

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.

Notetagningshjemmeside

NoteSort

Synopsis

Programmering

2020

Programmering

Eksamensprojekt

# Titelblad

## Vejledning

Søren Præstegaard (SPR)

## Fag

Programmering B

## Klasse og skole

3.D - Odense Tekniske Gymnasium

## Antal tegn

19193 af 19200 tegn (8 normalsider)

## Dato

4. maj 2020

# Indholdsfortegnelse

[1 Titelblad 2](#_Toc39484643)

[1.1 Vejledning 2](#_Toc39484644)

[1.2 Fag 2](#_Toc39484645)

[1.3 Klasse og skole 2](#_Toc39484646)

[1.4 Antal tegn 2](#_Toc39484647)

[1.5 Dato 2](#_Toc39484648)

[2 Indholdsfortegnelse 3](#_Toc39484649)

[3 Indledning 5](#_Toc39484650)

[3.1 Opgaveformulering 5](#_Toc39484651)

[4 Kravspecifikation 5](#_Toc39484652)

[4.1 Brugerhistorier 5](#_Toc39484653)

[4.1.1 Log ind 5](#_Toc39484654)

[4.1.2 Opret ny note 6](#_Toc39484655)

[4.1.3 Rediger note 6](#_Toc39484656)

[4.2 Iterationer 7](#_Toc39484657)

[4.2.1 Pre-Iteration - skitser 8](#_Toc39484658)

[4.2.2 Iteration 1 - Log ind system 9](#_Toc39484659)

[4.2.3 Iteration 2 - Hovedmenu 10](#_Toc39484660)

[4.2.4 Iteration 3 - Noteskrivning 10](#_Toc39484661)

[5 Programmets opbygning 11](#_Toc39484662)

[5.1 Tre-lags-modellen 11](#_Toc39484663)

[5.1.1 Præsentationslag/brugerflade 12](#_Toc39484664)

[5.1.2 Logiklag/applikationslag 12](#_Toc39484665)

[5.1.3 Datalag 13](#_Toc39484666)

[5.1.4 Implementeringen i programmet NoteSort 13](#_Toc39484667)

[5.2 Beskrivelse af machine learning algoritme 18](#_Toc39484668)

[5.3 Kort om mest relevante anvendte biblioteker 22](#_Toc39484669)

[5.3.1 Flask 22](#_Toc39484670)

[5.3.2 Sklearn 22](#_Toc39484671)

[6 Referencer 23](#_Toc39484672)

[7 Bilag 24](#_Toc39484673)

[7.1 Bilag 1 - Brugerhistorier 24](#_Toc39484674)

[7.2 Kildekode 24](#_Toc39484675)

[7.2.1 Python 24](#_Toc39484676)

[7.2.2 HTML 43](#_Toc39484677)

# 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.

## Opgaveformulering

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

* Sammensætningen af en hjemmeside, der funger 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.

# Kravspecifikation

## Brugerhistorier

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

### 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.
   1. Hvis kodeordet er det rigtige føres brugeren videre til brugerens startside.
   2. Hvis kodeordet ikke er rigtigt, siger programmet, at det er forkert, og brugeren kan indtaste igen.

### 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
   1. Brugeren kan trykke krydset for at gå tilbage til noten og ændre noget, inden den skal gemmes.
   2. Brugeren kan vælge at trykke ”Yes”-knappen for at acceptere algoritmens placering af noten.
   3. 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.

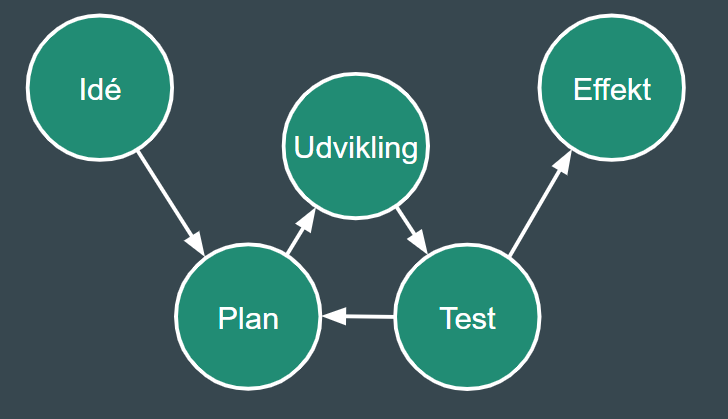
### Rediger note

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 redigeringsside, 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.

## 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:

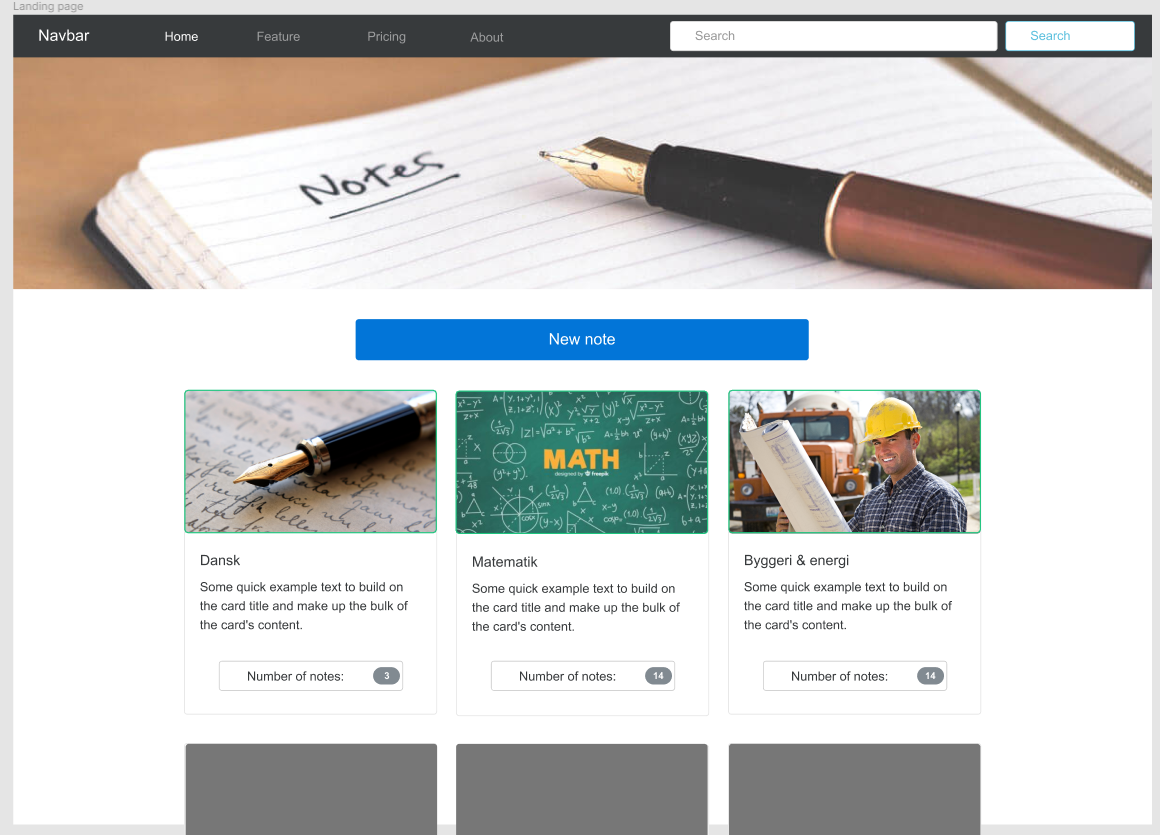


Figur Egen illustration af den iterative proces

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 fej 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.

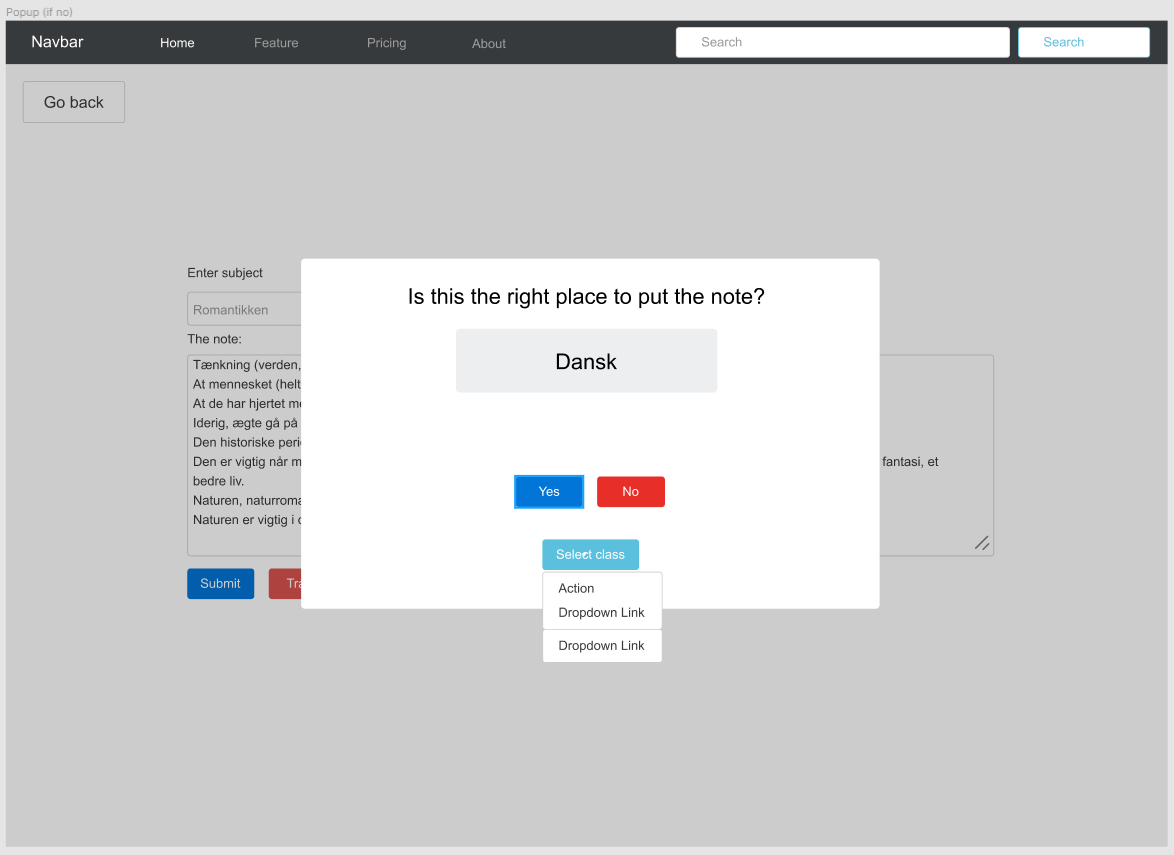
### Pre-Iteration - skitser

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.



Figur Egen illustration af designet af hjemmesidens forside.

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.



Figur Egen illustration af designet af siden for at skrive ny note.

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.

### 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]](#footnote-1) Herefter var det at 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.

### 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.

### Iteration 3 - Noteskrivning

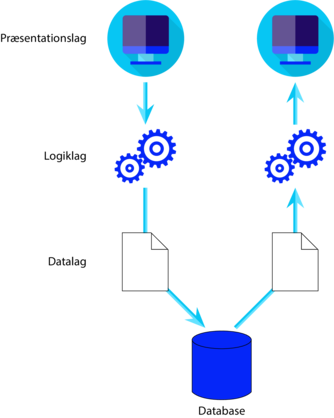
Sidste funktion jeg ville implementere for at kunne kalde hjemmesiden et funktionelt produkt, var en måde at skrive nye noter. Til dette startede jeg med igen at opstille siden vha. HTML og CSS. Hernæst skrev jeg koden i datalaget til at tilføje nye noter til databasetabellen for noter. Inden jeg ville 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 noterne for et specifik 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.

# 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.

## 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.



Figur Illustration af tre-lags-modellen hentet fra: (Trelags-arkitektur, 2020)

De tre lag er som følger:

### 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).

### Logiklag/applikationslag

Applikationslaget er det lag, hvor data bliver behandler. 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.

### 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.

### Implementeringen i programmet NoteSort

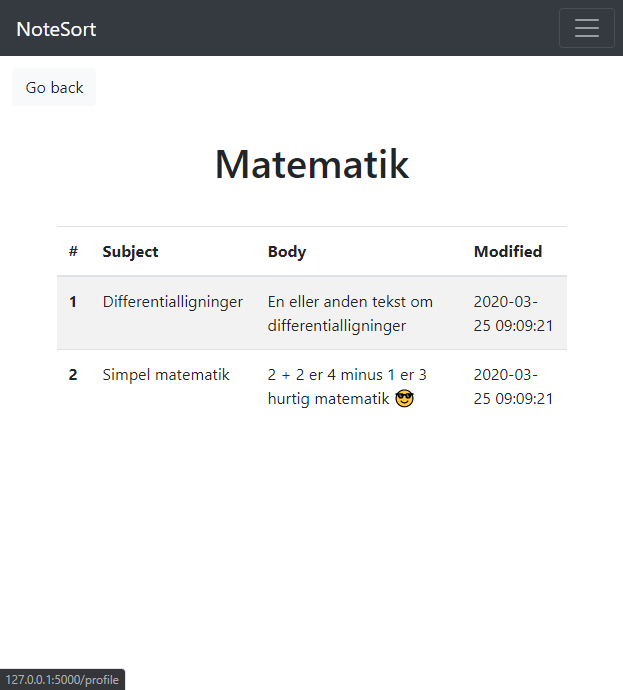
#### Brugerfladen

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:



Figur 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.



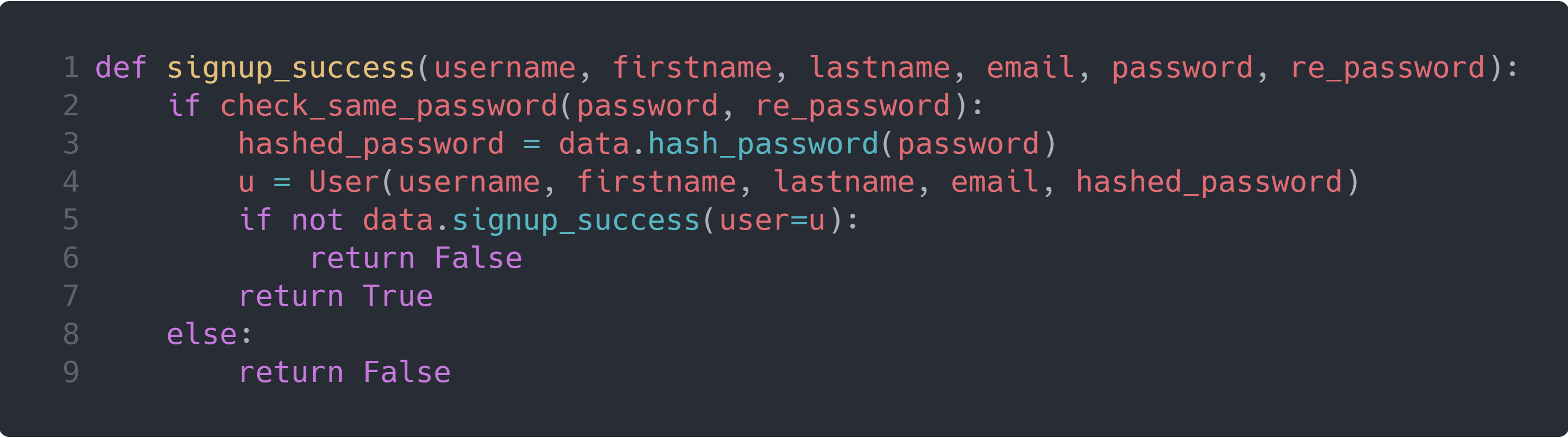
Figur Skærmbillede af fagsiden af hjemmesiden.

#### 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 (), siden det er ret omfangsrigt.

På figuren herover ses to kodestykker 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\_sucess”, 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” returlere false (stk. B linje 6 og 9), hvorefter funktionen ”signup” vil nulstille sessionen og genindlæse registreringssiden (stk. B linje 19-21).



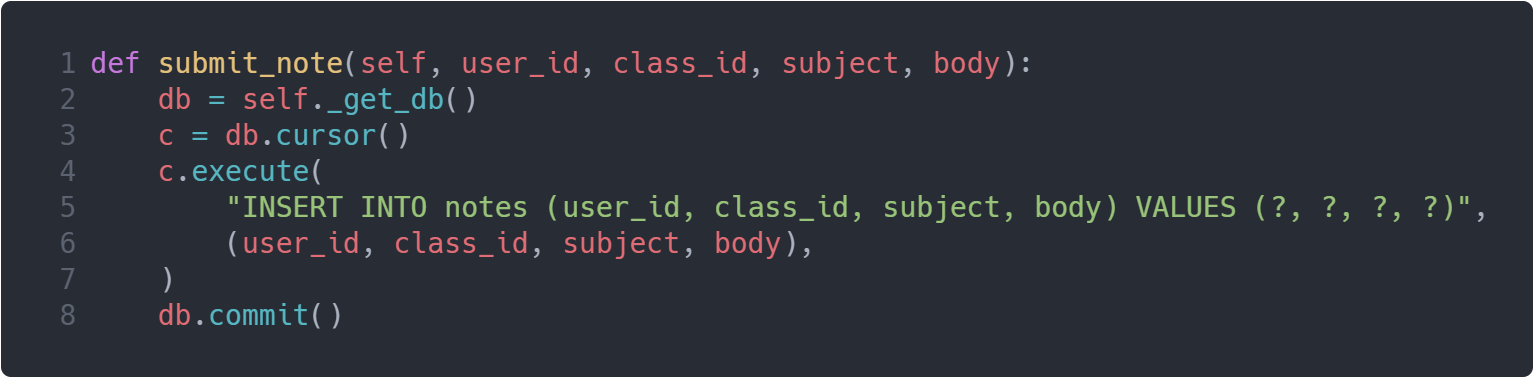
**A**

**B**

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

#### Datalaget

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



Figur 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]](#footnote-2) (se Figur 9 linje 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.

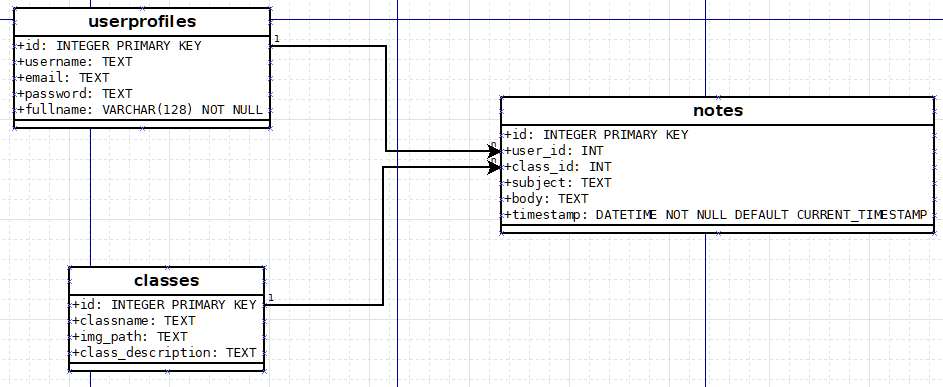


Figur Kodeeksempel på forbindelsesoprettelse til databasen. Kommer fra filen ”note\_data.py”.

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).

##### 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.



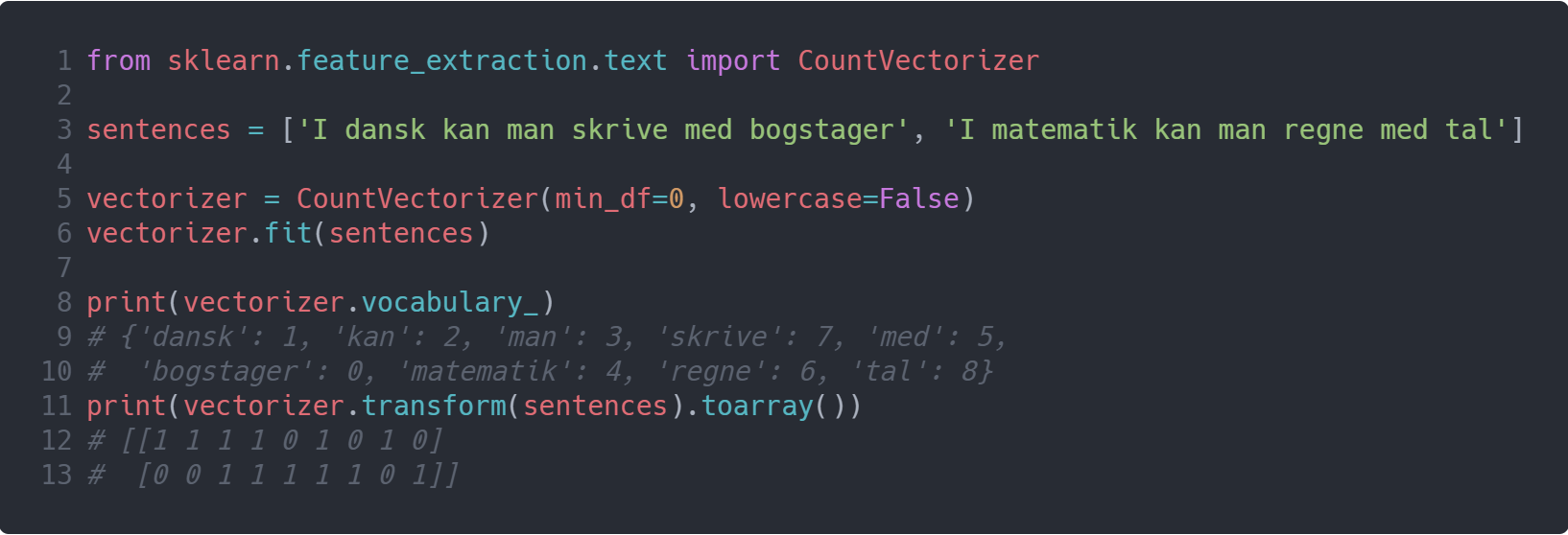
Figur Illustration af databasens opbygning (ER-diagram)

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.

## Beskrivelse af machine learning algoritme[[3]](#footnote-3)

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



Figur Kodeeksempel af ”vektorisering” af sætninger. Kommer fra filen ”test.py”.

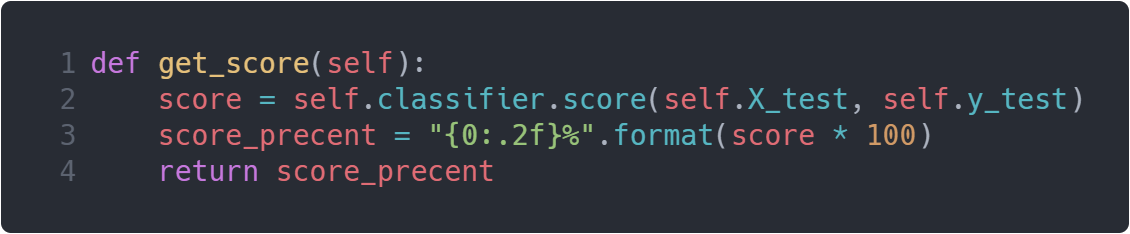
I dette eksempel (Figur 11) er der taget to sætninger, hvor den ene er fra dansk og den anden matematik (linje 3). Klassen ”CountVectorizer” fra biblioteket ”sklearn” bliver brugt til at omdanne sætningerne til en ordbog (linje 5-6). På linje 9-10 ses ordbogen, hvor man kan se placeringen/indekset af hvert ord. På linje 12-13 ses sætningerne konverterede til såkaldte feature vektorer,[[4]](#footnote-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.

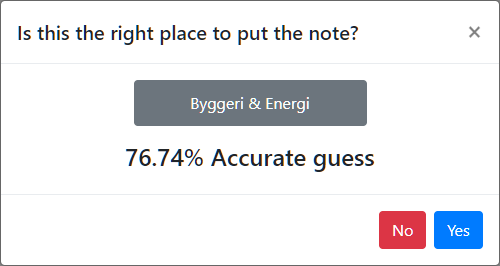
Næste skridt er at definere en model. Hvis man havde mere tid, ville denne simple model være første skridt til at bygge bedre modeller ud fra, men grundet tidsrummet af denne opgave, har der ikke været tid til det. Første skridt er at dele dataene op i trænings- og testdata (dette skridt er samtidig, 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]](#footnote-5) af éns model.

På Figur 12 har jeg defineret en klasse ”TextClassidierModel”, som indeholder alt med modellen. Når klassen initialiseres findes dataene (linje 3), som derefter bliver gemt i et ”dictionary”[[6]](#footnote-6) (linje 4-6). Herefter bliver ”create\_dataframe”-funktionen kaldt (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.



Figur Kodeeksempel på genereringen af modellen for notesortering.





Figur Illustration af hentning af nøjagtigheden af modellen.

## Kort om mest relevante anvendte biblioteker

Siden der er brugt 11 forskellige biblioteker[[7]](#footnote-7) i dette projekt, har jeg valgt kun kort at gå over Flask og Sklearn, da det er de biblioteker, som står

### Flask

Flask er et forholdsvist 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.

### 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.

# 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

# Bilag

## Bilag 1 - Brugerhistorier

* Registrering af bruger
* Log ud
* Slet note

## Kildekode

### Python

#### Main.py

1. **from** note\_data **import** Database, User
2. **from** API.class\_classifier\_model **import** TextClassifierModel
4. **from** flask **import** Flask, request, g, render\_template, session, redirect, url\_for
5. **import** os, random, string, json
7. app = Flask(\_\_name\_\_)
8. key = "very secret string"
9. app.secret\_key = key
11. class\_model = TextClassifierModel()
13. with app.app\_context():
14. data = Database()

17. @app.teardown\_appcontext
18. **def** close\_connection(exception):
19. data.close\_connection()

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["currentuser"], \*\*kwargs
27. )
28. **else**:
29. **return** render\_template(template, loggedin=login\_status, user="", \*\*kwargs)

32. **def** get\_login\_status():
33. **return** "currentuser" **in** session

36. **def** get\_user\_id():
37. **if** get\_login\_status():
38. **return** session["currentuser"]
39. **else**:
40. **return** -1

43. **def** login\_success(username, password):
44. **return** data.login\_success(username, password)

47. **def** check\_same\_password(password, re\_password):
48. **if** password == re\_password:
49. **return** True
50. **else**:
51. **return** False

54. **def** clean\_dict\_from\_req\_args(cluttered\_dict):
55. cluttered\_dict = cluttered\_dict[""].replace("'", '"')
56. cluttered\_dict = cluttered\_dict[1:-1]
57. info = json.loads(cluttered\_dict)
58. **return** info

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

72. @app.route("/")
73. @app.route("/login")
74. **def** index():
75. **return** my\_render("index.html", success=False, title="login")

78. @app.route("/login\_profile", methods=["POST"])
79. **def** login():
80. password = request.form["password"]
81. username = request.form["username"]
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"/")

92. @app.route("/logout")
93. **def** logout():
94. session.pop("currentuser", None)
95. **return** redirect(f"/")

98. @app.route("/signup")
99. **def** signup\_site():
100. **return** my\_render("sign\_up.html", title="signup")

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}"]
113. **return** my\_render(
114. "user\_main.html", title="Student", success=True, classes\_info=classes\_info,
115. )

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"]
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)

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)

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']}'")

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. )

164. @app.route("/take\_notes")
165. **def** take\_notes():
166. **return** my\_render("note\_writer.html")

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. class\_names = {
179. "class\_name": class\_name,
180. "other\_classes1": other\_classes[0],
181. "other\_classes2": other\_classes[1],
182. "prediction\_score": prediction\_score,
183. }
184. **return** class\_names

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, subject, body)
198. **return** redirect("/profile")

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}'")

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. )

220. **if** \_\_name\_\_ == "\_\_main\_\_":
221. with app.app\_context():
222. data = Database()
224. app.run(debug=True)

#### Note\_data.py

1. **from** flask **import** g, Flask
2. **import** sqlite3, hashlib, binascii, os, time
4. ABS\_FILEPATH = os.path.dirname(os.path.abspath(\_\_file\_\_))

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
15. **def** set\_id(self, ID):
16. self.id = ID
18. **def** \_\_str\_\_(self):
19. **return** f"""
20. username: {self.username}
21. email: {self.email}
22. password: {self.password}
23. rank: {self.rank}"""

26. **class** Database:
27. **def** \_\_init\_\_(self):
28. self.DATABASE = fr"{ABS\_FILEPATH}\main.db"
30. # self.\_create\_tables()
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
38. **def** close\_connection(self):
39. db = getattr(g, "\_database", None)
40. **if** db **is** **not** None:
41. db.close()
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
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")
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. pwdhash = hashlib.pbkdf2\_hmac(
67. "sha512", provided\_password.encode("utf-8"), salt.encode("ascii"), 100000
68. )
69. pwdhash = binascii.hexlify(pwdhash).decode("ascii")
70. **return** pwdhash == stored\_password
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)
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, description = 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
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 = ?", (class\_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
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
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 classname = ?", (class\_name,)
128. )
129. r = c.fetchall()
130. **for** class\_name **in** r:
131. other\_classes.append(class\_name)
132. **return** other\_classes
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
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
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 class\_id == ?",
162. (user\_id, class\_id),
163. )
164. r = c.fetchall()
165. **for** note **in** r:
166. note\_id, user\_id, class\_id, subject, body, timestamp = note
167. note\_dict = {
168. "note\_id": note\_id,
169. "user\_id": user\_id,
170. "class\_id": class\_id,
171. "subject": subject,
172. "body": body,
173. "timestamp": timestamp,
174. }
175. notes.append(note\_dict)
176. **return** notes
178. **def** get\_note\_info(self, note\_id, user\_id):
179. db = self.\_get\_db()
180. c = db.cursor()
181. c.execute(
182. "SELECT \* FROM notes WHERE id == ? AND user\_id == ?", (note\_id, user\_id),
183. )
184. r = c.fetchone()
185. note\_id, user\_id, class\_id, subject, body, timestamp = r
186. note\_dict = {
187. "note\_id": note\_id,
188. "user\_id": user\_id,
189. "class\_id": class\_id,
190. "subject": subject,
191. "body": body,
192. "timestamp": timestamp,
193. }
194. **return** note\_dict
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 = ?", (note\_id, user\_id),
201. )
202. db.commit()
204. **def** submit\_note(self, user\_id, class\_id, subject, body):
205. db = self.\_get\_db()
206. c = db.cursor()
207. c.execute(
208. "INSERT INTO notes (user\_id, class\_id, subject, body) VALUES (?, ?, ?, ?)",
209. (user\_id, class\_id, subject, body),
210. )
211. db.commit()
213. **def** edit\_note(self, note\_id, user\_id, subject, body):
214. db = self.\_get\_db()
215. c = db.cursor()
216. c.execute(
217. "UPDATE notes SET subject=?, body=? WHERE id = ? AND user\_id=?",
218. (subject, body, note\_id, user\_id),
219. )
220. db.commit()
222. **def** check\_existing\_username(self, username):
223. db = self.\_get\_db()
224. c = db.cursor()
225. c.execute("SELECT username from userprofiles WHERE username=?", (username,))
226. r = c.fetchone()
227. **if** r == None:
228. **return** False
229. **return** True
231. **def** signup\_success(self, user: User):
232. db = self.\_get\_db()
233. c = db.cursor()
234. fullname = f"{user.firstname};{user.lastname}"
235. **if** **not** self.check\_existing\_username(user.username):
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
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 None
250. **if** r **is** **not** None:
251. **return** r[0]
252. **else**:
253. **return** None
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 None
261. **if** r **is** **not** None:
262. fullname = r[0].replace(";", " ")
263. **return** fullname
264. **else**:
265. **return** None
267. **def** \_drop\_tables(self):
268. db = self.\_get\_db()
269. c = db.cursor()
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()
279. **def** \_create\_tables(self):
280. db = self.\_get\_db()
281. c = db.cursor()
283. **try**:
284. c.execute(
285. """CREATE TABLE IF NOT EXISTS userprofiles (
286. id INTEGER PRIMARY KEY,
287. username TEXT,
288. email TEXT,
289. password TEXT,
290. fullname VARCHAR(128) NOT NULL);"""
291. )
292. **except** Exception as e:
293. **print**(e)
295. **try**:
296. c.execute(
297. """CREATE TABLE IF NOT EXISTS classes (
298. id INTEGER PRIMARY KEY,
299. classname TEXT,
300. img\_path TEXT,
301. class\_description TEXT);"""
302. )
303. **except** Exception as e:
304. **print**(e)
306. **try**:
307. c.execute(
308. """CREATE TABLE IF NOT EXISTS notes (
309. id INTEGER PRIMARY KEY,
310. user\_id INT,
311. class\_id INT,
312. subject TEXT,
313. body TEXT,
314. timestamp DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP);"""
315. )
316. **except** Exception as e:
317. **print**(e)
319. db.commit()
321. # Create testing profiles
322. pass1 = self.hash\_password("1234")
324. **try**:
325. c.execute(
326. """
327. INSERT INTO userprofiles (username, email, password, fullname)
328. SELECT "user", "andreasgdp@gmail.com", ?, "Andreas;Petersen"
329. WHERE NOT EXISTS (
330. SELECT 1 FROM userprofiles WHERE username = "user" AND
331. email = "andreasgdp@gmail.com" AND
332. fullname = "Andreas;Petersen");
333. """,
334. (pass1,),
335. )
336. c.execute(
337. """
338. INSERT INTO userprofiles (username, email, password, fullname)
339. SELECT "user2", "mand@gmail.com", ?, "Mande;Manden"
340. WHERE NOT EXISTS (
341. SELECT 1 FROM userprofiles WHERE username = "teacher" AND
342. email = "mand@gmail.com" AND
343. fullname = "Mande;Manden");
344. """,
345. (pass1,),
346. )
347. **except** Exception as e:
348. **print**(e)
350. # Create testing classes
351. **try**:
352. c.execute(
353. """
354. INSERT INTO classes (classname, img\_path, class\_description) VALUES (
355. "Byggeri & Energi",
356. "./static/Images/byggeri & energi.jpg",
357. "Det er Byg og Hyg");
358. """
359. )
360. c.execute(
361. """
362. INSERT INTO classes (classname, img\_path, class\_description) VALUES (
363. "Dansk",
364. "./static/Images/dansk.png",
365. "Det er dansk");
366. """
367. )
368. c.execute(
369. """
370. INSERT INTO classes (classname, img\_path, class\_description) VALUES ("Matematik",
371. "./static/Images/matematik.jpg",
372. "Det er mat");
373. """
374. )
375. **except** Exception as e:
376. **print**(e)
378. # Create testing notes
379. **try**:
380. # ? Dansk
381. c.execute(
382. """
383. INSERT INTO notes (user\_id, class\_id, subject, body) VALUES (
384. 1,
385. 2,
386. "Romantikken",
387. "Romantikkens afgørende dyder er Intuition og Fantasi"
388. );
389. """
390. )
391. c.execute(
392. """
393. INSERT INTO notes (user\_id, class\_id, subject, body) VALUES (
394. 1,
395. 2,
396. "Romantikken",
397. "Den såkaldte organicisme eller organismetanke går ud på, at..."
398. );
399. """
400. )
401. c.execute(
402. """
403. INSERT INTO notes (user\_id, class\_id, subject, body) VALUES (
404. 1,
405. 2,
406. "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, subject, body) VALUES (
415. 1,
416. 3,
417. "Differentialligninger",
418. "En eller anden tekst om differentialligninger"
419. );
420. """
421. )
422. c.execute(
423. """
424. INSERT INTO notes (user\_id, class\_id, subject, body) VALUES (
425. 1,
426. 3,
427. "Simpel matematik",
428. "2 + 2 er 4 minus 1 er 3 hurtig matematik 😎"
429. );
430. """
431. )
432. # ? Byggeri & Energi
433. c.execute(
434. """
435. INSERT INTO notes (user\_id, class\_id, subject, body) VALUES (
436. 1,
437. 1,
438. "Dimensionering",
439. "Det er matematik men i byg"
440. );
441. """
442. )
444. **except** Exception as e:
445. **print**(e)
447. db.commit()
448. **print**("Database tables created")

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()

#### 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
7. ABS\_FILEPATH = os.path.dirname(os.path.abspath(\_\_file\_\_))

10. # Labels:
11. # 1: Byg
12. # 2: Dan
13. # 3: Mat

16. **class** TextClassifierModel:
17. **def** \_\_init\_\_(self):
18. self.model\_file = fr"{ABS\_FILEPATH}\data\classes\combined\_test.txt"
19. self.filepath\_dict = {
20. "combined": self.model\_file,
21. }
22. self.create\_dataframe()
23. self.create\_model()
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)
32. self.df = pd.concat(df\_list)
34. **def** create\_model(self):
35. self.df\_combined = self.df[self.df["text\_class"] == "combined"]
37. sentences = self.df\_combined["sentence"].values
38. y = self.df\_combined["label"].values
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)
47. self.vectorizer = CountVectorizer()
48. self.vectorizer.fit(self.sentences\_train)
50. self.X\_train = self.vectorizer.transform(self.sentences\_train)
51. self.X\_test = self.vectorizer.transform(self.sentences\_test)
53. self.classifier = LogisticRegression()
54. self.classifier.fit(self.X\_train, self.y\_train)
56. **def** add\_data(self, data):
57. with open(self.model\_file, "ab") as f:
58. f.write(data)
60. **def** refresh\_model(self):
61. self.create\_dataframe()
62. self.create\_model()
64. **def** print\_score(self):
65. score = self.classifier.score(self.X\_test, self.y\_test)
66. **print**("Accuracy:", score)
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
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.splitlines())
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
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(data)
87. **def** predict\_class(self):
88. prediction = self.classifier.predict(self.prepared\_data)
89. **return** prediction[0]

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=}")

#### 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
3. sentences = ['I dansk kan man skrive med bogstager', 'I matematik kan man regne med tal']
5. vectorizer = CountVectorizer(min\_df=0, lowercase=False)
6. vectorizer.fit(sentences)
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]]

### HTML

#### 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. **/>**
11. <!-- Bootstrap CSS -->
12. **<link**
13. rel="stylesheet"
14. href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
15. integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
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 title !=
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>**
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+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
91. crossorigin="anonymous"
92. **></script>** --**>**
93. **<script**
94. src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
95. integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49"
96. crossorigin="anonymous"
97. **></script>**
98. **<script**
99. src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js"
100. integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy"
101. crossorigin="anonymous"
102. **></script>**
103. **</body>**
104. **</html>**

#### Index.html

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

#### 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. <!-- https://mdbootstrap.com/docs/jquery/forms/basic/ -->
6. **<form** class="text-center border border-light p-5" action="signup\_profile" method="POST"**>**
7. **<p** class="h4 mb-4"**>**Sign up**</p>**
9. <!-- Username -->
10. **<input**
11. type="username"
12. id="defaultRegisterFormEmail"
13. class="form-control mb-4"
14. placeholder="Username"
15. name="username"
16. **/>**
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>**
41. <!-- E-mail -->
42. **<input**
43. type="email"
44. id="defaultRegisterFormEmail"
45. class="form-control mb-4"
46. placeholder="E-mail"
47. name="email"
48. **/>**
50. <!-- Password -->
51. **<input**
52. type="password"
53. id="defaultRegisterFormPassword"
54. class="form-control"
55. placeholder="Password"
56. aria-describedby="defaultRegisterFormPasswordHelpBlock"
57. name="password"
58. **/>**
59. **<small**
60. id="defaultRegisterFormPasswordHelpBlock"
61. class="form-text text-muted mb-4"
62. **>**
63. At least 8 characters and 1 digit
64. **</small>**
66. <!-- Repeat password -->
67. **<input**
68. type="password"
69. id="defaultRegisterFormPassword"
70. class="form-control"
71. placeholder="Repeat password"
72. aria-describedby="defaultRegisterFormPasswordHelpBlock"
73. name="re\_password"
74. **/>**
75. **</small>**
77. <!-- Sign up button -->
78. **<button** type="submit" class="btn m-4 btn-primary"**>**Sign in**</button>**
80. <!-- Register -->
81. **<p>**
82. Already a member?
83. **<a** href="/"**>**Log in**</a>**
84. **</p>**
85. **</form>**
87. <!-- Default form register -->
88. **</div>**
89. **</div>**
91. {% endblock content %}

#### User\_main.html

1. {% extends "header.html" %} {% block content %}
3. **<div** class="bg img-fluid"**></div>**
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. **<img**
21. class="card-img-top"
22. src="{{ class\_info['class\_img'] }}"
23. alt="Card image cap"
24. **/>**
25. **<div** class="card-body"**>**
26. **<h5** class="card-title"**>**{{ class\_info['class\_name'] }}**</h5>**
27. **<div** style="height: 60px"**>**
28. **<p**
29. class="card-text"
30. style="white-space: nowrap; overflow: hidden; text-overflow: ellipsis;"
31. **>**
32. {{ class\_info['description'] }}
33. **</p>**
34. **</div>**
35. **<div** class="card"**>**
36. **<div** class="container" style="text-align: center;"**>**
37. **<h5>**
38. 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>**
51. {% endblock content %}

#### 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;
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 = data['other\_classes1'];
50. let wrong\_class\_name2 = data['other\_classes2'];
51. let prediction\_score = data['prediction\_score'];
52. $('#correct-class').empty();
53. $('#correct-class').text(class\_name);
54. $('#accuracy').empty();
55. $('#accuracy').text(
56. `${prediction\_score} Accurate guess`
57. );
59. $('#wrong-class-name1').empty();
60. $('#wrong-class-name1').text(wrong\_class\_name1);
61. $('#wrong-class-name2').empty();
62. $('#wrong-class-name2').text(wrong\_class\_name2);
64. let wrong\_class\_name1\_inp = document.getElementById(
65. 'wrong-class-name1-inp'
66. );
67. wrong\_class\_name1\_inp.value = wrong\_class\_name1;
69. let wrong\_class\_name2\_inp = document.getElementById(
70. 'wrong-class-name2-inp'
71. );
72. wrong\_class\_name2\_inp.value = wrong\_class\_name2;
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 = document.getElementById(
138. 'dropdownFooter'
139. );
140. if (x.style.display === 'none') {
141. x.style.display = 'block';
142. } else {
143. x.style.display = 'none';
144. }
145. var knap = document.getElementById(
146. 'spoiler\_knap'
147. );
148. if (x.style.display === '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 = document.getElementById(
162. 'wrong-class-name1-inp'
163. ).value;
164. let class\_name\_hidden = document.getElementById(
165. 'correct-class-hidden'
166. );
167. class\_name\_hidden.value = wcn;
168. $('#correct-class').empty();
169. $('#correct-class').text(wcn);
170. }
171. </**script**>
172. <**script**>
173. function wrong\_class\_name2() {
174. let wcn = document.getElementById(
175. 'wrong-class-name2-inp'
176. ).value;
177. console.log(wcn);
179. let class\_name\_hidden = document.getElementById(
180. 'correct-class-hidden'
181. );
182. class\_name\_hidden.value = wcn;
183. $('#correct-class').empty();
184. $('#correct-class').text(wcn);
185. }
186. </**script**>
187. <**button** type="submit" class="btn btn-primary">
188. Yes
189. </**button**>
190. </**div**>
191. <**div**
192. 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\_inp\_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\_inp\_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 %}

#### 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. 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 %}

#### 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-heading"**>**{{ class\_info['class\_name'] }}**</h1>**
7. **<div** class="row h-100"**>**
8. **<div** class="col-sm-12 my-auto"**>**
9. **<div** class="card card-block w-100"**>**
10. **<h1>**{{ note\_info['subject'] }}**</h1>**
11. **<div** class="scrollbar scrollbar-primary pre-scrollable"**>**
12. {{ note\_info['body'] }}
13. **</div>**
14. **</div>**
15. **<br** **/>**
16. **<a**
17. name=""
18. id=""
19. class="btn btn-primary"
20. href="/edit\_note?='{{ note\_info }}'"
21. role="button"
22. **>**Edit**</a**
23. **>**
24. **<a**
25. name=""
26. id=""
27. class="btn btn-danger"
28. href="/remove\_note?='{{ note\_info }}'"
29. role="button"
30. **>**Trash**</a**
31. **>**
32. **</div>**
33. **</div>**
34. **</div>**
36. {% endblock content %}

#### 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 %}

1. (The Bootstrap team, 2020) [↑](#footnote-ref-1)
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. [↑](#footnote-ref-2)
3. Som inspiration til min machine learning algoritme, har jeg brugt forklaringen på denne hjemmeside: (Janakiev, 2019) [↑](#footnote-ref-3)
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]. [↑](#footnote-ref-4)
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. [↑](#footnote-ref-5)
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. [↑](#footnote-ref-6)
7. Anvendte biblioteker: Flask, Os, Random, String, Json, Sklearn, Pandas, Sqlite3, Hashlib, Binascii Time. [↑](#footnote-ref-7)