


Programming

Eksamensprojekt



Synopsis
NoteSort
Programming

2020

Notetagningshjemmeside

Projektets produkt er en hjemmeside, hvor man kan tage sine noter til skolearbejde, og så vil hjemmesiden selv sortere noten i det fag, noten er skrevet til, registreret med en "machine learning" algoritme.

1 TITELBLAD

1.1 VEJLEDNING

Søren Præstegaard (SPR)

1.2 FAG

Programmering B

1.3 KLASSE OG SKOLE

3.D - Odense Tekniske Gymnasium

1.4 ANTAL TEGN

19193 af 19200 tegn

1.5 DATO

4. maj 2020

2 INDHOLDSFORTEGNELSE

1	Titelblad	2
1.1	Vejledning	2
1.2	Fag.....	2
1.3	Klasse og skole	2
1.4	Antal tegn.....	2
1.5	Dato	2
2	Indholdsfortegnelse.....	3
3	Indledning	5
3.1	Opgaveformulering.....	5
4	Kravspecifikation	5
4.1	Brugerhistorier.....	5
4.1.1	Log ind.....	5
4.1.2	Opret ny note.....	6
4.1.3	Rediger note	6
4.2	Iterationer	7
4.2.1	Pre-Iteration - skitser	8
4.2.2	Iteration 1 - Log ind system	9
4.2.3	Iteration 2 - Hovedmenu	10
4.2.4	Iteration 3 - Noteskrivning.....	10
5	Programmets opbygning	11
5.1	Tre-lags-modellen	11
5.1.1	Præsentationslag/brugerflade.....	12
5.1.2	Logiklag/applikationslag.....	12
5.1.3	Datalag	13
5.1.4	Implementeringen i programmet NoteSort.....	13

5.2	Beskrivelse af machine learning algoritme	18
5.3	Kort om mest relevante anvendte biblioteker	22
5.3.1	Flask	22
5.3.2	Sklearn	22
6	Referencer	23
7	Bilag	24
7.1	Bilag 1 - Brugerhistorier	24
7.2	Kildekode	24
7.2.1	Python	24
7.2.2	HTML.....	43

3 INDLEDNING

Det kan fratage nogen lysten til at skrive noter, fordi det også indebærer en vis form for organisering. Dette er hvorfor jeg i dette projekt har sat mig for at lave en lettilgængelig notetagningshjemmeside, hvor man let kan lave en ny note.

Programmet skal efterfølgende undersøge indeholder af notatet og vurdere, hvilket fag noten hører til og selv lægge noten det rigtige sted hen. Dette skulle forhåbentlig give en bedre oplevelse for f.eks. gymnasieelever, når de skal tage noter til fagene.

3.1 OPGAVEFORMULERING

Herunder er mere specifikt beskrevet de områder, som jeg gerne vil gennemgå i dette projekt.

- Sammensætningen af en hjemmeside, der fungerer som en brugerflade til at interagere med en machine learning algoritmen. Herunder en forklaring af opdelingen mellem brugerflade, applikation og datalag (tre-lags-modellen).
- En machine learning algoritme (supervised learning), som givet nogle tekststykker med tilhørende mærkater vil kunne kategorisere nye tekststykker ud fra mønstre i de markerede tekststykker.

4 KRAVSPECIFIKATION

4.1 BRUGERHISTORIER

I dette afsnit er de mest essentielle funktioner af applikationen beskrevet ved brug af brugerhistorier. De resterende brugerhistorier ligger i Bilag 1 - Brugerhistorier.

4.1.1 Log ind

- 1) Brugeren skriver sit brugernavn og password i input.
- 2) Brugeren trykker på log-ind-klappen.
- 3) Programmet tjekker om begge felter er fyldt ud.
- 4) Programmet går ind i databasen og leder efter en bruger med det brugernavn.

- 5) Programmet tjekker, hvis brugeren er i databasen, og om den brugers kodeord er det samme, som brugeren har indtastet.
 - a) Hvis kodeordet er det rigtige føres brugeren videre til brugerens startside.
 - b) Hvis kodeordet ikke er rigtigt, siger programmet, at det er forkert, og brugeren kan indtaste igen.

4.1.2 Opret ny note

- 1) Når brugeren er logget ind, trykker brugeren på "New note"-knappen
- 2) Programmet navigerer til siden for at oprette nye noter.
- 3) Brugeren skriver et emne i "Subject"-inputfeltet og indholdet af noten i tekstområdet nedenunder.
- 4) Brugeren klikker "Submit"-knappen for at gemme noten.
- 5) Programmet tager indholdet af tekstområdet, kører det igennem machine learning algoritmen og returnerer algoritmens gæt på det fag.
- 6) Brugeren har nu tre valg
 - a) Brugeren kan trykke krydset for at gå tilbage til noten og ændre noget, inden den skal gemmes.
 - b) Brugeren kan vælge at trykke "Yes"-knappen for at acceptere algoritmens placering af noten.
 - c) Brugeren kan trykke "No"-knappen, hvorefter programmet viser en mulighed for brugeren at vælge, hvilket fag noten i stedet skal lægges i.
- 7) Når noten gemmes, bliver den tilføjet til machine learning algoritmens data, hvorefter algoritmen bliver opdateret. Herefter bliver noten lagt i databasen.
- 8) Programmet navigerer tilbage til brugerens startside.

4.1.3 Rediger note

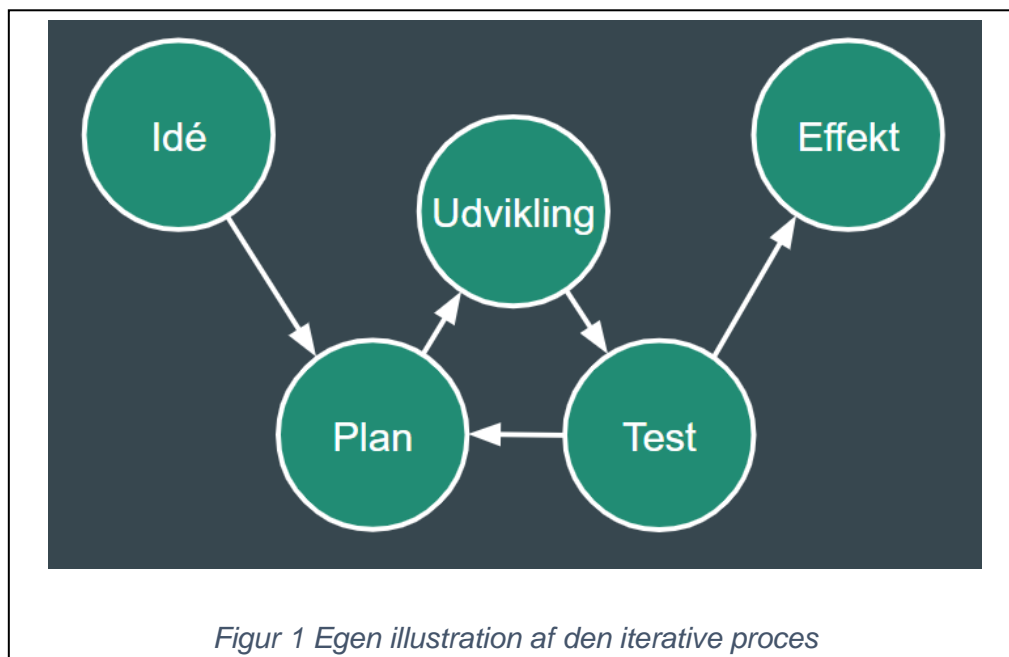
- 1) Når brugeren er logget ind, klikker brugeren på det fag, som noten ligger i.
- 2) Programmet navigerer til det fags side.
- 3) Brugeren klikker på den note i listen, som skal redigeres.
- 4) Programmet navigerer til en side, hvor noten bliver vist.
- 5) Brugeren klikker på "Edit"-knappen.
- 6) Programmet navigerer til en redigeringside, hvor notens indhold står (opbygget på samme måde som for siden, hvor man skriver en ny note).

- 7) Brugeren kan tilføje ændringer til noten og trykke "Submit"-knappen for at gemme ændringerne til noten.
- 8) Programmet går ind i databasen og tilføjer ændringerne og navigerer tilbage til faget, hvor den nye note også bliver vist.

4.2 ITERATIONER

Arbejdet på programmer foregår i iterationer, hvor der er fokus på den enkelte funktions tilføjelse. I dette afsnit har jeg medtaget de mest essentielle iterationer for programmets funktion (her er ikke medtaget machine learning algoritmen, da den bliver beskrevet senere).

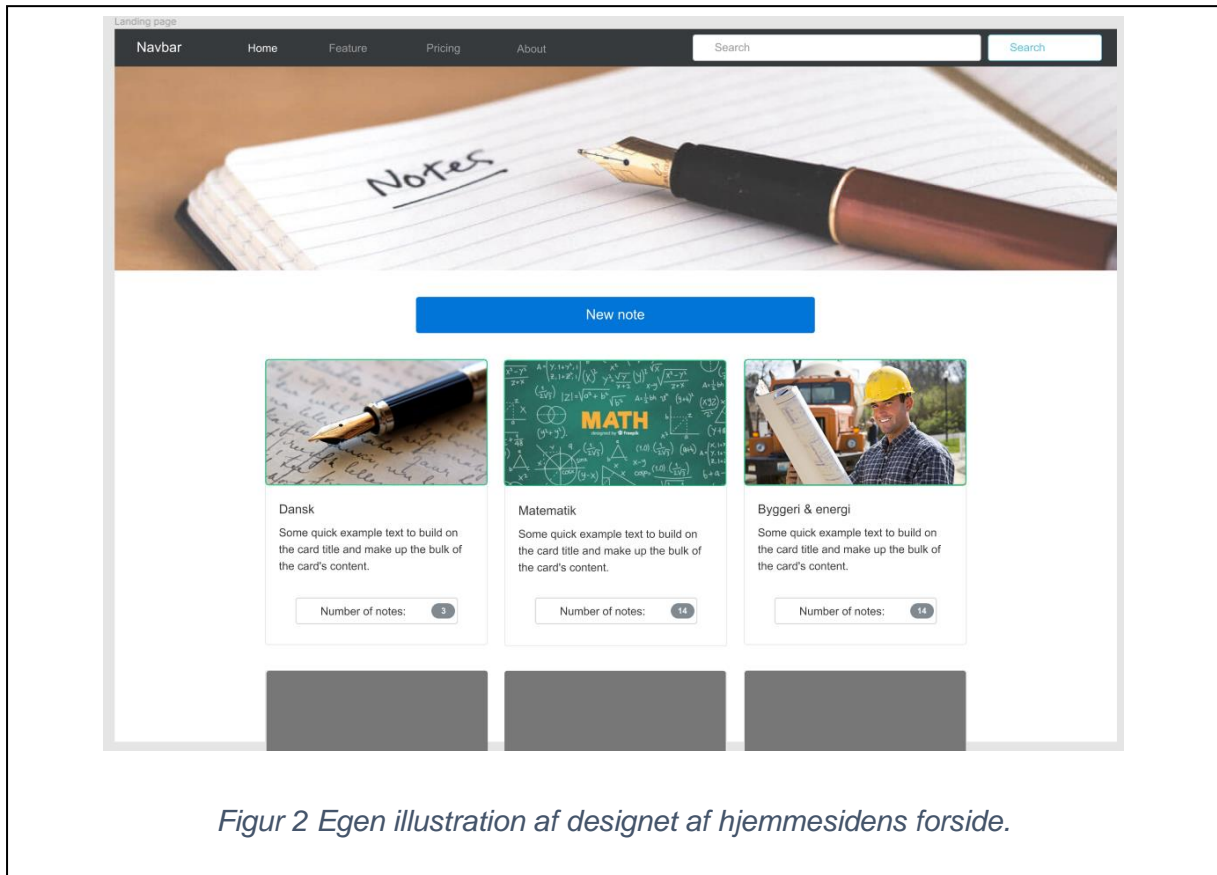
Den iterative proces ser ud som følger:



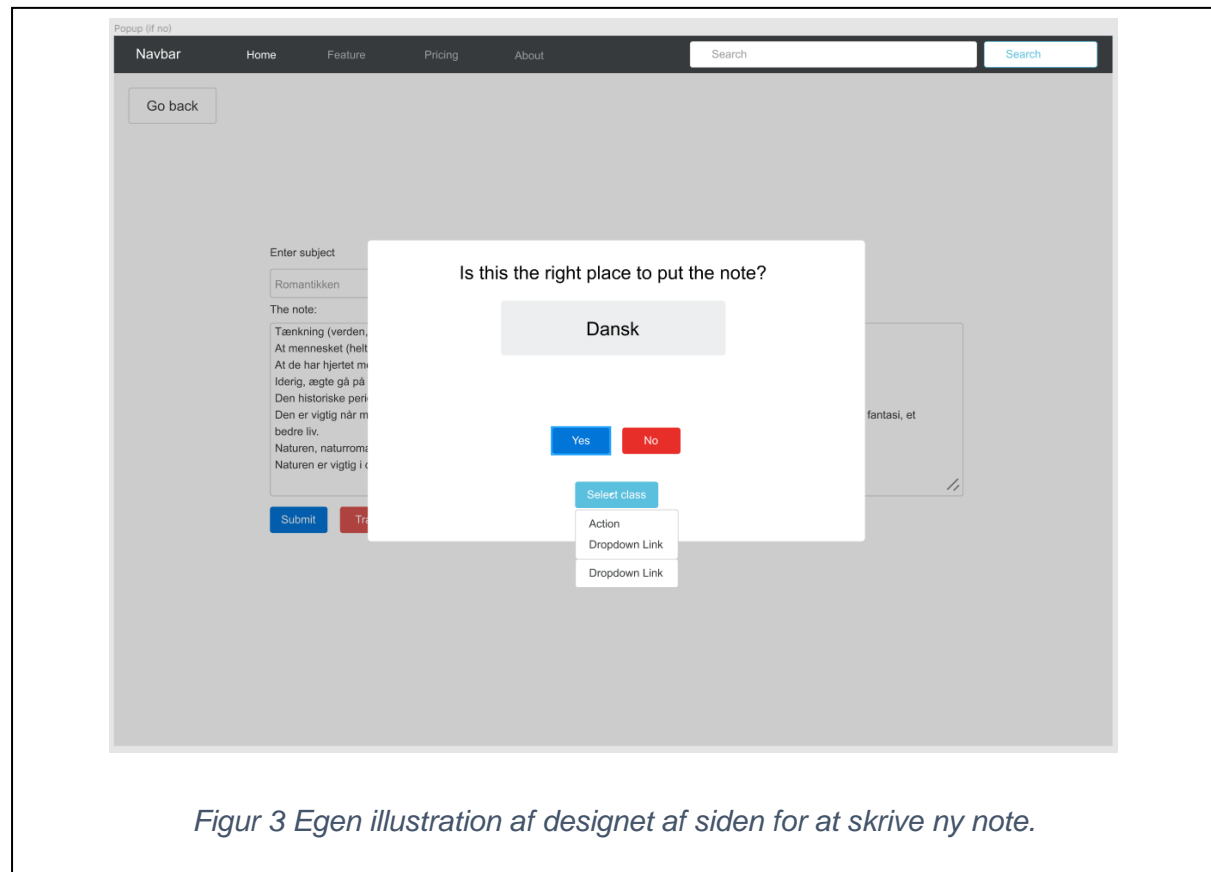
Den iterative proces består af flere elementer, som iterativt bliver gennemgået. Første skridt er, at man har en idé, dernæst lægger man en plan for, hvordan man vil gennemføre den idé. Så kommer udviklingsprocessen, hvorefter man tester, om den udvikling, man har gjort, er god nok til at kunne kaldes færdig. Hvis den ikke er det, eller man finder nogle fejl i sin funktion går man tilbage til at planlægge, men hvis testen lykkedes, implementeres idéen/funktionen. Det er efter denne opbygning jeg har forsøgt at lave mit projekt.

4.2.1 Pre-Iteration - skitser

Inden jeg begyndte at kode, lavede jeg nogle skitser for at visualisere mit projekt og for at få et overblik over de elementer, som skulle indgå.



På Figur 2 ses det første design af hjemmesidens forside. Her havde jeg tænkt mig at have en nem adgang til at skrive en ny note, men også en nem adgang til at komme til fagene, som indeholder alle noterne for det pågældende fag.



På Figur 3 ses det tænkte design for den side, hvor man skal kunne skrive en ny note. På figuren er det highlightet funktionen af, at når man har skrevet noten og gerne vil tilføje den til notesamlingen, skal et popop-vindue komme frem og vise, hvad for et fag, programmet har gættet på, at noten skal ligge i og muligheden for brugeren at ændre det fag, hvis programmet har gættet forkert.

4.2.2 Iteration 1 - Log ind system

Første iteration for programmet var at få lavet et log ind system, så det var muligt at holde styr på hjemmesidens brugere. Dette blev gjort ved at opskrive de parametre, som ud fra designet af hjemmesiden blev vurderet. Disse parametre blev skrevet ind i et ER-diagram (forklaring følger i afsnit 5.1.4.3.1) for at skabe overblik over programmets opbygning. Næste skridt var at kode selve systemet. Først valgte jeg at kode det visuelle, altså selve felterne for log in og opskrivning af ny bruger. Til dette brugte nogle en skabelon fra Bootstraps dokumentation.¹ Herefter var det at

¹ (The Bootstrap team, 2020)

implementere selve koden, hvilket jeg gjorde ved at oprette en databasetabel indholdene parametrene for brugeren. Med denne database tog jeg informationerne fra HTML siderne for log ind systemet (det visuelle/brugerfladen) og førte dem ind i databasetabellen. Efter at have det sat op testede jeg ved at tilføje en ny bruger vha. registrerings HTML siden, hvorefter jeg prøvede at logge ind som den bruger. For at være sikker på at det virkede, havde jeg ført informationerne for den pågældende bruger, som loggede ind, med ind på den side, som brugeren loggede ind på: Så jeg kunne tjekke, at det var den bruger, som loggede ind, hvis informationer blev vist på siden.

4.2.3 Iteration 2 - Hovedmenu

Næste skridt var at programmere hovedmenuen. Her startede jeg igen med at kode det visuelle ved brug af designet, som jeg havde lavet i forvejen. I alle felterne havde jeg skrevet fyldtekst, som skulle forestille indholdet til, når siden var forbundet med data fra databasen. Da det var sat op, skulle jeg ud fra ER-diagrammet (forklaring følger i afsnit 5.1.4.3.1) oprette databasetabeller for de fag, som jeg havde data for (Byggeri og Energi, Dansk og Matematik) og en tabel for de noter, som ville skulle ligge i hver sit fag. På trods af at noterne ikke direkte ville skulle ligge på forsiden, var jeg stadig nødt til at lave tabellen nu, siden jeg ville skulle have nogen noter liggende i tabellen for at vise antallet af noter i hvert fag (se Figur 2). Efter at have oprettet databasetabellerne, tilføjede jeg nogle standardnoter til hver af de tre fag, så jeg havde noget data at teste med. Jeg forbandt dataene til visningsfelterne i HTML siden vha. Jinja og testede dernæst ved at logge ind og se, at informationerne på fagene og at antallet af noter i fagene var rigtige.

4.2.4 Iteration 3 - Noteskrivning

Sidste funktion jeg ville implementere for at kunne kalde hjemmesiden et funktionelt produkt, var en måde at skrive nye noter. Til dette startede jeg med igen at opstille siden vha. HTML og CSS. Hernæst skrev jeg koden i datalaget til at tilføje nye noter til databasetabellen for noter. Inden jeg ville se på at få min machine learning algoritme til at virke, ville jeg bare tilføje noten direkte til databasen (algoritmen er beskrevet i et senere afsnit 5.2), for stadig at beholde en funktion/en iteration ad gangen. Efter at have fået programmet til at tilføje noten til databasen skrev jeg koden for fagsiden (siden med alle noter for et specifikt fag) og notesiden (siden for

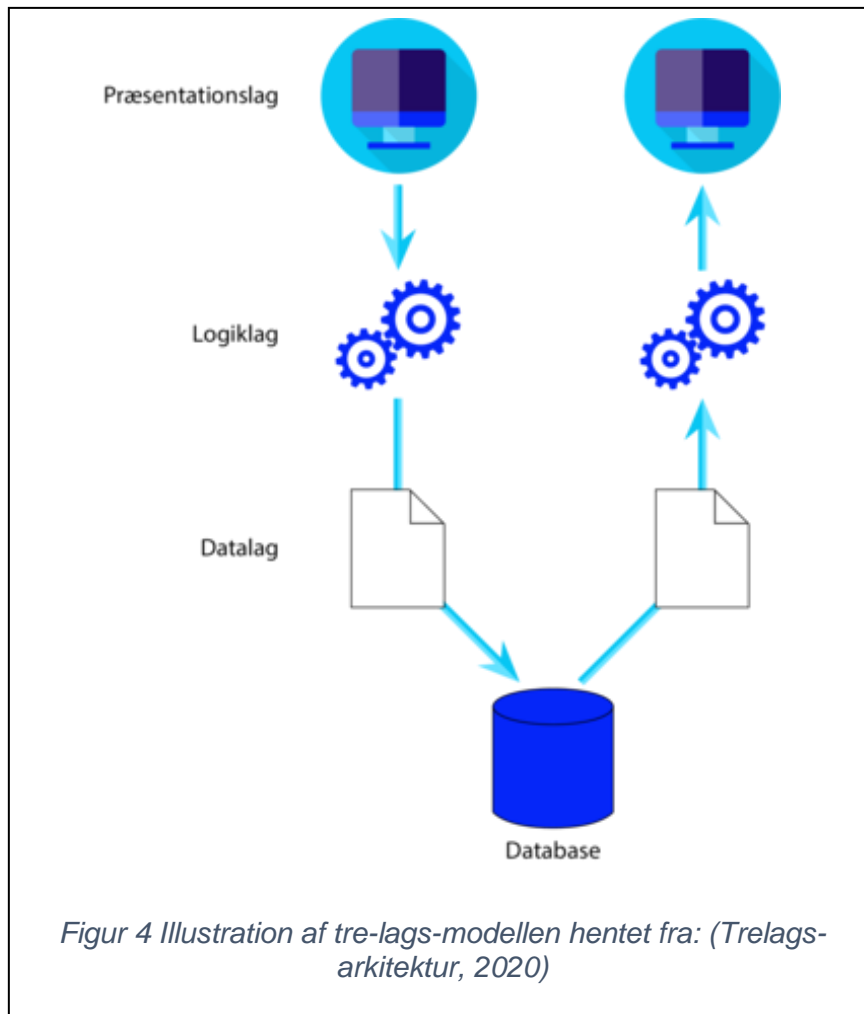
visningen af en note). Disse sider brugte jeg til at teste, at de noter jeg tilføjede til databasen, blev tilføjet som de skulle. Som ekstra tjek brugte jeg en udvidelse til mit tekstredigeringsprogram, VS Code, kaldet "SQLite", som kunne fremvise indholdet af en .db fil.

5 PROGRAMMETS OPBYGNING

I dette afsnit vil jeg gennemgå programmets opbygning, og hvordan de forskellige inddelinger/lag af programmet taler sammen med hinanden for at danne et fuldendt og funktionelt program.

5.1 TRE-LAGS-MODELLEN

Det første, jeg vil gennemgå, er tre-lags-modellen, da det er essentielt til at forstå sammenhængen af programmet. På Figur 4 ses en illustration af tre-lags-modellen. Modellen er en måde at opdele sit program på, så det er lettere at forstå og lettere at komme tilbage og tilføje funktioner eller ændringer.



De tre lag er som følger:

5.1.1 Præsentationslag/brugerflade

Dette lag er den del af programmet, som sørger for, at brugeren har noget at interagere med. Det er det visuelle lag, og her er der udelukkende fokus på at præsentere data for brugeren. Behandling af data sker ikke i dette lag (udover formatering).

5.1.2 Logiklag/applikationslag

Applikationslaget er det lag, hvor data bliver behandlet. Det er her der bliver indsamlet og behandlet data til enten at føre tilbage til brugeren i brugerfladen eller gemme via datalaget. En behandling af data betyder dog ikke bare, at dataen skal formateres. Det betyder også, at der f.eks. kan være noget af dataen, som skal tjekkes, om det er korrekt, eksempelvis at man har skrevet det samme kodeord, når man er blevet bedt om at gentage kodeordet i registreringen af ny bruger.

5.1.3 Datalag

Datalaget er den del af programmet, som står for at have forbindelse til databasen, om så det er en lokal eller ekstern database. Delle del af programmet sørger for at tage de data, som applikationslaget har bearbejdet og placere dem i databasen. Det er samtidig dette lag, som ændrer værdier eller sletter værdier fra databasen.

5.1.4 Implementeringen i programmet NoteSort

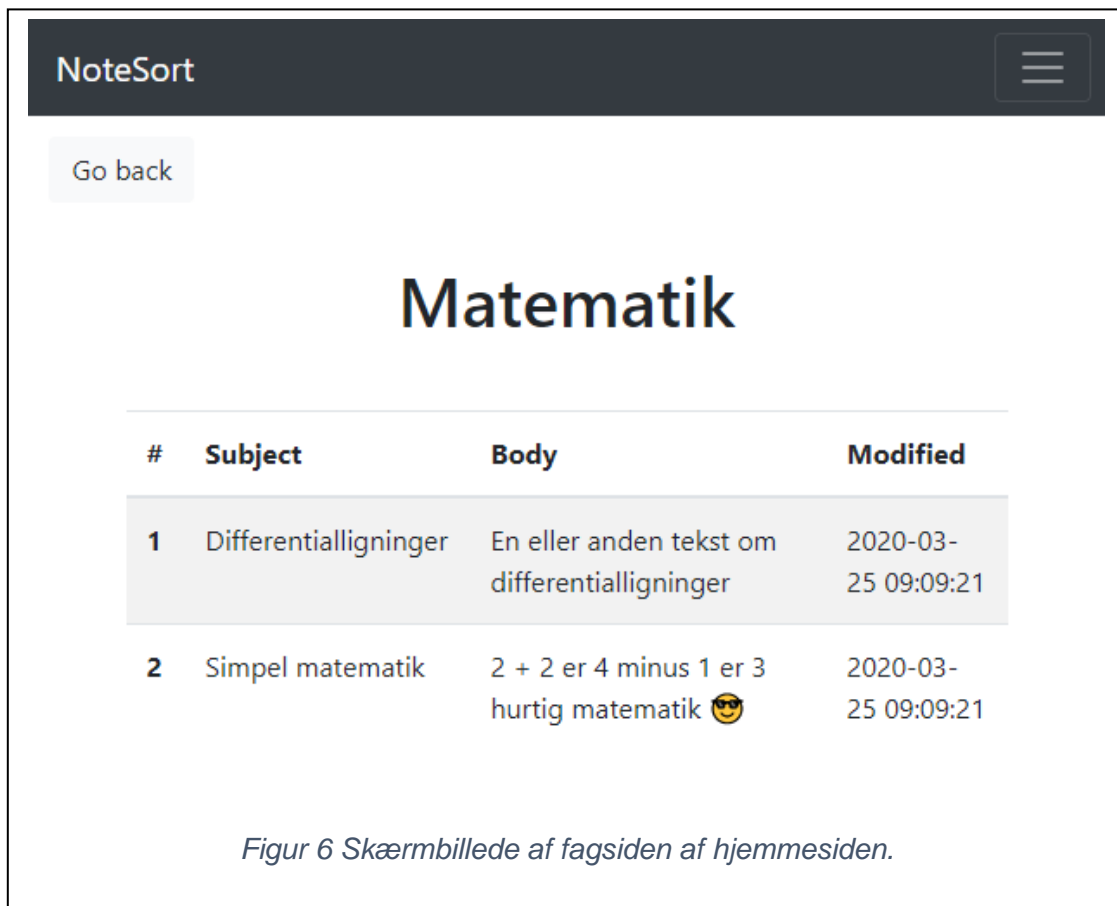
5.1.4.1 Brugerfladen

I mit program består brugerfladen af HTML- og CSS-kodet sider, som vha. skabelon værktøjet Jinja integrerer dataen fra applikationslaget ind i hjemmesiden. Et konkret eksempel på dette kan ses herunder:

```
1 {% extends "header.html" %} {% block content %}
2 <a href="/profile" class="btn btn-light back-btn" type="button">Go back</a>
3 <div class="container">
4     <h1 class="class-heading">{{ class_info['class_name'] }}</h1>
5     <div class="table-responsive">
6         <table class="table table-striped table-hover">
7             <thead>
8                 <tr>
9                     <th scope="col">#</th>
10                    <th scope="col">Subject</th>
11                    <th scope="col">Body</th>
12                    <th scope="col">Modified</th>
13                </tr>
14            </thead>
15            <tbody>
16                {% for note in notes %} {% set note_id = note['note_id'] %}
17                <tr>
18                    onclick="window.location.href='/read_note?={{ note_id }}'"
19                    >
20                    <th scope="row">{{ loop.index }}</th>
21                    <td>{{ note['subject'] }}</td>
22                    <td>{{ note['body'] }}</td>
23                    <td>{{ note['timestamp'] }}</td>
24                </tr>
25                {% endfor %}
26            </tbody>
27        </table>
28    </div>
29 </div>
30 {% endblock content %}
```

Figur 5 Kodeeksempel på implementering af brugerfladen. Kommer fra filen "class_page.html".

Koden i Figur 5 er HTML-koden for fagsiden, som viser alle noter, der ligger i det pågældende fag. Jinja bruges til at lave et for-loop, som løber igennem alle noterne tilhørende det pågældende fag, hvor der for hvert fag i linje 20-23 på figuren placeres dataene for hver note i faget ind i en tabel. Et visuelt eksempel på koden er matematikfaget på Figur 6, hvor dataene for henholdsvis "subject", "body" og "modified" er placeret i tabellen.



The screenshot shows a web application titled 'NoteSort'. At the top left is a 'Go back' button. The main heading is 'Matematik'. Below it is a table with four columns: '#', 'Subject', 'Body', and 'Modified'. The table contains two rows of data. The first row has an index of 1, subject 'Differentialligninger', body 'En eller anden tekst om differentialligninger', and a timestamp '2020-03-25 09:09:21'. The second row has an index of 2, subject 'Simpel matematik', body '2 + 2 er 4 minus 1 er 3 hurtig matematik 😎', and the same timestamp.

#	Subject	Body	Modified
1	Differentialligninger	En eller anden tekst om differentialligninger	2020-03-25 09:09:21
2	Simpel matematik	2 + 2 er 4 minus 1 er 3 hurtig matematik 😎	2020-03-25 09:09:21

Figur 6 Skærbillede af fagsiden af hjemmesiden.

5.1.4.2 Applikationslaget

Til applikationslaget har jeg valgt at beskrive registreringen af en ny bruger. machine learning algoritmen er den meste centrale del af applikationslaget af programmet, men jeg har valgt at beskrive det i sit eget afsnit (5.2), siden det er ret omfangsrigt.

```
1 @app.route("/signup_profile", methods=["POST"])
2 def signup():
3     username = request.form["username"]
4     firstname = request.form["firstname"]
5     lastname = request.form["lastname"]
6     email = request.form["email"]
7     password = request.form["password"]
8     re_password = request.form["re_password"]
9
10    if signup_success(
11        username=username,
12        firstname=firstname,
13        lastname=lastname,
14        email=email,
15        password=password,
16        re_password=re_password,
17    ):
18        return index()
19    else:
20        session.pop("currentuser", None)
21        return my_render("sign_up.html", success=False)
```

A

```
1 def signup_success(username, firstname, lastname, email, password, re_password):
2     if check_same_password(password, re_password):
3         hashed_password = data.hash_password(password)
4         u = User(username, firstname, lastname, email, hashed_password)
5         if not data.signup_success(user=u):
6             return False
7         return True
8     else:
9         return False
```

B

Figur 7 Kodeeksempel på registrering af bruger. Kommer fra filen "main.py".

På figuren herover ses to kodelokker som udgør applikationslaget for funktionen at registrere en ny bruger. Den første del er funktionen "signup" (stk. A, linje 2-21), som bliver kaldt af frameworket Flask, når brugeren klikker på "Sign in" på HTML siden. Først gemmes værdierne fra inputfelterne på siden som variabler (stk. A, linje 3-8).

Dernæst kaldes funktionen "signup_suces", som er vist i stk. B. Funktionen tager parametrene fra registreringssiden, tjekker om kodeordet og det gentagede kodeord er det samme (funktionskald "check_same_password", stk B, linje 2), herefter opretter den et nyt instans af den simple klasse "User", som blot er et objekt bestående af de indsatte parametre på linje 4 i stk. B. Til sidst sender funktionen oplysningerne (instansen af brugeren) til datalaget, hvor der tjekkes, at brugeren ikke allerede er i database. Hvis brugeren ikke er det, tilføjes brugeren til database, og if-tjekket i linje 5 af stk. B vil returnere True for funktionen, betydende at programmet vil navigere til brugerens forside tilbage i funktionen "signup" (stk. A, linje 18). Hvis ikke kodeordene er de samme, eller hvis brugernavnet allerede er taget i databasen, vil "signup_success" returnere false (stk. B linje 6 og 9), hvorefter funktionen "signup" vil nulstille sessionen og genindlæse registreringssiden (stk. B linje 19-21).

5.1.4.3 Datalaget

Som et eksempel på koden for datalaget har jeg valgt at forklare tilføjelsen af en ny note til databasen.

```
1 def submit_note(self, user_id, class_id, subject, body):
2     db = self._get_db()
3     c = db.cursor()
4     c.execute(
5         "INSERT INTO notes (user_id, class_id, subject, body) VALUES (?, ?, ?, ?)",
6         (user_id, class_id, subject, body),
7     )
8     db.commit()
```

Figur 8 Kodeeksempel på indsætning af en note i databasen. Kommer fra filen "note_data.py".

For at tilføje en note til databasen kræves brugerens id, det pågældende fags id, emnet for noten (titlen) og selve noten. Disse parametre kræves af funktionen "submit_note" i datalaget (se Figur 8 linje 1). Til at starte med skal der skabes forbindelse til selve databasen, som i dette tilfælde er en .db fil (med en anden opsætning, ville man kunne have f.eks. en MySQL database kørende over internettet). Forbindelsen oprettes ved brug af funktionen "_get_db"² (se Figur 9 linje

² Funktionens navn starter med en underscore for at indikere, at denne funktion kun er relevant inde i selve Database klassen, og at den ikke skal bruges, når klassen eksporteres til andre filer. I nogle andre objektorienterede programmeringssprog vil sådanne funktioner blive kaldt private funktioner.

4-8). Funktionen bruger først en klasse fra det importerede framework Flask ved navn "g" til at forsøge at hente en muligt allerede indlæst database (Figur 9 linje 5). Hvis den database ikke er indlæst, vil variablen "db" være "None" hvorefter programmet vil definere "db" samt "g._database" til at være lig med den forbindelse til databasen ("self.DATABASE" Figur 9 linje 2), som sqlite3 biblioteket danner (Figur 9, linje 6-7). Til sidst vil funktionen returnere forbindelsen til databasen, som så kan arbejdes med.

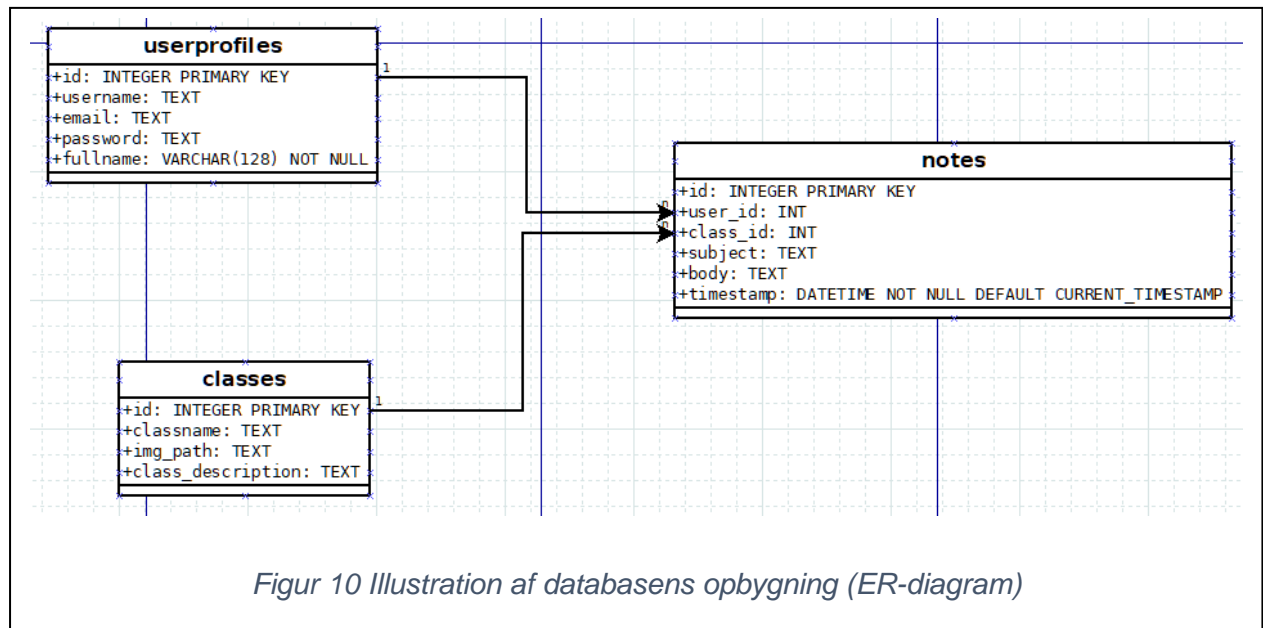
```
1 def __init__(self):
2     self.DATABASE = fr"{ABS_FILEPATH}\main.db"
3
4     def _get_db(self):
5         db = g.get("_database", None)
6         if db is None:
7             db = g._database = sqlite3.connect(self.DATABASE)
8         return db
```

Figur 9 Kodeeksempel på forbindelsesoprettelse til databasen. Kommer fra filen "note_data.py".

Tilbage i "submit_note" (Figur 8) oprettes en "cursor" til databasen (linje 3), som kan bruges til at sende SQL-kommandoer til databasen. I denne funktion sendes "INSERT"-kommandoen, som indsætter parametrene beskrevet til databasetabellen "notes" (linje 4-7). Efter det gemmer programmet ændringerne til tabellen til databasen (linje 8).

5.1.4.3.1 Beskrivelse af databasen – ER-diagram

Som det sidste fjerde element til tre-lags-modellen er databasen, hvor selve dataen bliver gemt. Dette er ikke et lag, da det blot er opbevaringen af data som sker her. Der skrives ikke noget kode i databasen, når man laver programmer. Her vil jeg dog like komme ind på opbygningen af min database. For at få et overblik over databasens sammensætning, har jeg lavet et ER-diagram (se Figur 10), som beskriver de tabeller og værdier deri, som jeg bruger i mit program.



Efter at have lavet programmets design i skitser, havde jeg dannet et overblik over de elementer, som skulle med, hvorefter jeg lavede ER-diagrammet, for at danne mig en idé om, hvordan jeg skulle kode datalaget til at håndtere den data, jeg forventede ville skulle bruges i programmet.

5.2 BESKRIVELSE AF MACHINE LEARNING ALGORITME³

Som det mest centrale element af mit programs funktion er en machine learning algoritme for klassificering af tekst. Metoden, som er brugt her, er "Supervised learning", hvilket er hvor man giver programmet nogle data med tilhørende mærkater, som kategoriserer de data. I dette tilfælde har jeg indsamlet en række data og givet dem mærkater (1, 2, 3) for fagene byggeri og energi, dansk og matematik.

For at komme videre med dataene, er de blevet delt op i sætninger (stadig med deres tilhørende mærkater). For at få givet programmet oplysninger om, hvilken form for sprogtilgang, der skal til, for hvert af fagene, kan frekvensen af hvert ord pr. sætning for hvert fag tælles. Ved at gøre dette kan en form for ordbog for hvert fag dannes. Det man får ud af det, at en vektor for hver sætning, der er som en liste med

³ Som inspiration til min machine learning algoritme, har jeg brugt forklaringen på denne hjemmeside: (Janakiev, 2019)

antallet af gange et bestemt ord optrådte i en sætning ud fra den samlede ordbog for faget. Et eksempel kunne være:

```
1 from sklearn.feature_extraction.text import CountVectorizer
2
3 sentences = ['I dansk kan man skrive med bogstager', 'I matematik kan man regne med tal']
4
5 vectorizer = CountVectorizer(min_df=0, lowercase=False)
6 vectorizer.fit(sentences)
7
8 print(vectorizer.vocabulary_)
9 # {'dansk': 1, 'kan': 2, 'man': 3, 'skrive': 7, 'med': 5,
10 #  'bogstager': 0, 'matematik': 4, 'regne': 6, 'tal': 8}
11 print(vectorizer.transform(sentences).toarray())
12 # [[1 1 1 1 0 1 0 1 0]
13 #   [0 0 1 1 1 1 1 0 1]]
```

Figur 11 Kodeeksempel af "vektorisering" af sætninger. Kommer fra filen "test.py".

I dette eksempel (Figur 11) er der taget to sætninger, hvor den ene er fra dansk og den anden matematik (linje 3). Klassen "CountVectorizer" fra biblioteket "sklearn" bliver brugt til at omdanne sætningerne til en ordbog (linje 5-6). På linje 9-10 ses ordbogen, hvor man kan se placeringen/indekset af hvert ord. På linje 12-13 ses sætningerne konverterede til såkaldte feature vektorer,⁴ som har antallet af hvert ord i hver sætning. Begge sætninger har ordet "kan", og ud fra ordbogen kan man se, at de ligger på indekset 2, hvilket de også gør for på linje 12-13.

Næste skridt er at definere en model. Hvis man havde mere tid, ville denne simple model være første skridt til at bygge bedre modeller ud fra, men grundet tidsrummet af denne opgave, har der ikke været tid til det. Første skridt er at dele dataene op i trænings- og testdata (dette skridt er samtidig, hvordan jeg har implementeret modellen til brug i hjemmesiden – NoteSort). Det gøres for at kunne holde kontrol over, hvor god modellen er. Det er også for at undgå "overfitting"⁵ af ens model.

⁴ Det er blot en vektor med en serie af numre. Man kan se det som en matrix, men kun med en række og flere kolonner (eller omvendt). Eksempel: [9,8,7,6,5,4,3,2,1].

⁵ Overfitting er, hvor modellen er trænet for godt til træningsdatasættet. Man vil gerne undgå dette, da det, selvom det giver en høj nøjagtighed for forsøg med træningssættet, giver en lille nøjagtighed for testsættet.

```
1 class TextClassifierModel:
2     def __init__(self):
3         self.model_file = fr"{ABS_FILEPATH}\data\classes\combined_test.txt"
4         self.filepath_dict = {
5             "combined": self.model_file,
6         }
7         self.create_dataframe()
8         self.create_model()
9
10    def create_dataframe(self):
11        df_list = []
12        for text_class, filepath in self.filepath_dict.items():
13            df = pd.read_csv(filepath, names=["sentence", "label"], sep="\t")
14            df["text_class"] = text_class
15            df_list.append(df)
16
17        self.df = pd.concat(df_list)
18
19    def create_model(self):
20        self.df_combined = self.df[self.df["text_class"] == "combined"]
21
22        sentences = self.df_combined["sentence"].values
23        y = self.df_combined["label"].values
24
25        (
26            self.sentences_train,
27            self.sentences_test,
28            self.y_train,
29            self.y_test,
30        ) = train_test_split(sentences, y, test_size=0.25, random_state=1000)
31
32        self.vectorizer = CountVectorizer()
33        self.vectorizer.fit(self.sentences_train)
34
35        self.X_train = self.vectorizer.transform(self.sentences_train)
36        self.X_test = self.vectorizer.transform(self.sentences_test)
37
38        self.classifier = LogisticRegression()
39        self.classifier.fit(self.X_train, self.y_train)
```

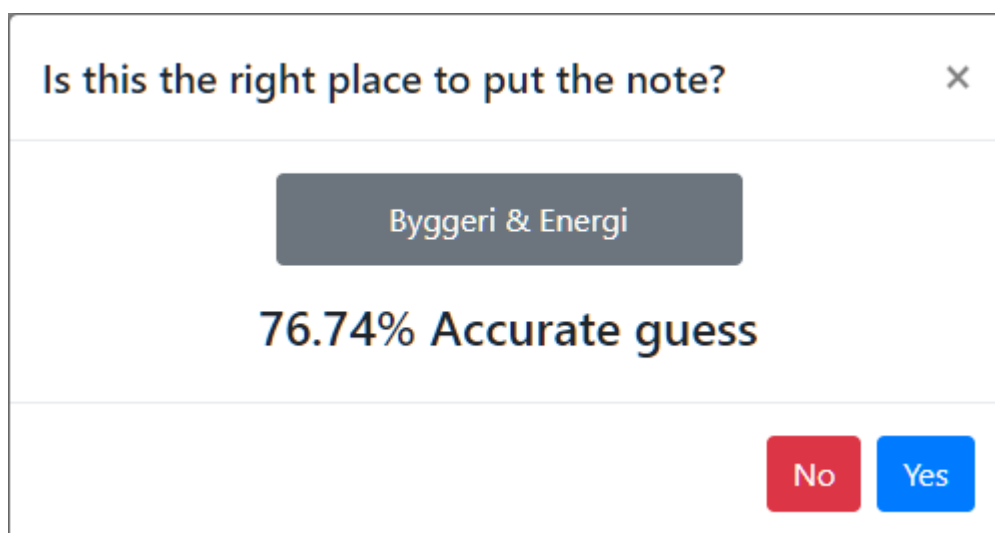
Figur 12 Kodeeksempel på genereringen af modellen for notesortering.

På Figur 12 har jeg defineret en klasse "TextClassifierModel", som indeholder alt med modellen. Når klassen initialiseres findes dataene (linje 3), som derefter bliver gemt i et "dictionary"⁶ (linje 4-6). Herefter bliver "create_dataframe"-funktionen kaldt

⁶ Dette var for at have muligheden for i fremtiden at kunne tilføje data fra forskellige kilder, og hurtigt fjerne dem igen, hvis den kilde blev upålidelig.

(linje 7). Funktionen har ingen særlig funktion nu, siden der ikke er andre kilder, som er hentet data fra i "self.filepath_dict". Hernæst bliver "create_model"-funktionen kaldt. Denne funktion står for at danne selve modellen. Sætningerne og deres tilhørende mærkater bliver adskilt på linje 22-30. Trænings- og test datasettene bliver transformeret til feature vektorer på Figur 12 på linje 35-36 (ligesom i eksemplet før Figur 11). Hernæst defineres en klassificerings variabel til et objekt af typen " LogisticRegression" fra biblioteket "sklearn" (Figur 12, linje 38). Denne variabel bliver tilpasset træningsdatasettet, hvor "X_train" er sætningerne, og "y_train" er mærkaterne for hver sætning. Nu er modellen defineret ud fra datasettet, jeg har indsamlet, og det næste er bare, at finde ud af, hvor sikker algoritmen er i dens gæt. For at få nøjagtigheden af min model har jeg brugt funktionen "score" på modellen (Figur 13), og med de data, jeg har, har jeg fået en nøjagtighed på 76.74%. På figuren kan man også se, hvordan jeg viser til brugeren, hvor høj en nøjagtighed modellen har.

```
1 def get_score(self):  
2     score = self.classifier.score(self.X_test, self.y_test)  
3     score_precent = "{0:.2f}%".format(score * 100)  
4     return score_precent
```



Figur 13 Illustration af hentning af nøjagtigheden af modellen.

5.3 KORT OM MEST RELEVANTE ANVENDTE BIBLIOTEKER

Siden der er brugt 11 forskellige biblioteker⁷ i dette projekt, har jeg valgt kun kort at gå over Flask og Sklearn, da det er de biblioteker, som står

5.3.1 Flask

Flask er et forholdsvis let webapplikationsframework. Det er designet til at være let at starte på og har samtidig muligheden for at kunne skaleres til større og mere komplekse applikationer. Flask er skrevet i programmeringssproget Python og bruger samtidig Jinja, til at indsætte data fra programmet ind i hjemmesiderne, for at kunne vise dem til brugeren.

5.3.2 Sklearn

Sklearn (også kendt som "scikits.learn") er et gratis machine learning bibliotek, som bliver brugt i programmeringssproget Python. Det har en lang række funktioner, som bl.a. klassifikation, regression, klyngedannelse og mange andre algoritmer brugt i sammenhæng med machine learning. I dette projekt har jeg brugt biblioteket til at klassificere tekst ved brug af en logistisk regressions funktion fra biblioteket.

⁷ Anvendte biblioteker: Flask, Os, Random, String, Json, Sklearn, Pandas, Sqlite3, Hashlib, Binascii Time.

6 REFERENCER

Janakiev, N. (6. oktober 2019). *Practical Text Classification With Python and Keras*.

Hentet 2. maj 2020 fra Realpython: <https://realpython.com/python-keras-text-classification/>

The Bootstrap team. (april. 28 2020). *Bootstrap*. Hentet 29. april 2020 fra Bootstrap:

<https://getbootstrap.com/>

Trelags-arkitektur. (29. april 2020). Hentet fra Informatikbeux Systime:

<https://informatikbeux.systime.dk/?id=1158>

7 BILAG

7.1 BILAG 1 - BRUGERHISTORIER

- Registrering af bruger
- Log ud
- Slet note

7.2 KILDEKODE

7.2.1 Python

7.2.1.1 *Main.py*

```
1. from note_data import Database, User
2. from API.class_classifier_model import TextClassifierModel
3.
4. from flask import Flask, request, g, render_template, session,
   redirect, url_for
5. import os, random, string, json
6.
7. app = Flask(__name__)
8. key = "very secret string"
9. app.secret_key = key
10.
11.     class_model = TextClassifierModel()
12.
13.     with app.app_context():
14.         data = Database()
15.
16.
17.     @app.teardown_appcontext
18.     def close_connection(exception):
19.         data.close_connection()
20.
21.
22.     def my_render(template, **kwargs):
23.         login_status = get_login_status()
24.         if login_status:
25.             return render_template(
26.                 template, loggedin=login_status, user=session[
27.                     "currentuser"], **kwargs
28.             )
```



```
28.         else:
29.             return render_template(template, loggedin=login_status, user="", **kwargs)
30.
31.
32.     def get_login_status():
33.         return "currentuser" in session
34.
35.
36.     def get_user_id():
37.         if get_login_status():
38.             return session["currentuser"]
39.         else:
40.             return -1
41.
42.
43.     def login_success(username, password):
44.         return data.login_success(username, password)
45.
46.
47.     def check_same_password(password, re_password):
48.         if password == re_password:
49.             return True
50.         else:
51.             return False
52.
53.
54.     def clean_dict_from_req_args(cluttered_dict):
55.         cluttered_dict = cluttered_dict[""].replace("'", '')
56.
57.         cluttered_dict = cluttered_dict[1:-1]
58.         info = json.loads(cluttered_dict)
59.         return info
60.
61.     def signup_success(username, firstname, lastname, email, password, re_password):
62.         if check_same_password(password, re_password):
63.             hashed_password = data.hash_password(password)
64.             u = User(username, firstname, lastname, email, hashed_password)
65.             if not data.signup_success(user=u):
66.                 return False
67.             return True
68.         else:
69.             return False
70.
71.
72. @app.route("/")
```

```
73.     @app.route("/login")
74.     def index():
75.         return my_render("index.html", success=False, title="l
    ogin")
76.
77.
78.     @app.route("/login_profile", methods=["POST"])
79.     def login():
80.         password = request.form["password"]
81.         username = request.form["username"]
82.
83.         if login_success(username=username, password=password)
            :
84.             # Create user object, store in session.
85.             session["currentuser"] = data.get_user_id(username
        )
86.             return redirect(f"/profile")
87.         else:
88.             session.pop("currentuser", None)
89.             return redirect(f"/")
90.
91.
92.     @app.route("/logout")
93.     def logout():
94.         session.pop("currentuser", None)
95.         return redirect(f"/")
96.
97.
98.     @app.route("/signup")
99.     def signup_site():
100.        return my_render("sign_up.html", title="signup")
101.
102.
103.     @app.route("/profile")
104.     def profile():
105.         # Get general class info
106.         classes_info = data.get_classes_info()
107.         # Get amount of notes in each class for current user
108.         num_notes_in_class_dict = data.get_num_notes_in_class(
            session["currentuser"])
109.         for i, class_info in enumerate(classes_info):
110.             i += 1
111.             class_info["num_notes"] = num_notes_in_class_dict[
                f"{i}"]
112.
113.         return my_render(
114.             "user_main.html", title="Student", success=True, c
            lasses_info=classes_info,
115.         )
```

```
116.
117.
118.     @app.route("/signup_profile", methods=["POST"])
119.     def signup():
120.         username = request.form["username"]
121.         firstname = request.form["firstname"]
122.         lastname = request.form["lastname"]
123.         email = request.form["email"]
124.         password = request.form["password"]
125.         re_password = request.form["re_password"]
126.
127.         if signup_success(
128.             username=username,
129.             firstname=firstname,
130.             lastname=lastname,
131.             email=email,
132.             password=password,
133.             re_password=re_password,
134.         ):
135.             return index()
136.         else:
137.             session.pop("currentuser", None)
138.             return my_render("sign_up.html", success=False)
139.
140.
141.     @app.route("/edit_note")
142.     def edit_note():
143.         note_info = clean_dict_from_req_args(request.args)
144.         return my_render("edit_note.html", note_info=note_info
145.         )
146.
147.     @app.route("/remove_note")
148.     def remove_note():
149.         note_info = clean_dict_from_req_args(request.args)
150.         data.remove_note(note_info["note_id"], session["currentuser"])
151.         return redirect(f"/showclass?='{note_info['class_id']}'")
152.
153.
154.     @app.route("/showclass")
155.     def showclass():
156.         class_id = clean_dict_from_req_args(request.args)
157.         class_info = data.get_class_info(class_id)
158.         notes = data.get_notes_in_class(session["currentuser"], class_id)
159.         return my_render(
```

```
160.         "class_page.html", success=True, class_info=class_
        info, notes=notes
161.     )
162.
163.
164.     @app.route("/take_notes")
165.     def take_notes():
166.         return my_render("note_writer.html")
167.
168.
169.     @app.route("/get_class_prediction", methods=["GET"])
170.     def get_class_prediction():
171.         cluttered_dict = request.args
172.         body = cluttered_dict[""].replace("'", '')
173.         class_model.prepare_data(body)
174.         prediction = class_model.predict_class()
175.         prediction_score = class_model.get_score()
176.         class_name = data.get_class_name(prediction)
177.         other_classes = data.get_other_class_names(class_name)
178.
179.         class_names = {
180.             "class_name": class_name,
181.             "other_classes1": other_classes[0],
182.             "other_classes2": other_classes[1],
183.             "prediction_score": prediction_score,
184.         }
185.         return class_names
186.
187.     @app.route("/submit_note", methods=["POST"])
188.     def submit_note():
189.         subject = request.form["subject"]
190.         body = request.form["body"]
191.         class_name = request.form["class_name"]
192.         class_id = data.get_class_id_from_name(class_name)
193.         body_for_model = class_model.prepare_model_data(body,
            class_id)
194.         class_model.add_data(body_for_model)
195.         class_model.refresh_model()
196.         data.submit_note(session["currentuser"], class_id, sub
            ject, body)
197.
198.         return redirect("/profile")
199.
200.
201.     @app.route("/submit_note_edit", methods=["POST"])
202.     def submit_note_edit():
203.         subject = request.form["subject"]
204.         body = request.form["body"]
```

```
205.         note_id = request.form["note_id"]
206.         data.edit_note(note_id, session["currentuser"], subject, body)
207.         return redirect(f"/read_note?='{note_id}''")
208.
209.
210.     @app.route("/read_note")
211.     def read_note():
212.         note_id = clean_dict_from_req_args(request.args)
213.         note_info = data.get_note_info(note_id, session["currentuser"])
214.         class_info = data.get_class_info(note_info["class_id"])
215.         return my_render(
216.             "read_note.html", success=True, note_info=note_info, class_info=class_info
217.         )
218.
219.
220.     if __name__ == "__main__":
221.         with app.app_context():
222.             data = Database()
223.
224.         app.run(debug=True)
```

7.2.1.2 Note_data.py

```
1. from flask import g, Flask
2. import sqlite3, hashlib, binascii, os, time
3.
4. ABS_FILEPATH = os.path.dirname(os.path.abspath(__file__))
5.
6.
7. class User:
8.     def __init__(self, username, firstname, lastname, email, password):
9.         self.username = username
10.        self.firstname = firstname
11.        self.lastname = lastname
12.        self.email = email
13.        self.password = password
14.
15.    def set_id(self, ID):
16.        self.id = ID
17.
18.    def __str__(self):
19.        return f"""
20.        username: {self.username}
21.        email: {self.email}
```

```
22.         password: {self.password}
23.         rank: {self.rank}""
24.
25.
26.     class Database:
27.         def __init__(self):
28.             self.DATABASE = fr"{ABS_FILEPATH}\main.db"
29.
30.             # self._create_tables()
31.
32.         def _get_db(self):
33.             db = g.get("_database", None)
34.             if db is None:
35.                 db = g._database = sqlite3.connect(self.DATABASE)
36.             return db
37.
38.         def close_connection(self):
39.             db = getattr(g, "_database", None)
40.             if db is not None:
41.                 db.close()
42.
43.         def get_user_id(self, username):
44.             db = self._get_db()
45.             c = db.cursor()
46.             c.execute("SELECT id FROM userprofiles WHERE username = ?", (username,))
47.             r = c.fetchone()
48.             # If the user doesn't exist, the result will be None
49.             if r is not None:
50.                 return r[0]
51.             else:
52.                 return None
53.
54.         def hash_password(self, password):
55.             # https://www.vitoshacademy.com/hashing-passwords-in-python/
56.             # Hash a password for storing.
57.             salt = hashlib.sha256(os.urandom(60)).hexdigest().encode("ascii")
58.             pwdhash = hashlib.pbkdf2_hmac("sha512", password.encode("utf-8"), salt, 100000)
59.             pwdhash = binascii.hexlify(pwdhash)
60.             return (salt + pwdhash).decode("ascii")
61.
62.         def verify_password(self, stored_password, provided_password):
```

```
63.         # Verify a stored password against one provided by
        user
64.         salt = stored_password[:64]
65.         stored_password = stored_password[64:]
66.         pdhash = hashlib.pbkdf2_hmac(
67.             "sha512", provided_password.encode("utf-
8"), salt.encode("ascii"), 100000
68.         )
69.         pdhash = binascii.hexlify(pdhash).decode("ascii"
)
70.         return pdhash == stored_password
71.
72.     def login_success(self, username, password):
73.         db = self._get_db()
74.         c = db.cursor()
75.         c.execute("SELECT password FROM userprofiles WHERE
username = ?", (username,))
76.         r = c.fetchone()
77.         if r is not None:
78.             db_pw = r[0]
79.         else:
80.             return False
81.         return self.verify_password(stored_password=db_pw,
provided_password=password)
82.
83.     def get_classes_info(self):
84.         class_info_list = []
85.         db = self._get_db()
86.         c = db.cursor()
87.         c.execute("SELECT * FROM classes")
88.         r = c.fetchall()
89.         for class_info in r:
90.             class_id, class_name, class_img_path, descript
ion = class_info
91.             class_info_dict = {
92.                 "class_id": class_id,
93.                 "class_name": class_name,
94.                 "class_img": class_img_path,
95.                 "description": description,
96.             }
97.             class_info_list.append(class_info_dict)
98.         return class_info_list
99.
100.    def get_class_info(self, class_id):
101.        db = self._get_db()
102.        c = db.cursor()
103.        c.execute("SELECT * FROM classes WHERE id = ?", (c
lass_id,))
104.        r = c.fetchone()
```

```
105.         class_id, class_name, class_img_path, description
    = r
106.         class_info_dict = {
107.             "class_id": class_id,
108.             "class_name": class_name,
109.             "class_img": class_img_path,
110.             "description": description,
111.         }
112.         return class_info_dict
113.
114.     def get_class_name(self, class_id):
115.         db = self._get_db()
116.         c = db.cursor()
117.         c.execute("SELECT classname FROM classes WHERE id
    = ?", (int(class_id),))
118.         r = c.fetchone()
119.         class_name = r[0]
120.         return class_name
121.
122.     def get_other_class_names(self, class_name):
123.         other_classes = []
124.         db = self._get_db()
125.         c = db.cursor()
126.         c.execute(
127.             "SELECT classname FROM classes WHERE NOT class
    name = ?", (class_name,)
128.         )
129.         r = c.fetchall()
130.         for class_name in r:
131.             other_classes.append(class_name)
132.         return other_classes
133.
134.     def get_class_id_from_name(self, class_name):
135.         db = self._get_db()
136.         c = db.cursor()
137.         c.execute("SELECT id FROM classes WHERE classname
    = ?", (str(class_name),))
138.         r = c.fetchone()
139.         class_id = r[0]
140.         return class_id
141.
142.     def get_num_notes_in_class(self, user_id):
143.         num_notes_in_class_dict = {}
144.         db = self._get_db()
145.         c = db.cursor()
146.         for i in range(1, 4):
147.             c.execute(
148.                 "SELECT COUNT(id) FROM notes WHERE user_id
    == ? AND class_id == ?",
```



```

149.         (user_id, i),
150.     )
151.     r = c.fetchone()
152.     (num,) = r
153.     num_notes_in_class_dict[f"{i}"] = int(num)
154.     return num_notes_in_class_dict
155.
156.     def get_notes_in_class(self, user_id, class_id):
157.         notes = []
158.         db = self._get_db()
159.         c = db.cursor()
160.         c.execute(
161.             "SELECT * FROM notes WHERE user_id == ? AND cl
162.             ass_id == ?",
163.             (user_id, class_id),
164.         )
165.         r = c.fetchall()
166.         for note in r:
167.             note_id, user_id, class_id, subject, body, tim
168.             estamp = note
169.             note_dict = {
170.                 "note_id": note_id,
171.                 "user_id": user_id,
172.                 "class_id": class_id,
173.                 "subject": subject,
174.                 "body": body,
175.                 "timestamp": timestamp,
176.             }
177.             notes.append(note_dict)
178.         return notes
179.
180.     def get_note_info(self, note_id, user_id):
181.         db = self._get_db()
182.         c = db.cursor()
183.         c.execute(
184.             "SELECT * FROM notes WHERE id == ? AND user_id
185.             == ?", (note_id, user_id),
186.         )
187.         r = c.fetchone()
188.         note_id, user_id, class_id, subject, body, timesta
189.         mp = r
190.         note_dict = {
191.             "note_id": note_id,
192.             "user_id": user_id,
193.             "class_id": class_id,
194.             "subject": subject,
195.             "body": body,
196.             "timestamp": timestamp,
197.         }

```

```
194.         return note_dict
195.
196.     def remove_note(self, note_id, user_id):
197.         db = self._get_db()
198.         c = db.cursor()
199.         c.execute(
200.             "DELETE FROM notes WHERE id = ? AND user_id =
201.             ?", (note_id, user_id),
202.             )
203.         db.commit()
204.
205.     def submit_note(self, user_id, class_id, subject, body
206.         ):
207.         db = self._get_db()
208.         c = db.cursor()
209.         c.execute(
210.             "INSERT INTO notes (user_id, class_id, subject
211.             , body) VALUES (?, ?, ?, ?)",
212.             (user_id, class_id, subject, body),
213.             )
214.         db.commit()
215.
216.     def edit_note(self, note_id, user_id, subject, body):
217.
218.         db = self._get_db()
219.         c = db.cursor()
220.         c.execute(
221.             "UPDATE notes SET subject=?, body=? WHERE id =
222.             ? AND user_id=?",
223.             (subject, body, note_id, user_id),
224.             )
225.         db.commit()
226.
227.     def check_existing_username(self, username):
228.         db = self._get_db()
229.         c = db.cursor()
230.         c.execute("SELECT username from userprofiles WHERE
231.             username=?", (username,))
232.         r = c.fetchone()
233.         if r == None:
234.             return False
235.         return True
236.
237.     def signup_success(self, user: User):
238.         db = self._get_db()
239.         c = db.cursor()
240.         fullname = f"{user.firstname};{user.lastname}"
241.         if not self.check_existing_username(user.username)
242.         :
```

```
236.         c.execute(
237.             "INSERT INTO userprofiles (username, email
    , password, fullname) VALUES (?, ?, ?, ?)",
238.             (user.username, user.email, user.password,
    fullname),
239.             )
240.         db.commit()
241.         return True
242.     return False
243.
244.     def get_username(self, user_id):
245.         db = self._get_db()
246.         c = db.cursor()
247.         c.execute("SELECT username FROM userprofiles WHERE
    id = ?", (user_id,))
248.         r = c.fetchone()
249.         # If the user doesn't exist, the result will be No
    ne
250.         if r is not None:
251.             return r[0]
252.         else:
253.             return None
254.
255.     def get_fullname(self, user_id):
256.         db = self._get_db()
257.         c = db.cursor()
258.         c.execute("SELECT fullname FROM userprofiles WHERE
    id = ?", (user_id,))
259.         r = c.fetchone()
260.         # If the user doesn't exist, the result will be No
    ne
261.         if r is not None:
262.             fullname = r[0].replace(";", " ")
263.             return fullname
264.         else:
265.             return None
266.
267.     def _drop_tables(self):
268.         db = self._get_db()
269.         c = db.cursor()
270.
271.         try:
272.             c.execute("""DROP TABLE IF EXISTS userprofiles
    ;""")
273.             c.execute("""DROP TABLE IF EXISTS classes;""")
274.             c.execute("""DROP TABLE IF EXISTS notes;""")
275.         except Exception as e:
276.             print(e)
```

```
277.         db.commit()
278.
279.     def _create_tables(self):
280.         db = self._get_db()
281.         c = db.cursor()
282.
283.         try:
284.             c.execute(
285.                 """CREATE TABLE IF NOT EXISTS userprofiles
286.                 (
287.                     id INTEGER PRIMARY KEY,
288.                     username TEXT,
289.                     email TEXT,
290.                     password TEXT,
291.                     fullname VARCHAR(128) NOT NULL);"""
292.             )
293.         except Exception as e:
294.             print(e)
295.
296.         try:
297.             c.execute(
298.                 """CREATE TABLE IF NOT EXISTS classes (
299.                     id INTEGER PRIMARY KEY,
300.                     classname TEXT,
301.                     img_path TEXT,
302.                     class_description TEXT);"""
303.             )
304.         except Exception as e:
305.             print(e)
306.
307.         try:
308.             c.execute(
309.                 """CREATE TABLE IF NOT EXISTS notes (
310.                     id INTEGER PRIMARY KEY,
311.                     user_id INT,
312.                     class_id INT,
313.                     subject TEXT,
314.                     body TEXT,
315.                     timestamp DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP);"""
316.             )
317.         except Exception as e:
318.             print(e)
319.
320.         db.commit()
321.
322.         # Create testing profiles
323.         pass1 = self.hash_password("1234")
```

```
324.         try:
325.             c.execute(
326.                 """
327.                 INSERT INTO userprofiles (username, email,
328.                 password, fullname)
329.                 SELECT "user", "andreasgdp@gmail.com", ?,
330.                 "Andreas;Petersen"
331.                 WHERE NOT EXISTS (
332.                 SELECT 1 FROM userprofiles WHERE usern
333.                 ame = "user" AND
334.                 email = "andreasgdp@gmail.com" AND
335.                 fullname = "Andreas;Petersen");
336.                 """,
337.                 (pass1,),
338.             )
339.             c.execute(
340.                 """
341.                 INSERT INTO userprofiles (username, email,
342.                 password, fullname)
343.                 SELECT "user2", "mand@gmail.com", ?, "Mand
344.                 e;Manden"
345.                 WHERE NOT EXISTS (
346.                 SELECT 1 FROM userprofiles WHERE usern
347.                 ame = "teacher" AND
348.                 email = "mand@gmail.com" AND
349.                 fullname = "Mande;Manden");
350.                 """,
351.                 (pass1,),
352.             )
353.         except Exception as e:
354.             print(e)
355.
356.         # Create testing classes
357.         try:
358.             c.execute(
359.                 """
360.                 INSERT INTO classes (classname, img_path,
361.                 class_description) VALUES (
362.                 "Byggeri & Energi",
363.                 "./static/Images/byggeri & energi.jpg"
364.                 ,
365.                 "Det er Byg og Hyg");
366.                 """,
367.             )
368.             c.execute(
369.                 """
370.                 INSERT INTO classes (classname, img_path,
371.                 class_description) VALUES (
372.                 "Dansk",
```

```
364.         "./static/Images/dansk.png",
365.         "Det er dansk");
366.         """
367.     )
368.     c.execute(
369.         """
370.         INSERT INTO classes (classname, img_path,
371.         class_description) VALUES ("Matematik",
372.         "./static/Images/matematik.jpg",
373.         "Det er mat");
374.         """
375.     )
376. except Exception as e:
377.     print(e)
378.
379. # Create testing notes
380. try:
381.     # ? Dansk
382.     c.execute(
383.         """
384.         INSERT INTO notes (user_id, class_id, subj
385.         ect, body) VALUES (
386.             1,
387.             2,
388.             "Romantikken",
389.             "Romantikens afgørende dyder er Intui
390.             tion og Fantasi"
391.         );
392.         """
393.     )
394.     c.execute(
395.         """
396.         INSERT INTO notes (user_id, class_id, subj
397.         ect, body) VALUES (
398.             1,
399.             2,
400.             "Romantikken",
401.             "Den såkaldte organicisme eller organi
402.             smetanke går ud på, at..."
403.         );
404.         """
405.     )
406.     c.execute(
407.         """
408.         INSERT INTO notes (user_id, class_id, subj
409.         ect, body) VALUES (
410.             1,
411.             2,
412.             "Dokumentar",
```

```
407.         "Dokumentaren handler om MKs barndom,  
    hans forældre og..."  
408.         );  
409.         ""  
410.     )  
411.     # ? Matematik  
412.     c.execute(  
413.         ""  
414.         INSERT INTO notes (user_id, class_id, subj  
    ect, body) VALUES (  
415.             1,  
416.             3,  
417.             "Differentialligninger",  
418.             "En eller anden tekst om differentiall  
    igninger"  
419.         );  
420.         ""  
421.     )  
422.     c.execute(  
423.         ""  
424.         INSERT INTO notes (user_id, class_id, subj  
    ect, body) VALUES (  
425.             1,  
426.             3,  
427.             "Simpel matematik",  
428.             "2 + 2 er 4 minus 1 er 3 hurtig matema  
    tik 😊"  
429.         );  
430.         ""  
431.     )  
432.     # ? Byggeri & Energi  
433.     c.execute(  
434.         ""  
435.         INSERT INTO notes (user_id, class_id, subj  
    ect, body) VALUES (  
436.             1,  
437.             1,  
438.             "Dimensionering",  
439.             "Det er matematik men i byg"  
440.         );  
441.         ""  
442.     )  
443.  
444.     except Exception as e:  
445.         print(e)  
446.  
447.     db.commit()  
448.     print("Database tables created")  
449.
```

```
450.  
451.     if __name__ == "__main__":  
452.         app = Flask(__name__)  
453.         key = "very secret string"  
454.         app.secret_key = key  
455.         with app.app_context():  
456.             data = Database()  
457.             data._drop_tables()  
458.             data._create_tables()
```

7.2.1.3 Class_classifier_model.py

```
1. from sklearn.linear_model import LogisticRegression  
2. from sklearn.model_selection import train_test_split  
3. from sklearn.feature_extraction.text import CountVectorizer  
4. import pandas as pd  
5. import os  
6.  
7. ABS_FILEPATH = os.path.dirname(os.path.abspath(__file__))  
8.  
9.  
10.     # Labels:  
11.     # 1: Byg  
12.     # 2: Dan  
13.     # 3: Mat  
14.  
15.  
16.     class TextClassifierModel:  
17.         def __init__(self):  
18.             self.model_file = fr"{ABS_FILEPATH}\data\classes\c  
combined_test.txt"  
19.             self.filepath_dict = {  
20.                 "combined": self.model_file,  
21.             }  
22.             self.create_dataframe()  
23.             self.create_model()  
24.  
25.         def create_dataframe(self):  
26.             df_list = []  
27.             for text_class, filepath in self.filepath_dict.items():  
28.                 df = pd.read_csv(filepath, names=["sentence",  
"label"], sep="\t")  
29.                 df["text_class"] = text_class  
30.                 df_list.append(df)  
31.  
32.             self.df = pd.concat(df_list)  
33.
```



```
34.         def create_model(self):
35.             self.df_combined = self.df[self.df["text_class"] =
= "combined"]
36.
37.             sentences = self.df_combined["sentence"].values
38.             y = self.df_combined["label"].values
39.
40.             (
41.                 self.sentences_train,
42.                 self.sentences_test,
43.                 self.y_train,
44.                 self.y_test,
45.             ) = train_test_split(sentences, y, test_size=0.25,
random_state=1000)
46.
47.             self.vectorizer = CountVectorizer()
48.             self.vectorizer.fit(self.sentences_train)
49.
50.             self.X_train = self.vectorizer.transform(self.sent
ences_train)
51.             self.X_test = self.vectorizer.transform(self.sente
nces_test)
52.
53.             self.classifier = LogisticRegression()
54.             self.classifier.fit(self.X_train, self.y_train)
55.
56.         def add_data(self, data):
57.             with open(self.model_file, "ab") as f:
58.                 f.write(data)
59.
60.         def refresh_model(self):
61.             self.create_dataframe()
62.             self.create_model()
63.
64.         def print_score(self):
65.             score = self.classifier.score(self.X_test, self.y_
test)
66.             print("Accuracy:", score)
67.
68.         def get_score(self):
69.             score = self.classifier.score(self.X_test, self.y_
test)
70.             score_precent = "{0:.2f}%".format(score * 100)
71.             return score_precent
72.
73.         def prepare_model_data(self, body, class_id):
74.             body_for_model = body.replace("\t", " ")
75.             body_for_model = str.join(" ", body_for_model.spli
tlines())
```

```

76.         body_for_model = body_for_model + "\t" + str(class
    _id)
77.         body_for_model = "\n" + body_for_model
78.         body_for_model = body_for_model.encode("UTF-8")
79.         return body_for_model
80.
81.     def prepare_data(self, data):
82.         data = [data]
83.         self.prepare_vectorizer = CountVectorizer()
84.         self.prepare_vectorizer.fit(data)
85.         self.prepared_data = self.vectorizer.transform(dat
    a)
86.
87.     def predict_class(self):
88.         prediction = self.classifier.predict(self.prepared
    _data)
89.         return prediction[0]
90.
91.
92.     if __name__ == "__main__":
93.         model = TextClassifierModel()
94.         model.print_score()
95.         test_data = str(input("Give me some text!: "))
96.         model.prepare_data(test_data)
97.         prediction = model.predict_class()
98.         print(f"Prediction: {prediction}")

```

7.2.1.4 Test.py

Denne fil har ikke noget med hovedprogrammet at gøre, men jeg har brugt det til at skrive noget kode, for at finde ud af, hvordan det virkede. Koden der står nu er brugt til at eksemplificere noget af brugen af machine learning algoritmen.

```

1. from sklearn.feature_extraction.text import CountVectorizer
2.
3. sentences = ['I dansk kan man skrive med bogstager', 'I matema
    tik kan man regne med tal']
4.
5. vectorizer = CountVectorizer(min_df=0, lowercase=False)
6. vectorizer.fit(sentences)
7.
8. print(vectorizer.vocabulary_)
9. # {'dansk': 1, 'kan': 2, 'man': 3, 'skrive': 7, 'med': 5, '
10.   # bogstager': 0, 'matematik': 4, 'regne': 6, 'tal': 8}
11. print(vectorizer.transform(sentences).toarray())
12. # [[1 1 1 1 0 1 0 1 0]
13.   # [0 0 1 1 1 1 1 0 1]]

```

7.2.2 HTML

7.2.2.1 Header.html

```
1. <!DOCTYPE html>
2. <html lang="en">
3.   <head>
4.     <!-- Required meta tags -->
5.     <meta charset="utf-8" />
6.     <meta
7.       name="viewport"
8.       content="width=device-width, initial-
scale=1, shrink-to-fit=no"
9.     />
10.
11.     <!-- Bootstrap CSS -->
12.     <link
13.       rel="stylesheet"
14.       href="https://stackpath.bootstrapcdn.com/boot-
strap/4.1.3/css/bootstrap.min.css"
15.       integrity="sha384-
MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLP
MO"
16.       crossorigin="anonymous"
17.     />
18.     <link
19.       rel="stylesheet"
20.       type="text/css"
21.       href="{ { url_for('static', filename='main.css'
) } }"
22.     />
23.     {% if title != "login" and title != "signup" and t
itle !=
24.       "login_profile" and success==true %}
25.     <nav class="navbar navbar-expand-lg navbar-
dark bg-dark">
26.       {% if title == "login" or title == "signup" %}
27.       <a class="navbar-
brand" href="/">NoteSort</a>
28.       {% elif success %}
29.       <a class="navbar-
brand" href="/profile">NoteSort</a>
30.       {% endif %}
31.       <button
32.         class="navbar-toggler"
33.         type="button"
34.         data-toggle="collapse"
```

```

35.         data-target="#navbarSupportedContent"
36.         aria-controls="navbarSupportedContent"
37.         aria-expanded="false"
38.         aria-label="Toggle navigation"
39.     >
40.         <span class="navbar-toggler-
icon"></span>
41.     </button>
42.
43.     <div class="collapse navbar-
collapse" id="navbarSupportedContent">
44.         <ul class="navbar-nav mr-auto">
45.             <!-- <li class="nav-item active">
46.                 <a class="nav-
link" href="/">Home <span class="sr-
only">(current)</span></a>
47.             </li> -->
48.         </ul>
49.         <form class="form-inline my-2 my-lg-0 mr-
sm-2">
50.             <input
51.                 class="form-control mr-sm-2"
52.                 type="search"
53.                 placeholder="Search"
54.                 aria-label="Search"
55.             />
56.             <button
57.                 class="btn btn-outline-success my-
2 my-sm-0"
58.                 type="submit"
59.             >
60.                 Search
61.             </button>
62.         </form>
63.         <ul class="navbar-nav mr-sm-0">
64.             <li class="nav-item active">
65.                 <a class="btn btn-outline-
danger" href="/logout"
66.                     >Log out <span class="sr-
only">(current)</span></a>
67.             >
68.         </li>
69.     </ul>
70. </div>
71. </nav>
72. {% else %}
73. <nav class="navbar navbar-expand-lg navbar-
dark bg-dark">

```

```
74.         {% if title == "login" or title == "signup" %}
75.         <a class="navbar-
brand" href="/">NoteSort</a>
76.         {% else %}
77.         <a class="navbar-
brand" href="/profile">NoteSort</a>
78.         {% endif %}
79.     </nav>
80.     {% endif %}
81.     <title>NoteSort</title>
82. </head>
83. <body>
84.     {% block content %}{% endblock content %}
85.     <!-- Optional JavaScript -->
86.     <!--
- jQuery first, then Popper.js, then Bootstrap JS -->
87.     <script src="https://code.jquery.com/jquery-
3.1.1.min.js"></script>
88.     <!-- <script
89.         src="https://code.jquery.com/jquery-
3.3.1.slim.min.js"
90.         integrity="sha384-
q8i/X+965Dz00rT7abK41JStQIAVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6ji
zo"
91.         crossorigin="anonymous"
92.     ></script> -->
93.     <script
94.         src="https://cdnjs.cloudflare.com/ajax/libs/po
pper.js/1.14.3/umd/popper.min.js"
95.         integrity="sha384-
ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIpM
49"
96.         crossorigin="anonymous"
97.     ></script>
98.     <script
99.         src="https://stackpath.bootstrapcdn.com/bootst
rap/4.1.3/js/bootstrap.min.js"
100.        integrity="sha384-
ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEUL
Ty"
101.        crossorigin="anonymous"
102.    ></script>
103. </body>
104. </html>
```

7.2.2.2 Index.html

```
1. {% extends "header.html" %} {% block content %}
2. <div class="center">
3.     <div class="container log-in-container">
4.         <!-- Default form login -->
5.         <!--
6.         - https://mdbootstrap.com/docs/jquery/forms/basic/ -->
7.         <form
8.             class="text-center border border-light p-5"
9.             action="login_profile"
10.            method="POST"
11.        >
12.            <p class="h4 mb-4">Sign in</p>
13.            <!-- Username -->
14.            <input
15.                type="username"
16.                id="defaultLoginFormEmail"
17.                class="form-control mb-4"
18.                placeholder="Username"
19.                name="username"
20.            />
21.            <!-- Password -->
22.            <input
23.                type="password"
24.                id="defaultLoginFormPassword"
25.                class="form-control mb-2"
26.                placeholder="Password"
27.                name="password"
28.            />
29.            <!-- Sign in button -->
30.            <button type="submit" class="btn btn-
31.                primary m-3">Sign in</button>
32.            <!-- Register -->
33.            <p>
34.                Not a member?
35.                <a href="/signup">Register</a>
36.            </p>
37.        </form>
38.        <!-- Default form login -->
39.        {% endblock content %}
40.    </div>
41. </div>
```

7.2.2.3 Sign_up.html

```
1. {% extends "header.html" %} {% block content %}
2. <div class="center">
3.     <div class="container sign-up-container">
4.         <!-- Default form register -->
5.         <!--
6.         - https://mdbootstrap.com/docs/jquery/forms/basic/ -->
7.         <form class="text-center border border-light p-
8.         5" action="signup_profile" method="POST">
9.             <p class="h4 mb-4">Sign up</p>
10.             <!-- Username -->
11.             <input
12.                 type="username"
13.                 id="defaultRegisterFormEmail"
14.                 class="form-control mb-4"
15.                 placeholder="Username"
16.                 name="username"
17.             />
18.             <div class="form-row mb-4">
19.                 <div class="col">
20.                     <!-- First name -->
21.                     <input
22.                         type="text"
23.                         id="defaultRegisterFormFirstName"
24.                         class="form-control"
25.                         placeholder="First name"
26.                         name="firstname"
27.                     />
28.                 </div>
29.                 <div class="col">
30.                     <!-- Last name -->
31.                     <input
32.                         type="text"
33.                         id="defaultRegisterFormLastName"
34.                         class="form-control"
35.                         placeholder="Last name"
36.                         name="lastname"
37.                     />
38.                 </div>
39.             </div>
40.             <!-- E-mail -->
41.             <input
42.                 type="email"
```

```
44.         id="defaultRegisterFormEmail"
45.         class="form-control mb-4"
46.         placeholder="E-mail"
47.         name="email"
48.     />
49.
50.     <!-- Password -->
51.     <input
52.         type="password"
53.         id="defaultRegisterFormPassword"
54.         class="form-control"
55.         placeholder="Password"
56.         aria-
57.         describedby="defaultRegisterFormPasswordHelpBlock"
58.         name="password"
59.     />
60.     <small
61.         id="defaultRegisterFormPasswordHelpBlock"
62.         class="form-text text-muted mb-4"
63.     >
64.         At least 8 characters and 1 digit
65.     </small>
66.     <!-- Repeat password -->
67.     <input
68.         type="password"
69.         id="defaultRegisterFormPassword"
70.         class="form-control"
71.         placeholder="Repeat password"
72.         aria-
73.         describedby="defaultRegisterFormPasswordHelpBlock"
74.         name="re_password"
75.     />
76.     </small>
77.     <!-- Sign up button -->
78.     <button type="submit" class="btn m-4 btn-
79.     primary">Sign in</button>
80.     <!-- Register -->
81.     <p>
82.         Already a member?
83.         <a href="/">Log in</a>
84.     </p>
85. </form>
86.
87. <!-- Default form register -->
88. </div>
```



```
89.     </div>
90.
91.     {% endblock content %}
```

7.2.2.4 User_main.html

```
1. {% extends "header.html" %} {% block content %}
2.
3. <div class="bg img-fluid"></div>
4.
5. <div class="container">
6.     <a
7.         href="/take_notes"
8.         type="button"
9.         class="btn btn-primary btn-lg btn-block center "
10.        style="width:50%; margin-top:1.5rem"
11.    >
12.        New note
13.    </a>
14.    <div class="card-deck" style="margin-top:1.5rem">
15.        {% for class_info in classes_info %}
16.        <a
17.            href="/showclass?={{ class_info['class_id'] }}"
18.            class="card class-card"
19.        >
20.            
25.            <div class="card-body">
26.                <h5 class="card-
27.                title">{{ class_info['class_name'] }}</h5>
28.                <div style="height: 60px">
29.                    <p
30.                        class="card-text"
31.                        style="white-
32.                        space: nowrap; overflow: hidden; text-overflow: ellipsis;"
33.                    >
34.                        {{ class_info['description'] }}
35.                    </p>
36.                </div>
37.                <div class="card">
38.                    <div class="container" style="text-
39.                    align: center;">
40.                        <h5>
41.                            Number of notes: <br /><span
```

```

39.                                     class="badge badge-
    secondary"
40.                                     >{{ class_info['num_notes'
    ] }}</span>
41.                                     >
42.                                     </h5>
43.                                     </div>
44.                                 </div>
45.                            </div>
46.                        </a>
47.                    {% endfor %}
48.                </div>
49.            </div>
50.
51.    {% endblock content %}

```

7.2.2.5 Note_writer.html - ser anderledes ud grundet problemer med "www.planetb.ca"

```

1. {% extends "header.html" %} {% block content %}
2. <a href="/profile" class="btn btn-light back-btn" type="button">Go
   back</a>
3. <div class="container">
4.     <form action="submit_note" method="POST">
5.         <div class="form-group">
6.             <label for="exampleFormControlInput1">Enter
subject</label>
7.             <input
8.                 type="text"
9.                 class="form-control"
10.                 id="exampleFormControlInput1"
11.                 placeholder="Subject"
12.                 name="subject"
13.                 required
14.             />
15.         </div>
16.         <div class="form-group">
17.             <label for="exampleFormControlTextarea1">The
note:</label>
18.             <textarea
19.                 class="form-control"
20.                 id="exampleFormControlTextarea1"
21.                 rows="3"
22.                 name="body"
23.                 required
24.             ></textarea>
25.         </div>

```

```
26.         <!-- Button trigger modal -->
27.         <button
28.             type="button"
29.             class="btn btn-primary"
30.             data-toggle="modal"
31.             data-target="#exampleModalCenter"
32.             onclick="get_class_prediction1()"
33.         >
34.             Submit
35.         </button>
36.         <script>
37.             function get_class_prediction1() {
38.                 let body = document.getElementById(
39.                     'exampleFormControlTextarea1'
40.                 ).value;
41.
42.                 $.ajax({
43.                     url: '/get_class_prediction?=' + body,
44.                     cache: false,
45.                     type: 'GET',
46.                     success: function (data) {
47.                         console.log('shit works!' + data);
48.                         let class_name = data['class_name'];
49.                         let wrong_class_name1 =
50.                             data['other_classes1'];
51.                         let wrong_class_name2 =
52.                             data['other_classes2'];
53.                         let prediction_score =
54.                             data['prediction_score'];
55.                         $('#correct-class').empty();
56.                         $('#correct-class').text(class_name);
57.                         $('#accuracy').empty();
58.                         $('#accuracy').text(
59.                             `${prediction_score} Accurate
60.                             guess`
61.                         );
62.                         $('#wrong-class-name1').empty();
63.                         $('#wrong-class-
64.                             name1').text(wrong_class_name1);
65.                         $('#wrong-class-name2').empty();
66.                         $('#wrong-class-
67.                             name2').text(wrong_class_name2);
68.
69.                         let wrong_class_name1_inp =
70.                             document.getElementById(
71.                                 'wrong-class-name1-inp'
72.                             );
73.                         wrong_class_name1_inp.value =
74.                             wrong_class_name1;
75.                         let wrong_class_name2_inp =
76.                             document.getElementById(
77.                                 'wrong-class-name2-inp'
78.                             );
79.                         wrong_class_name2_inp.value =
80.                             wrong_class_name2;
81.                         $('#prediction-score').empty();
82.                         $('#prediction-score').text(
83.                             prediction_score
84.                         );
85.                     }
86.                 });
87.             }
88.         </script>
89.     </div>
90. </div>
```

```

67.                                wrong_class_name1_inp.value =
wrong_class_name1;
68.
69.                                let wrong_class_name2_inp =
document.getElementById(
70.                                'wrong-class-name2-inp'
71.                                );
72.                                wrong_class_name2_inp.value =
wrong_class_name2;
73.
74.                                let class_name_hidden =
document.getElementById(
75.                                'correct-class-hidden'
76.                                );
77.                                class_name_hidden.value = class_name;
78.                                },
79.                                error: function (err) {},
80.                                });
81.                                }
82.                                </script>
83.                                <a href="/profile" type="button" class="btn btn-
secondary">Cancel</a>
84.                                <!-- Modal -->
85.                                <div
86.                                    class="modal fade"
87.                                    id="exampleModalCenter"
88.                                    tabindex="-1"
89.                                    role="dialog"
90.                                    aria-labelledby="exampleModalCenterTitle"
91.                                    aria-hidden="true"
92.                                >
93.                                    <div class="modal-dialog modal-dialog-
centered" role="document">
94.                                        <div class="modal-content">
95.                                            <div class="modal-header">
96.                                                <h5 class="modal-
title" id="exampleModalLongTitle">
97.                                                    Is this the right place to put the
note?
98.                                                </h5>
99.                                                <button
100.                                                    type="button"
101.                                                    class="close"
102.                                                    data-dismiss="modal"
103.                                                    aria-label="Close"
104.                                                >
105.                                                    <span aria-
hidden="true">&times;</span>
106.                                                </button>

```

```
107.         </div>
108.         <div class="modal-body class-guess-
            container">
109.             <div
110.                 type="button"
111.                 class="btn btn-secondary class-
                    guess"
112.                     id="correct-class"
113.                 >
114.                     Dansk
115.             </div>
116.             <h4 id="accuracy" style="margin-top:
                1rem;">
117.                 0% Accurate
118.             </h4>
119.             <input
120.                 type="hidden"
121.                 name="class_name"
122.                 id="correct-class-hidden"
123.                 value="Dansk"
124.             />
125.         </div>
126.         <div class="modal-footer">
127.             <button
128.                 id="spoiler_knap"
129.                 type="button"
130.                 class="btn btn-danger"
131.                 onclick="show_spoiler()"
132.             >
133.                 No
134.             </button>
135.             <script type="text/javascript">
136.                 function show_spoiler() {
137.                     var x =
138.                         document.getElementById(
139.                             'dropdownFooter'
140.                         );
141.                     if (x.style.display ===
142.                         'none') {
143.                         x.style.display = 'block';
144.                     } else {
145.                         x.style.display = 'none';
146.                     }
147.                     var knap =
148.                         document.getElementById(
149.                             'spoiler_knap'
150.                         );
151.                     if (x.style.display ===
152.                         'block') {
```

```

149.         document.querySelector(
150.             '#spoiler_knap'
151.         ).innerText = 'Woops';
152.     } else {
153.         document.querySelector(
154.             '#spoiler_knap'
155.         ).innerText = 'No';
156.     }
157. }
158. </script>
159. <script>
160.     function wrong_class_name1() {
161.         let wcn =
162.             document.getElementById(
163.                 'wrong-class-name1-inp'
164.             ).value;
165.         let class_name_hidden =
166.             document.getElementById(
167.                 'correct-class-hidden'
168.             );
169.         class_name_hidden.value = wcn;
170.         $('#correct-class').empty();
171.         $('#correct-class').text(wcn);
172.     }
173. </script>
174. <script>
175.     function wrong_class_name2() {
176.         let wcn =
177.             document.getElementById(
178.                 'wrong-class-name2-inp'
179.             ).value;
180.         console.log(wcn);
181.         let class_name_hidden =
182.             document.getElementById(
183.                 'correct-class-hidden'
184.             );
185.         class_name_hidden.value = wcn;
186.         $('#correct-class').empty();
187.         $('#correct-class').text(wcn);
188.     }
189. </script>
190. <button type="submit" class="btn btn-
primary">
191.     Yes
192. </button>
</div>
<div
class="modal-footer"

```

```
193.         style="display: none;"
194.         id="dropdownFooter"
195.     >
196.         <div class="dropdown">
197.             <button
198.                 class="btn btn-info dropdown-
toggle"
199.                 type="button"
200.                 id="dropdownMenuButton"
201.                 data-toggle="dropdown"
202.                 aria-haspopup="true"
203.                 aria-expanded="false"
204.             >
205.                 Select right class
206.             </button>
207.             <div
208.                 class="dropdown-menu"
209.                 aria-
labelledby="dropdownMenuButton"
210.             >
211.                 <a
212.                     class="dropdown-item"
213.                     href="#"
214.                     id="wrong-class-name1"
215.                     onclick="wrong_class_name1
()"
216.                 >Action</a>
217.             >
218.             <input
219.                 type="hidden"
220.                 name="wrong_class_name1_in
p_name"
221.                 id="wrong-class-name1-inp"
222.                 value="Action"
223.             />
224.             <a
225.                 class="dropdown-item"
226.                 href="#"
227.                 id="wrong-class-name2"
228.                 onclick="wrong_class_name2
()"
229.             >Action</a>
230.             >
231.             <input
232.                 type="hidden"
233.                 name="wrong_class_name2_in
p_name"
234.                 id="wrong-class-name2-inp"
235.                 value="Action"
```

```

236.         />
237.     </div>
238. </div>
239. </div>
240. </div>
241. </div>
242. </div>
243. </form>
244. </div>
245. {% endblock content %}

```

7.2.2.6 Class_page.html

```

1. {% extends "header.html" %} {% block content %}
2. <a href="/profile" class="btn btn-light back-
   btn" type="button">Go back</a>
3. <div class="container">
4.     <h1 class="class-
   heading">{{ class_info['class_name'] }}</h1>
5.     <div class="table-responsive">
6.         <table class="table table-striped table-hover">
7.             <thead>
8.                 <tr>
9.                     <th scope="col">#</th>
10.                    <th scope="col">Subject</th>
11.                    <th scope="col">Body</th>
12.                    <th scope="col">Modified</th>
13.                </tr>
14.            </thead>
15.            <tbody>
16.                {% for note in notes %} {% set notenote_id
   = note['note_id'] %}
17.                <tr>
18.                    <td>{{ note['subject'] }}</td>
19.                    <td>{{ note['body'] }}</td>
20.                    <td>{{ note['timestamp'] }}</td>
21.                </tr>
22.            </tbody>
23.        </table>
24.    </div>
25. {% endfor %}
26. </div>
27. </div>
28.
29.

```



```
30.    {% endblock content %}
```

7.2.2.7 Read_note.html

```
1.  {% extends "header.html" %} {% block content %}
2.  <a href="/showclass?={{ class_info['class_id'] }}" class="btn btn-light back-btn" type="button"
3.      >Go back</a>
4.  >
5.  <div class="container">
6.      <h1 class="class-
7.      heading">{{ class_info['class_name'] }}</h1>
8.      <div class="row h-100">
9.          <div class="col-sm-12 my-auto">
10.             <div class="card card-block w-100">
11.                 <h1>{{ note_info['subject'] }}</h1>
12.                 <div class="scrollbar scrollbar-
13.                 primary pre-scrollable">
14.                     {{ note_info['body'] }}
15.                 </div>
16.             </div>
17.             <br />
18.             <a
19.                 name=""
20.                 id=""
21.                 class="btn btn-primary"
22.                 href="/edit_note?={{ note_info }}"
23.                 role="button"
24.                 >Edit</a>
25.             >
26.             <a
27.                 name=""
28.                 id=""
29.                 class="btn btn-danger"
30.                 href="/remove_note?={{ note_info }}"
31.                 role="button"
32.                 >Trash</a>
33.             >
34.         </div>
35.     </div>
36. {% endblock content %}
```

7.2.2.8 *Edit_node.html* - ser anderledes ud grundet problemer med "www.planetb.ca"

```
1. {% extends "header.html" %} {% block content %}
2. <a href="/read_note?={{ note_info['note_id'] }}" class="btn btn-
   light back-btn" type="button"
3.     >Go back</a>
4. >
5. <div class="container">
6.     <h1 class="class-heading">Editing note</h1>
7.     <form action="submit_note_edit" method="POST">
8.         <div class="form-group">
9.             <input type="hidden" name="note_id" id="correct-class-
   hidden" value="{{ note_info['note_id'] }}">
10.             <label for="exampleFormControlInput1">Enter
   subject</label>
11.             <input
12.                 type="text"
13.                 class="form-control"
14.                 id="exampleFormControlInput1"
15.                 name="subject"
16.                 value="{{ note_info['subject'] }}"
17.                 required
18.             />
19.         </div>
20.         <div class="form-group">
21.             <label for="exampleFormControlTextarea1">The
   note:</label>
22.             <textarea
23.                 class="form-control"
24.                 id="exampleFormControlTextarea1"
25.                 rows="7"
26.                 name="body"
27.                 required
28.             >
29.                 {{ note_info['body'] }}
30.             </textarea>
31.             >
32.         </div>
33.         <!-- !<button type="button" class="btn btn-
   primary"></button> -->
34.         <!-- Button trigger modal -->
35.         <a href="/read_note?={{ note_info['note_id']
   }}" type="button" class="btn btn-secondary"
36.             >Cancel</a>
37.     >
38.     <button
39.         type="submit"
40.         class="btn btn-primary"
```

```
41.         data-target="#exampleModalCenter"
42.     >
43.         Submit
44.     </button>
45.     <a
46.         href="/remove_note?={{ note_info }}"
47.         type="button"
48.         class="btn btn-danger"
49.     >Trash</a>
50. >
51. </form>
52. </div>
53. {% endblock content %}
```