

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

11990 af 19200 tegn

## Dato

30. april 2020

# Indholdsfortegnelse

[1 Titelblad 2](#_Toc37929471)

[1.1 Vejledning 2](#_Toc37929472)

[1.2 Fag 2](#_Toc37929473)

[1.3 Klasse og skole 2](#_Toc37929474)

[1.4 Antal tegn 2](#_Toc37929475)

[1.5 Dato 2](#_Toc37929476)

[2 Indholdsfortegnelse 3](#_Toc37929477)

[3 Indledning 5](#_Toc37929478)

[3.1 Opgaveformulering 5](#_Toc37929479)

[4 Kravspecifikation 5](#_Toc37929480)

[4.1 Brugerhistorier 5](#_Toc37929481)

[4.1.1 Log ind 5](#_Toc37929482)

[4.1.2 Opret ny note 6](#_Toc37929483)

[4.1.3 Rediger note 6](#_Toc37929484)

[4.2 Iterationer 7](#_Toc37929485)

[4.2.1 Pre-Iteration - skitser 7](#_Toc37929486)

[4.2.2 Iteration 1 - 9](#_Toc37929487)

[5 Programmets opbygning 9](#_Toc37929488)

[5.1 Tre-lags-modellen 9](#_Toc37929489)

[5.2 Beskrivelse af datalaget 9](#_Toc37929490)

[5.2.1 ER-diagram 9](#_Toc37929491)

[5.3 Beskrivelse af ”machine learning” algoritme 10](#_Toc37929492)

[5.4 Beskrivelse af centrale funktioner 10](#_Toc37929493)

[5.5 Kort om anvendte biblioteker 10](#_Toc37929494)

[6 Forslag til forbedringer 10](#_Toc37929495)

[7 Arbejdsprocessen 10](#_Toc37929496)

[8 Referencer 11](#_Toc37929497)

[9 Bilag 12](#_Toc37929498)

[9.1 Bilag 1 - Brugerhistorier 12](#_Toc37929499)

# Indledning

Det kan være 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 i det øverste inputfelt.
2. Brugeren skriver sit password i det nederste inputfelt.
3. Brugeren trykker på log-ind-klappen.
4. Programmet tjekker om begge felter er fyldt ud.
5. Programmet går ind i databasen og leder efter en bruger med det brugernavn.
6. 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, som noten burde tilhøre.
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 (for at gøre den bedre over tid med tilføjelser af flere noter). Herefter bliver noten lagt i databasen.
8. Programmet navigerer tilbage til brugerens startside.

### Rediger note

1. Når brugeren er logget ind, klikker brugeren på den klasse, hvor noten, som skal redigeres, ligger i.
2. Programmet navigerer til den klasses side.
3. Brugeren klikker på den note i listen, som skal redigeres.
4. Programmet navigerer til en side, hvor noten alene bliver vist.
5. Brugeren klikker på ”Edit”-knappen.
6. Programmet navigerer til en redigeringsside, hvor værdierne af henholdsvis emnet og tekst allerede er i felterne for dem (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 klasse, 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:

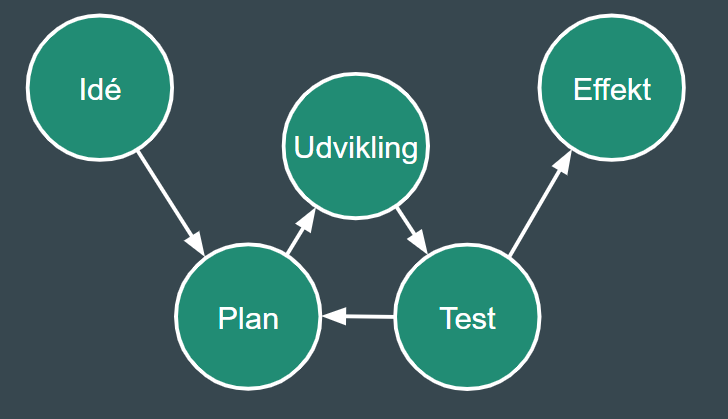
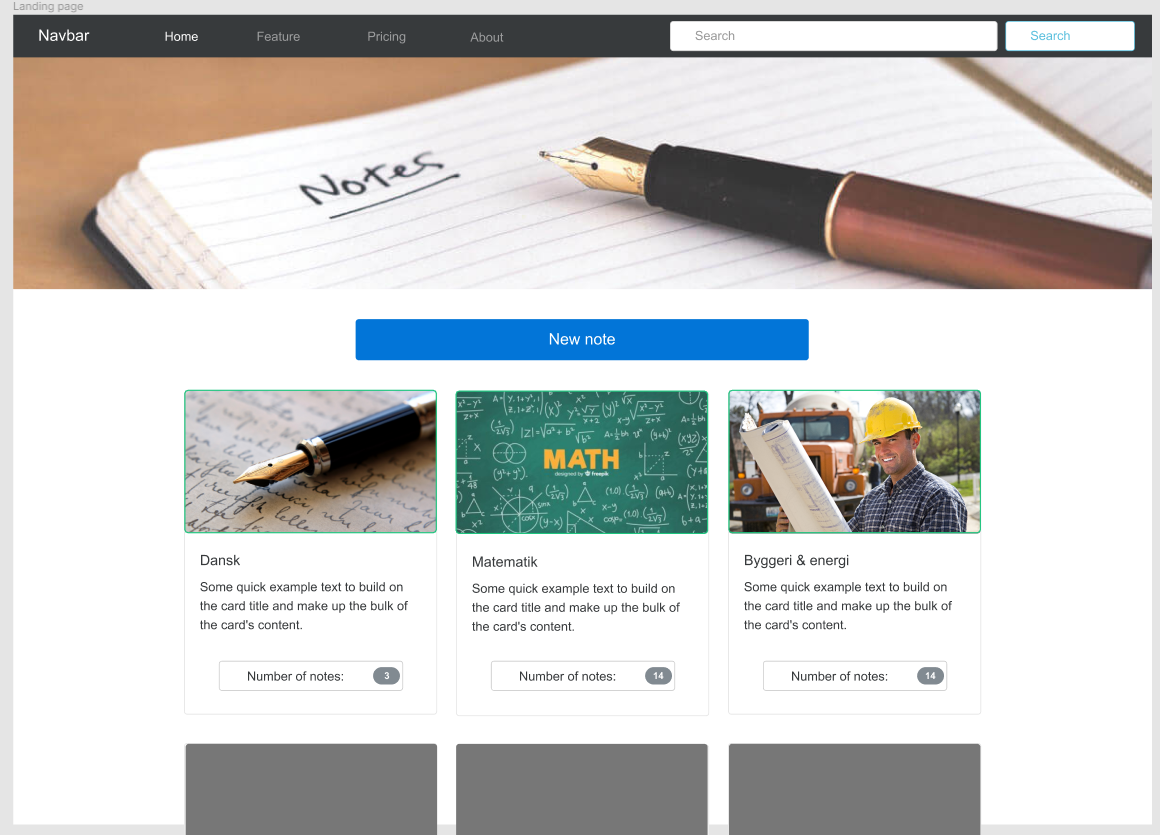


Figure 1 Egen illustration af den iterative proces

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.

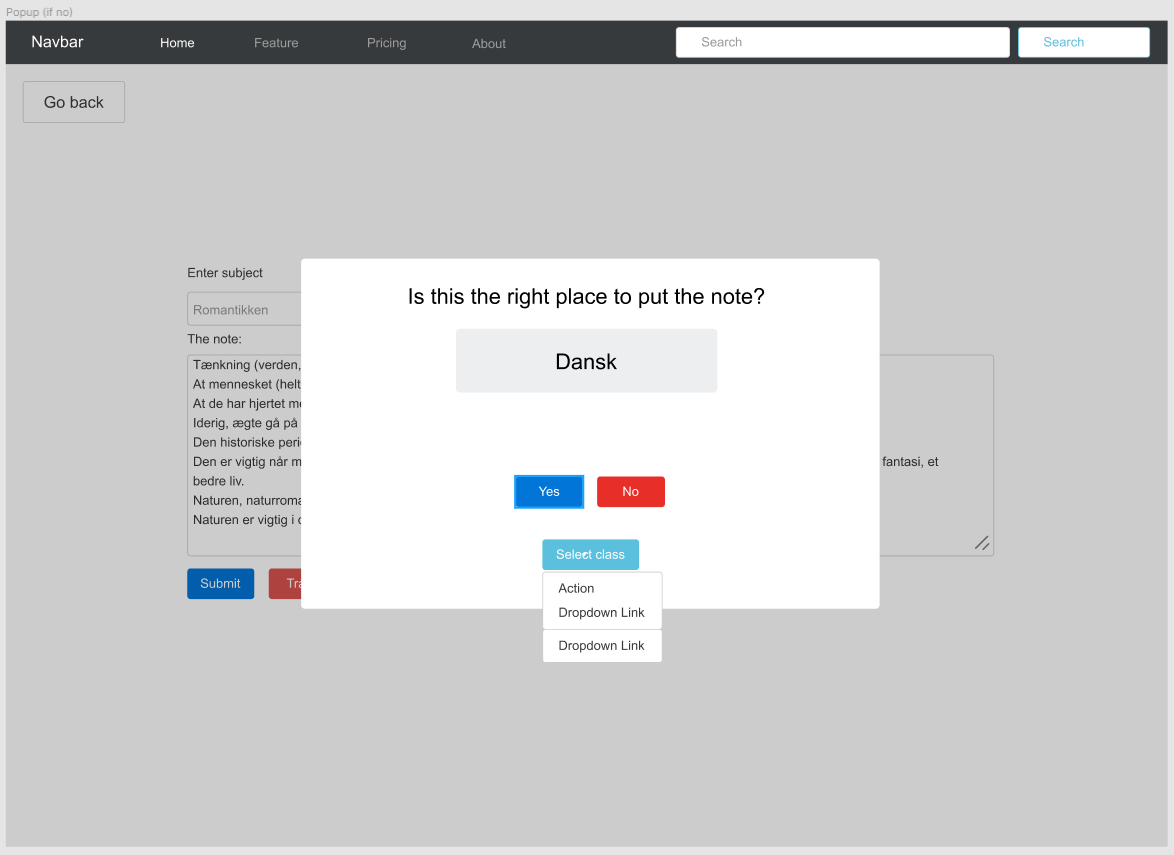
### Pre-Iteration - skitser

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 1 Egen illustration af designet af hjemmesidens forside.

På Figur 1 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 klasserne, som indeholder alle noterne for den pågældende klasse.



Figur 2 Egen illustration af designet af siden for at skrive ny note.

På Figur 2 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 5.2.1) 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, der skulle være, når siden var forbundet med data fra databasen. Da det var sat op, skulle jeg ud fra ER-diagrammet (forklaring følger i afsnit 5.2.1) oprette databasetabeller for de klasser, som jeg havde data for (Byggeri og Energi, Dansk og Matematik) og en tabel for de noter, som ville skulle ligge i hver sin klasse. 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 hver klasse (se Figur 1). Efter at have oprettet databasetabellerne, tilføjede jeg nogle standardnoter til hver af de tre klasser, 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 klasserne og at antallet af noter i klasserne var rigtige.

### Iteration 3 - Noteskrivning

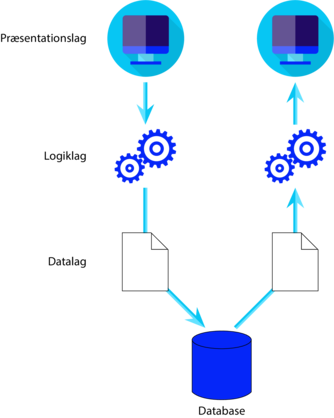
Sidste funktion jeg ville implementere for at kunne kalde hjemmesiden et funktionelt produkt, var en måde at skrive nye noter. Til dette startede jeg med igen at opstille siden vha. HTML og CSS. Hernæst skrev jeg koden i datalaget til at tilføje nye noter til databasetabellen for noter. Inden jeg ville 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 5.3), 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 klassesiden (siden med alle noterne for en specifik klasse) 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 3 ses en illustration af tre-lags-modellen. Modellen er en måde 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

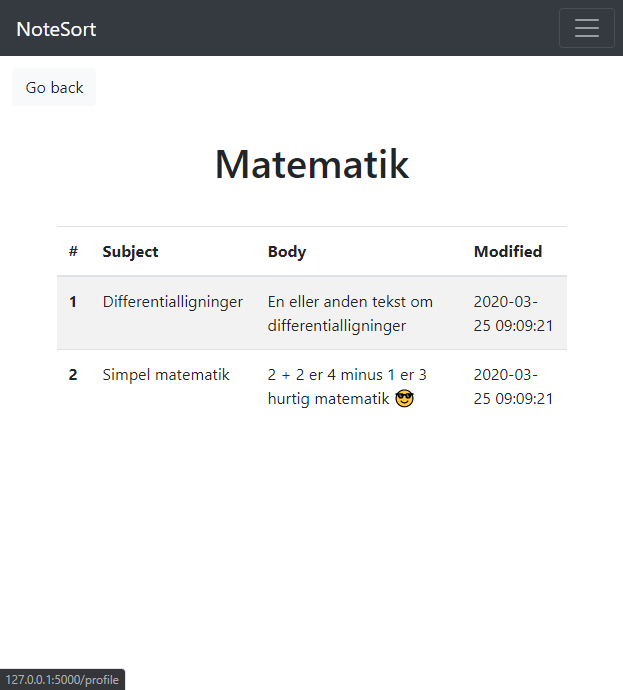
#### Brugerfladen

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 4 Kodeeksempel på implementering af brugerfladen.

Koden i Figur 4 er HTML-koden for klassesiden, som viser alle noter, der ligger i den pågældende klasse. Jinja bruges til at lave et for-loop, som løber igennem alle noterne tilhørende den pågældende klasse, hvor der for hver klasse i linje 20-23 på figuren placeres dataene for hver note i klassen ind i en tabel. Et visuelt eksempel på koden er matematikklassen på Figur 5, hvor dataene for henholdsvis ”subject”, ”body” og ”modified” er placeret i tabellen.



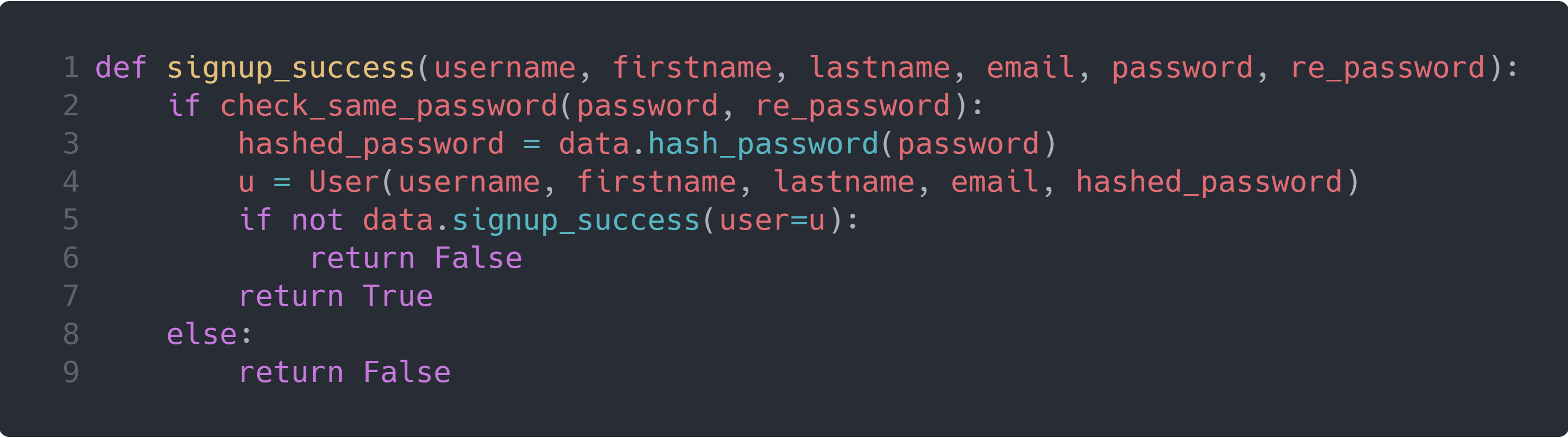
Figur 5 Skærmbillede af klassesiden af hjemmesiden.

#### 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**

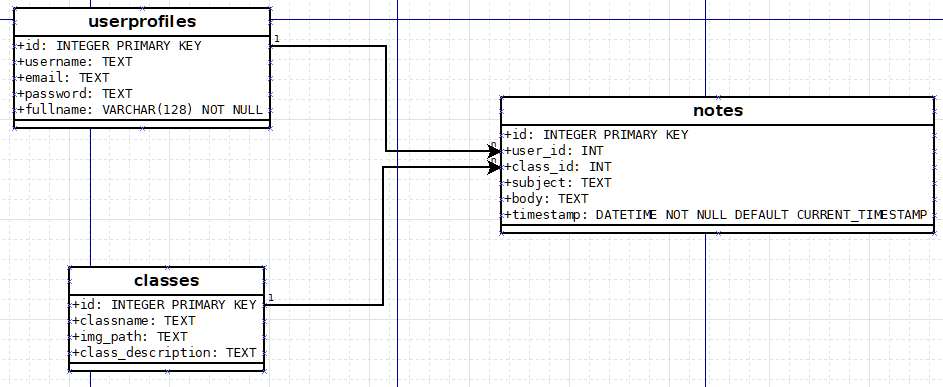
Figur Kodeeksempel på registrering af bruger.

#### Datalaget

## Beskrivelse af databasen

### ER-diagram

For at få et overblik over databasens sammensætning, har jeg lavet et ER-diagram (se Figur 3), som beskriver de tabeller og værdier deri, som jeg bruger i mit program.



Figur 7 Illustration af databasens opbygning (ER-diagram)

## Beskrivelse af ”machine learning” algoritme

## Kort om anvendte biblioteker

# Forslag til forbedringer

# Arbejdsprocessen

# Referencer

**There are no sources in the current document.**

# Bilag

## Bilag 1 - Brugerhistorier

* Registrering af bruger
* Log ud
* Slet note

1. (The Bootstrap team, 2020) [↑](#footnote-ref-1)