Elektronisk Butikk med Django

Før oppstart av Django prosjektet:

Før vi starter med å lage prosjektet er det naturlig å legge opp en plan på hvordan nettsiden skal struktureres. **Design og en mal** for nettsidens utsende. Jeg velger å sette opp nettsiden slik.

Velger å sette opp 4 sider for å håndtere følgende:

***hoved.html*** – for å behandle design. Denne filen vil alle andre html-filer arve fra. Det betyr at i stedet for å gjenta html kode, kan vi planlegge hvilke kode som skal gjentas og arve dette de andre html-filene. Denne vises ikke på nettsiden. Oppgaven til denne filen er å arve bort kode så det ikke trenger å gjentas.

***butikk.html*** – butikkens hovedside. Her vil alle produkter som er tilgjengelig vises til kunder. Design og struktur er viktig.

***handlevogn.html*** – kobles opp med brukeren og viser alle produkter kunden har lagt i sin handlevogn. Her får brukeren valget mellom å øke eller minske mengden av produktet samt slette den fra handlevognen.

***utsjekk.html*** – siste siden i prosessen av å kjøpe et produkt. Her blir betalingsmetoder integrert. Betalingsmetoder for visa og pay-pal er et eksempel. En oppsummering av produktene i handlevognen er også vist.

Vi skal også i denne gjennomgangen ha muligheten til å logge inn som en kunde eller registrere seg som en ny kunde. Det skal også være mulig for å handle uten å lage bruker, altså handle som en gjest. Dette går vi gjennom senere.

Oppstart av Django prosjekt:

For å lage et Django-prosjekt må vi installere Django via powerShell eller lignende. Først må vi installere pip for så å installere Django ved hjelp av disse kommandoene.

***1.Steg: Installerer pip***

****

Ved installering av Python 3.4 og høyere er følger pip og dette steget kan dermed utelates. Nå må vi installere Django ved hjelp av pip.

***2. Steg: Installerer Django***



For å sjekke hvilken versjon av Django som ble installert kan vi bruke følgende kommando.

***Sjekker Django versjon***



I dette eksemplet skal vi jobbe med Django 3.0.2

Nå som alt er installert gjenstår det å starte prosjektet. Før vi starter prosjektet må vi ***cd*** til stedet vi skal lagre prosjektet.

***Plassering av prosjektet***



For å starte prosjektet må vi kjøre følgende kommando etterfølgt av navnet vi gir vårt prosjekt.

***3. Steg: Starter opp prosjektet***



Nå må vi ***cd*** inn i prosjektet for å starte vår app.

***4. Steg: cd til prosjektet***



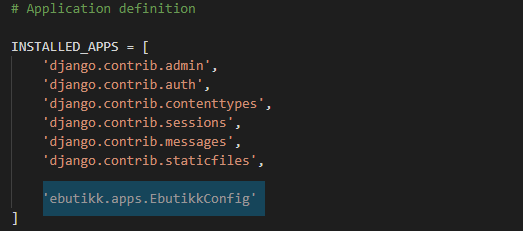
Starter appen ved følgene kommando.

***5. Steg: Starter appen***



Nå gjenstår det å knytte appen vi har laget med vårt Django-prosjekt. Ved hjelp av en tekst editor, må vi åpne prosjektet. Vi går i filen **ehandel > settings.py** for å knytte disse to sammen. Merk hvordan app navnet er skrevet med store bokstaver på slutten.

***6.Steg: Knytter appen med prosjektet***

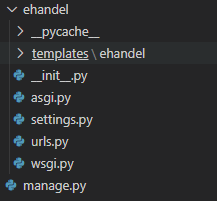


## Nå er appen knyttet sammen med prosjektet. Vi har nå vellykket laget et Django-prosjekt.

Html, Design og Struktur

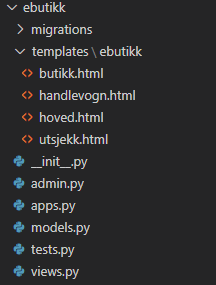
Nå som vi er ferdig med å lage prosjektet kan vi starte med å sette opp appen med sidene vi har planlagt. Django har gitt oss mulighet til å lagre html sider i form av templates. Disse blir lagres i en mappe i selve appen. Det er vanligst å lage en mappe med navn ‘templates’, inne i denne mappen må vi lage en til mappe med navnet til appen vi ønsker å knytte den sammen med. Vi gjør dette slik.

1. ***Steg: Mappe lagring for html filer***



Nå som vi har laget de nødvendige mappene, kan vi lage alle de 4 planlagte sidene.

1. ***Steg: Html filer***

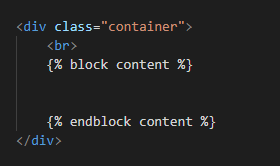


Oppsettet og strukturen til sidene vil jeg utelate i denne gjennomgangen. Poenget er å lage sider som er fornuftige i denne sammen hengen. Jeg vil nå gå gjennom hvordan arv fungerer får så å koble CSS og JavaScript til disse sidene.

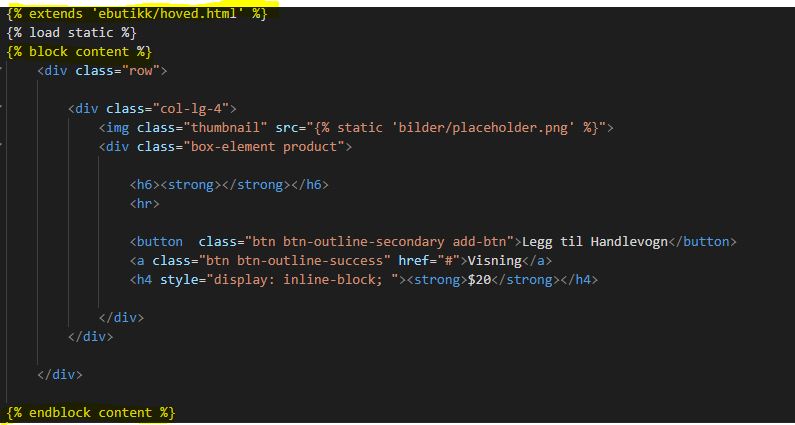
Arv av html filer

Måten det er mulig å arve html kode i Django er slik at du lager oppsettet på hvordan du vil hver side skal se ut. Det betyr hvis du har samme navigasjons baren og footer i hver side vil du kode dette i hoved.html. Deretter lager du en blokk av kode du vil der alle de andre sidene skal sitte.

Dette er hoved.html. Alt som er i mellom den kode blokken der vil være spesielt for hver side. Alt utenfor vil vises i hver side.



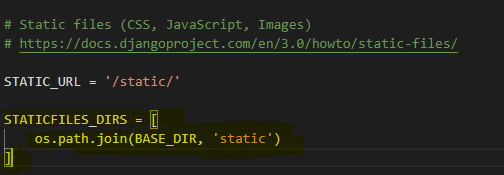
Måten du arver dette med andre htmlfiler er slik.



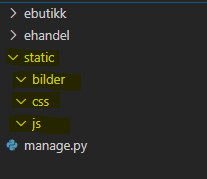
I toppen må du ha extends koden for å gi beskjed om fra hvilken fil du vil den skal arve fra. Deretter lager du de samme blokkene i denne filen. De som står i denne koden vil være det spesielle for denne siden. Så alt utenfor den blokken i hoved.html er arvet kode.

CSS, Javascript og Bilder

For å tilknytte dette opp mot html filene må vi først lage en mappe som heter static i vårt prosjekt. Etter denne er laget må vi i settings.py koble dette opp for at Django skal vite hvor disse filene ligger.

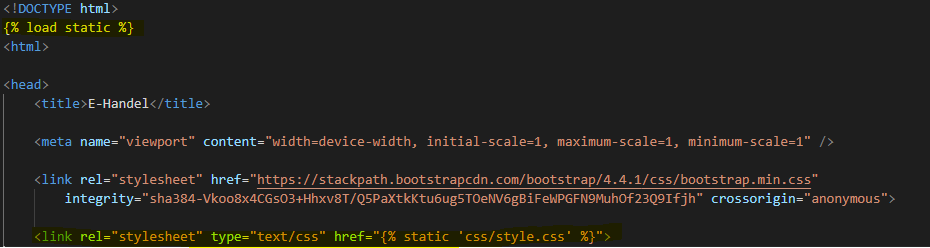
1. Steg: Lager static mappen og deretter kobler opp mot setting.py

ehandel/settings.py



Når har vi koblet opp filene med appen. Nå gjenstår det å legge til en CSS fil og deretter gjøre den synlig på html filen. Dette gjøres ved å bruke kode blokken {% load static %} på toppen av filen. Dette gjør det mulig for å vise CSS i siden. Uten denne kodeblokken vil vi få en error.

1. ***Steg: Kobler CSS til hoved.html***

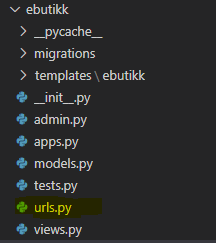


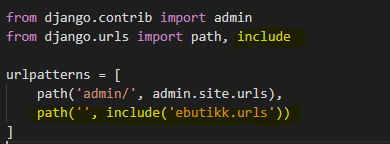
## Nå er vi ferdig med å sett opp html filene.

Kobling opp mot views og visning på server

For at disse filene skal vises på vår server må gjøre 3 steg.

1. Lage en urls.py i vår applikasjon, koble den så opp mot ehandel/urls.py i vårt prosjekt.
2. Lage en funksjon i views.py som gjengir vår html-side.
3. Koble så denne funksjonen opp mot en url i vår ebutikk/urls.py

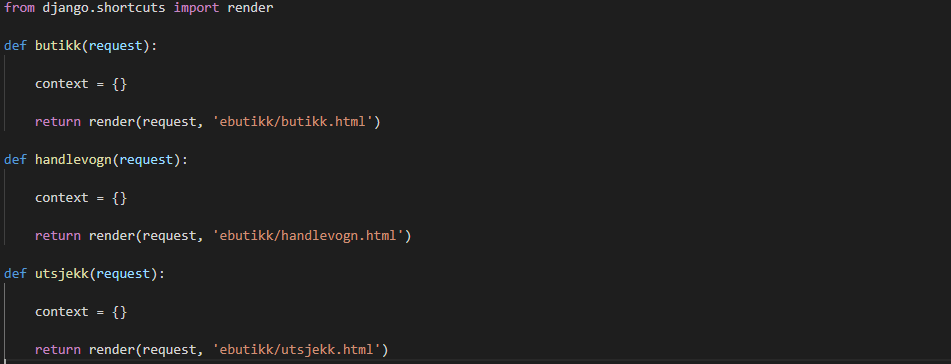
***1. Steg: lager urls.py, kobler opp med ehandel/urls.py***



ehandel/urls.py

🡪

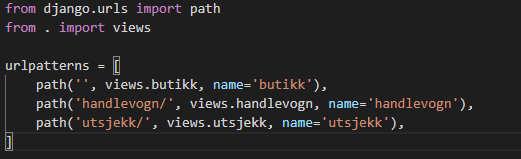
2***. Steg: lager en funksjon i views.py for å gjengi denne html filen. OBS: dette må gjentas for hver side vi ønsker å vise. Slutt resultatet blir:***



ebutikk/views.py

Plasseringen av vår html fil må vi skrive etter request. Html filene brukt her ligger i templates/ehandel/html.html, men Django vet allerede de befinner seg i en templates mappe dermed kan vi dropp denne.

***3. Steg: Vi koble opp alle funksjonene til en url for serveren.***



ebutikk/urls.py

## Nå har vi koblet opp funksjonene og siden vises ved hjelp av denne KOMMANDOEN.



Porten vil være:



Database oppsett og integrasjon

Nå som vi er ferdig med håndtering av html stukturen er det tid å ta en titt på backenden av vår side. I denne siden trenger vi databaser for å håndtere følgende.

* Lagring av kunder
* Lagring av produkter
* Lagring av ordre
* Lagring av ordre elementer så vi har mulighet til å oppdatere dem i vår handlevogn.
* Lagring av leverings adresse

I models.py har vi muligheten til å håndtere og integrere disse databasene. Dette gjør vi ved å lage klasser av database objektet for så å migrere det til databasen som er gitt av Django. Django gir oss tilgang til å kunne tilpasse vår database i forhold til hva vi trenger. Det er viktig å få med seg kodeord for hva som representerer hva. Vi har tilgang til å lage en database for navn, telefonnr, epost osv og flere foskjellige felt. Alle disse står i Django sin dokumentasjon under models. La oss starte med kunde modellen.

***Kunde Modell***

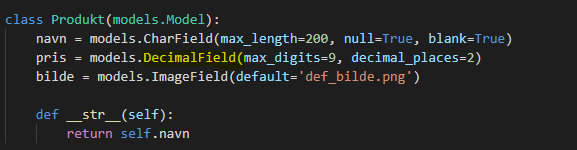
Vi må nå importere User fra Django (vist i bildet under) siden hver kunde skal være en bruker. En kunde skal ha navn, etternavn, en bruker og email.



Grunnen til vi bruker OneToOneField for bruker modellen, er fordi at en kunde kan bare ha en bruker og en bruker kan bare være en kunde. For at Django skal forstå dette må vi bruke OneToOneField i feltet. on\_delete=models.CASCADE gjør at kunden blir slettet hvis brukeren blir slettet. Dette gjør vi for i vårt oppsett må en kunde ha en bruker, dermed hvis denne brukeren blir slettet er han automatisk ikke en kunde. Blank = True betyr at databasen vil godta verider som er tomme og null = True gjør at databasen godtar verdier som er **null**. CharField er for felt med tekst og vi bestemmer maks lengde av den ved bruk av max\_length. Funksjonen under avgjør hva som blir synlig for oss i backenden i databasen.

***Produkt Modell***

Nå må vi lage en modell for produkter. Et produkt skal ha et navn, id får den automatisk ved laging av produktet så det trenger vi ikke å tenke på, pris og et bilde. Vi setter opp modellen slik:

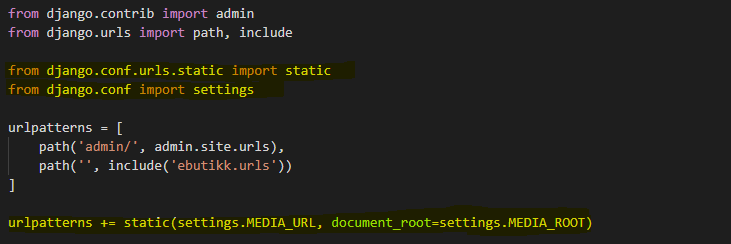


DecimalField gjør det mulig å lagre data med desimaler. Vi ønsker å vise 2 desimaler og en maks lengde på 9 tall.

Vi har en ImageField som gjør mulig å lagre bilder i vår database. Default betyr, det bilde som skal vises hvis intet bilde er lagt inn.

 Dette vil vi videreutvikle og gjør dette dynamisk. Vi skal kunne laste opp en valgt bilde og vise dette på siden vår. For å kunne laste opp bilder på vår database må vi først angi bildenes bane, hvor de skal bli lagret, så gi alle en url så det kan bli vist på siden. Vi går i settings.py filen og angir banen for bildene.

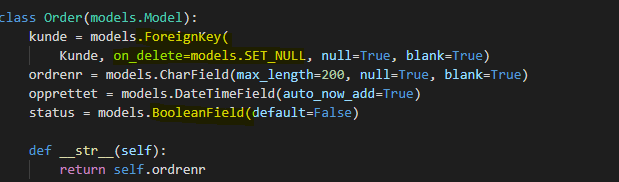
Nå har vi angitt bane på hvor bildene vi velger å laste opp skal lagres. Nå gjenstår det å generere en url for hvert bilde vi laster opp så vi har muligheten til å vise bildene på siden. Vi går i ehandel/urls.py og gjør følgende kode.



Vi må importere static og settings for å få tilgang til variabler. Nå har vi mulighet til å laste opp bilder.

***Order Modell***

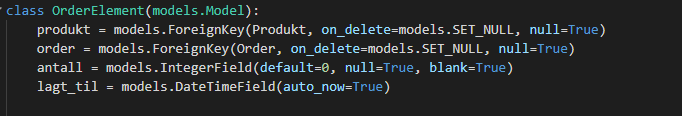
Nå må vi lage en modell for håndtering av ordre. En kunde skal ha muligheten til å ha flere ordrer. Så forholdet mellom kunden og ordre blir en til mange, One To Many. En ordre skal ha en ordre nr, dato opprettet og en status på om ordre er bekreftet eller ikke.



ForeginKey blir laget for å gi kunden muligheten til å ha flere enn en ordre. One To Many relationship som nevnt over. on\_delete=models.SET\_NULL betyr, hvis kunden blir slettet, ikke slett ordrett gjør verdiene til NULL. DateTimeField er for lagring av tid, auto\_now\_add=True gjør at tiden settes når ordren blir laget.

***Order Elementer***

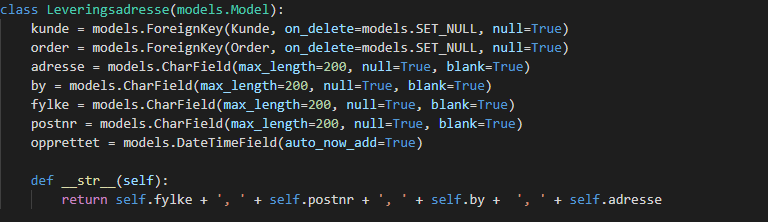
Dette blir laget for at vi skal kunne manipulere verdiene til hvert order. Hvis en kunde vil ha fler av et produkt så skal det være mulig, samt hvis personen vil ha mindre. Siden et order element kan ha flere av samme produkt må vi ha et en til mange forhold med produkter. Dette betyr at et order element kan være flere produkter, dette blir det hvis brukeren velger flere antall av et produkt. Siden vi skal ha mange order elementer i et order må vi også benytte dette forholdet til order. Dette gjøres ved bruk av nøkkelordet ForeginKey.



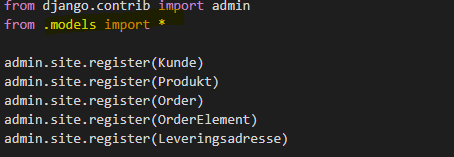
IntegerField er for lagring av heltall. Siden vi skal bare manipulere disse verdiene trenger vi å spesifisere noen funksjon for navnet.

***Leverings Adresse Modell***

For leverings adresse trenger vi felt for adresse, by, fylke, postnr og dato lagret. En kunde skal også muligheten til å ha flere leverings adresser ved flytting eller lignende. Dermed må vi ha en til mange forhold her også. Siden flere ordrer kan være knyttet til en leveringsadresse må vi også ha dette forholdet med order.

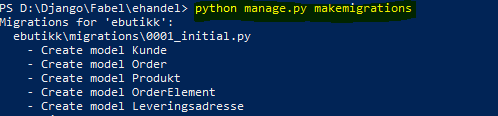


Nå er vi ferdig med å sette opp vår database i første omgang. For at disse skal vises i vår admin side må vi registrer modellene i admin.py filen.

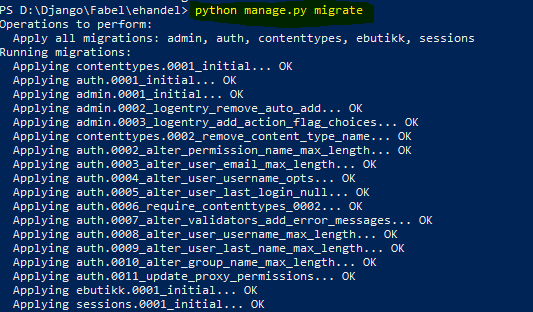


Vi må også importere alle modellene for å få tilgang til disse.

Nå gjenstår det migrere vår data base. Dette gjør vi ved følgende kommando. Merk du må være i samme filplasseringen som applikasjonen, dvs du må ha tilgang til manage.py filen.



Det som er markert i gul er kommandoen og responsen skal være som vist på bildet. Det gjenstår en kommando til.

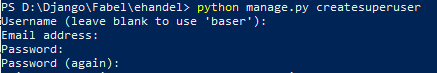


Siden det er førstegang vi gjør dette får vi flere migrasjoner som allerede var innebygd inne i prosjektet.

## Nå har laget en database struktur for vårt prosjekt.

Visning av lagred data på side

Siden vi skal selge produkter, må gjøre produktene synlig i vår side. Vil starte med å legge til noen produkter i vår data base. Før vi kan gjøre dette må vi lage en bruker som har evnen til å logge seg inn og kunne manipulere verdier i vår database. Vi kjørere den kommandoen og setter inn verdier som er etterspurt. Dette vil se slik ut.



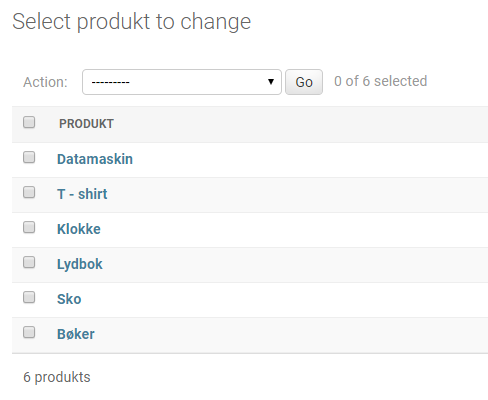
Nå som vi har lagd en bruker må vi starte opp serveren og logge inn i vår database. Starter serveren med kommandoen. Etter det går vi inn i vår lokal adresse etterfulgt med /admin i søkefeltet.



Jeg bruker google chrome. Nå etter vi har logget inn legger vi til noen produkter med navn og diverse. Jeg velger å legge til 6 produkter med passende bilder.

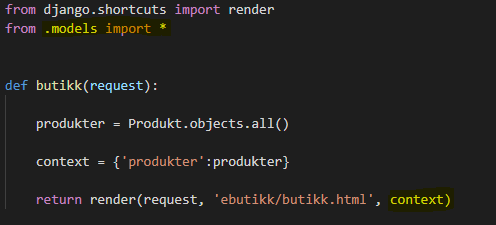
* Bok
* Datamaskin
* T-shirt
* Sko
* Klokke
* Lydbok

Admin siden etter innlagte produkter skal se slik ut:



Vi har nå 6 produkter. Disse produktene ønsker vi å vise på hovedsiden vår. Vi vil vise disse dynamisk. Dette betyr at vi ikke trenger å kode hvert produkt selv, vi ønsker å sette en kode som generere en visning av produktet og denne koden skal ta utgangspunkt i vår database.

Vi starter med å importere modellene i views.py filen. Etter vi har gjort det må vi lagre de ønskede produktene i en database. Deretter må vi mate den lagrede variabelen i konteksten deretter har vi mulighet til å vise disse på siden vår.



Når har vi importert alle modellene fra models.py, deretter lagret vi produktene i en variabel og lagt det i konteksten. I konteksten er det en Dictionary, det som står i ‘’ er hvordan vi vil referere til denne variabelen i html siden. Man kan i teorien sette hvilket som helst navn men det er viktig å bruke fornuftige variabelnavn.

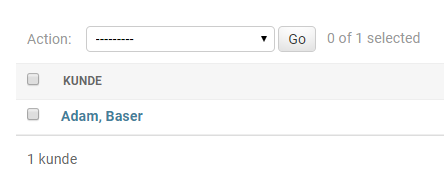
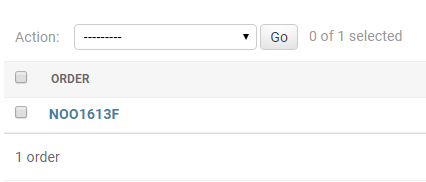
Nå gjenstår det å vise alle produktene i vår butkk.html side. Dette gjør vi ved å lage en forløkke som går gjennom hvert produkt og skriver ut informasjon vi vil skal vises på siden vår.



Vi sier nå til datamaskinene, for hvert produkt i variabelen produkter, lag denne kodeblokken her. I kodeblokken skriver vi ut produktets navn, pris og bilde. Merk at bilde skrives ut ved å sette .url på enden siden vi skal referere til en url og ikke selve bildet.

Nå viser vi alle produktene i på butikk.html siden vår. Nå ønsker vi å gjøre det samme for alle andre sidene. For å gjøre dette, må vi opprette en kunde, denne kunden må ha et order fylt med order elementer. Kunden vi lager nå er vår superuser som vi lagde for å logge inn på admin panelet i serveren vår.

Vi gjør dette ved å gå i admin panelet og oppretter en kunde. Deretter må vi gjøre tilsvarende for et order og order elementer.

Nå har vi laget en kunde, og et order tilknyttet denne kunden. Nå gjenstår det å putte produkter i dette ordret. Dette gjør vi ved å legge til et order element for gjeldene order.



Nå har vi laget 2 order elementer for tilsvarende order. Dette ønsker vi nå å vise på gjeldene sider.

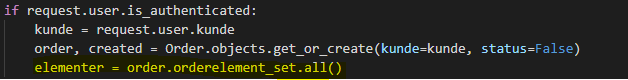
Før vi viser denne daten må vi sjekke om en kunde er logget inn. Hvis kunden er logget inn må vi hente kundens spesifikke order og vise dette i handlevognen. Denne kodes skriver vi først i handlevogn viewen for å gjengi kundens handlevogn.



Hvis brukeren som er etterspurt er autorisert, så setter lagrer vi denne kunden i et variabel. Ettersom når en kunde logger inn, er det ikke sikkert at denne kunden har noe i handlevognen. Dermed må vi lage en variabel som enten finner en order knyttet til denne brukeren eller lager en tom order med status = False .



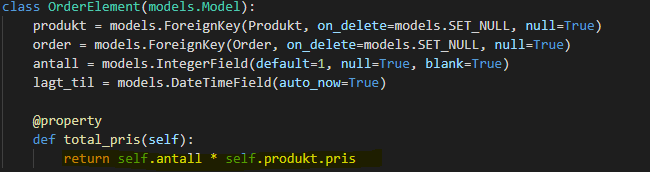
Nå har vi hentet eller laget et order og lagret den i variabelen order. Nå må vi hente alle order objektene tilknyttet til denne ordren.



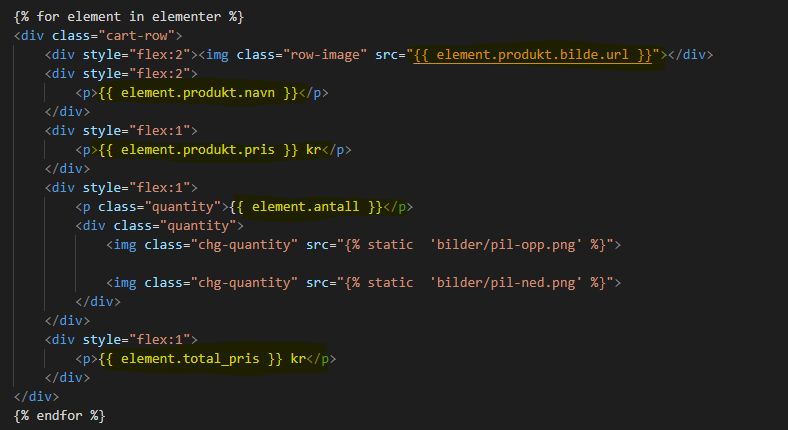
Siden vi allerede har lagret ordre i variabelen over. Kan vi bruke dette for å finne elementene for det spesifikke ordre vi leter etter. Måten vi kan gjengi data ved å bruke foreldreklassen er å skrive barneklassen med små bokstaver. Derfor har vi skrevet orderelement med små bokstaver. Nå har vi hentet alle elementer for det spesifikke ordre. Siden vi må også håndtere brukere som ikke er logget inn, må vi lage en else som håndterer dette.

Siden det er elementene i et order vi skal vise, trenger vi ikke brukeren. Vi lager en tom array i variabelen elementer. Vi skal videre utvikle dette etter hvert.

Nå har vi muligheten til å gjengi dataen fra brukeren sitt order. For å finne total prisen for en liste med elementer må vi lage en funksjon i models.py som håndterer dette. Det vi må tenkte på nå er at vi trenger å gange sammen produktene pris med antall av produktet som er i ordre.



Nå har vi laget en funksjon i models.py for dette. Denne blir dekorert med @property. Nå kan vi bearbeide koden så disse veridene vises på handlevogn.html. Siden vi nå jobber med et objekt som arver må vi ta hensyn til dette i koden vår.

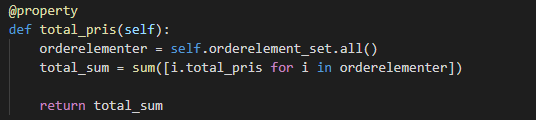


Her bruker vi samme oppsett som i butikk siden vår, men siden vi jobber med et objekt som arver for vi tilgang til daten i foreldre objektet ved å sette . bak og deretter ønskede ønsket arvk lasse og deretter ønsket verdi.

Vi ønsker nå å finne total prisen for hele ordret, og total antall elementer i ordret. Vi må gjøre dette i en funksjon i models.py. For å finne pris for hele ordret blir logikken slik. Vi ønsker å finne summen av prisen til alle ordre elementer. Denne funksjonen må vi lage i Order klassen siden, det er for denne modellen det gjelder.

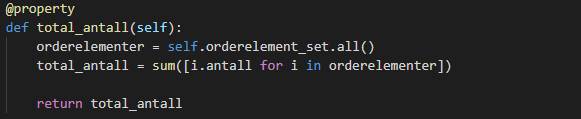


Vi lagrer først alle elementene som tilhører dette ordret i en variabel. Nå trenger vi å lage en forløkke som går gjennom alle elementene og finner totalprisen til hvert rekke av elementer. Siden vi har denne funkjonene i order elementer, kan vi kalle den for hver gang for hvert element.

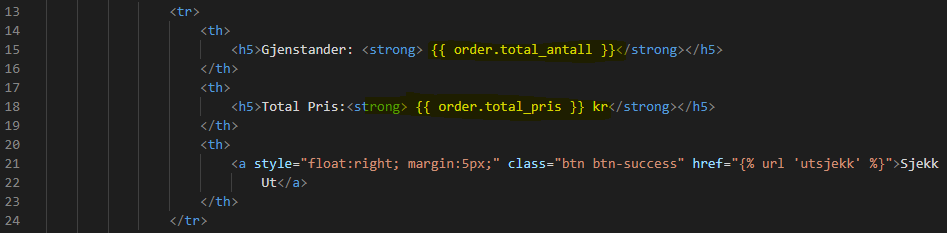


Nå går vi gjennom hvert element og kaller total pris funksjonen og deretter lagerer summen av den i variabelen total\_sum.

Det gjenstår å lage en funksjon som henter antall elementer i et order. Vi trenger denne gangen å sumere antall elementer i ordre elementer. Vi henter først alle elementen for dette ordret for så å summere antallet.

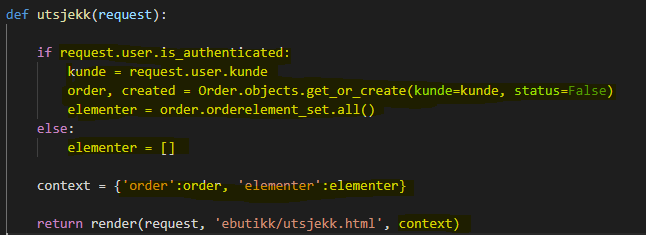


Vi lagrer først alle elementene i er variabel for så å summere antallet. Nå ønsker vi å vise dette på html siden. Dette gjør vi ved å kalle for ordret som vi har gjort før.



Siden vi ikke har en rekke med objekter trenger vi ikke å lage en for loop.

Vi ønsker å vise en oppsumering av handlevognen i utsjekk siden. Vi må no gjenta koden som vi gjorde i handlevogn viewen i utsjekk viewen.

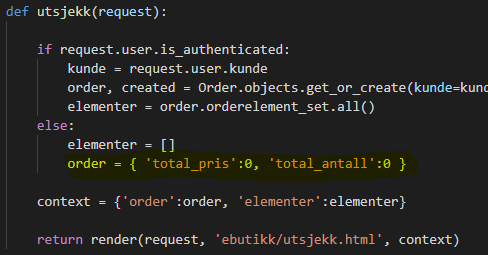


Vi sjekker om brukeren er autorisert, lagrer denne brukeren i en variabel. Finner eller lager ordre knyttet til denne kunden. Finner alle orderelementene tilknyttet til dette ordret og lagrer det også i en variabel. Nå bruker vi samme logikk som før for å vise dateen på utsjekk.html.



Vi lager en loop for alle elementene og skriver ut daten vi trenger. Kaller funksjonen til ordre for å finne total antall og pris.

Siden vi nå har laget et order total for brukere. Må vi gjøre det samme for gjest brukere. Vi il vidreutvikle denne i neste del. For nå lager vi en tom dictionary for å erstatte order for brukere som ikke er logget inn.

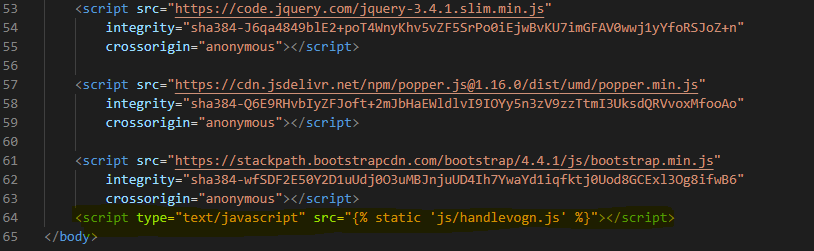


Nå sier vi til datamaskinen, hvis brukeren ikke er logget inn, så setter vi total\_pris og total\_antall til 0. Hvis vi ikke hadde gitt datamaskinen disse veridene ville vi ha fått en errror der datamaskinen prøver å finne disse totalene.

## Nå har vi skrevet ut order data i vår nettside

Funksjonalitet, legge, trekke og slette i handlevogn

Nå ønsker vi å lage kode som håndterer legge til i handlevogn, sletting av elementer, lagring av ordret, lagring av leveringsadresse osv. For å håndtere denn logikken må vi lage en JavaScript fil. Denne må vi først koble opp mot hoved.html for at alle html filer skal kunne få tilgang til den for så å starte å skrive logikken.

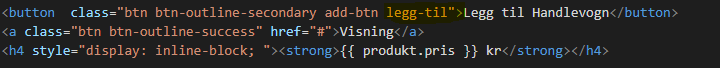


Nå har vi koblet javascript filen opp mot våre html filer. Vi vil at siden skal først lastest for så ta hensyn til javascript og derfor plasserer vi denne i bunnen av filen.

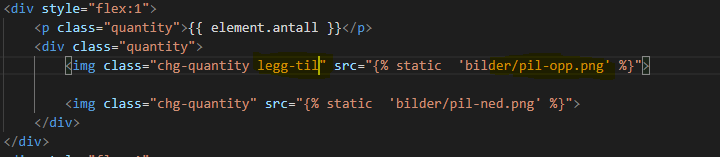
Vi ønsker å kunne legge til et produkt i vår handlevogn ved å trykke på legg til handlevogn knappen i vår butikk.html.



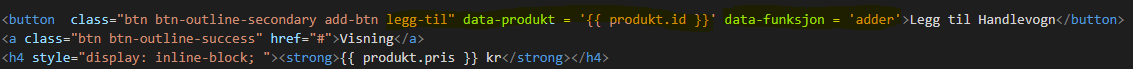
For å gjøre dette lager vi ført en funksjon som skriver ut noe når denne knappen er trykket for så å intergrere logikken.



Vi legger til en klasse legg-til for denne knappen for å kunne finne denne knappen og gi den en funksjon når vi klikker på den. I vår handlevogn, har vi også muligheten til å legge til flere av et produkt, så vi må gi pilen som peker opp samme klassen. Dette kan vi gjøre fordi disse to knappene skal ha samme logikk.



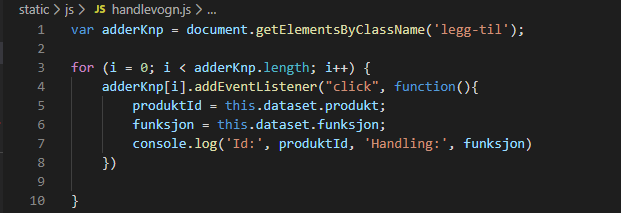
Nå har vi lagt til en klasse for pilen som peker opp i vår handlevogn. Vi ønsker også å gi disse to klassene en egendefinert atributt. En atributt gir deg muligheten til å skille denne ut fra andre tagger. I stedet for å bruke id eller klasse kan vi lage egendefinerte atributter til våre knapper. Måten vi gjør dette på er å gi først et atributt navn for så dens verdi. Vi ønsker å lage to atributter, en for henting av produkt id og en for funksjonaliteten. Grunnen til at vi lager et atributt for å hente produkt id er for at vi skal kunne manipulere verdien til produktet. Legge til fler eller trekke fra. Vi må bruke data-(navnet til atributten).



Nå har vi gitt vår knapp to atributter. Den første henter produkt iden til produktet du trykket på og den andre sier informerer at denne knappen har funkjsonen adder. Vi adde også disse to at atributtene til pilen som peker opp i vår handlevogn.



Med det første, ønsker vi å skrive ut i konsollen, hvilket produkt id som ble trykket og funksjonen til knappen. Vi gjør dette i javascript. Vi lager først en variabel som henter alle elementene med klassen ‘legg-til’, deretter må vi lage en forløkke for hver knapp som har denne iden. I denne forløkken skal vi skrive logikken for å skrive ut iden og funksjonen. Vi gjør dette slik.



Nå sier vi til datamaskinen, for hver knapp som har denne klassen, lagg en EVENTLISTNER. Denne skal aktiveres ved et klikk, og dette er funksjonen som kjøres ved dens klikk. Vi lagerer produkt iden i en variabel og funksjonen i en variabel, deretter skriver vi den ut i konsoll en til nettleseren. This nøkkelordet sier til datamaskinen dette elementet, alså hent datasetten til dette elementet som ble trykket på. Konsollen på nettelesern skal se slik ut .

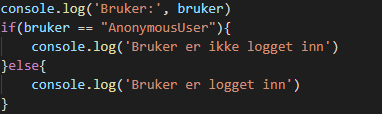
Nå klikket jeg på første produktet, og dette har iden 1 med handligen adder.

Siden vi må håndtere brukere som er ikke er logget inn, må vi lage en annen type logikk for disse brukerene. Vi ønsker nå å lage logikk som skiller brukere som er logget inn og brukere som er gjester. Det betyr at vi ved klikk av ‘legg til i handlevogn’ på butikk siden vår, ønsker vi å skille brukere som er logget inn eller ikke.

Dette gjør vi ved å først ved å koble sammen brukeren med vårt nettsted. Vi går i hoved.html og legger til følgende kode.



Nå lagrer vi nå brukeren i en variabel ved hjelp av python sin syntaks. Denne variabelen vil variere for brukere som er logget inn og brukere som ikke er logget in. Ved hjelp av denne kan vi skrive logikk som hånterer handlevogn funksjonalietet for brukere som ikke er logget inn. Siden vi har lagret denne variabelen i hoved.html, har vi tilgang til denne i vår java script fil. Vi går inn på handlevogn.js og skriver følgende logikk.



Nå sier vi, hvis brukeren vi har lagret er en AnonymousUser, dette er slik en bruker som ikke er autorisert blir lagret som i java dermed bruker vi nøkkelordet AnonymousUser, så skal vi skrive brukeren er ikke logget inn i konsollen. Tilsvardene for brukere som er logget inn, skal vi skrive ut bruker er logget inn. Nå har vi klar å skille mellom brukere som er logget inn og ikke. Grunnen til at vi gjør dette er at, logikken og funksjonaliteten er annerledes i forhold til om brukeren er logget inn eller ikke.

Slik blir loggen for brukere som er logget inn:



Dette er for brukere som er ikke logget inn:

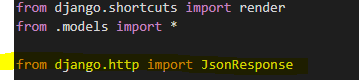


Vi ser at brukere som ikke er logget inn, får en verdi på AnonymousUser og dermed lager vi loggiken som vi gjorde over.

Vi staret med å lage funksjonaliteten for brukere som er logget inn.

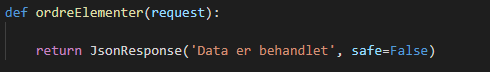
Får å kunne manipulerer verdiene som vi gir inn, må vi lage en funksjonalitet som tar imot data som vi sender inn ved et klikk. Vi ønsker å lage en view som hånterer antall ordre elementer i en handlevogn. Hvis vi klikker på legg til, ønsker vi å legge til et ordre element i vår handlevogn tilsvardene vil vi trekke fra et element. Vi skal ha muligheten til å slette produkten fra handlevognene eller trekke fra antallet av produktet. Dette skal vi kunne gjøre på handlevogn siden. Vi starter først med å lage den view som tar imot data vi trykker på.

Vi setter opp grunnlegende elementene for viewen, lager en funksjon og deretter kobler den til url.py.

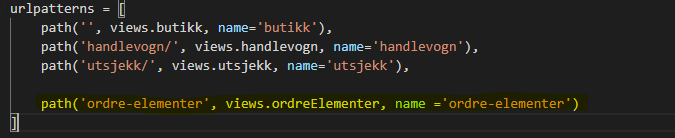


ebutikk/views.py

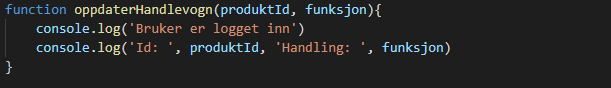
Vi importerer først JsonResponse, siden vi ikke ønsker å gjengi en html side, kan vi bare sende brukeren en meldig om at dataen har blitt sendt til denne viewen. Vi lager en funksjon i views.py og returnerer en response.



Når denne viewen blir kalt vil denne returnere en JsonResponse, dette betyr kort sagt at den vil returnere en melding. Nå må vi lage en url som kobler seg mot denne funksjonen. Dette gjør vi i ebutikk/urls.py



Vi ønsker nå å lage en funksjon som sender daten som vi lagrer til vår view og der kan vi håndtere den. Vi ønsker først å gjøre dette for brukere som er logget inn. Så i stedet for å printe ut ‘bruker er logget inn’ i vår konsoll logg, ønsker vi å lage en funksjon som sender dataen vi har klikket på til vår view. Dette gjør vi slik:

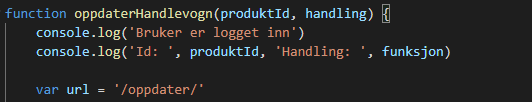


Vi lager først en funksjon som lagrer datene vi ønsker å manipulere. Denne funksjonen tar imot to parametere, produktid og funksjon. Disse to blir unik for hver klikk dermed ønsker vi å sende den inn som et parameter. Denne funksjonen ønsker vi å kalle for hver gang et produkt blir klikket.

Vi skriver inne i vår forløkke, kall denne funksjonen hver gang en bruker som er logget inn klikker på et produkt. Send in parameterne funksjonen tar inn. Ved hjelp av denne koden blir dette nå dynamisk.

Vi ønsker nå å sende denne dataen til backenden for å kunne manipulere antall order elementer osv. For at vi skal kunne gjøre dette må vi lage logikk i vår oppdataerHandlevogn funksjon. Denne logikken går ut på å omforme dataen vi får inn til et JSON objekt. Dette objektet blir da sendt vår view url. Det bruker vi en fetch api til å gjøre en POST request. POST request gjør at apien forventer en verdi, sammenlignet med GET der apien forventer å sende ut en verdi.

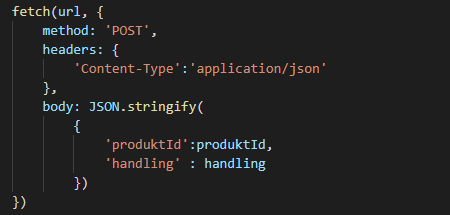
Vi henter først url vi skal sende dataene til.



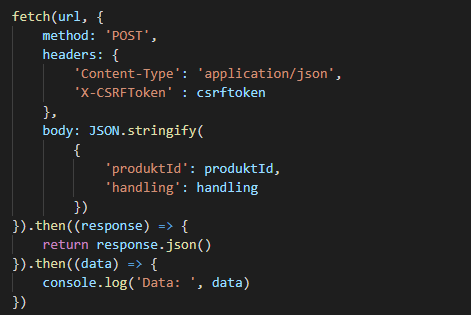
Nå har vi lagret urlen vi skal sende POST forespørselen til. Nå må vi bruke innebygde fetch funksjonen til javascript til å kunne sende denne forespørselen. Vi definerer etterspørselen i method feltet.



Urelen vi skal bruke skrives først. I dictionary skrives hva vi ønsker å sende inn. Måten vi bruker fetch apien på er å først sende inn urlen vi ønsker å sende daten til, deretter beskriver vi hvilken type data vi ønsker å sende.



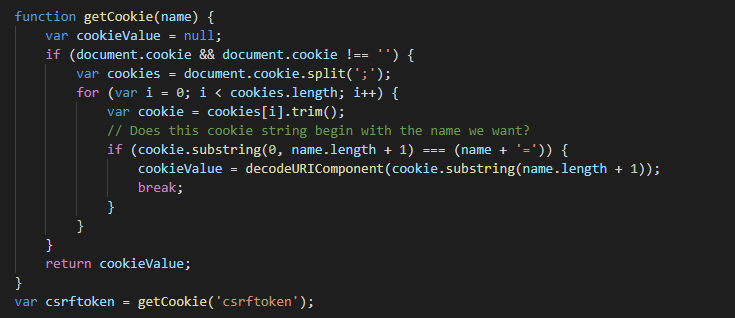
Nå har vi lagd en fetch kall til urlenen vi lagret, deretter sier vi til apien at dette er en POST forespørsel. Headers definerer hvilke type data apien kan forvente å få, i bodyen deklarer vi ønsker å behandle. Siden backenden forventer å få inn en string, gjør vi objektet inne i body om til en string før vi sender den.



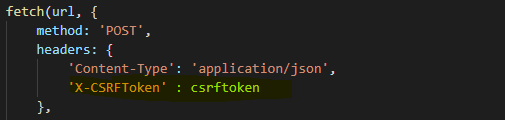
.then brukes til å kjøre logikk asynkront. Vi omgjør daten til en json objekt. => definerer en funksjon, så det som står inne i {} er funksjonens logikk. Data variabelen er hvilken type data vår view sender tilbake. Så i view sender vi tilbake en jsonResponse og det er denne datan vi printer ut i konsollen.

Siden vi bruker Django, forventes det en CSRF – Token. Denne må lage i JavaScript for så å send den inn til backenden ved hjelp av headers. Vi henter en funksjon laget av Django for at vi skal kunne hente en CSRF – Token ved bruk av javascript. Denne funksjonene legger vi i hoved.html for at alle andre html filer skal ha tilgang til denne.

I hoved.html



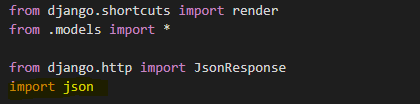
Denne funksjonen henter en CSRF Token som vi trenger får å kunne sende inn dataen til vår backend. Vi trenger nå i vår headers i fetch, sende inn denne med vår data.

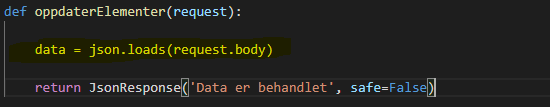


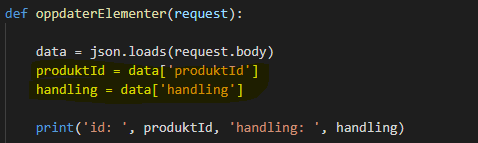
Siden vi har deklarert denne i hoved.html har vi tilgang til den. Nå har vi mulighet til å sende inn data til vår backend. I konsoll loggen ser vi Json Responsen vi har skrevet i vår view. Dette tyder på at daten gikk gjennom som den skulle og vi kan nå jobbe med å manipulere veridene.



Nå ønsker vi å kunne hente denne dataen vi vår view og kunne manipulere den.

Hvis vi husker tilbake, sende vi inn dataen vi ønsket å manipulere gjennom vår body. Så nå ønsker vi å lagre denne dataen i er variabel og deretter printe den ut for å se om det var dataen vi forventet å få. Først importerer vi json.

Lagrer nå dataen vi sende inn i en variabel.

Dataen er nå laget som et Json objekt. Dette betyr at dataen er i en dictionary. Måten vi henter ut data i en dictionary er å referere til dens key. Vi ønsker nå å lage en varabel for produkt iden vi sendte inn og for handlingen. 

Vi har nå lagret dataene vi sende inn i to forskjellige variabler. Disse variablene ønsker vi å skrive ut i komanndo prompten. Resultatet ved klikk på første produkt blir da.

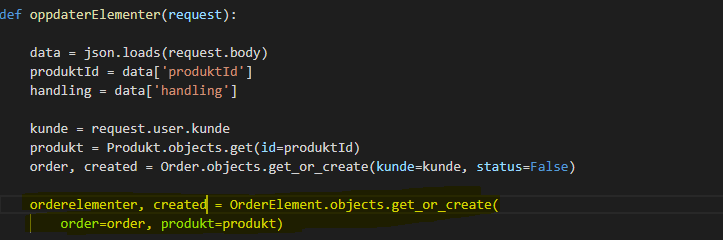


Her ser vi at iden til produktet er 1, og handlingen vi trykket på var adder. Under ser vi tidspunktet vi klikket på knappen og hvilke type etterspørsel dette var. Som vi har koded sendte vi en POST etter spørsel til urellen /oppdatar/. Alt gikk som planlagt.

Siden vi nå har muligheten til å forandre på dataen vi sendte inn, trenger vi å gjøre justeringer på order elementene til kundenes order. Vi må først lagre kunden som har trykket på denne knappen i en variabel. Deretter må vi denne sende inn denne daten brukeren har trykket på inn på order elementer. Vi gjør dette i oppdaterElemtener funksjonen.

Nå henter vi kunden, lagrer produktet han trykket på i en variabel. Denne lagrer vi ved å kalle produktens id i vår database. Så vi sier finn produktet med denne iden i vår Produkt database. Deretter lager vi eller finner et order tilknyttet til kunden med status verdi faslk. Dette gjøres for hvis kunden ikke har et order, ønsker vi å lage dette et order som er tomt.

Deretter må vi gjøre det samme for order elementer. Hvis det er første gang kunden legger til det produktet kunden har trykket i handlevognen må vi derfor lage dette elementet i vår logikk. Hvis det allerede finnes henter finner vi den også lagrer vi den i variabelen.



Nå har vi laget eller hentet et order element knyttet til produktet vi har trykket på. Siden vi har nå koblet alt sammen med riktig produkt og riktig bruker kan vi starte å skrive logikken for å kunne trekke fra antallet av produktet i vår handlevogn, samt addere eller fjerne helt.

Vi må utføre logikken basert på handlingen av knappen kunden har trykket på.

Feil i handlevogn, i stedet for produkt.id skal vi få element.produkt.id.

Dette er fordi vi ønsker å finne iden til det produktet som er et element i ordret til kunden.



Vi har nå bare en knapp. En knapp som legge til et antall i vårt order element. Vi ønsker nå å legg til to knapper. En knap for å trekke et antall av produktet og en knapp for å slette hele produktet fra ordret. Dette gjøres igjen ved å lage egendefinerte atributter som vi har gjort for pil opp iconet.

Gjør det samme for pil ned ikonet men forandrer handlingne til trekk siden vi ønsker å trekke et antall ved klikk.



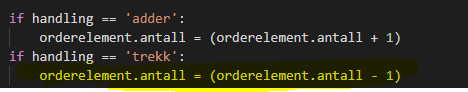
Vi legget til en knapp for å slette et order element i ordret.



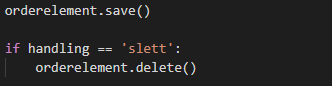
Ønsket er også igjen å sende produktet med dens id for å at vi skal kunne redigerer på elementet. Det som gjenstår nå er å lage logikk for disse handlingene vi har gitt knappene våre. Som visst over har vi nå funnet kunden og tilsvarende order og order elementer. Nå gjenstår det å lage logikk på disse handlingen i vår view. Ved klikk på pil opp, ønsker vi legge til et antall i ordere elementet.

oppdaterElement()

Ønsker å trekke fra et antall ved handling trekk.

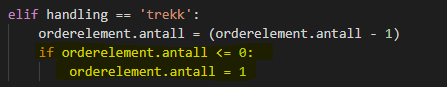


Siste er å slette ordret ved slett handlingen.



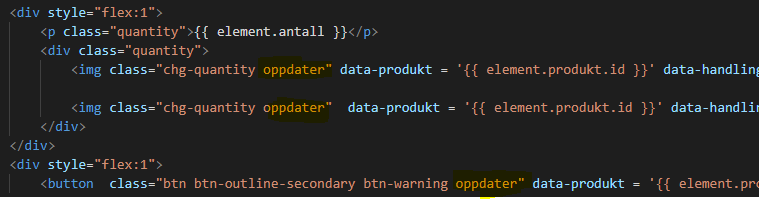
Vi ønsker å lagre antallet etter klikke av handlingen adder eller trekk og deretter ha muligheten til å kunne slette elementet fra ordret. Dette gjøres pga hvis man lagrer elementet etter man sletter det vil det ikke forsvinne fra handlevognen. Derfor velger vi å lagre først for så å kunne slette. Nå er det mulig å få antall til en negativ verdi. For å fikse dette må vi lage logikk som stopper brukeren før antallet kan bli negativt. Dette må gjøres i if settingen til handlignen trekk.

Dette er pga at dette er den funkjsonen som trekker fra et antall. Så hvis vi hadde puttet den uten for hadde den først trekket lagret for så å sjekke om antallet var mindre eller lik null. Da blir logikken slik.

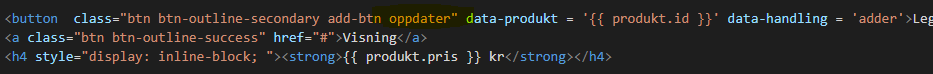


Vi sier nå, hvis orderelement antallet blir 0 eller lavere enn null, reset antallet til å være 1.

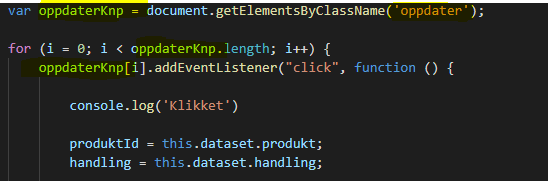
Nå har vi laget logikk og funksjonalitet for disse tre handlingene. Det som gjenstår nå er å legge til en addeventlistner for når disse knappen blir tykket. En løsning er å gi hver av de en forskjellg id for så å gjenta koden vi har laget i vår handlevogn.js. For å unngå repitasjon av kode velger vi å gi alle tre knappene klassen oppdater og deretter kalle en eventlistner ved klikk av knapper med denne klassen. Da blir koden vi vår handlevogn.html slik.



Vi må også gjøre dette i vår butikk.html for knappen legg til i handlevogn.



Nå må vi forandre kode i vår javascript fil for å høre etter elementer med dette klasse navnet. Da blir koden seende ut slik:



Vi ønsker nå å oppdatere handlevognen øverst i høyre hjørne i forhold til brukerens order elementer.

## Nå har vi muligheten til å legge til, trekke og slette elementer til vår handlevogn.

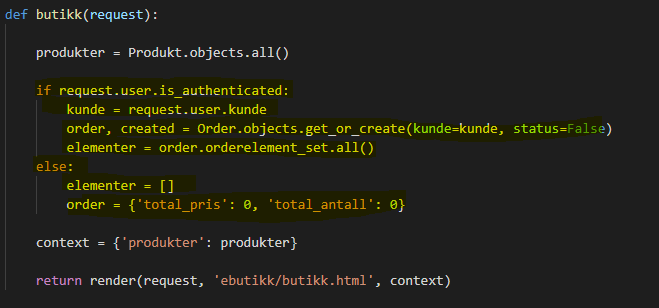
Oppsummering

Vi skriver en liten oppsummering på hva vi har gjort for å gjøre ting litt klarere. Vi har nå satt opp 4 sider som håndterer en kunde i en elektronisk butikk. Vi har laget butikk.html, handlevogn.html, utsjekk.html og hoved.html. Hoved html filen er en arver bort kode til alle andre sidene. Butikk er siden vi viser produktene, handlevogn er der vi viser kundens handlevogn og utsjekk er den siste siden kunden ser før betaling.

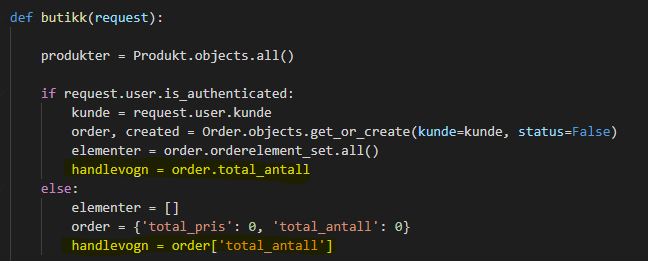
Vi har også laget funksjoner som viser våre produkter dynamisk. Dette betyr at vi ikke trenger å gjenta kode for et produkt, dette blir generert automatisk. Siste vi gjorde var å lage funksjonalitet for å kunne legge til, slette produkter i handlevognen.

Lagre data fra order

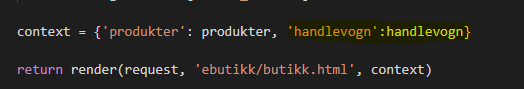
Kunden kan nå legge til elementer i handlevognene. Vi ønsker nå å lagre ordre kunden har og sende disse informasjonene til backden for å håndtere ordre. Leveringsadressen til kunden samt kundens navn og bruker skal bli sendt til oss for behandling. For å gjøre dette må vi først lagre alle elementene i handlevognen i en variabel. For å få tilgang til dette må vi i vår butikk view gjenta kode vi allerede har brukt for å hente brukeren og brukerens order.



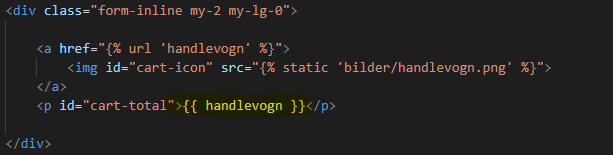
Igjen vi har funnet brukeren og ordre tilknyttet til denne brukeren. Nå må vi hente antallet av elementer i ordre til kunden.



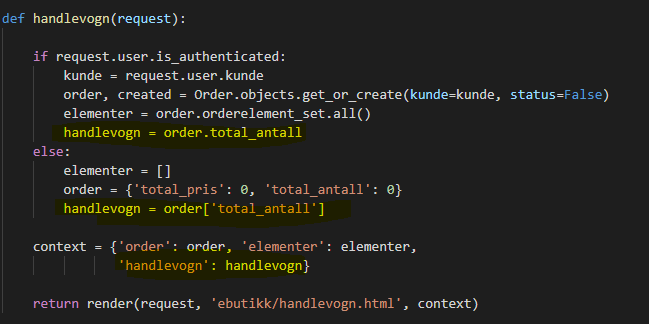
Nå har vi lagret antallet av elementer i et order i variabelen handlevogn. For brukere som ikke er logget in har vi sagt at handlevogn er keyen totalt antall til order objektet. Siden order er en dictionary kan vi skrive dette slik henter vi keyen på måten illustrert over. Siden vi har nå lagret antallet i en variabel kan vi sende den inne i konteksten for å bruke i vår html side.



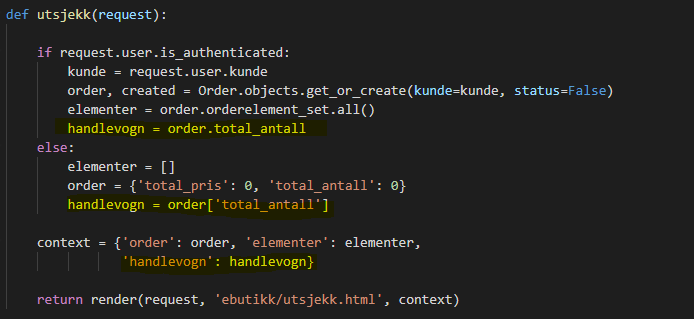
Nå sender vi inn denne på hoved.html for visning av antall elementer i handlevognen.



Som koden står nå vil oppdateringen av handlevogn antallet bare vises på butikk.html siden. Dette er fordi det er bare denne siden som har tilgang til denne variabelen. Vi må nå legge til handlevogn variabelen i de to andre funksjonene.



Slik blir koden for handlevogn viewen.

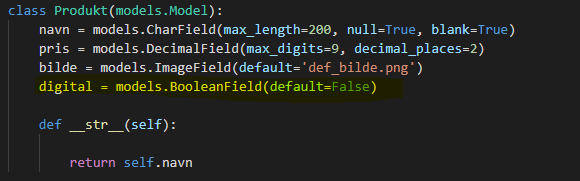


Vi henter antallet av ordre elementer fra tilhørende order og sender denne variabelen inn i vår html side for å kunne jobbe med den der.



Vi ser at tallet oppdatere seg i forhold til antallet av produkter i ordre i hver side. Som koden våres er nå laster vi siden på nytt hver gang et produkt blir lagt til i handlevognen. Siden vi dette liten nettside med lite data går dette relativt fort. For sider med mye data og mange produkter ville dette være uaktuelt. For å håndtere denne funksjonen blir da JavaScript brukt, utvikling av dette skal vi jobbe med senere.

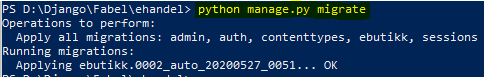
Vi vil nå bearbeide funksjonene til leveringsadressen. Det viser seg at det er noen produkter i vår butikk som ikke trenger å bli fraktet. Dette betyr at en leveringsadresse ikke er nødvending. For å håndtere dette må vi implementere logikk som tar hensyn til dette. Det vi kan gjøre er i vår produkt database, definere om et produkt er et digitalt produkt eller et fysisk produkt. Vi går nå inne i models.py og skriver denne logikken.



Vi oppdatere vårt produkt modell slik. Alle produkter vil vanligvis være et fysisk produkt med mindre vi gir verdien digital – True.

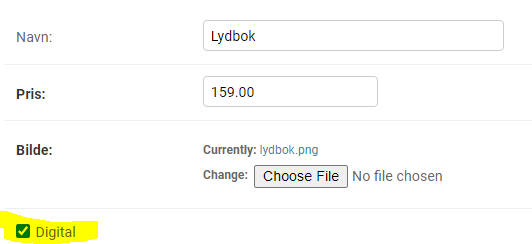
For å se disse endringene må vi migrere databasen vår. Vi bruker først kommandoen makemigrations og deretter migrate.



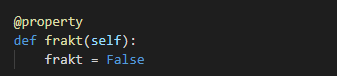


Vi forventer dette resultatet.

I våre produkter, finnes det en lydbok. Denne lydboken er ikke en fysisk gjenstand og trenger heller ikke en leveringsadresse. Vi går inn i admin siden og forandrer verdien på digital fra false til true.

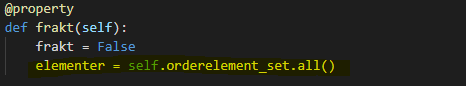


Målet nå er å håndtere ordre med bare digitale gjenstander. Dette må gjøres i order modellen der vi lager en variabel og forandrer dens verdi i forhold til digital verdien til ordre elementene. Vi trenger nå å skrive logikk som fjerner leveringsadressen hvis alle produktene i handlevognen er digitale. Det betyr ar hvis en av produktene i handlevognen ikke er digital, skal vi vise formen for leveringsadressen.

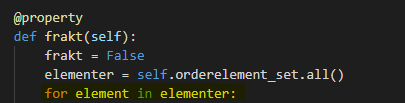


models.py – order()

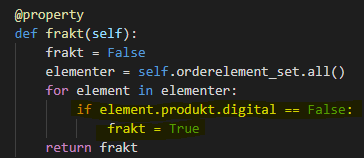
Vi starter først med å lage en variabel for frakt. Denne setter vi til å være False. Vi må nå hente alle elementene ordre det gjelder, så vi gjør som vi har gjort tidligere.



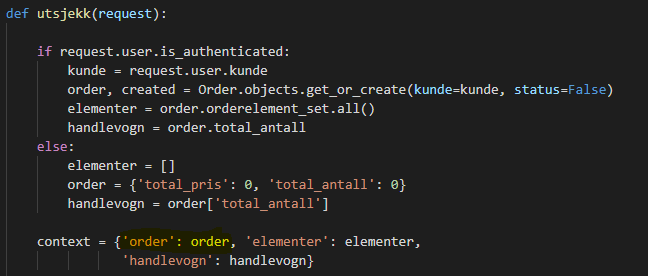
Nå har vi hentet alle ordre elementene tilhørende til dette spesifikke ordre og lagret det i en variabel elementer. Vi trenger nå å lage en forkløkke som går gjennom alle disse elementene.

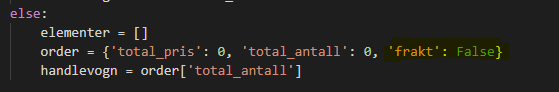


Vi ønsker å se om noen av elementene i dette ordre ikke er digitale. Hvis en av ordrene ikke er digitale, setter iv frakt til å være true.

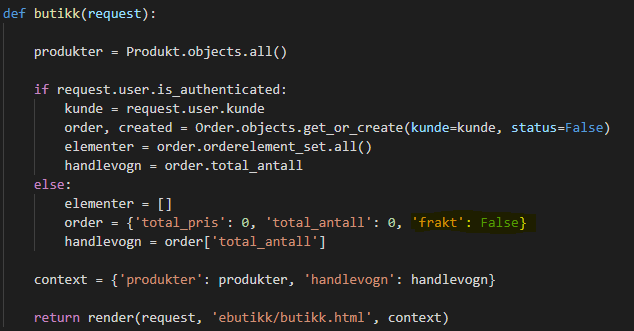
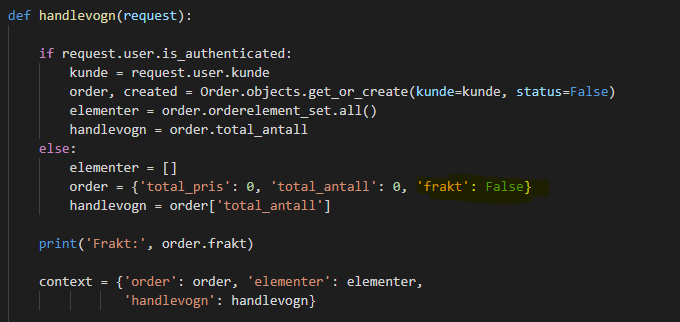


Denne funksjonen vil nå returnere frakt med hensyn til om produktet er digital eller ikke. Dette kan vi nå bruke til å vise eller fjerne frakt informasjon på vår utsjekk side. Siden vi har allerede sendt in order modellen i vår kontekts har vi tilgang til denne metoden.

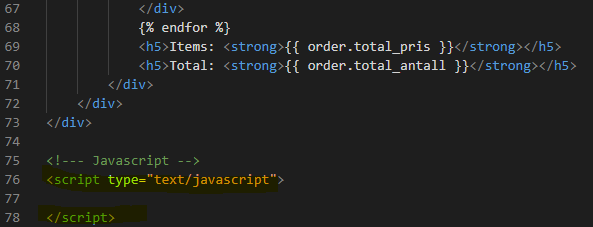


Nå har vi håndtert frakt for brukere som er logget inn. Vi trenger nå å oppgi en frakt for brukere som er gjester. 

Vi gir order dictionaryen en key frakt med verdien false. Dette blir brukt for brukere som ikke er logget inn. Deretter må vi legge dette til i handlevogn viewen og brutikk viewen. Dette gjøres for å gi tilgang til frakten for gjest brukere i hver av sidene.



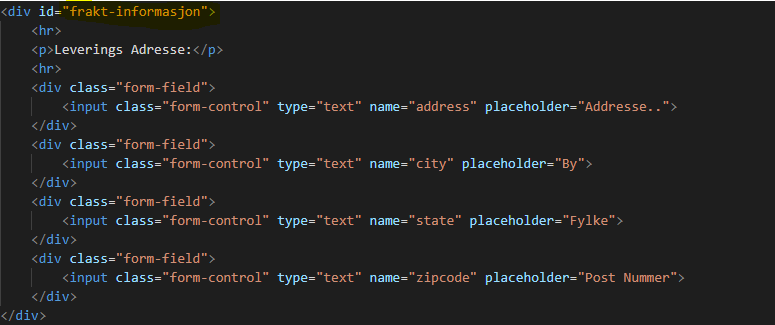
Det vi prøver å gjør er hvis frakten på ordre er false, skal vi gjemme frakt informasjons formen. Vi går nå i utsjekk.html og skriver java kode på bunnen av siden. Vi gjør det på bunnen fordi siden må lastes først før vi kan manipulere verdiene på siden.



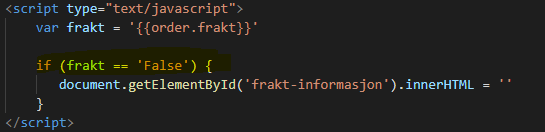
Vi trenger nå å hente resultatet av frakten til ordre. Dette gjøres på en spesiell måte siden vi henter variabelen fra python.



Siden vi henter variabelen fra python må vi lagre den som en string. Vi ønsker nå å skrive logikk som gjemmer feltet med frakt-informasjon hvis frakten til orde er false. Altså vi ønsker å gjemme frakt feltet hvis elementene i ordre er digitale.



Vi trenger denne iden. Hivs frakten er false, ønsker vi å fjerne all innhold i dette feltet.



Siden variabelen frakt er lagret i en streng, finner må vi skrive False i en streng.

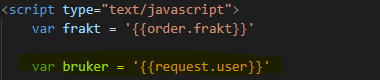
Resultatet blir nå som vi ville. Vi har laget en funksjon i order modellen som returnerer frakt til true eller false. Hvis noen av produktene i ordre ikke er digitale, skal vi returnere frakt til å være true og motsatt. Dette resultatet lagrer vi i variabelen frakt ved hjelp av javascript i vår utsjekk.html side. Vi sier da, hvis denne variabelen er false, setter vi innholdet til elementet med iden ‘frakt-informasjon’ til å være en som streng.

Vi må nå håndtere følgende senarioer.

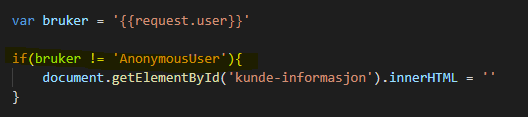
For brukere som er inlogget:

* Gjemme navn og email feltet.
* Ved klikk på betaling kanppe, gjemme frakt-informasjon, hvis den ikke er gjemt og deretter vise feltet for betaling.

Vi henter brukeren ved hjelp av lignende syntaks for frakt.



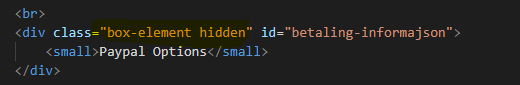
Lagrer brukeren i en variabel. Vi trenger logikk for hvis denne brukeren ikke er Anonymous så skal vi gjemme feltet for kunde-informasjon. Altså hvis brukeren er en autorisert bruker skal vi bli kvitt kunde-informasjons feltet.



Hvis brukeren nå ikke er en Anonymous bruker så skal vi gjemme kunde-informasjons feltet.

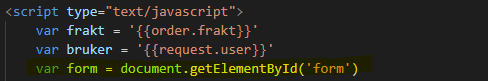
Det må nå skrives kode som gjør ved klikk av betalings knappen at det feltet for betalings-informasjon åpnes.

Vi ser på vår html side på utsjekk.html

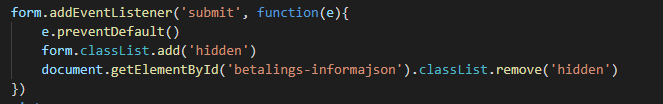


Denne boksen er ment for betalings informasjon. Vi ser på klassen at den har klassen hidden. Dette gjør at alt inne i denne boksen ikke blir vist på nettleseren. Vi ønsker nå å vise denne ved klikk på betalings knappen.

Først må vi hente gjeldene form og legge til en eventlistner som lyter etter submit siden dette er en form.



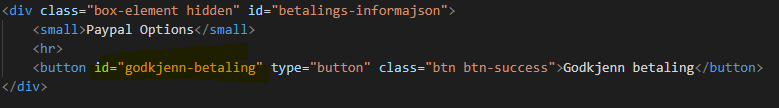
Vi lagrer formen i en variabel.



e.preventDefault() gjør at formen ikke blir send inn automatisk. I vår logikk sier vi, ved submit, skal vi adde klassen hidden til formen, deretter skal vi fjerne klassen fra betalings informasjon blokken.

Hvis et leveringsadressen er fylt ut og betalingen er godkjent ønsker vi å sende ordre informasjonen samt kunde informasjonen til vår backend. For brukere som er logget inne, blir det sendt brukernavn til backenden mens for brukere som ikke er logget in blir informasjonen i kunde-informasjon blokken sendt. Hvis produktet trenger frakt blir leveringsadressen sendt til vår backend. Vi må gjøre lik programmering som order element oppdaterings funksjonen vi lagde. Vi ønsker å sende data i json format til backenden der vi kan lagre den.

Vi legger til en knapp for betalings informasjon får å gjøre kunne utføre denne logikken.



Vi lagrer så denne i en variabel.



Når denne knappen trykkes skal vi sende informasjonen til backenden.

**Behandling av ordre.**

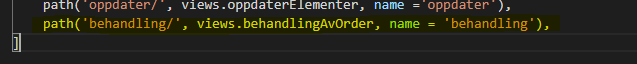
For å behandle order, trenger vi først å tenke på om brukeren er logget inn eller ikke. For en bruker som er logget vil kunde informasjon feltet være irrelevant fordi kunden allerede er registrert på vårt nettsted. For brukere som ikke logget inn vil vi ta informasjonen skrevet på kunde informasjon feltet. For å kunne gjøre dette må vi lage if else kode som håndterer dette.

Vi må også gjøre det samme for frakt informasjon. For ordre som ikke behøver frakt, kan vi sende informasjon om dette til vår backend. For ordre som trenger frakt, sender vi leveringsadressen til vår backend.

Første vi må gjøre er å lage en view som hånterer disse dataene. Vi går inn i views.py og lager følgende funksjon.



Så må vi koble denne viewen opp mote en url



Denne viewen returnerer en JsonResponse der det skrives order godkjent. Hvis vi husker tilbake til oppdatering av order, lagde vi en fetch i javascript for å sende dataen i Jsonformat tilbake til viewen. Vi må gjøre lignende logikk her også.

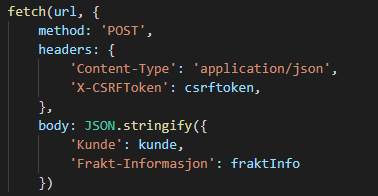
Vi beveger oss til utsjekk.html og lager en url vi skal sende fetch til.



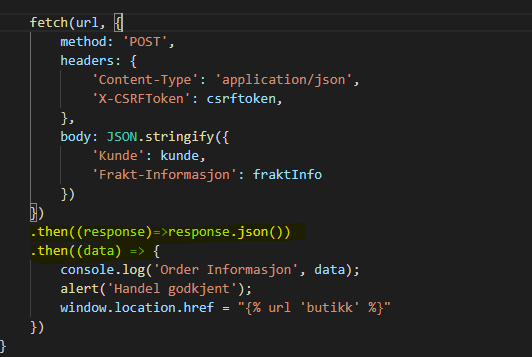
Setter denne deretter inn i fetch.



Likt som vi har gjort før må vi spesifiser hvilken metode vi ønsker å sende inn. Hvilken type format det kan forvente og sendes og hvilken informasjon vi ønsker å sende. Vi gjør følgende.



Vi ønsker å sende en POST metode, så det betyr at vi skal endre på noe. Hvis det hadde vært en GET metode forventer apien at vi skal hente data, men i dette tilfellet skal vi sende data og dermed blir metoden POST. I vår headers spesefiserer vi hvilken type format vi ønsker å sende den i og legger til vår csrftoken for sending av informasjon til Django. I vår body spesifiserer vi hvilken type data vi ønsker å sende. JSON. Stringifiy gjør at informasjonen vi sender blir send i json format, altså i en dictionary med keys og values. Vi ønsker å sende informasjon om kunden som bestilte dette ordret og frakt informasjonen hvis det er frakt. Videre sende dataen i form av en json.response

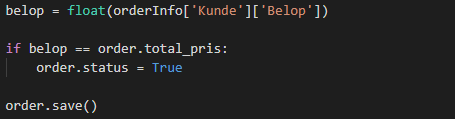


Vi ønsker også å konsolle dataen får tilbake av backenden. Data i denne sammenheng er det vi får som JsonResponse i vår view. Vi husker at vi lagde en JsonResponse der vi skrev Order godkjent, og dette er dataen vi får tilbake da. Vi øsnker også å gi brukeren en alert om at handelen er godkjent og deretter sende han tilbake til butikken sin hovedside.

For å kunne prossesere ordret og lagre orderet må vi gjør logikk i vår view. Vi må først få tilgang til dataen vi sende fra utsjekk.html. Vi går i views.py og skriver følgende.

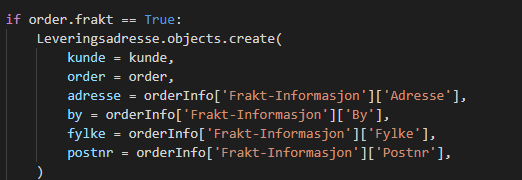


Dataen vi sendte i vår view er nå lagret i denne variabelen. Nå må vi sjekke om brukeren er logget inn og lagre hans bruker og ordre i en variabel.

For å ungå at noen manipulerer dataene våre og ikke endrer på verdiene i vår database må vi sammenligne om totalen på ordret er samme totalen vi har fått inn av responesen. Vi ønsker ført å lagre totalen vi har sendt inn via json og sammen ligne den med totalen i ordret tilknyttet til denne kunden.

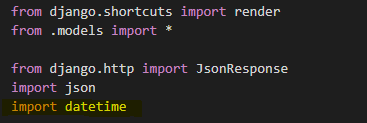
Vi lagrer først totalen vi sendte inn i en variabel belop, deretter sammenligner vi den og totalen til ordret tilknyttet til kunden. Hvis disse er like, setter vi order statusen til True, dette gjør at den forsvinner fra handlevognene. Hvis vi ser tilbake til vår kode, henter vi bare ordre med status = false og når denne blir satt til true blir den dermed ikke hentet. Vi ønsker uansett om beløpene er like eller ikke å lagre ordret. Hvis beløpene ikke er like blir ikke order status = True og dermed blir den ikke godkjent.

Nå må vi lagre leveringsadressen vi sende inn.

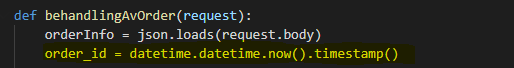


Vi sier nå, hvis frakt er true, det betyr hvis ordre elementene inneholder en et element som er fysisk, skal vi lage en leveringadresse i vår database. Kunden blir lagret, ordret blir også lagret samt adrese, by, fylke og postnr. Disse er alle dataene vi sendte inn.

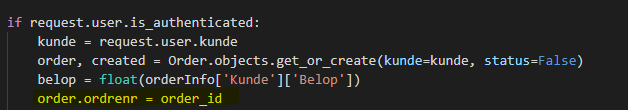
Får å sette et order id på ordre gjør vi følgende. Vi importerer date time.



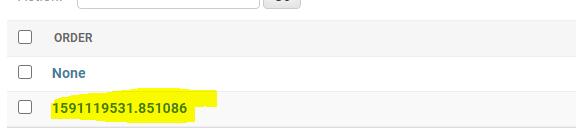
Deretter lager vi en variabel som setter en timestamp på datoen og deretter gir setter denne variabelen til order id vi har i databasen.



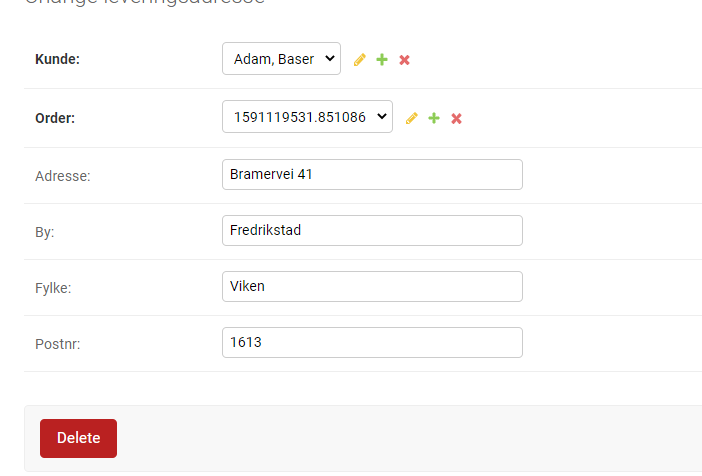
Lager først variabelen. Timestamp gjør at det blir en lang sett av karakterer og denne blir unik for hver gang. Vi ønsker nå å lagre order iden til denne variabelen hvis brukeren er innlogget.



Nå sier vi hvis brukeren er logget inn, gå inn i order sin ordernr og sett denne til å være den variabelen vi nettopp lagde. Da skal data basen se slik ut etter vi har prossesert et order og fulført betalingen.



Vi skal få et order som har status = True, dette ordret er nettopp det ordre vi har vi har godkjent. Grunnen til at et annet order er lagt til er fordi vi lager automatisk et order for en bruker som er inne på butikk siden vår med status False. Vi ser inne på levering adresser at adressen vi har skrevet inn har også blitt lagret. Adressen er tilknyttet en kunde og et order.



## Nå Har vi sendt og lagret et order.

Oppsumering

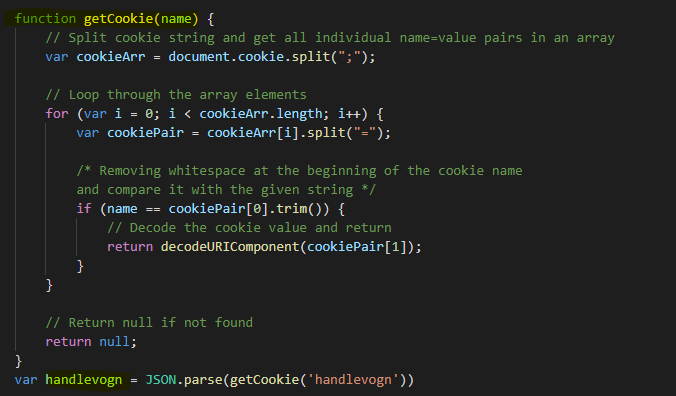
Vi har nå gjort det mulig å kunne behandle ordre. Dette betyr at en kunde kan nå bestille et produkt. Kunden sin informasjon blir lagret i vår database der vi kan behandle ordre. Vi gjør dette ved å sende en POST forespørsel fra vår butikk.html, vi lagrer først informasjonen om kunden i variabler og disse variabelene sender vi til vår backend view via JSON format. Denne dataen åpner vi opp vi vår view. Deretter lager vi en variabel for orde nummer og sjekker om totalen er vi sender inn via html siden er lik totalen på ordre som gjelder. Deretter lager vi en ny leveringsadresse i vår database der informasjonen vi sende inn erstatter adresse osv. for ordre som ikke trenger en leverings adresse blir denne ikke laget.

Gjest bruker funksjonalitet

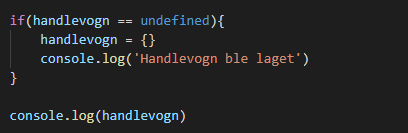
Nå skal vi gjøre det mulig for en gjest bruker å kunne legge til produkter til sin handlevogn og deretter plassere et order. For å kunne gjøre dette, trenger vi å bruke noe som heter cookies. Cookies er minne som er lagret på nettsiden for hver enkelt bruker, minnet blir lagret for så lenge vi trenger. Det betyr at når brukeren lukker nettsiden vil ordre fortsatt være lagret. Ved hjelp av cookies får vi lagt til elementer i ordre for en gjest bruker, samt fjerne et element.

For å kunne manipulere cookie data, trenger vi en funksjon som lager cookien for oss, deretter kan vi manipulere cookien vi har laget. Vi må nå lage en funksjon som lager en cookie for oss. Denne funksjonen ligner veldig på funkjsonen vi hentet for å lage en csrf token på vår nettside. Grunnen til dette er at vi lager en csrftoken som en cookie i vår nettside.

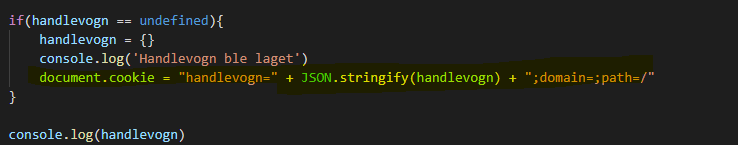
Vi lager denne funksjonen i hoved.html så alle andre sidene har tilgang til denne. 

Vi går i vår forgje getCookie funksjon og forandrer navnet til getCsrfToken. Nå lager vi en ny funksjon.

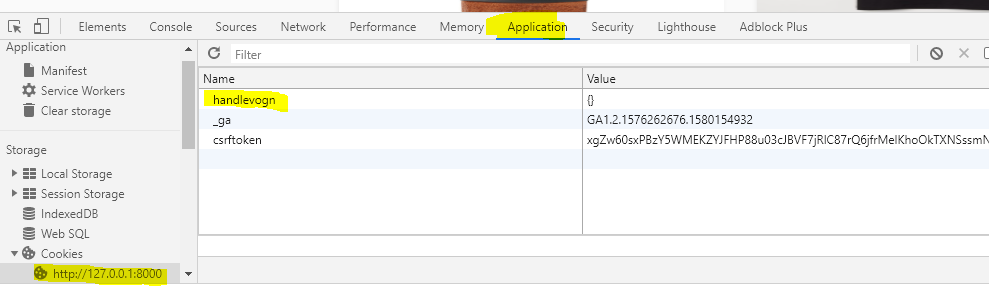
Denne funksjonen sjekker om det finnes en om det finnes en cookie som heter handlevogn. Hvis denne ikke finnes returnere den null eller undefined. Nå må vi lage logikk for at hvis denne handlevognen ikke finnes, skal vi lage en cookie som er tom og den skal vi kalle handlevogn.



Denne funksjonen sjekker først om handlevogn variabelen er udefinert, hvis den er det lager det en tom JavaScript objekt og konsoller ut at denne ble laget. Vi må videre lagre denne variabelen som siden sin cookie. Dette gjør vi ved å sette domene for cookien, altså hvor den tilhører. I dette tilfellet er det siden brukeren er på. Vi gjør følgende.

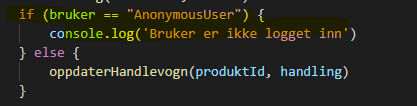


Nå lager vi en cookie som heter handlevogn. Vi gir denne cookien verdien av handlevogn og deretter gjør denne verdien om til JSON format. Vi lager så domenet cookien skal være lagret til, og den siste backlashen gjør at denne cookien blir lagret i butikk.html for det er slik vi satt opp våre urls.

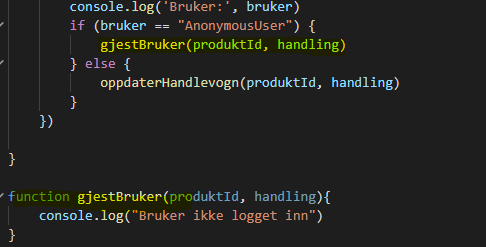


I vår applikasjon sine cookies skal vi nå se dette. Hvis vi refresher vår side nå, ser vi at cookien blir ikke lagret fordi vi allerede har gjort det. Dette betyr at vi har nå lagret en cookie på vår side. Denne skal vi bruke til å kunne legge til produkter til vår handlevogn samt lage et order.

Hvis vi beveger oss til handlevogn.js ser vi at vi har skrevet logikk for når brukeren ikke er logget inn.

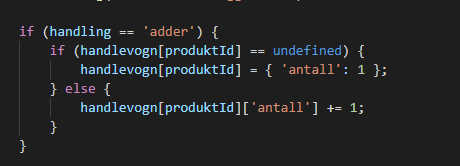


Vi ser at vi konsoller ut brukeren er ikke logget inn hvis en kunde ikke har en bruker. Nå ønsker vi å lage en funksjon som gjør det mulig for kunder som ikke er logget inn å ha samme funksjonalitet.



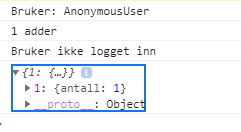
Denne funksjonen skal trigges hver gang en bruker som ikke er logget inn trykker på en av våre produkter. Vi ønsker å sende med 2 parameter som vi gjorde for funksjonen vi lagde for kunder som hadde en bruker.

Vi ønsker nå å legge til produktet brukeren har trykket på i vår handlevogn cookie. Vi gjør følgende.

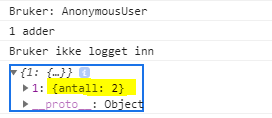


Vi sier nå, hvis handlingen er adder skal vi først sjekke om cookien vi lagde har en key som heter produkt id. Hvis den ikke har det, skal vi lage en ny key med den produkt iden vi sendte inn og sette en dictionary inne i denne keyen med antall til veriden av 1. hvis denne allerede finnes skal vi legge til 1 i antallet til dette produktet.

Dennes funksjonalitet er slik, hvis produktet brukeren trykker på ikke finnes i handlevognen(cookien vi har laget) skal vi legge den til, hvis den allerede finnes skal vi legge til 1. Hvis vi konsoller denne handlevognen ser den slik ut.

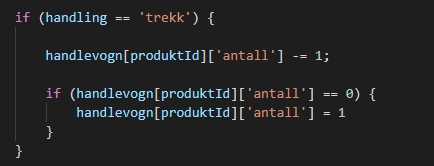


Vi ser at det blir laget en key med produktens id og en inner objekt der antallet er satt til veriden av 1. hvis vi legger til et produkt med samme iden for vi følgende konsoll.

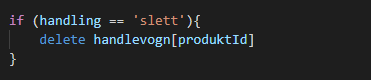


Vi ser at antallet har økt.

Nå må vi gjøre det samme for å fjerne et produkt fra handlevognen.



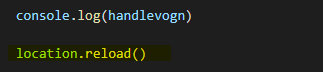
Hvis handlingen nå er trekk, skal vi trekke antallet med en og hvis dette antallet er null skal vi sette tilbake til 1. Dette gjør vi for å ikke får minus verdier. Vi gjør nå det samme for handlingen slett.



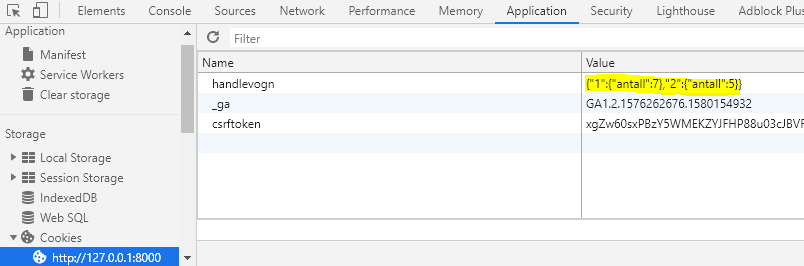
Hvis handlingen er slett, ønsker vi å slette produktet fra handlevognen. Nå gjenstår det å lagre dette som en cookie i vår funksjon. Vi legger til samme kode som vi gjorde i vår hoved.html der vi lagret cookien, men denne gangen gjør vi det inne i funksjonen. Dette gjør vi for at vi skal kunne lagre det vi gjør med cookien. Manipulere verdiene til vår cookie uten å miste det.



Vi skriver denne koden i vår funksjon for gjester. Denne lagrer cookien etter endringene vi lager til den.

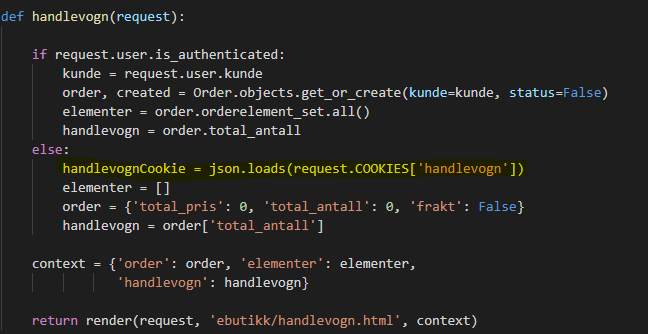


Vi ønsker å gjenopprette siden etter hver gang vi legger til noe for å kunne se endringene.

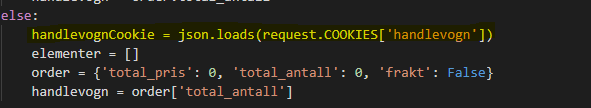


Vi ser i vår applikasjon at produktene vi legger til nå blir lagret i vår cookie. Nå må vi gjøre det mulig å vise disse verdiene i handlevognen så vi kan slette eller trekke fra elementer i ordre. Vi gjør lignende som vi har gjort før. Først trenger vi å hente ut cookiesene vi lager i vår views.py

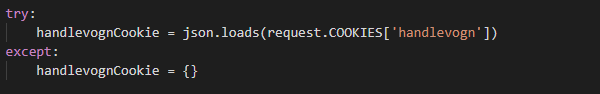
Vi går i handlevogn funksjonen og gjør følgende.



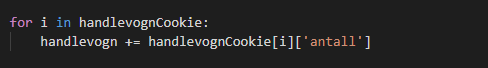
Vi lager en variabel handlevogn og henter cookies med navn handlevogn. Vi må gjøre verdien vi henter om til en python dictionary siden det vi får nå er en streng.



Når en bruker først går til vår side, vil brukeren få en error ettersom denne verdien ikke finnes. Vi må løse dette ved å gjøre en try catch funksjon.

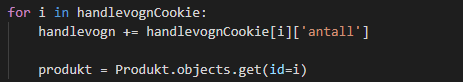


Nå har vi hentet handlevognen som en python dictionary. Vi må nå lage en loop som går gjennom hvert av antallene til dette objektet og adderer dette sammen for å vise handlevogn totalen øverst høyre hjørne.

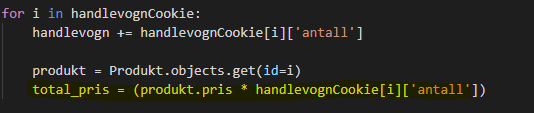


Nå lager vi en loop for hver eneste produkt i vår cookie handlevogn og legger til denne veriden til handlevogn variabelen. 

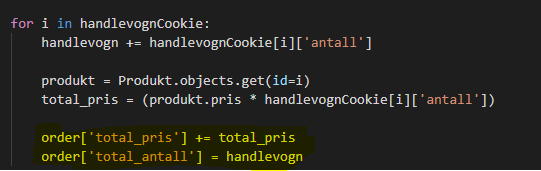
Vi ser nå at totalen oppdaterer seg i forhold til hvor mange elementer vi har i vår handlevogn cookien vår. Nå ønsker vi å vise alle produktene vi har i vår handlevogn på handlevogn siden vår. Vi må først hente hvert produkt i handlevognen, deretter hente totale prisen i cookien for å legge disse til våre kontekst variabler så de vises på siden.



Vi henter produktet ved hjelp av keyen til cookien vi har laget. Husk vi henter produktet sin id i cookien så produkt blir laget riktig. Vi henter produktet for å få tilgang til produktene pris for å finne totalprisen til ordre.



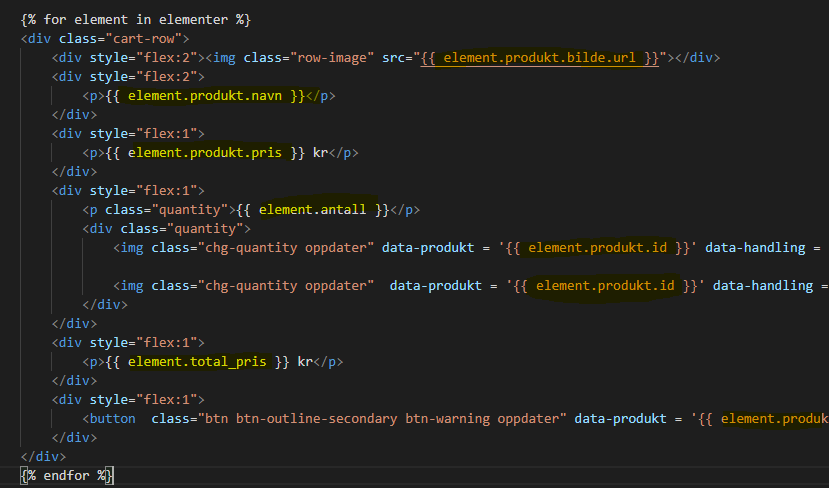
Vi lagrer total prisen ved hjelp av å gange sammen produktet sin pris med antallet av produktet det finnes i handlevognen. Nå kan vi legge disse verdiene til vårt order dictionary.



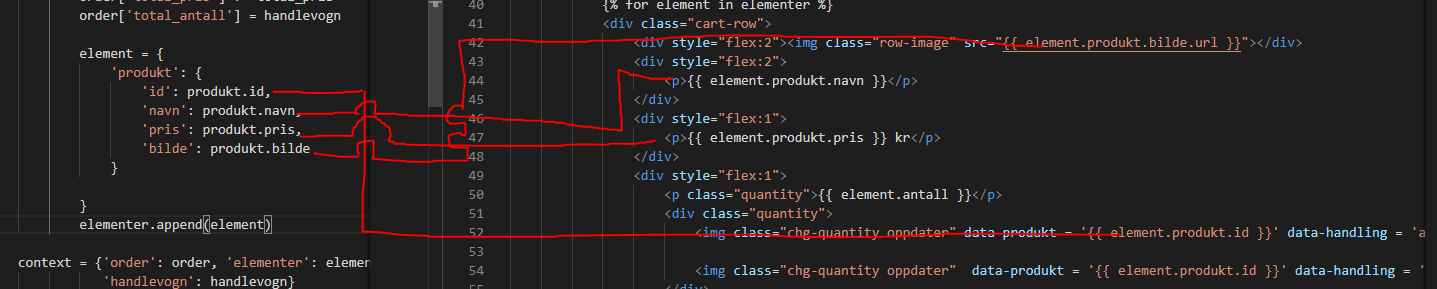
For å finne total prisen legger vi til hver total pris for alle antall elementer av et produkt. For å finne totale antall elementer av ordre henter vi var handlevogn variabelen.

For at vi skal kunne vise alle produktene i ordret må vi lage en dictionary som har de samme keyene vi leter vi har i vår elementer dictionary. Hvis vi ser hvordan vi gjorde for brukere som er logget inn, hentet vi ordre via databasen og viste dem på vår side. Vi må nå lage en dictionary som har de samme keysene som vår database. Dette må vi gjøre for hvert produkt i ordre og deretter sende denne verdien til elementer objektet vi har laget over.

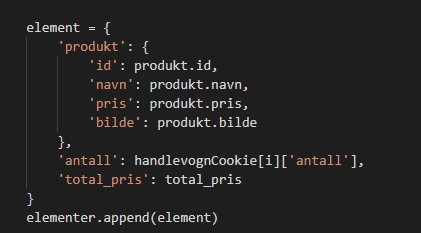
Måten vi bygger element objektet på er ved å ta utgangspunktet i hvordan vi viser et produkt i vår html side. Hvis vi ser på handlevogn.html.



Vi trenger å lage en dictionary for alle disse variablene. Dette gjøres for å kunne vise samme bilde som vi har gjort for brukere som er logget inn.

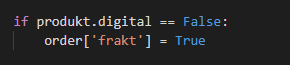


Vi lager en representasjon av hver eneste query vi vise i vår side. Grunnen til at vi lager et produkt objekt i et objekt som heter element, er for at hvis vi ser på hvordan vi viser produktene ser vi at vi først henter elementet, deretter produktet og deretter navnet. Nå må vi gjøre det samme for antall, og totalpris.



For antallet henter vi antall til iden av elementet vi looper gjennom, for totalpris bruker vi variabelen vi har laget der vi ganger sammen gjeldene produkt med dens pris. Nå vises alle elementene i ordret for en bruker som ikke er logget inn. Vi må nå håndtere om ordret inneholder et produkt som er en fysisk gjenstand.

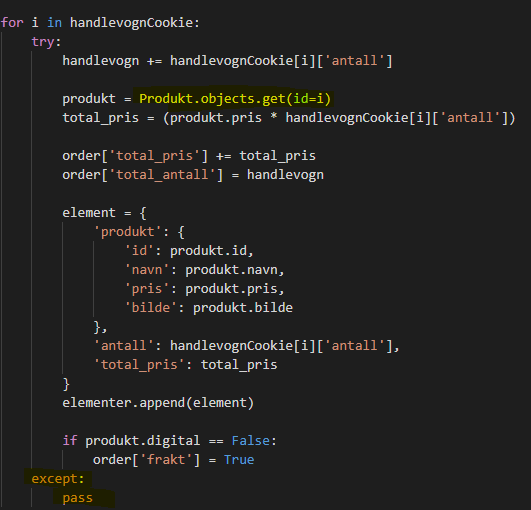
Hvis ordre har et produkt som er fysisk ønsker vi å sette frakt til å være sann, hvis det ordre bare inneholder produkter som er digitale ønsker vi ikke å sette frakt til å være sann.



Vi ønsker å setter ordre sin frak til true hvis det finnes et produkt som ikke er digitalt i ordre.

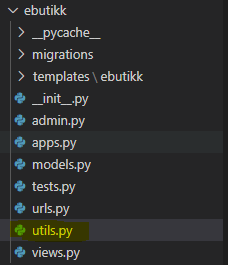
Husk denne dataen tas ikke i bruk i handlevogn viewen men i utsjekk viewen, vi skriver denne her så kan vi kopiere alt vi har gjort over til utsjekk viewen.

For å unngå en error der hvis et produkt blir slettet fra vår database, må vi lage en try pass kode for all logikken vi nettopp gjorde.

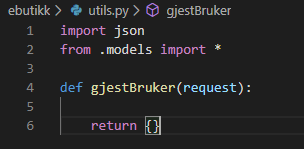


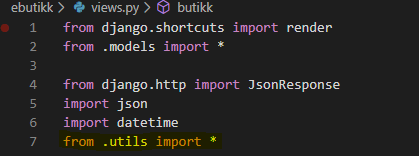
Hvis et produkt hadde blitt slettet fra vår database ville produkt variabelen ikke funnet iden til tilhørende produkt, dette ville produsert en error. Får å unngå dette lager vi en try exept der hvis produkt variabelen ikke finnes skal vi hoppe over dette produktet. Produktet som er slettet vil fortsatt være i våre cookies, derfor vil vi få denne erroren. Det er mulig å fikse dette på en annen måte, men jeg vil ikke legge vekt på dette akkurat nå.

Vi skal nå rydde opp all kode vi repeterer. All denne koden skal vi legge i en ny fil som vi lager, denne skal hete utils.py. vi skal i første omgang lage en funksjon som inneholder all kode for håndtering av gjest brukere. Nå vi har laget den som en funksjon må vi først importere filen vi lagde og deretter kalle funksjonen der den trengs. Vi lager først utils.py.

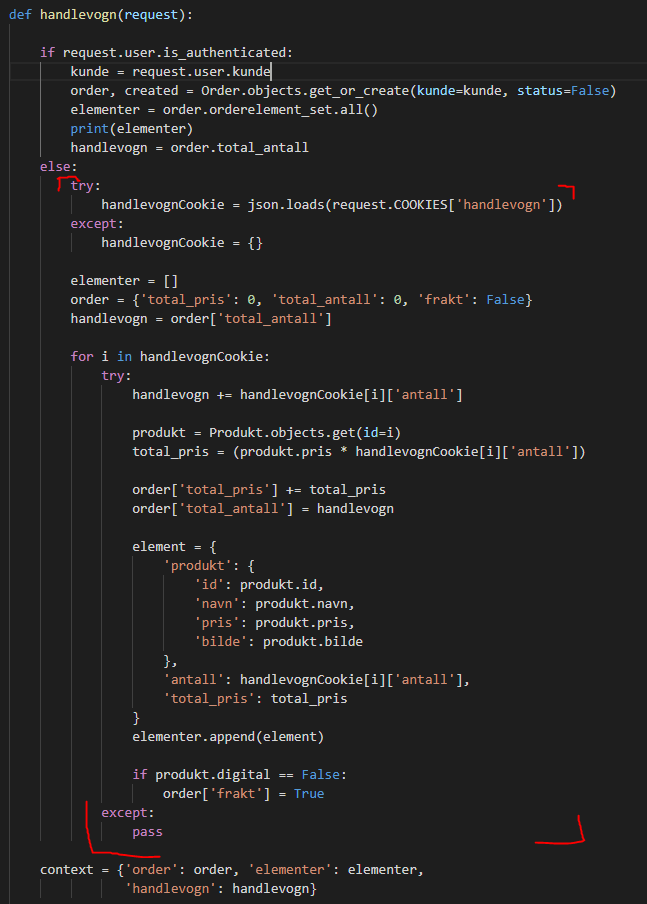


Vi lager denne filen i samme filplassering som alle andre filene vi bruker for vår applikasjon. Vi importerer models i den ny filen, vi trenger full tilgang til alle modellene og vi importerer også json.

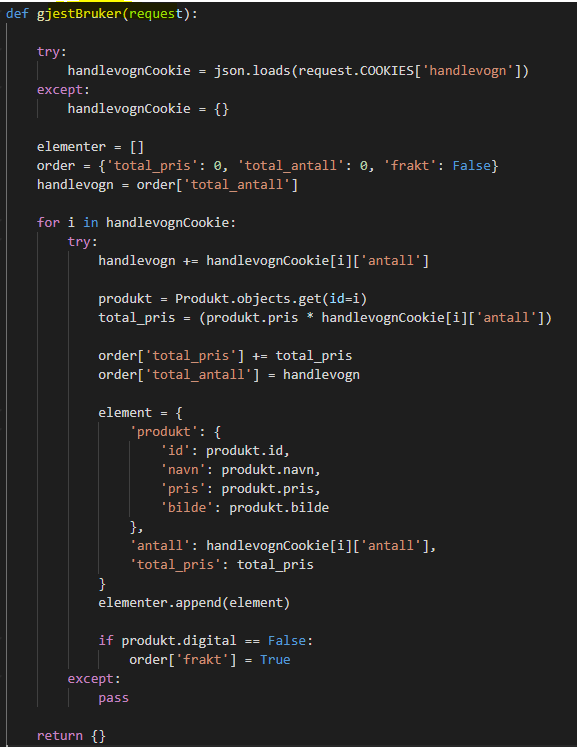


Vi lager en ny funksjon som heter gjest bruker, denne skal inneholde all koden vi trenger for å håndtere en gjest bruker. Det betyr håndtering av handlevogn funksjonalitet for en bruker som ikke er logget inn. Vi må nå importere denne filen for å få tilgang til dens funksjoner. Vi importerer den i views.py filen vår. 

Nå skal vi flytte all koden vi gjorde for en gjest bruker over til denne filen. Dette gjør vi for å rydde opp i koden så vi ikke trenger å repirere denne logikken for hver view. Vi skal også rydde opp i flere kode blokker senere.

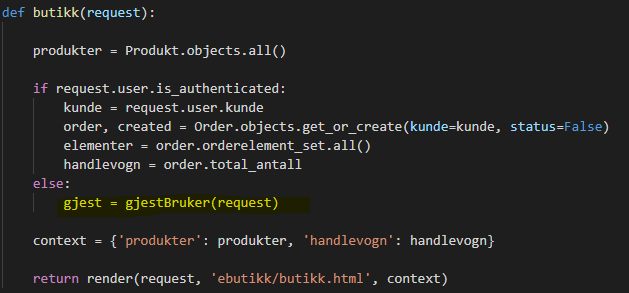


Vi klipper ut all denne koden og limer den i vår nye fil vi har laget.

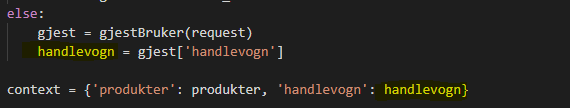


Nå ønsker vi å returnere samme verdiene vi har i vår kontekst. Dette gjøres for å gi alle verdiene konteksten forventer å få. Konteksten gjør det mulig å vise produkter i handlevogn osv i vår html side. Dette skal etterspørre i vår view.py fil. Vi skriver følgende.

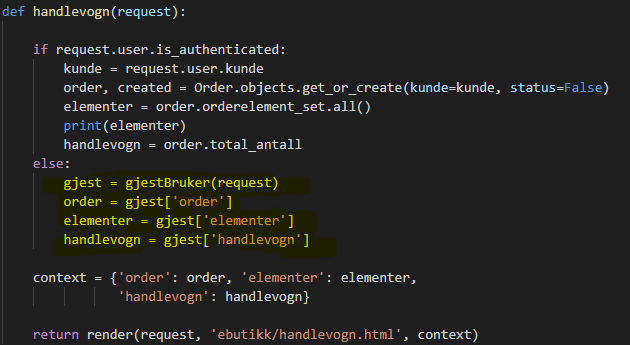


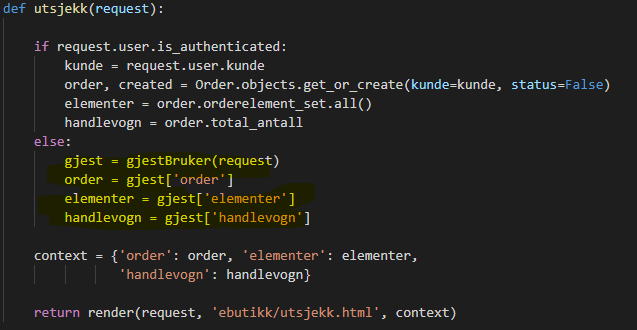
Nå beveger vi oss til vår butikk view. Her ser vi at vi trenger å bare antall elementer i handlevognen. Siden vi i vår gjestBruker() funkjson returnerer vi en dictionary, vi kan få tilgang til denne ved å skrive følgende kode i vår views.py butikk funksjon. 

Nå har vi tilgang til retur verdien til funkjsonnen vi lagde. Vi husker at vi returnerer en dictionary med forskjellige keys. Vi ser i vår butikk view vi trenger handlevogn keyen. Vi henter den slik.



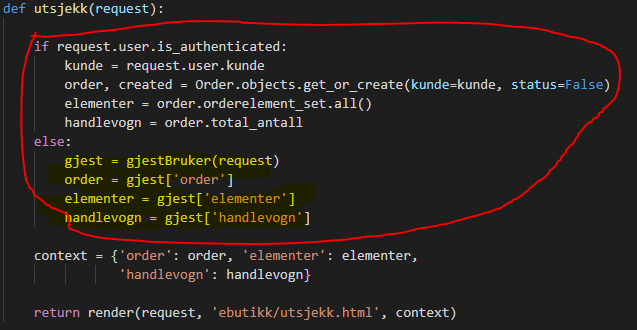
Nå har vi tilgang til antall produkter en gjest har i sin handlevogn i vår butikk view. Nå gjenstår det å gjøre tilsvarende for de 2 siste viewene våre. I handlevogn viewen trenger vi å hente alle retur verdiene. Vi henter først retur verdiene og deretter angir riktig verdi for de forskjellige variabelene.



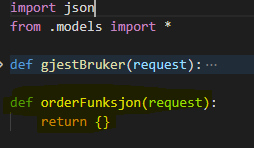
Vi henter først order, og vi husker at vi returnerer en python dictionary i vår funksjon og derfor får vi tilgang til order keyen slik. Order inneholder totalverdiene til et order, det betyr totalsummen til ordre og total antall av elementer til ordre. Elementer gir oss prisen til de produktene som er i elementene og antallet av et produkt som er i ordre. Handlevogn varabelene gir oss totalen av antall elementer i ordre som vi bruker øverst i høyre hjørne for å angi antallet av elementer i handlevognen. Vi beveger oss videre til utsjekk viewen vi har. Der må vi gjøre det samme som vi gjorde i handlevogn viewen siden vi trenger tilgang til alle tre verider. 

Nå ser vi at vi har tilgang til samme funksjonalitet som for kunder med en bruker. Vi kan legge til elementer i handlevognene og se disse elementene bli vist i handlevogn.html. vi ser også at handlevognen oppdaterer seg i forhold til hvor mange produkter vi har i vår handle vogn. En oppsummering av ordre er også vist på utsjekk.html siden vi lagde. Vi ser også at leveringsadresse feltet forsvinner hvis vi har et order med bare digitale elementer. Dette er fordi vi finner frakten i vår logikk ved å loope gjennom alle produktene og se om de har verdiene av digital til False. Dette betyr at hvis en av produktene i vår gjest bruker sin handlevogn har et fysisk element vil frakt være True og dermed vil leveringsadressen vises.

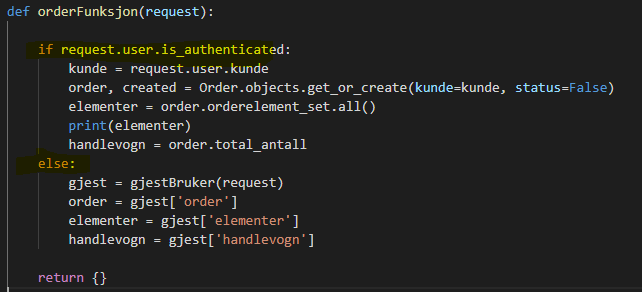
Det bi ønsker å gjøre nå er å gjøre koden mye mer ryddigere. Vi ønsker å legge all kode som vi har fra if til else og legge den i en funkjson i utils.py.



All denne koden ønsker vi å legge til en ny funksjon i utils.py. Det første vi gjør er å lage denne funksjonen i utils.py.



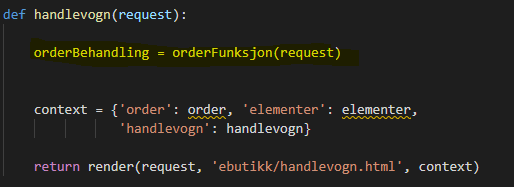
Denne funksjonen skal håndtere vår if else kode. Dette betyr vi trenger bare å tenke på retur verdiene ti vår views.py fil. Vi lager denne under gjestebruker funksjonen fordi vi må sjekke om en bruker er gjest først. Vi legger til en request i funksjonen fordi vi vil ha tilgang til request.user osv. Vi hadde ikke fått tilgang til denne hvis vi ikke hadde tatt med request i vår funksjon. Vi henter all koden i vår handlevogn view og legger den i vår nye funksjon vi lagde.



Vi trenger nå å returnere riktige verdier så vi har tilgang til disse i vår view.



Nå må vi få tilgang til disse verdiene og ansette riktig verdig for hver variabel i vår views.py.



Henter først tilgang til retur verdenen.

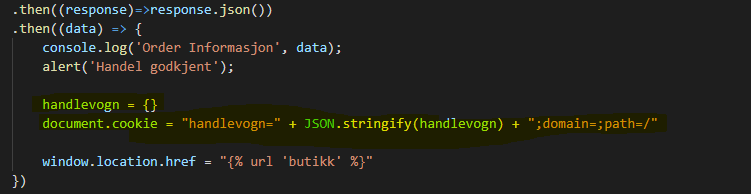


Ansetter riktig verdi for hver variabel som vi gir inn i konteksten. Det som skjer nå, er at vi kjører logikken vi hadde i to forskjellige funksjoner. Den første funksjonen er for gjest brukere. Hvis en person ikke er logget inn vil vi order funksjonen kalle gå til vår else statement. Der kaller vi på fiksjonen som håndterer brukere som ikke er logget inn. Dette funksjonen returnerer verdier som brukere som ikke er logget inn. Der er kan en gjest legge til elementer eller fjerne elementer til sin order. Dette får han gjort fordi vi har laget en JavaScript som sender cookie representasjon av et order. Denne cookien får vi tilgang til i gjest bruker funksjonen. Vi åpner denne cookien opp og ansetter riktig verider til variabler. Disse variabelen returnerer vi, de returnerte variabelen blir fanget opp av order funksjonen. Igjen blir verdenen ansatt til riktige variabler og returnert. Denne retur verdien blir da igjen fanget opp av vår views.py fil der vi igjen ansetter riktig verdi til variabler. Hvis en bruker er logget inn vil vi ikke kalle på gjest bruker funksjonen, vi vil bare hente ut de forskjellige verdiene som er tilknyttet ordre og gjellende kunde.

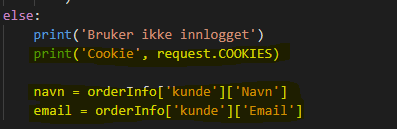
Vi gjør nå det samme vi gjorde for handlevogn viewen, der vi erstatter all logikken ved å kalle på order funksjonen og hente ut de riktige verdiene.

***Nå har vi forkortet koden vår, håntering av gjest bruker og bruker som er innlogget gjøres nå på utils.py. Og disse funksjonene kaller vi til i vår view.***

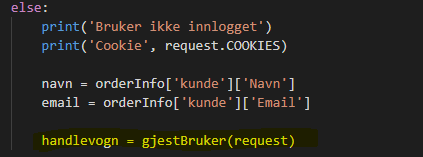
Det vi skal gjøre nå, er behandle ordre for en gjest bruker. Dette skal skje på samme måte som vi gjorde før. Vi skal sende data via json til vår backend og behandle dataen der. Siden brukeren ikke er logget inn, vil vi nå kreve navn, email og hvis gjesten har et produkt som skal fraktes skal vi også sende leveringsadresse infromasjon. Vi må forandre litt på koden vår i utsjekk.html. Vi vil at ved klikk på godkjenn betaling skal handlevogn cookien tømmes. Vi gjør følgende.



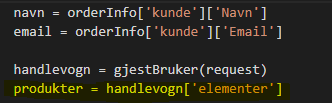
Vi tømmer først handlevogn cookien og deretter lagrer den. Det gjør vi ved den siste koden document.cookie, derfor må vi tømme den først og deretter lagre den. Husk når vi trykker på godkjenn betaling vil vi trigge godkjenn order funksjonen, den vil da ta all informasjonen i feltene og sende den til vår backend. Som koden står nå, vil ingen informasjon vi sender inn bli lagret fordi vi ikke har skrevet noe på else statementen vår i vår view for godkjenning av order, det eneste vi gjør er å printe ut at brukeren ikke er logget inn. Vi må nå skrive kode som håndtere disse dataene og lagrer det i vår database. Vi sender dataen til backenden men vi lagrer ingen av feltene siden brukeren ikke er logget inn, dette må vi gjør nå.



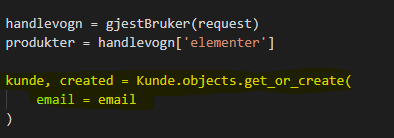
Vi printer først ut cookien som ble sendt med, og deretter henter vi navnet og emailen. Husk vi sendte kunde informasjon ved å lage en dictionary. For å hente navn og email må vi inn i kunde keyen og navn og email veriden til denne dictionaryen. For å hente produktene i handlevognen har vi allerde laget en funksjon som returnerer disse veridene. Vi kaller på denne funksjonen og sett ting på riktig plass.



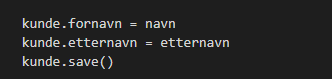
Henter først ut all informasjonen om brukeren. Deretter henter vi alle elementene gjest brukeren har i sin handlevogn.



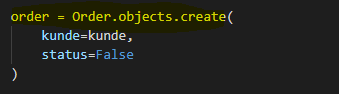
Nå har vi hentet alle produktene gjest brukeren har i sin handlevogn. Siden vi lagret alle order elementene i retur verdien elementer henter vi de slik. Nå må vi lage en bruker for gjesten som handler.



Ved bruk av get or create for vi lagret brukeren ved email. Dette betyr at hvis en kunde som er gjest bruker den samme emailen som han har brukt før, vil ordre registreres på hans bruker. Navn feltet lagrer vi ikke, vi bruker bare email. Hvis kunden handler som gjest gjentatte ganger vil vi kunne lagre dette ved å tilknytte email til hans order. Vi kan dermed se hvor mange ganger en gjest bruker har handlet hos oss. Vi ønsker nå å gi kunden et for navn og etternavn.



Dette betyr at hvis kunden vil bruke et annet navn eller etternavn så lager vi ikke en ny bruker, men hvis emailen er det samme vil ordre registerets på kundens email. Gjest brukeren kan ha flere navn men siden vi lagrer kunden i vår database basert på email kan gjest brukeren ha flere navn. Nå må vi lage et order tilknyttet til denne brukeren.



Nå lager vi et order tilknyttet den til kunde med status FALSE. Vi trenger nå en loop som går gjennom alle produktene i handlevognen som er lager i cookien, og vi lagrer dette i en variabel.



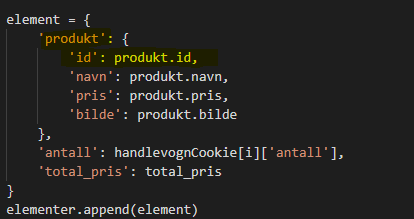
Vi husker at vi lagret alle elementene i handlevognen i en variabel produkter ved å kalle på cookie funksjonen.



Her henter vi cookien ved å kalle på funksjonen som omgjør alle elementen i cookien til en dictionary, vi får tilbake dette.

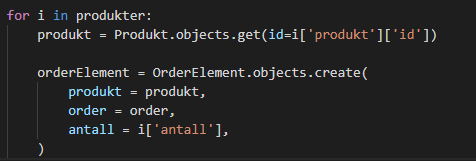


Vi husker at vi sendte elementene i handlevognene slik:

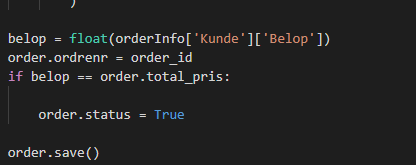


For å finne iden til produktet må vi dermed først gå inn i produkt sin dictionary og dertter finne dens id. Derfor har vi skrevet koden over på den måten. Nå må vi lagre en rekke med order elementer for vårt order i vår database. Dette gjør vi inne i loopen fordi for hvert produkt det finnes i elementene som vi sender ved, må vi lage et order element objekt for dem.

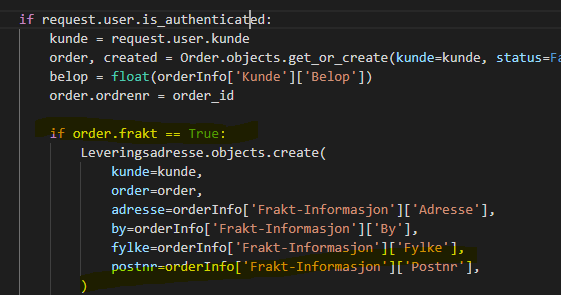
Hvis vi husker tilbake til vår models for order elementer, trenger vi 3 verdier for å lage et order element. Vi trenger produkt, order og antall. Disse må vi nå sette i vår loop.



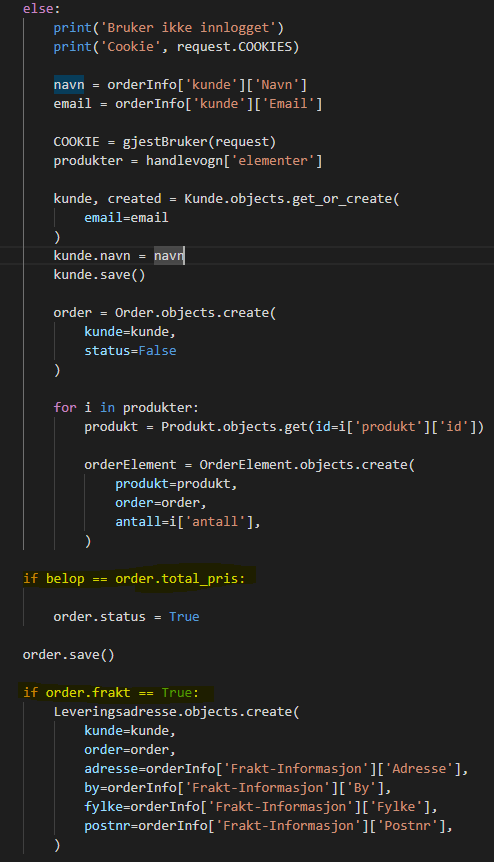
Vi setter produkt til å være det produkt variabelt vi nettopp lagde, vi fant produktet ved å finne iden til produktet. Vi setter order til å være det ordre vi nettopp lagde og antallet får vi tilgang til ved å gå i elementer sin antall key. Vi vil nå sjekke om totalen vi fikk in i json format er lik totalen vi har lagret i vårt order. Vi vet denne totalen fordi vi lagrer ordre og order element i vår database før vi sjekker om beløpene er like. For å ikke gjenta kode klipper vi ut logikken vi har laget for brukere som er logget inn og limer den inn helt under så begge deler har tilgang til denne logikken



Vi legget denne kodeblokken på bunnen av if else logikken. Det samme gjør vi nå for fraktinformasjon. Vi klipper ut frakt informasjon for brukere som er logget inn og limer det inn helt på bunnen av if else for at begge skal få tilgang til denne logikken.



Limer denne inn på bunnen, blir sende slik ut.

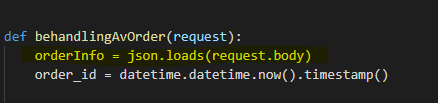


etternavn = order[‘kunde’][‘Etternavn’]

Kunde.fornavn = navn

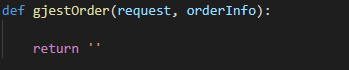
Kunde.etternavn = etternavn

Vi har tilgang til feltene fordi vi deklarerer orderInfo før if else logikken.

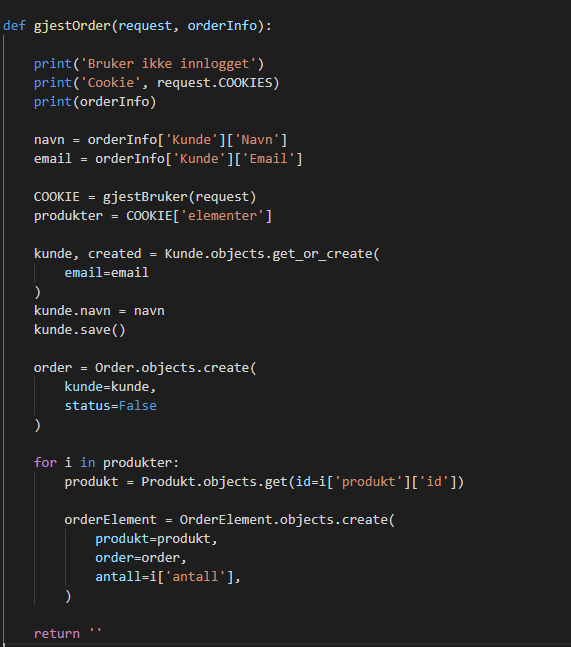
¨

Nå ønsker vi å putte all funksjonalitet for behandling av et gjest order i en funksjon i utils.py. så Når vi skal behandle ordre for gjester kaller vi på denne funksjonen som gjør jobben for oss. Dermed blir koden mindre og mer oversiktbar. For denne funksjonen trenger vi å sende inn order info variabelene som vi får i vår behandling av order funkjsonen. Så funksjonen må forvente et parameter for order info.

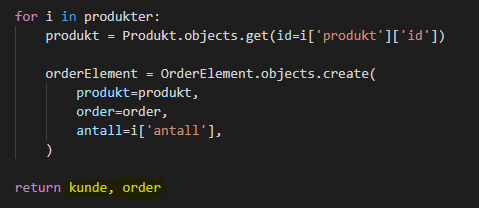
Vi lager først en funksjon i utils.py.



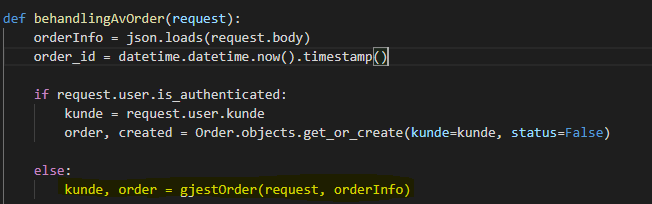
I vår view har vi importert alle funksjonene i utils.py så vi har tilgang til denne funksjonen også. Vi klipper ut all koden fra else i vår views.py behandlig av order funkjsonen og limer den inn i gjest order i utils.py.



For å kunne behandle order fra en gjest bruker trenger vi å returnere ordre og kunde verdien i vår funksjon.



Nå kan vi hente disse verdiene ved å kalle til funksjonen. Siden vi har lagret all informasjonen om ordre og kunde i linjen orderInfo, må vi mate funksjonen disse verdiene først, deretter vil funksjonen behandle disse verdiene, dette gjøres i order behandling i vår view.



Vi henter requesten og dertter mater inn dataen vi skal ha håndtert. Denne dataen lager vi i linjen over if logikken i orderinfo.

Der order informasjon blir hentet ved hjelp av en json request, denne requesten lagrer vi i order info. Dette mater vi inn i funksjonen, den vil da behandle disse datene og detter sette kunde til å være kunde retur verdien og det samme for order veriden. Merk hvordan vi deklarer våre variabler, vi returnerer kunde så order i vår funksjon derfor må vi deklarere kunde og order for at disse skal ha riktig verdi.

## Nå Har gjort koden vår enklere og laget funksjonalitet for en gjest bruker.

Vi har nå laget funksjonalitet for en gjest bruker. Dette betyr at en gjest bruker kan bestille produkter på vår side. Når gjest brukeren har lagt inn et order, lagrer vi informasjonen om ordre og informasjonen om kunden. Vi sender dette til vår behandling av order funksjon i vews.py. Der blir dataene håndtert og dretter lagret i vår database.

Datene blir send inn i behandling av order viewen via json format. Der lagrer vi dataene i orderInfo. Deretter har vi laget en else funkjson for gjester som ikke er logget inn. informasjon om kunden samt totalbeløpet blir lagret i orderinfo variabelen. Denne sender vi til gjest order funksjonen. Der setter vi kundes navn, etternavn og email. Vi lagrer brukeren via emailen brukeren skrev inn i email feltet.

Dertter lager vi et order i vår database tilknyttet til denne brukeren vi har laget med status false.

Deretter vet vi at ordre gjest brukeren lager blir lagret i en cookie. Vi har laget funksjonalitet for denne koden. Når gjest brukeren trykker på legg til i handlevogn på en av produktene blir handlingen og produkt iden trigget ved hjelp av en addevent listner. Vi lagde funksjonalitet for sletting og trekking av produkter også. For hver gang gjest brukeren trykker på en av handlingene blir cookien oppdatert og lagret deretter blir siden forfrisket. Vi har laget en funksjon gjest bruker som leser og setter dataene til cookien i en dictionary.

Denne funksjonen kaller vi på i vår gjest order funksjon, da har vi fått tilgang til kundens order ved hjelp av gjest order funksjonen. Deretter setter looper vi gjennom elementene i cookiene og lagrer produktet ved hjelp av iden vi sendte med i cookien. Nå som vi har produktet lager vi et order element i vår database med produktet vi nettopp fant, fortsatt i loopen. Ordre har vi laget ved å tilknytte kunden denne legger vi til i det order elementet vi lager. Vi lagrer antallet på samme måte som vi finner produktens id. etter dette returnerer vi kunden og ordre.

I vår view håndterer vi dette videre. Vi sjekker om beløpet vi fikk inn er likt beløpet vi har i vår order vi laget, hvis det er det, lagrer vi ordre i vår database. Deretter sjekker vi om ordre vi behandler behøver frakt, hvis det gjør det lager vi en leveringadresse i vår database med adresse, postnr osv.

Paypal betalings intergrasjon

Vi skal nå bygge videre på vårt prosjekt. For å kunne godkjenne betalinger og for å kunne kjøpe produkter må våre kunder ha tilgang til å kunne betale på paypal eller med kredit kort. Vi skal nå utvikle vårt prosjekt og integrere paypal betalings muligheter. Kunder skal ha mulighet til å betale med paypal brukeren sin eller bruke et kredit kort.

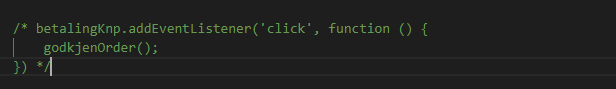
<https://developer.paypal.com/demo/checkout/#/pattern/client> gir oss tilgang til å kunne integrere dette i vår nettside. På siden står det hva slags kode vi skal bruke. Vestre på siden er det en integrasjon felt. Vi skal i denne utviklingen jobbe med Client side Integration. Dermed velger vi denne koden. vi skal sette opp en sandbox bruker for vår paypal som betyr vi skal lage to brukere, en bruker for eieren av nett butikken og en bruker for å virke som en kunde. Deretter skal vi sende falske penger fra vår kunde bruker til butikken og se om pengene har gått gjennom. Vi starter først med å legget til knappene i vår utsjekk.html.



Vi kopierer denne koden og limer den i vår utsjekk.html kode. Knappene skal vises etter brukeren trykker på fortsett dermed blir plasseringen på koden slik.



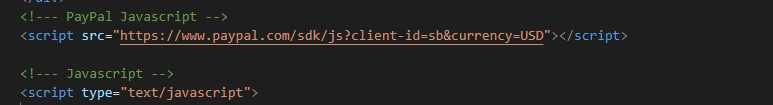
Vi kommenterer ut denne knappen siden vi ikke skal bruke den. Vi kommenter også ut addeventlistneren for betalingsknappen.



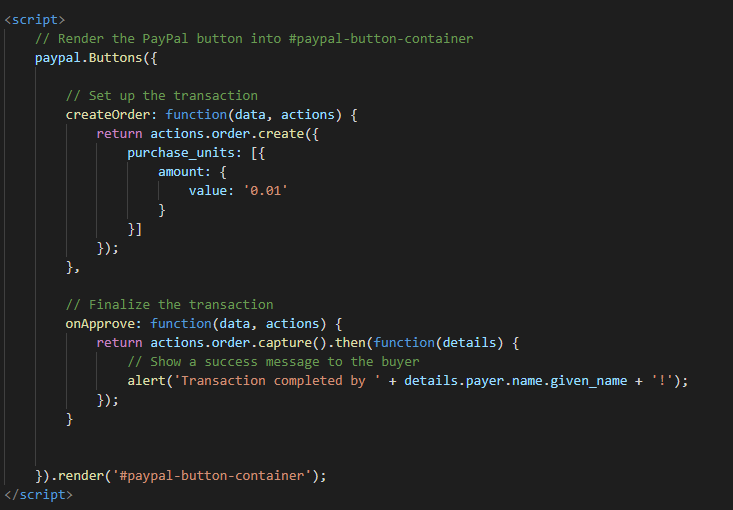
Vi må nå legge til koden som gir oss tilgang til paypal sin api.



Denne koden legger vi til rett over vår originale javascript kode i utsjekk.html.

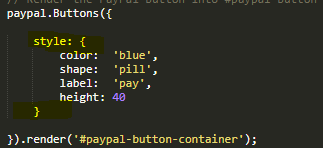


Nå kopierer vi koden vi fikk fra paypal for å få alt til å fungere. Vi kopierer koden fra linje 17 til linje 42. denne koden legger vi i sin egen script block.

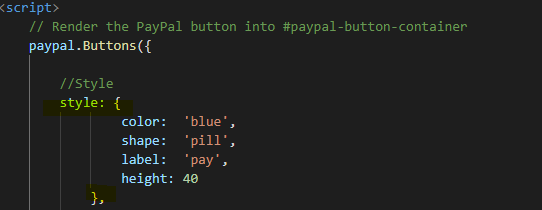


Det som skjer i denne kode blokken er at paypal har laget en kode som lager knapper. Disse knappene blir først hentet når paypal.Buttons blir kalt. Hver knapp har sin funksjonalitet derfor ser vi en funksjon inne i knappen. Funksjonalitetene til knappene blir først skrevet og deretter blir de renderet/vist i vår div kontainter vi lagdte for å holde alle paypal knappene. Vi har to metoder for disse knappene. Create Order kjører paypal expirience metode. Value er hvor hva brukeren skal betale vi skal gjøre dette dynamsik i forhold til kundens order. Den åpner et nytt vindu der brukeren kan logge in til sin paypal bruker og godkjenne betaling. I onApprove metoden blir en alert sendt til kunden om at tranaksjonen er vellykket. Dette skal vi forandre på senere.

Knappene er nå synlige på vår side. Vi har en knapp for paypal og en knapp for kredit kort. Vi har også envnen til å kunne designe disse kanppen. <https://developer.paypal.com/demo/checkout/#/pattern/client> venstre på denne siden på features ser vi style. Vi ønsker å ha våre knapper på pille form så vi kopierer koden og sett inn i vår utsjekk.html.

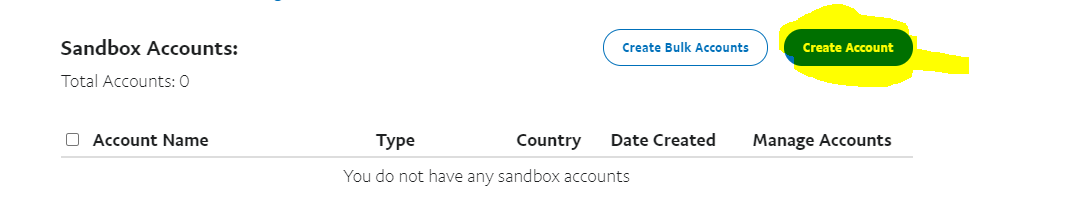
¨

For å kunne designe knappene må vi skrive style i vår javascript kode etter vi har skrevet paypal.buttons.

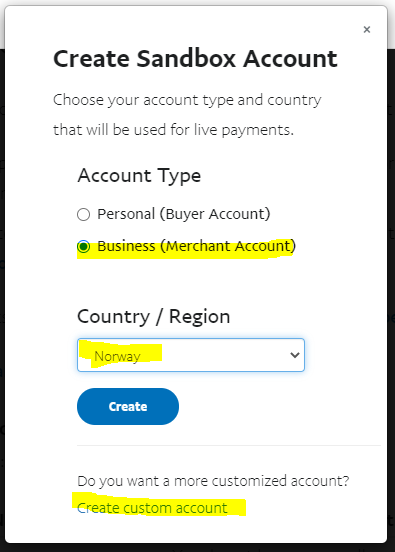


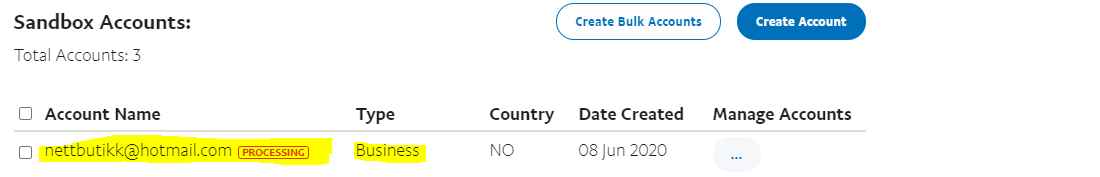
Siden datamaskinen tolker dette som et javascript objekt må vi putte et komme på slutten. Nå har vi lagt til litt av vår egen design på knappene vi gjør synlig. På <https://developer.paypal.com> finnes det mer doukumentasjon på hvordan vi kan designe knappene og flere endrings muligheter.

Vi må nå lage to sandbox brukere for vår nettside. Vi må lage en bruker som representerer eiern av nettbutikken og en bruker som representerer en kunde som har kommet for å handle. Vi går inn på <https://developer.paypal.com/developer/accounts/> og lager to brukere. For å kunne lage sandbox brukere må vi også ha en paypal bruker.



Trykker på create Account.

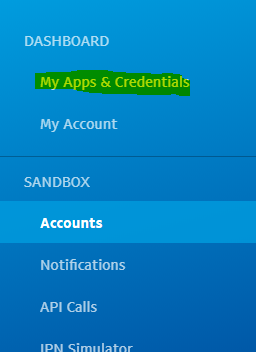


For å lage bruker for eier av nettbutikk velger vi buissnes, vi velger deretter norge og vi trykker helt under for å lage en tilpasset bruker. Vi fyller ut informasjonen og lager en bruker med en balanse på 50 000 kr. 

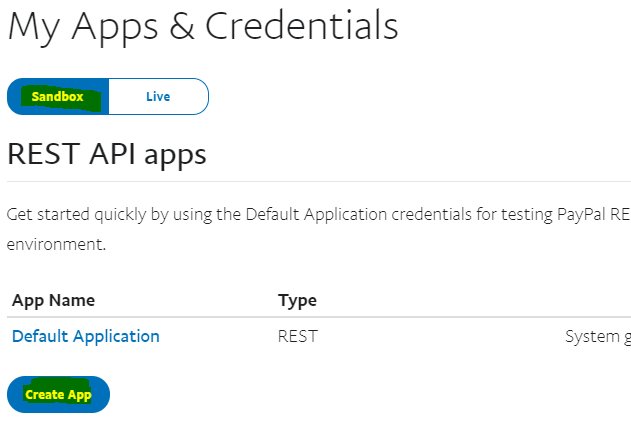
Nå har vi laget en bruker for der pengene skal sendes til. Vi gjør nå det samme men velger personal bruker.



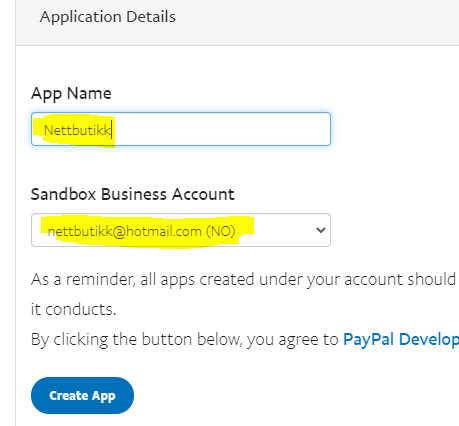
Nå har vi laget to brukere, en for kunde og en for eier av nettsiden. Personlige brukeren sin penge balanse satt jeg til å være 25 000kr. Nå må vi lage en app som genererer en link der pengene skal sendes. Appen må vi linke opp til nettbutikk eieren. Vi skal etter hvert gjøre en handel og se om den går gjennom.



Vi går inn i denne linken her og lager en app.



Vi velger sandbox og trykker på create app.



Vi setter navnet til å være nettbutikk og linker appen opp mot brukeren vi lagte for eieren av nettbutikken. Dette betyr at når vi linker opp denne appen i vår kode, vil pengene bli sendt til denne brukeren som er tilknyttet til appen.



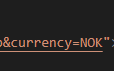
Nå som vi har laget appen har vi fått tilbake en client id. Denne må vi legge til i vår javascript kode. Når denne er lagt til i koden vår har vi koblet appen opp mot vårt nettsted. Vi kopierer client iden og limer den inn i vår script link vi lagde for paypal.



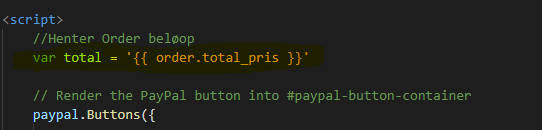
Vi skal er erstatte der det står id = med vår client id. iden blir senendes slik ut.



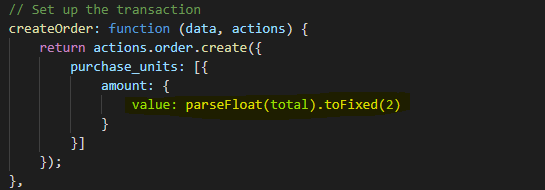
Client iden er unik for hver person som lager brukeren så vær sikker på at du har skrevet inn riktig client id. vi forandrer currency også til NOK



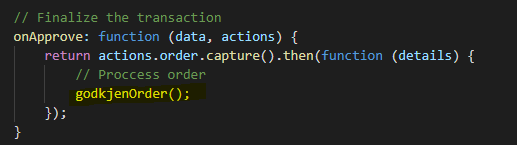
Instedet for at totalbeløpet skal være en øre skal vi nå gjøre dette mer dynamsik. Vi ønsker at totalbeløpet skal være totalen til ordre til brukeren som handler. Vi deklarer først en variabel som henter dette beløpet.



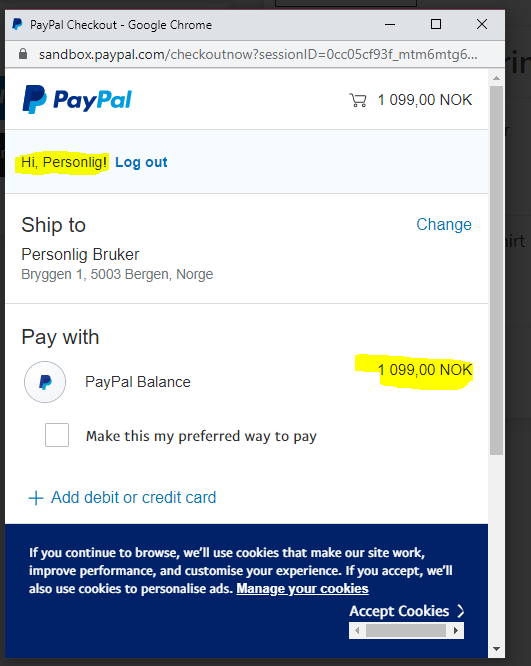
Vi henter den ved hjelp av vårt order. Vi går inn på funkjsonen vi har laget for å hente order totalen og lagrer det på variabelen total. Denne er nå lagret som en streng siden vi henter den ved hjelp av python. Før vi setter verdien må gjøre strengen om til en float og deretter gi den to desimal plasser.



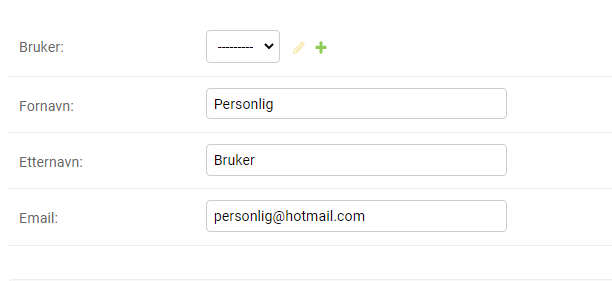
Nå har vi gjort paypal beløp veriden dynamisk i forhold til order totalen. Nå en bruker nå godkjenner sin betaling og betaler for produktene han har kjøpt, lager vi en alert som beskriver dette. Vi ønsker nå for brukere som har godkjent sitt order og alt er som det skal og lagre deres informasjon i vår database. Så for en bruker som kjøper et produkt og går gjennom paypal metoden og betaler for produketet ønsker vi å lagre deres kunde informasjon og fraktinformasjon hvis detter nødvendig. Vi har laget logikk for lagring av kunde informasjon og frakt informasjon vi må nå bare kalle til denne metoden og denne vil gjøre jobben.



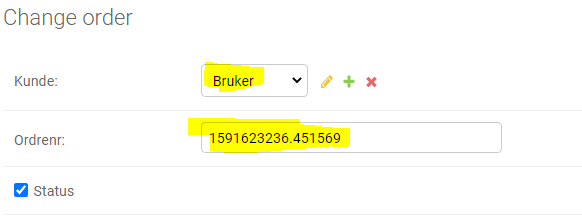
Vi husker at funksjonen som lagrer ordre navnet vi godkjen order. Vi kaller på denne funksjonen etter betalingen er godtatt. Vi skal nå teste om alt fungerer som det skal. Vi leger til noen produkter til vår handlevogn og godkjenner betalingen med den personlige sandbox brukeren vi lagte.



Vi ser at order totalen er på 1099 kr, vi trykker på pay now. Vi ser at alerten kommer opp som det skal. Vi sjekker vår database for om dataene er lagret.



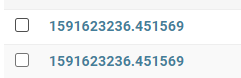
Kunden ble lagret i vår database. La oss nå sjekke order.



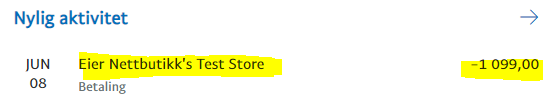
Ordre ble også lagret i vår database.



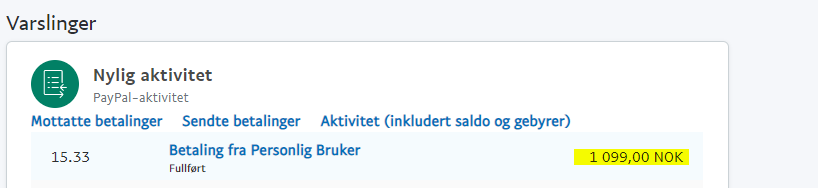
Siden vi hadde produkter som måtte fraktes har vi også lagret leveringsadressen i vår database.



Ordre elementene har også blitt lagret. Nå som vi har kjøpt disse produktene på vår side kan det ta opp til 15 min før vi ser pengene gå ut av vår personlige konto og inn på eieren sin konto. Si sjekker om et beløp på 1099 kr har blitt overført.

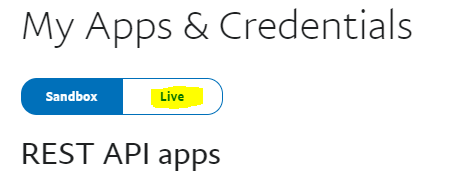


Vi ser at pengene har gått ut fra vår personlige konto. Logger nå inn på eier sin konto og ser om pengene har kommet inn.



Vi ser nå at pengene har blitt godkjent.

For gjøre dette med en ordentlig nettside og ikke sandbox må vi erstatte client iden fra sandbox til en live id. Dette gjør vi ved å lage en app og velge live i stedet for sandbox.



Deretter kopierer vi client iden og erstatter den med sandbox client iden.

## Paypal metode intergret for vår nettside.

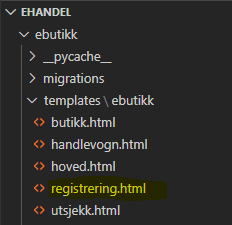
Oppsumering.

Vi ønsket i denne delen å intergrere paypal betalingsmetode i vår nettside. Vi gjorde dette ved hjelp av koden vi får av paypal. Vi la til først en script som ga oss tilgang til paypal sin api, deretter hentet vi koden som la til knapper og funksjonalitet for knappene. Deretter designet vi litt på knappene og la til en variabel total. Denne representerer totalen til ordre til handleren av produktene. Deretter la vi til å egen funksjon som vi lagde som lagret data fra kjøperen. Vi kjørte denne funkjsonen når betalingen var godkjent og lagret dataene kjøperen skrev inn i vår database.

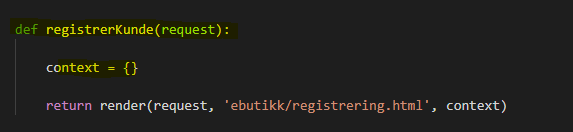
Vi testet så om dette funket.

Login og registrering

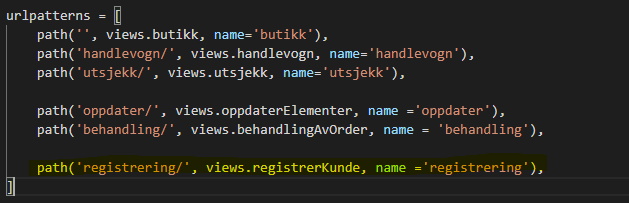
I dette segmentet skal vi fokusere på å lage et login og registering skjema. Vi ønsker at en bruker skal kunne lage en bruker som skal være registrert i vår database. Vi ønsker også at brukeren skal kunne logge inn. Vi starter først med registrerings skjema. Lager en registrering.html i vår templates.



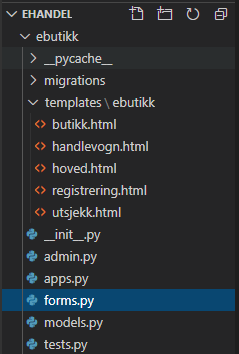
Nå må koble denne opp mot en view.



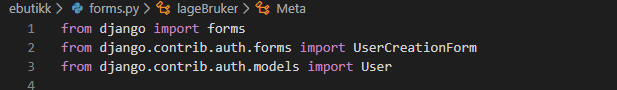
Detter må vi koble viewen opp mot en url i vår applikasjon.



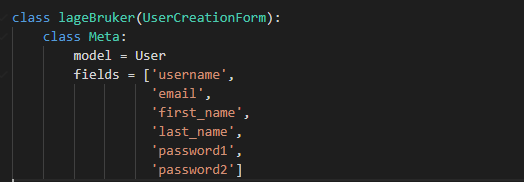
Django har en metode som håndterer lagring av brukere i vår database. Denne må vi importere. Dette er kjempefint fordi da trenger vi ikke å tenke på å lage if else for hvis passordene ikke matcher eller om brukeren allerede er registret. For at vi skal kunne optimalisere denne registrerings metoden må vi ha tilgang til å kunne redigere på den. Derfor lager vi vår egen metode men vi arver også metoden vi får av django. Vi lager først en python fil som heter forms.py i vår applikasjon.



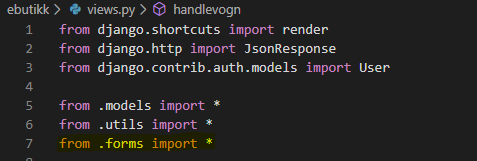
Her må vi gjøre følgende import for å kunne håndtere registering av brukere.



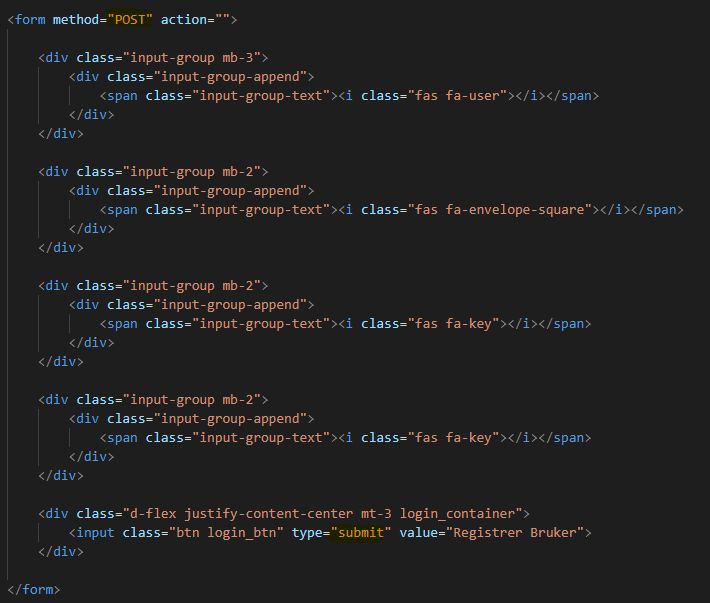
Vi importerer forms fra django for å kunne lage en egendefinert form. UserCreationForm er formen vi skal knytte opp mot vår egen form for å kunne lagre brukere i vår database. Vi importer User også fordi formen skal registrere brukere. For å kunne spesifisere hvilken type form vi lager og hvilken felt vi skal tillate gjør vi følgende kode.



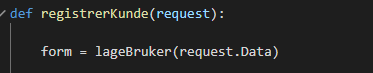
Nå spesifiserer vi at modellen skal håndtere en bruker. Og vi spesifiserer hvilken type felter vi skal tillate i formen. Nå kan vi importere denne filen i vår view og lage kode for å håndtere registering av brukere.



I viewen vi har laget for registering må vi først sjekke om vi får en POST request fra siden. Dette betyr om vi vår informasjon at vi sjekker om vi får informasjon inn av nettsiden. Vi må nå lage en form som ved sumbit sender tilbake POST data til viewen. I vår registrering.html skriver vi følgende.



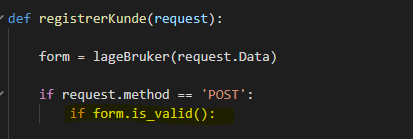
Vi har nå laget en form med metoden POST. Dette betyr at vi skal sende informasjon. Vi har et input med typen av submit helt underst. Når denne blir trykket blir all informasjonen vi har fylt ut sendt til vår view. Nå må i vår view håndtere denne informasjonen. Først må vi sjekke om metoden vi får tilbake er en POST metode.



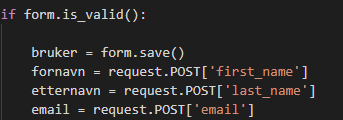
Først lagrer vi dataen vi får tilbake i variabelen form. Før vi lagrer den sett vi dataen vi år av nettsiden i vår metode som vi lagde. Metoden omgjør dataene til en form, men brukernavn osv.



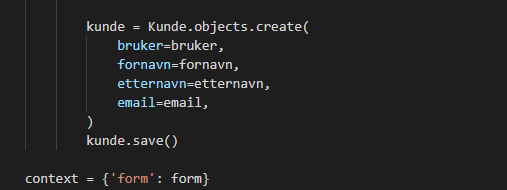
Vi sier nå, hvis dataen vi får tilbake har en metode som er POST etter, så skal vi gjøre følgende.

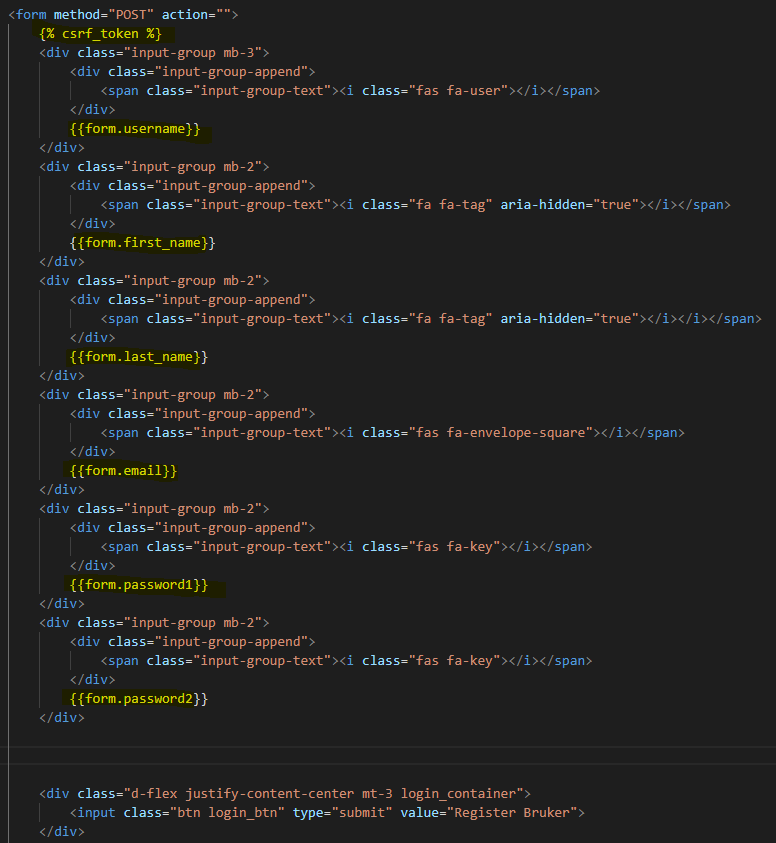


Før vi lagerer brukeren må vi se om det brukeren har skrevet inn er gyldig. Denne funksjonen sjekker om passordene stemmer og om brukeren allerede er lagret. Hvis formen vi får inn er gyldig skal vi gjøre følgende. For vi må først lagre brukeren vi vår database, så lagre fornavn etter navn og email i en variabel. Deretter må vi lage et kunde objekt som vi knytter brukeren til.

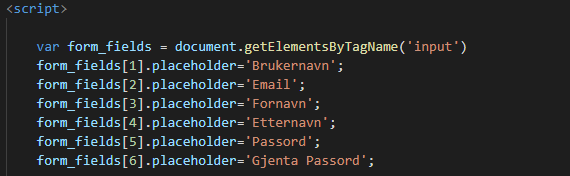


Vi lagrer brukeren ved å kalle på save funksjonen for vår data. Dataene vi har fått inn nå har blitt lagret i databasen. Vi lagrer så brukeren i varabelen bruker, så lagrer vi fornavn, og for å få fornavnet til brukeren må vi querie first name til formen vi har fått inn. Altså ut fra dataen vi har fått inn velger vi first name. Deretter gjør vi det samme for etternavn og email. Nå gjenstår det å lage et kundeobjekt som denne brukeren er koblet til.

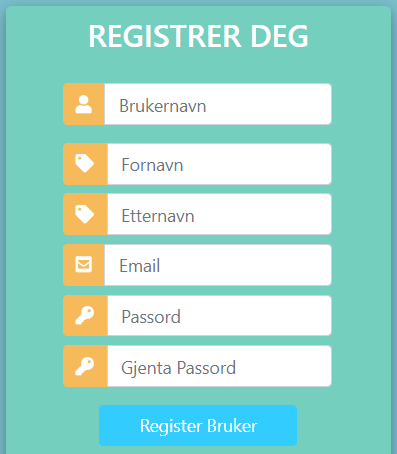


Vi setter veridene sammen og lagrer kunden ved hjelp av save funksjonen. Deretter for å vise registerings formen må vi mate den i contexten. Nå må vi redigere vår form så det blir på norsk. Før vi gjør dette lager ønsker vi å i stedet for å vise hele formen, vise alle inputene saparat. For å gjøre dette må vi gjøre følgende. 

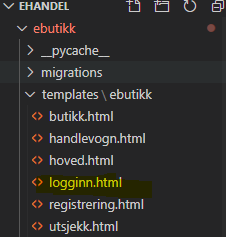
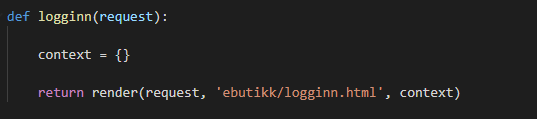
Mer at vi har en csrf token på toppen for å sende dataene gjennom riktig. Vi ser at vi har gjentit alle disse veidene vi har i vår form. Siden vi disse ikke er en html tag kan vi ikke putte en placeholder i hver input med html kode. Vi må gjøre dette i javascript. Vi lager en script tag på bunnen av vår html kode og skriver følgende.



Med cssen jeg har laget for html siden, ser dette da slik ut.

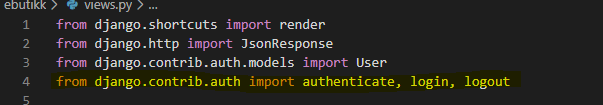


Etter bruk av litt css ser min registering side slik ut. Registreingen er nå på plass. En bruker kan registrere seg i vår nettsted og automatisk bli kunde. Før vi linker opp siden lager vi først login logikken. Vi starter først må lage et html dokument og deretter koble den opp mot view for så å koble det opp mot en url.

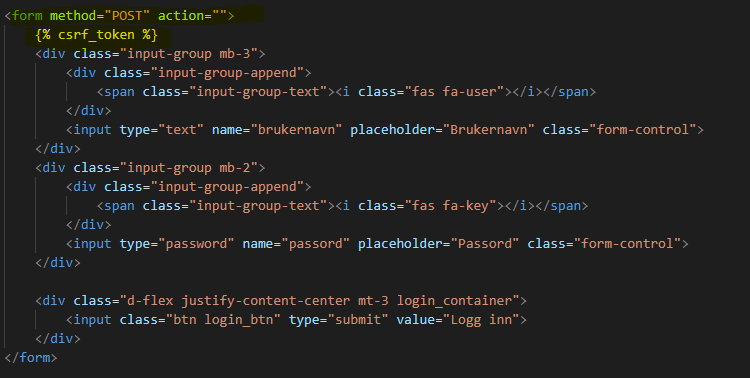




I vår logg inn view må vi lage logikk for å håndtering av innlogging av brukere. Før vi kan starte med logikken må vi lage noen import.



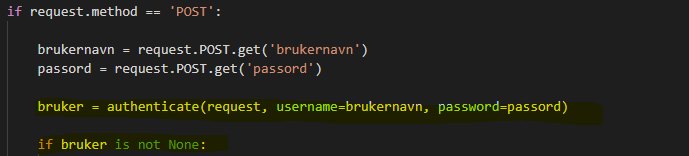
Vi må nå lage en form i logginn.html som sender tilbake POST data. Disse dataene skal vi da håndtere i viewen og logge brukeren inn. Lager først en form i logginn.html.



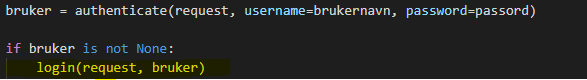
Vi har nå laget en form som sender tilbake POST DATA til vår backend. Vi må også legge til en csrftoken som vist. Nå må vi i vår view håndtere dataene vi får tilbake og logge brukeren inn basert på input. Vi sjekker først om metoden er POST, som vi gjorde i vår registerings logikk. Hivs metoden er post, ønsker vi å hente ut brukernavnet og passordet vi fikk inn.



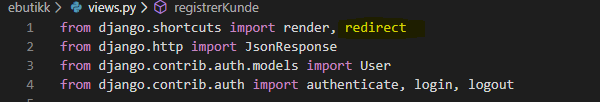
Vi kan hente brukernavn og passord slik fordi vi navnga disse i vår form for input av 2 felter. Først må vi autorisere brukeren og deretter sjekk om brukeren finnes.



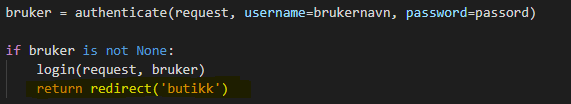
Hvis brukeren finnes skal vi bruke login metoden vi importerte. Login metoden tar inn to parametre, request og brukeren.



Vi ønsker å redirecte brukeren til butikk siden ved innlogging, for å oppnå dette må vi først importere redirect.

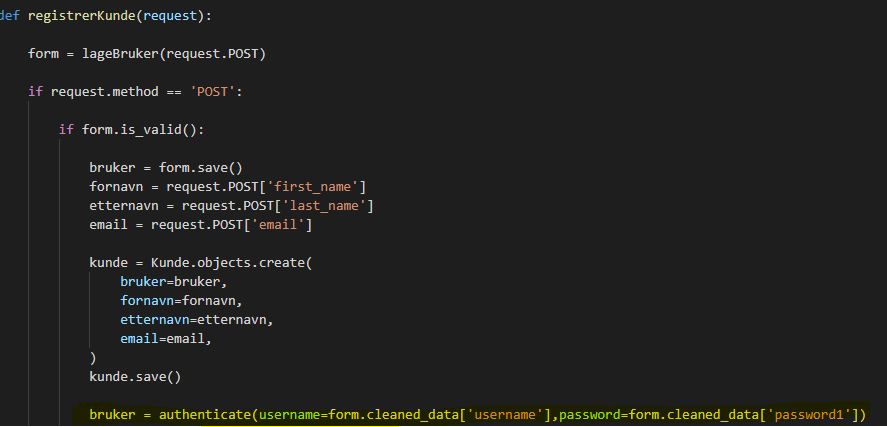


Deretter returnerer vi redirect tilbake til butikk siden.

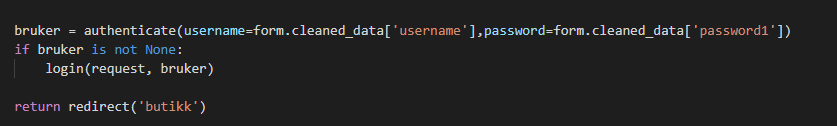


Siden vi navnga butikk.html butikk i vår urls.py kan vi skrive dette i redirect. Nå ser vi at vi kan logge inn en bruker og dertter finne brukeren ordre. Vi kan også prøve å lage nye brukere og deretter se at handlevognen endrer seg i forhold til hvilken bruker som er innlogget. Nå har våre kunder mulighet til å logge inn. Vi ønsker nå ved registrering å logge kunden direkte inn.

Hvis brukeren som ønsker å registrere seg fyller ut formen rikitg og alt er ok, så ønsker vi å logge den nylige registrerte brukeren inn og henvis brukeren til butikk siden. Vi gjør dette ved å tilføye kode i vår registerings view. Det første vi trenger å gjøre er å autoriserer brukeren.



Vi bruker form.cleaned\_data for å hente dataen fra formen som vi validerer. Hvis vi kunne også ha brukt request.POST.get(‘username’) men da hadde hente den ikke validerte dataen. Nå ønsker vi å logge brukeren inn ved hjelp av django sin login metode.

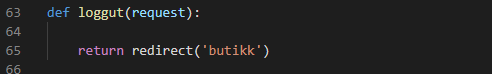


vi lager en if for å sjekke om brukeren finnes i vår database, siden vi lager brukeren først vil brukeren bli lagret i databasen. Hvis brukeren finnes logger vi brukeren inn.

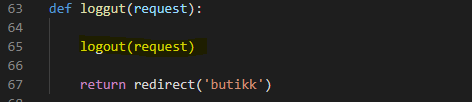
Ved registering vil en bruker automatisk bli logget inn og henvist til butikksiden.

**LOGGUT FUNKSJONALITET**

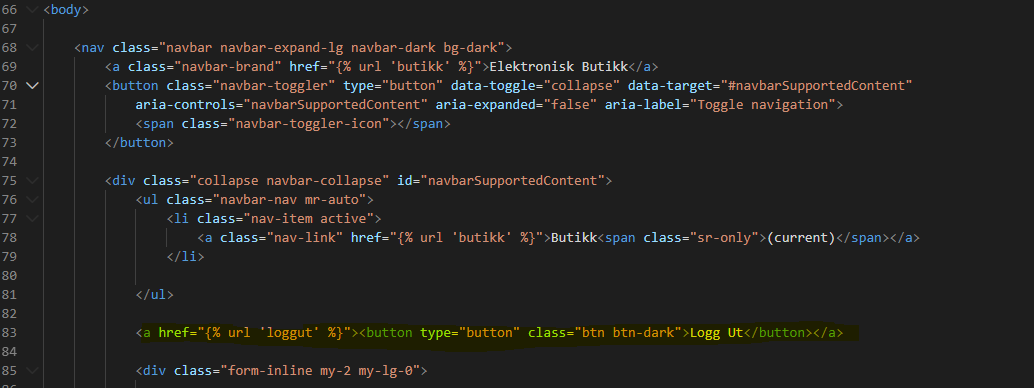
Nå som en bruker kan logge inn, må vi gi kunden evnen til å logge ut også. Det første vi gjør er å lage en view som heter logg ut rett under registerings viewen.



Dennes funksjon er å sende den ut loggete brukeren tilbake til butikk siden, derfor har vi ingen html side vi rendrer her. Dertter lager vi en url i vår urls.py for denne viewen. Det eneste vi trenger å gjøre nå er å kalle på log out metoden vi får av django og deretter sende brukeren tilbake til butikksiden.



Denne funksjonen må nå linke til i vår navigasjon bar. Vi går i hoved.html og lager først en knapp for å utlogging og deretter linker vi denne knappen til denne funksjonen vi nettopp lagde.



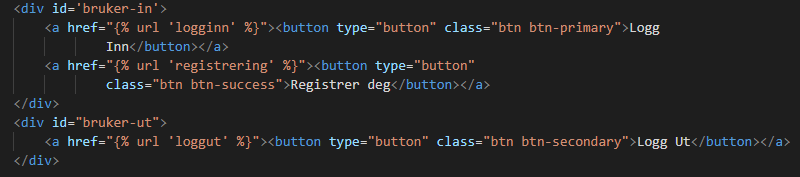
Når brukeren trykker på denne knappen vil kunden bli logget ut og send til butikksiden. Vi legger til en vellkommen melding for brukere som er logget inn. Vi lager den rett over knappen. Vi henter den innloggete brukeren ved følgende kode.



For hver bruker som logger inn nå blir dette vist.



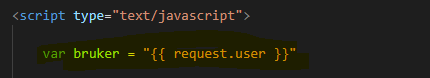
Vi legger så til knapper for inn logging og registrering.



Jeg har lagt disse knappene i forskjellige diver for at det skal være enklere å fjerne.

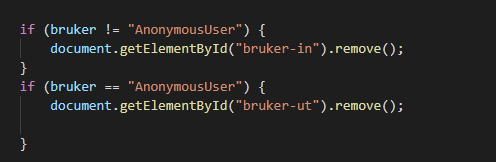
Logg inn knappen skal linke til logg inn viewn vår og registerings knapper skal linke til registering viewen. Siden disse tre knappen vises nå for hver bruker, selv om brukeren er logget inn eller ikke. Vi ønsker nå å gjemme logg inn og registerings knappen for brukere som er logget inn og gjemme log ut knappen for brukere som ikke er inlogget. Vi får dette til med javascript. Vi kan legge denne koden i vår hoved.html. Denne script taggen må vi legge på bunnen av html siden fordi vi må laste siden først for å kunne slette knappene.

Siden vi lager koden i en ny script tag må vi deklarere bruker varabelen igjen.



Logikken vi prøver å få til er. Hvis en bruker er logget inn, skal brukeren ikke se login og registrer knappen. disse knappene ligger i bruker-in diven. Hvis en bruker ikke er logget inn skal vi fjerne log ut knappen og denne ligger i bruker ut diven.

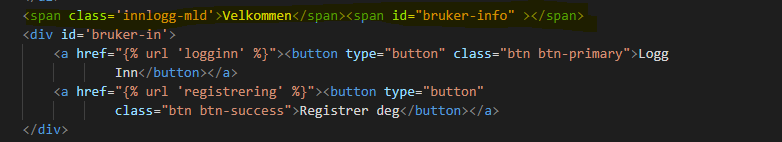
Denne logikken blir da slik:



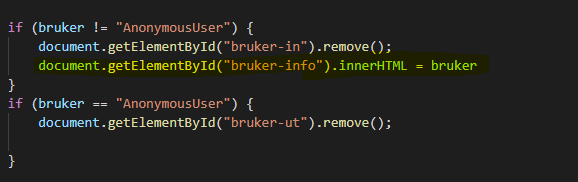
Vi sier først, hvis brukeren ikke er en AnonymousUser så skal vi fjerne elementet iden bruker in, og dette var diven for registering og login knappen. Hvis derimot brukeren er en annonymous user som betyr at brukeren ikke er logget inn, så skal vi fjerne bruker ut, som var diven som innholdet log ut knappen.

Vi ser nå at i vår navigasjons bar for brukere som er logget ut at det står en hilsen:

Velkommen Anonymous user. Denne ønsker vi å fjerne for brukere som er ikke er innlogget og legge til for brukere som er inlogget. Vi først en hilsen som er tom.



Vi skal skrive en velkomst for brukere som er logget inn ved å sende inn brukernavnet deres i velkomst melidngen. Dette gjør vi i javascript koden vi nettopp lagde.



Vi legger til denne kodeblokken i vår if for brukere som er innlogget. Vi forandrer på inner htmlen til span taggen vi nettopp lagde. Denne erstatter vi med brukeren vi hentet i variabelen over. Vi ser at vår logg inn og registrasjons form sender error meldinger på engelsk. Vi ønsker å redigere disse error meldingene og gjøre de om til norsk.

## Logg inn og registrering fullført.

API intergrering

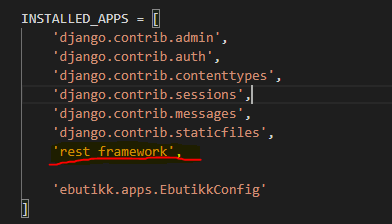
Når vår database blir stor nå, blir det vanskelig å holde styr på data. Hvis vi har noen ansatte i vår elektronisk butikk ønsker vi å gi dem tilgang til å legge inn nye produkter og prossesere ordre. For å gjøre dette enklere for alle brukere er det best å lage en api som håndterer dette. Vi kan lage en ny applikasjon som en api i vårt prosjekt og gi denne elhandel applikasjonen tilgang til dette. Denne apien skal håntere CRUD, create, read, update, and delete funksjonalitet. Vi skal bruke react.js for å hente informasjonen fra apien. Javascript kan også brukes men vi velger react.

Hva er en rest api?

Api står for application programing interface, det er en mellom man for to progarmmer. En api lar et program til å snakke med et annet. En rest api fungerer som en server. Sender en etterspørsel om data fra en client til en server og vi får data tilbake med http protokol. Store selskap som google, facebook, finn osv har en api. Denne har som oppgave å sende tilbake etterspurt data til clienten. Denne dataen er ofte begrenset i forhold til brukren som etterspør. Det betyr at hvis en ansatt på for eksempel facebook kan gjøre mer detalijert og bruker sensetive etterspørsler enn en vanlig facebook bruker. Denne dataen blir sendt tilbake ofte i json format. Dette gjøres så progammere har tilgang til data for sitt eget program. Et eksempel er facebook sin graph api. Hvis vi skriver graph.facebook.com/youtube i vår adresse linje får vi tilbake dataen til facebook brukreen til youtube i json format.

Vi skal i dette eksempelet bruke django rest api, denne gjør ting litt enklere når det gjelder crud funksjonalitet osv. for å instalere dete må vi gjøre en install i vår comando prompt.

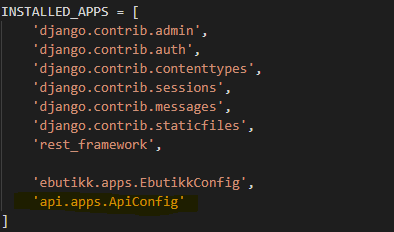
  
Deretter må vi legge den til i vår installed apps i setting,py.

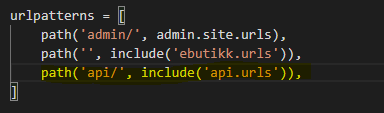


For å lage apien må lage et ny applikasjon i vårt prosjekt og gir den navnet api.

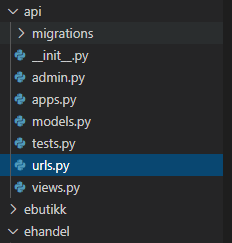


Vi installerer denne appen i vår settings.py

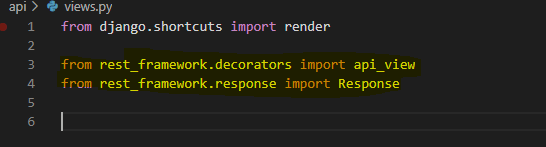


Vi ser nå denne er laget i vårt prosjekt. Vi må nå legge til en url for denne apien i vårt ehandel/urls.py. 

Vi lager så denne urls.py i vår api applikasjkon.



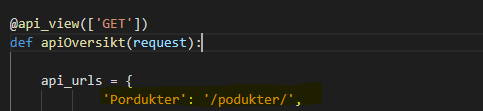
Vi lager først en view i vår api applikasjon. Denne viewen skal vise all data i vår api. For å vise disse datene må vi importere et par ting. Vi må importere resopnse som har ansvar for å sende data som json forman, og api\_view som gjengir dataene på som en api.



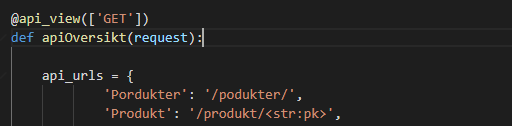
Vi skal nå lage en view for vising av apien og dens protokollen. Vi gir denne viewen en GET funksjonalitet med en api\_view.



Denne viewen skal gi ut data derfor kaller vi den for GET i api viewen. før vi gir ut en response, må vi lage url ruter for funksjonaliteten i apien. Disse urlene skal representere apien kallene som en bruker kan gjøre ved bruk av vårt api. Vi lager en url for oversikt av produkter.

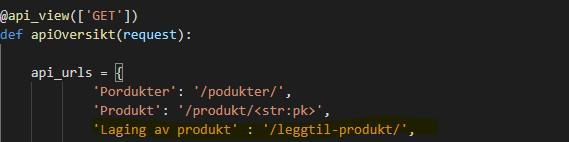


For visning av et produkt:

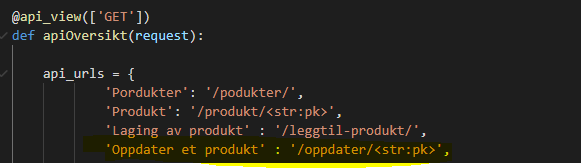


Merk: <str:pk> representerer iden til produktet. Denne skal vi sende inn for å vise et spesielt produkt ved hjelp av dens id.

Lager en url for laging av et produkt.

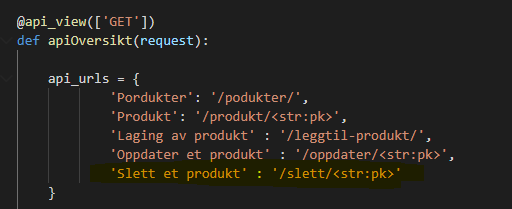


En for oppdatering av et produkt.



Vi oppdaterer produktet igjen ved hjelp av dens id.

Et for sletting av produkt.

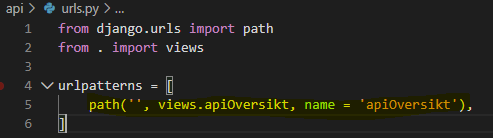


Vi bruker igjen iden til å slette produktet vi leter etter. Disse urlene skal brukres for crud funksjonalitet. Når bruker får tilgang til vårt api, skal brukeren ha en oversikt over hvilke funkjsoner brukeren har tilgang til.

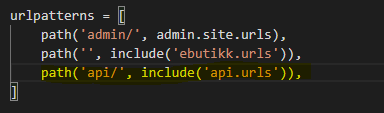
Nå må vi returnere en Json Response som gjengir disse urlenene.



Vi må lage en url som gjengir denne viewen. dette gjøres i api/urls.py



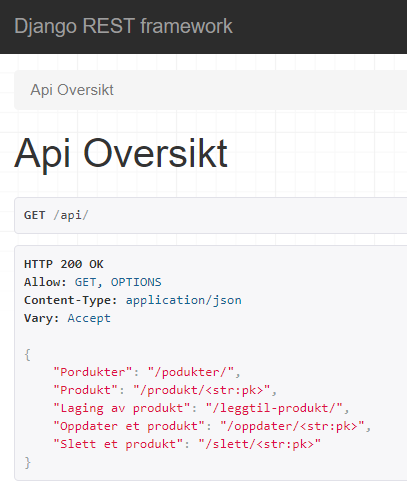
Merk importene må også være med. Husk at vi lagret koblet sammen prosjektets urls med api apliaksjonen sin url slik:



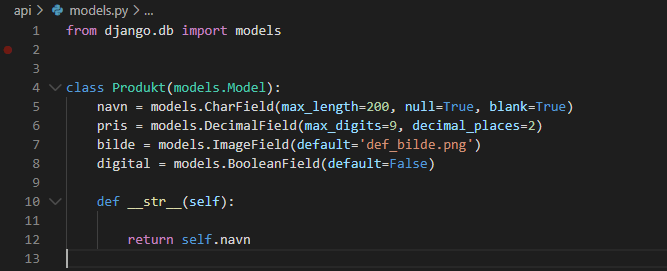
Dermed må vi skrive api i søke feltet for å gjengi denne oversikten.



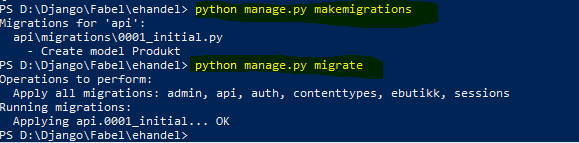
Da skal vi se denne siden.



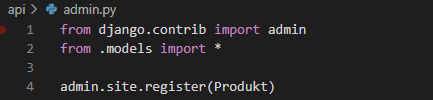
Dette gir brukeren en oversikt overfor hvilke api kalle som kan gjøres. Vi skal gjengi produktene i vår data base ved hjelp av en api, vi henter modelen for produkter i vår ebutikk/modells.py og legger den til i api/models.py. Vi kopierer koden og limer den inn.



Nå må vi migrere databasen.

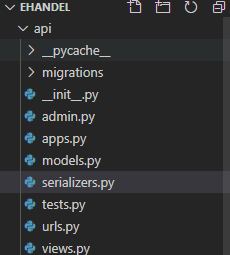


For at denne skal vises i vår database må vi gjøre følgende i admin.py i vår api applikasjon.

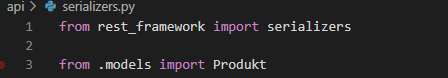


Vi må først importere alle modellenen og deretter registerer de på siden. Vi legger så til de samme produktene vi har i vår ebutikk database.

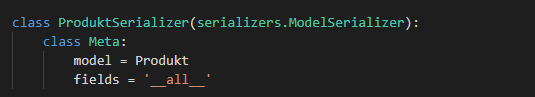
For å kunne vise denne modellen som et json objekt må vi først serialisere dataene. For å gjøre dette legger vi til en fil som heter serializers.py i vårt api applikasjon.



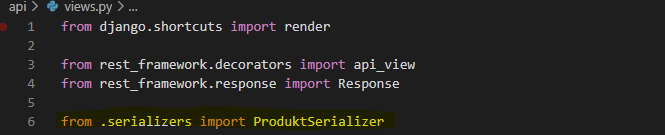
I denne filen skal vi lage en klasse og spesifisere hvilken modell vi skal serialisere. Først må vi importere produkt modellen og serialiser metoden som django tilbyr.



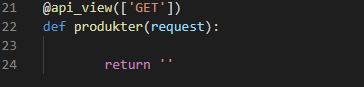
Vi lager så en klasse og spesifiserer modellen.



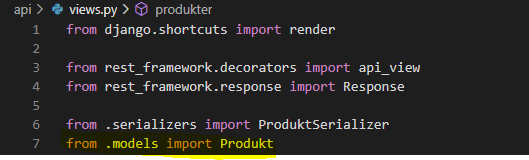
Vi spesifiserer hvilken model vi skal serialisere og antall felter vi skal tillate. Vi har nå laget en serializer for dataene, nå kan vi importere dette inne i viewen.



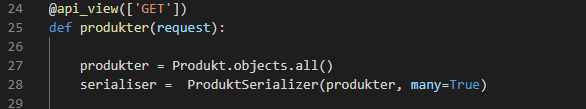
Nå må vi lage en funksjon som gjengir produktene i vår database. Vi lager denne i views med en dekarator api\_view. Denne viewen skal være en GET metode siden vi skal bare gjengi produktene ikke endre på noe data.



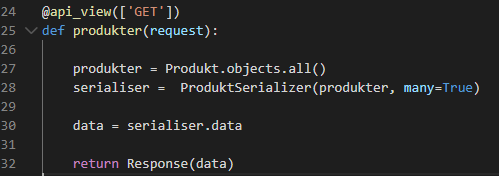
Før vi kan serialisere dataene må vi importere modellen produkt i vår view.



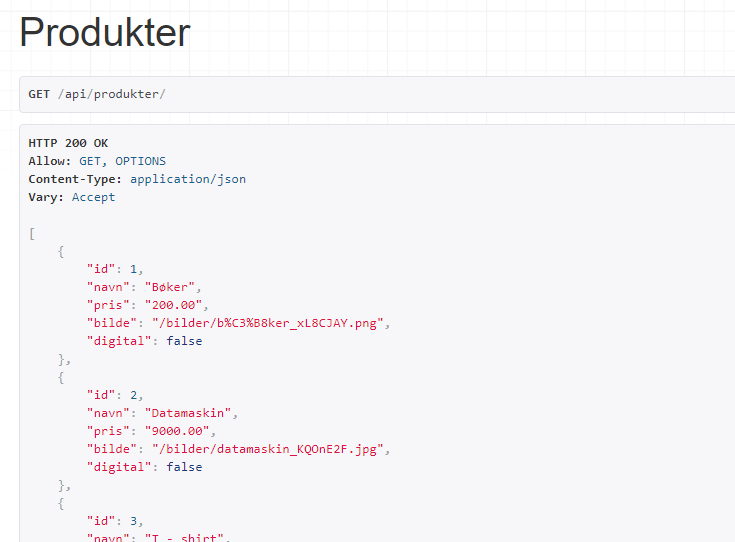
Først må vi hente alle produktene vi har lagt til i vår database. Deretter bruker vi serialiser metoden vi nettopp laget til omgjøre datene til json format.



Får å kunne gjengi disse dataen på siden må vi gjøre følgende.



Når vi går inn på vår produkter api, skal vi da se dette.



Listen skal vise alle produktene vi har i vår database med info.