salary"
    )
270.                self.entry_job_salary = ttk.Entry(self.button_panel, width = 23)
271.                self.label_job_payday = ttk.Label(self.button_panel, text = "Days between pa
    yments")
272.                self.entry_job_payday = ttk.Entry(self.button_panel, width = 23)
273.                self.button_add_job = ttk.Button(self.button_panel, text = "Add job", comman
    d = self.add_job, width = 23)
274.
275.                self.label_add_cat.grid(row = 1, column = 0, padx = (113,0), pady = 2)
276.                self.entry_add_cat.grid(row = 1, column = 1, pady = 2)
277.                self.button_add_cat.grid(row = 1, column = 2, pady = 2, padx = (0,10))
278.                self.label_sel_cat.grid(row = 2, column = 0, pady = 2)
279.                self.combo_sel_cat.grid(row = 2, column = 1, pady = 2)
280.                self.label_money_obtained.grid(row = 3, column = 0, padx = (132,0), pady = 2
    )
281.                self.entry_money_obtained.grid(row = 3, column = 1, pady = 2)
282.                self.button_money_obtained.grid(row = 3, column = 2, pady = 2, padx = (0,10)
    )
283.                self.label_money_used.grid(row = 4, column = 0, padx = (154,0))
284.                self.entry_money_used.grid(row = 4, column = 1)
285.                self.button_money_used.grid(row = 4, column = 2, padx = (0,10))
286.                self.label_error.grid(row = 0, column = 1)
287.                self.label_job_name.grid(row = 5, column = 0, padx = (144,0), pady = (10,2))

288.                self.entry_job_name.grid(row = 5, column = 1, pady = (10,2))
289.                self.label_job_salary.grid(row = 6, column = 0, padx = (143,0), pady = 2)
290.                self.entry_job_salary.grid(row = 6, column = 1, pady = 2)
291.                self.label_job_payday.grid(row = 7, column = 0, padx = (91,0), pady = 2)
292.                self.entry_job_payday.grid(row = 7, column = 1, pady = 2)
293.                self.button_add_job.grid(row = 9, column = 1, pady = (2,10))
294.
295.                #Data_panel
296.                if self.data.has_job(self.userID):
297.                    job = self.data.get_job(self.userID)
298.                    self.data.calc_days_for_payday(self.userID)
299.                    self.label_djob_name = ttk.Label(self.data_panel, text = 'Current job na
    me:')
300.                    self.label_djob_name_v = ttk.Label(self.data_panel, text = f'{job[0]}')
```

```
301.                 self.label_djob_salary = ttk.Label(self.data_panel, text = 'Current sala
    ry:')
302.                 self.label_djob_salary_v = ttk.Label(self.data_panel, text = f'{job[1]}'
    )
303.                 self.label_djob_payday = ttk.Label(self.data_panel, text = 'Payment ever
    y :')
304.                 self.label_djob_payday_v = ttk.Label(self.data_panel, text = f'{job[2]}
    days')
305.                 self.label_djob_nextpayment = ttk.Label(self.data_panel, text = 'Next pa
    yment:')
306.                 self.label_djob_nextpayment_v = ttk.Label(self.data_panel, text = f'{job
    [3]}')
307.                 self.button_djob_remove = ttk.Button(self.data_panel, text = 'Remove job
    ', command = self.remove_job, width = 35)
308.                 self.label_djob_name.grid(row = 1, column = 0, padx = (0,0))
309.                 self.label_djob_name_v.grid(row = 1, column = 1, padx = (0,100))
310.                 self.label_djob_salary.grid(row = 2, column = 0, padx = (0,0))
311.                 self.label_djob_salary_v.grid(row = 2, column = 1, padx = (0,100))
312.                 self.label_djob_payday.grid(row = 3, column = 0, padx = (0,0))
313.                 self.label_djob_payday_v.grid(row = 3, column = 1, padx = (0,100))
314.                 self.label_djob_nextpayment.grid(row = 4, column = 0, padx = (0,0))
315.                 self.label_djob_nextpayment_v.grid(row = 4, column = 1, padx = (0,100))

316.                 self.button_djob_remove.grid(row = 5, column = 0, columnspan = 2, pady =
     (0,93), padx = (0,30))
317.
318.             balance_v = self.data.calc_current_balance(self.userID)
319.             self.label_dbalance = ttk.Label(self.data_panel, text = 'Account balance:')

320.             self.label_dbalance_v = ttk.Label(self.data_panel, text = f'{balance_v}')
321.             self.label_dbalance.grid(row = 6, column = 0, padx = (130,0))
322.             self.label_dbalance_v.grid(row = 6, column = 1, padx = (0,100))
323.
324.             #Statisics_panel
325.             self.x, self.y = self.data.calc_date_balance(self.userID,0)
326.             self.f = Figure(figsize=(10,5), dpi=100)
327.             self.f.set_facecolor('#F0F0F0')
328.             self.a = self.f.add_subplot(111,)
329.             self.a.plot(self.x, self.y, marker = 'o')
330.             for i_x, i_y in zip(self.x, self.y):
331.                 self.a.text(i_x, i_y, '   {}'.format(i_y))
332.             canvas = FigureCanvasTkAgg(self.f, self.statistics_panel)
333.             canvas.get_tk_widget().pack(side=tk.BOTTOM, expand=True)
334.             #Packing
335.             self.statistics_panel.pack(side = tk.BOTTOM)
336.             self.data_panel.pack(side = tk.RIGHT)
337.             self.button_panel.pack(side = tk.TOP)
338.             self.pack()
339.
340.
341.     def loginGui():
342.         root = tk.Tk()
343.         screen_width = root.winfo_screenwidth()
344.         screen_height = root.winfo_screenheight()
345.         width = int(screen_width/6)
346.         height = int(screen_height/5.5)
347.         x = int((screen_width/2) - (width/2))
348.         y = int((screen_height/2) - (height/2))
349.         root.geometry(f'{width}x{height}+{x}+{y}')
350.
351.         app = EconomyLoginGui(root)
352.         app.master.title('Economy Login')
353.         app.mainloop()
```

```
354.
355.     def signUpGui():
356.         root = tk.Tk()
357.         screen_width = root.winfo_screenwidth()
358.         screen_height = root.winfo_screenheight()
359.         width = int(screen_width/8)
360.         height = int(screen_height/3)
361.         x = int((screen_width/2) - (width/2))
362.         y = int((screen_height/2) - (width/2))
363.         root.geometry(f'{width}x{height}+{x}+{y}')
364.
365.         app = EconomySignupGui(root)
366.         app.master.title('Economy Signup')
367.         app.mainloop()
368.
369.     def mainGui(userID: str):
370.         root = tk.Tk()
371.         root.geometry('1920x1080')
372.         root.state('zoomed')
373.         app = EconomyMainGUI(userID, root)
374.         app.master.title('Economy logged in')
375.         app.mainloop()
376.
377.
378.     loginGui()
```

BILAG 4 - ECONODATA.PY

```
1.  import sqlite3, hashlib, binascii, os, datetime
2.  from datetime import datetime as dt
3.  from collections import OrderedDict
4.
5.  class User():
6.      def __init__(self, first_name: str, last_name: str, username: str, email: str, passwor
    d: str):
7.          self.first_name = first_name
8.          self.last_name = last_name
9.          self.username = username
10.         self.password = password
11.         self.email = email
12.
13.
14. class EconomyData():
15.     def __init__(self):
16.         self.db = sqlite3.connect('economy.db')
17.
18.     def check_username(self, username: str):
19.         c = self.db.cursor()
20.         c.execute("""SELECT username FROM users;""")
21.         usernames = c.fetchall()
22.         for i in range(0, len(usernames)):
23.             if username == usernames[i][0]:
24.                 return False
25.         return True
26.
27.     def hash_password(self, password: str):
28.         # https://www.vitoshacademy.com/hashing-passwords-in-python/
29.         # Hash a password for storing.
30.         salt = hashlib.sha256(os.urandom(60)).hexdigest().encode("ascii")
31.         pwdhash = hashlib.pbkdf2_hmac(
32.             "sha512",
33.             password.encode("utf-8"),
```

```
34.             salt,
35.             100000)
36.         pwdhash = binascii.hexlify(pwdhash)
37.         return (salt + pwdhash).decode("ascii")
38.
39.     def verify_password(self, stored_password: str, provided_password: str):
40.         # Verify a stored password against one provided by user
41.         salt = stored_password[:64]
42.         stored_password = stored_password[64:]
43.         pwdhash = hashlib.pbkdf2_hmac(
44.             "sha512",
45.             provided_password.encode("utf-8"),
46.             salt.encode("ascii"),
47.             100000)
48.         pwdhash = binascii.hexlify(pwdhash).decode("ascii")
49.         return pwdhash == stored_password
50.
51.     def add_user(self, user: User):
52.         c = self.db.cursor()
53.         hashed_password = self.hash_password(user.password)
54.         c.execute("""INSERT INTO users (username, first_name, last_name, email, password)
    VALUES (?, ?, ?, ?, ?);""", (user.username, user.first_name, user.last_name, user.email, h
    ashed_password))
55.         userID = c.lastrowid
56.         self.db.commit()
57.         current_date = datetime.date.today()
58.         previous_day = current_date - datetime.timedelta(days = 1)
59.         c.execute("""INSERT INTO obtained_economy (user_id, category, money_obtained, date
    ) VALUES (?, 1, 0, ?);""", (userID, previous_day))
60.         c.execute("""INSERT INTO used_economy (user_id, category, money_spent, date) VALUE
    S (?, 1, 0, ?);""", (userID, previous_day))
61.         self.db.commit()
62.
63.     def add_cat(self, category: str):
64.         category = category
65.         c = self.db.cursor()
66.         c.execute("""INSERT INTO category (category) VALUES (?);""", (category,))
67.         self.db.commit()
68.
69.     def add_money_obtained(self, userID: str, categoryID: int, money_obtained: float):
70.         userID = userID
71.         categoryID = categoryID
72.         money_obtained = money_obtained
73.         c = self.db.cursor()
74.         c.execute("""INSERT INTO obtained_economy (user_id, category, money_obtained) VALU
    ES (?, ?, ?);""", (userID, categoryID, money_obtained))
75.         self.db.commit()
76.         return True
77.
78.     def add_money_used(self, userID: str, categoryID: int, money_used: float):
79.         userID = userID
80.         categoryID = categoryID
81.         money_used = money_used
82.         c = self.db.cursor()
83.         c.execute("""INSERT INTO used_economy (user_id, category, money_spent) VALUES (?,
    ?, ?);""", (userID, categoryID, money_used))
84.         self.db.commit()
85.         return True
86.
87.     def add_job(self, userID: int, job_name: str, job_salary: float, job_payday: str):
88.         userID = userID
89.         job_name = job_name
90.         job_salary = job_salary
```

```
91.         job_payday = job_payday
92.         current_date = datetime.date.today()
93.         next_payment = current_date + datetime.timedelta(days = job_payday)
94.         c = self.db.cursor()
95.         c.execute("""INSERT INTO job (user_id, job_name, salary, payday, next_payment) VAL
    UES (?, ?, ?, ?, ?);""", (userID, job_name, job_salary, job_payday, next_payment))
96.         self.db.commit()
97.         return True
98.
99.     def calc_days_for_payday(self, userID: int):
100.             c = self.db.cursor()
101.             c.execute("""SELECT last_login FROM users WHERE id = ?;""", (userID,))
102.             l = c.fetchone()
103.             ll = self.convert_str_to_date(l[0])
104.             c.execute("""SELECT salary, next_payment, payday FROM job WHERE user_id = ?;
    """, (userID,))
105.             k = c.fetchone()
106.             kk = self.convert_str_to_date(k[1])
107.             if kk <= ll:
108.                 self.add_money_obtained(userID, 3, k[0])
109.                 if self.update_nextpayment(userID) == False:
110.                     return
111.
112.         def update_nextpayment(self, userID: int):
113.             c = self.db.cursor()
114.             c.execute("""SELECT next_payment, payday FROM job WHERE user_id = ?;""", (us
    erID,))
115.             p = c.fetchone()
116.             current_date = datetime.date.today()
117.             oldpayment = self.convert_str_to_date(p[0])
118.             next_payment = oldpayment + datetime.timedelta(days = int(p[1]))
119.             c.execute("""UPDATE job SET next_payment = ? WHERE user_id = ?;""",(next_pay
    ment, userID))
120.             self.db.commit()
121.             if next_payment < current_date:
122.                 self.calc_days_for_payday(userID)
123.             else:
124.                 return False
125.
126.         def calc_obtained(self, userID: int):
127.             userID = userID
128.             m = self.get_obtained(userID)
129.             obtained = 0
130.             for date in m:
131.                 obtained += m[date]
132.             return obtained
133.
134.         def calc_used(self, userID: int):
135.             userID = userID
136.             m = self.get_used(userID)
137.             used = 0
138.             for date in m:
139.                 used += m[date]
140.
141.             return used
142.
143.         def calc_current_balance(self, userID: int):
144.             obtained = self.calc_obtained(userID)
145.             used = self.calc_used(userID)
146.             balance = obtained - used
147.
148.             return balance
149.
```

```
150.         def calc_date_balance(self, userID: int, start_money: int = 0):
151.             #F�r alt indtjent �konomi med datoer som et dict
152.             obtained = self.get_obtained(userID)
153.             #F�r alt brugt �konomi med datoer som et dict
154.             used = self.get_used(userID)
155.             #Laver et dict med obtained dict
156.             balance_dict = obtained
157.             #For hver dato (key) i used
158.             for use in used:
159.                 # Hvis datoen ikke er i modtaget dict (balance_dict)
160.                 if not use in balance_dict:
161.                     # Tilf�j v�rdien (value) til den dato (key) til balance_dict, da d
    et er den
162.                     # brugte m�ngde for den dato
163.                     balance_dict[use] = -used[use]
164.                 else:
165.                     # Tr�k den brugte m�ngde(value) fra, hvad der ellers var tilf�jet
     p� den
166.                     # dato(key)
167.                     balance_dict[use] -= used[use]
168.             # https://stackoverflow.com/questions/34129391/sort-python-dictionary-by-
    date-key
169.             # Sorter dict, s� den f�rste dato kommer f�rst. For at kunne beregne �ndr
    ingen i
170.             # saldo, for dagene der g�r
171.             balance_dict = OrderedDict(sorted(balance_dict.items(),
172.             key = lambda x:dt.strptime(x[0], "%Y-%m-%d")))
173.             # L�ber igennem alle dagene(key), hvor der er sket �ndringer
174.             for balance in balance_dict:
175.                 # Tilf�jer den forrige m�ngde penge til dagen. Dette beregner den tota
    le m�ngde
176.                 # penge p� kontoen for datoen
177.                 balance_dict[balance] += start_money
178.                 # Opdater saldoen for dagen, for at kunne regne videre for n�ste dato
179.                 start_money = balance_dict[balance]
180.             x = []
181.             y = []
182.             for bal in balance_dict:
183.                 x.append(bal)
184.                 y.append(balance_dict[bal])
185.             return x,y
186.
187.         def check_category(self, cat: str):
188.             c = self.db.cursor()
189.             c.execute("""SELECT category FROM category;""")
190.             categories = c.fetchall()
191.             for i in range(0, len(categories)):
192.                 if cat == categories[i][0]:
193.                     return False
194.             return True
195.
196.         def convert_str_to_date(self, string: str):
197.             return datetime.datetime.strptime(string, "%Y-%m-%d").date()
198.
199.         def get_obtained(self, userID: int):
200.             userID = userID
201.             c = self.db.cursor()
202.             c.execute("""SELECT money_obtained, date FROM obtained_economy WHERE user_id
     = ?;""", (userID,))
203.             obtained = c.fetchall()
204.             obtained_money = 0
205.             obtained_money_date_dict = {}
```

```
206.             for i in range(0, len(obtained)):
207.                 obtained_money = obtained[i][0]
208.                 date = self.remove_time(obtained[i][1])
209.                 if not date in obtained_money_date_dict:
210.                     obtained_money_date_dict[date] = obtained_money
211.                 else:
212.                     obtained_money_date_dict[date] += obtained_money
213.             return obtained_money_date_dict
214.
215.         def get_used(self, userID: int):
216.             userID = userID
217.             c = self.db.cursor()
218.             c.execute("""SELECT money_spent, date FROM used_economy WHERE user_id = ?;""
    ", (userID,))
219.             used = c.fetchall()
220.             used_money = 0
221.             used_money_date_dict = {}
222.             for i in range(0, len(used)):
223.                 used_money = used[i][0]
224.                 date = self.remove_time(used[i][1])
225.                 if not date in used_money_date_dict:
226.                     used_money_date_dict[date] = used_money
227.                 else:
228.                     used_money_date_dict[date] += used_money
229.             return used_money_date_dict
230.
231.         def get_cat_list(self):
232.             c = self.db.cursor()
233.             c.execute("""SELECT category FROM category WHERE NOT category='start';""")
234.             cat_list = []
235.             for cat in c:
236.                 cat_list.append(cat[0])
237.             return cat_list
238.
239.         def get_cat_id(self, category: str):
240.             category = category
241.             c = self.db.cursor()
242.             c.execute("""SELECT id FROM category WHERE category = ?;""", (category,))
243.             p = c.fetchone()
244.             return p[0]
245.
246.
247.         def get_userID(self, username: str):
248.             c = self.db.cursor()
249.             c.execute("""SELECT id FROM users WHERE username = ?; """, (username,))
250.             ui = c.fetchone()
251.             return ui[0]
252.
253.         def get_job(self, userID: int):
254.             userID = userID
255.             c = self.db.cursor()
256.             c.execute("""SELECT job_name, salary, payday, next_payment FROM job WHERE us
    er_id = ?;""", (userID,))
257.             j = c.fetchone()
258.             return j
259.
260.         def has_job(self, userID: int):
261.             userID = userID
262.             c = self.db.cursor()
263.             c.execute("""SELECT * FROM job WHERE user_id = ?;""", (userID,))
264.             if len(c.fetchall()) < 1:
265.                 return False
266.             else:
```

```
267.                 return True
268.
269.         def update_date(self, username: str):
270.             current_date = datetime.date.today()
271.             userID = self.get_userID(username)
272.             c = self.db.cursor()
273.             c.execute("""UPDATE users SET last_login = ? WHERE id = ?;""", (current_date
    , userID))
274.             self.db.commit()
275.
276.         def user_login(self, username: str, password: str):
277.             c = self.db.cursor()
278.             c.execute("""SELECT password FROM users WHERE username = ?;""", (username,))

279.             p = c.fetchone()
280.             if self.verify_password(p[0], password):
281.                 self.update_date(username)
282.                 if self.has_job(self.get_userID(username)):
283.                     self.calc_days_for_payday(self.get_userID(username))
284.                 return True
285.             else:
286.                 return False
287.
288.         def remove_time(self, datetime: str):
289.             datetime = datetime
290.             date = datetime.split()
291.             return date[0]
292.
293.         def remove_job(self, userID: int):
294.             userID = userID
295.             c = self.db.cursor()
296.             c.execute("""DELETE FROM job WHERE user_id = ?;""",(userID,))
297.             self.db.commit()
298.
299.         def create_tables(self):
300.             c = self.db.cursor()
301.
302.             try:
303.                 c.execute("""DROP TABLE IF EXISTS users;""")
304.                 c.execute("""DROP TABLE IF EXISTS used_economy;""")
305.                 c.execute("""DROP TABLE IF EXISTS obtained_economy;""")
306.                 c.execute("""DROP TABLE IF EXISTS category;""")
307.                 c.execute("""DROP TABLE IF EXISTS job;""")
308.             except Exception as e:
309.                 print(f'Error while deleting tables: {e}')
310.
311.             try:
312.                 c.execute("""CREATE TABLE IF NOT EXISTS users (
313.                     id INTEGER PRIMARY KEY,
314.                     username TEXT,
315.                     first_name TEXT,
316.                     last_name TEXT,
317.                     email TEXT,
318.                     password TEXT,
319.                     last_login DATETIME NOT NULL DEFAULT CURRENT_DATE);""")
320.
321.                 c.execute("""CREATE TABLE IF NOT EXISTS used_economy (
322.                     id INTEGER PRIMARY KEY,
323.                     user_id INTEGER,
324.                     category INTEGER,
325.                     money_spent FLOAT,
326.                     date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP);""")
327.
```

```
328.                     c.execute("""CREATE TABLE IF NOT EXISTS obtained_economy (
329.                         id INTEGER PRIMARY KEY,
330.                         user_id INTEGER,
331.                         category INTEGER,
332.                         money_obtained FLOAT,
333.                         date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP);""")
334.
335.                     c.execute("""CREATE TABLE IF NOT EXISTS category (
336.                         id INTEGER PRIMARY KEY,
337.                         category TEXT);""")
338.
339.                     c.execute("""CREATE TABLE IF NOT EXISTS job (
340.                         id INTEGER PRIMARY KEY,
341.                         user_id INTEGER,
342.                         job_name TEXT,
343.                         salary FLOAT,
344.                         payday TEXT,
345.                         next_payment DATETIME NOT NULL
346.                         );""")
347.
348.                     print('All tables created successfully')
349.                 except Exception as e:
350.                     print(f'Error when creating tables: {e}')
351.
352.                 test_password1 = "1234"
353.                 test_password2 = "4321"
354.                 test_password1 = self.hash_password(test_password1)
355.                 test_password2 = self.hash_password(test_password2)
356.                 c.execute("""INSERT INTO users (username, first_name, last_name, email, pass
    word) VALUES ('Tester1', 'Jens', 'Tester','jenstester@gmail.com',?);""", (test_password1,)
    )
357.                 c.execute("""INSERT INTO users (username, first_name, last_name, email, pass
    word) VALUES ('Tester2', 'Tester', 'Jens','testerjens@gmail.com',?);""", (test_password2,)
    )
358.                 c.execute("""INSERT INTO obtained_economy (user_id, category, money_obtained
    ) VALUES (1, 1, 2000);""")
359.                 c.execute("""INSERT INTO obtained_economy (user_id, category, money_obtained
    ) VALUES (1, 1, 3000);""")
360.                 c.execute("""INSERT INTO obtained_economy (user_id, category, money_obtained
    , date) VALUES (1, 1, 3000, '2020-05-02 17:43:04');""")
361.                 c.execute("""INSERT INTO obtained_economy (user_id, category, money_obtained
    , date) VALUES (1, 1, 4000, '2020-05-02 17:43:04');""")
362.                 c.execute("""INSERT INTO obtained_economy (user_id, category, money_obtained
    , date) VALUES (1, 1, 500, '2020-05-01 17:43:04');""")
363.                 c.execute("""INSERT INTO category (category) VALUES ('start');""")
364.                 c.execute("""INSERT INTO category (category) VALUES ('HEJ');""")
365.                 c.execute("""INSERT INTO category (category) VALUES ('Salary');""")
366.                 c.execute("""INSERT INTO job (user_id, job_name, salary, payday, next_paymen
    t) VALUES (1, 'spurgt', 200, 1, '2020-05-01');""")
367.                 c.execute("""INSERT INTO used_economy (user_id, category, money_spent, date)
     VALUES (1, 1, 1000, '2020-05-03 17:43:04');""")
368.                 c.execute("""INSERT INTO used_economy (user_id, category, money_spent) VALUE
    S (1, 1, 1000);""")
369.                 self.db.commit()
```