UNIVERSITETET I AGDER

# DAT215

# Prosjekt Rapport
# Rusta Vrak Bilutleige

*Student:*
Martin Engen

*Veileder:*
Arne Wiklund

*Ved*

Institutt for IKT
Fakultet for Teknologi og Realfag

26. november 2016

# Innhold

**Innhold** i

**List of Figures** ii

**1 Innlending** 1
- 1.1 Bakgrunn . . . 1
- 1.2 Problemdefinisjon . . . 1
  - 1.2.1 Oppgaven . . . 1
  - 1.2.2 Krav . . . 2
- 1.3 Problemløsning . . . 2
  - 1.3.1 Arbeidsmetode . . . 2
  - 1.3.2 Prosjekt Plan . . . 3
  - 1.3.3 Workflow . . . 4
    - 1.3.3.1 Kunde . . . 4
    - 1.3.3.2 Administrasjon . . . 4
- 1.4 Litteratur Studie . . . 4

**2 Oversikt over valgte Teknologier** 6
- 2.1 Publisering . . . 6
  - 2.1.1 Platform as a Service . . . 6
  - 2.1.2 Fordeler ved PaaS . . . 6
  - 2.1.3 Valg av PaaS Leverandør . . . 7
- 2.2 Programmeringsspråk og Rammeverk . . . 8
  - 2.2.1 Python . . . 8
  - 2.2.2 Django . . . 8
- 2.3 Frontend . . . 9
- 2.4 Verktøy . . . 9

**3 Løsning** 10
- 3.1 Krav . . . 10
  - 3.1.1 Ikke-funksjonelle krav . . . 10
- 3.2 Designspesifikasjoner . . . 10
- 3.3 Implementasjon . . . 10
- 3.4 Priser og Rabatt . . . 11
- 3.5 Administrasjon Side . . . 12
  - 3.5.1 Reservasjoner i Administrasjonsside . . . 13
- 3.6 Database . . . 14
- 3.7 Testing og Validering . . . 15

# Figurer

1.1 Iterativ Utviklings Diagram . . . . . 3

2.1 Django Oversikt . . . . . 8

3.1 Utleiepris Diagram . . . . . 11
3.2 Forside i Administrasjons Side . . . . . 12
3.3 Administrasjonsside - Oversikt over reservasjoner . . . . . 13
3.4 Django Oversikt . . . . . 13

# Kapittel 1

# Innlending

## 1.1 Bakgrunn

Rusta Vrak bilutleige er et lite firma basert i Førde og Jøster i Sogn og Fjordane. Utleie firmaet blir driftet av verkstedmesteren Stein Olav Erikstad, og målet er å kunne tilby utleie av greie og fult funksjonelle biler til en rimelig pris. Firmaet ble startet i 2013, og har i dag en flåte på over 20 biler. Rusta Vrak fungerer i dag som et sideprosjekt, men det har gode muligheter for vekst i fremtiden.[1]

Dersom et bilutleige firma ønsker å være kompetitivt, er det viktig å kunne tilby en digital form for tilgjengelighet på lik linje som de mer etablerte bedriftene.

## 1.2 Problemdefinisjon

Rusta Vrak Bilutleie har i dag en nettside basert på tjenesten Blogspot [CITATION], som gjør at det er en enkel informasjonsside som oppgir informasjon om bilene, priser og kontaktinformasjon. Deretter må kunden ta kontakt enten vha. telefon eller epost for å foreta selve bestilling. Da firmaet blir driftet av én person som et sideprosjekt, kan det være vanskelig å alltid være tilgjengelig, og samtidig ha en oversikt over hvilke biler som er ledige til enhver tid. (ASD)

Dette kan bli forbedret ved å gjøre nettsiden mer interaktiv både for kunde og bedrift.

### 1.2.1 Oppgaven

Lag en nettside for bilutleie firmaet Rusta Vrak Bilutleige. Denne nettsiden skal kunne benyttes av kunder for å finne hvilke biler som passer best for kundens formål, få en oversikt over når de spesifikke bilene er ledige for utleie, og foreta en reservasjon av valgt bil.

Nettsiden skal også kunne benyttes av firmaet for å få en oversikt over bilene som er for utleie, samt kunder som leier eller har leid bil. Det må være mulig å legge til, fjerne og redigere bilene for utleie forløpende.

### 1.2.2 Krav

I samarbeid med firmaet utarbeidet vi noen krav over hva som er viktig å få med i nettsiden. Disse kan være både funksjonelle og ikke-funksjonelle.

**Lett å bruke.** Nettsiden skal være enkel å bruke, og skal helst være så intuitiv som mulig. Det betyr at den ikke skal for komplisert i bruk.

**Selvstendig og Billig.** Firmaet er lite, og derfor skal nettsiden kunne være så selvstendig som mulig. Dvs. arbeid med drift og opprettholdning skal være minimal. Utover dette burde hosting av nettsiden også være rimelig.

**Håndere alle reservasjoner.** Nettsiden skal både fungere som et reservasjons system for brukere, og som et verktøy for bedriften. Derfor skal brukere kunne benytte nettsiden for reservasjoner, og ansatt skal ha et administrasjons område hvor det er mulig å legge til nye bestillinger.

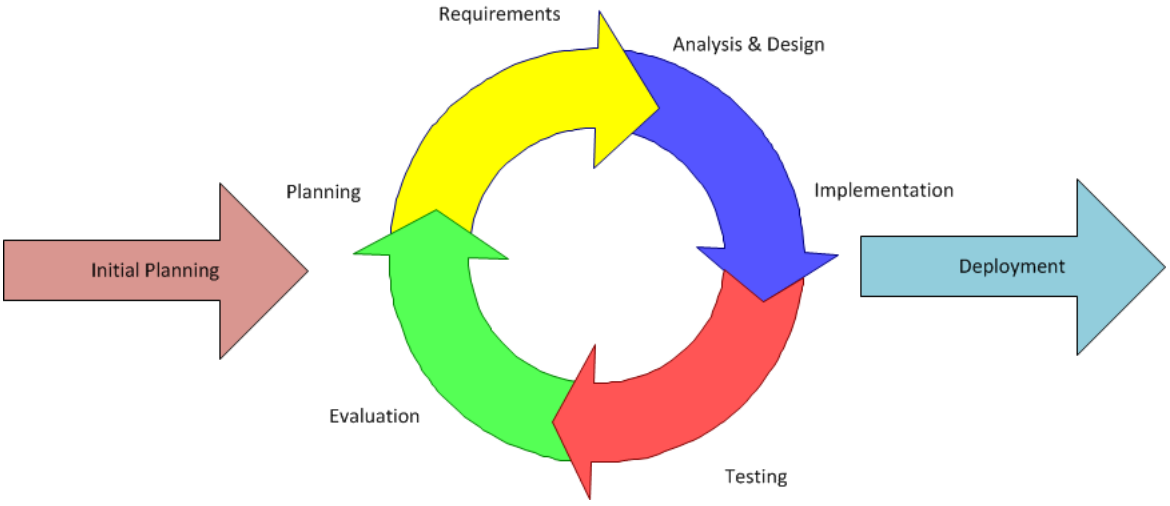
**Fordel å leie i lange perioder.** Det skal være en fordel å leie over lengre perioder. Systemet må ta hensyn til dette og det skal implementeres en funksjon for å kalkulere prisen.

## 1.3 Problemløsning

### 1.3.1 Arbeidsmetode

Utviklingen av dette prosjektet har benyttet en iterativ og inkrementell arbeidsmetode. Denne metoden går ut på å gjennomføre utviklingen stegvis, man begynner utviklingen med et skjelett, som man vil videre legge til litt funksjonalitet, og deretter komme tilbake og legge til mer på toppen av dette igjen. Ved å benytte en slik prosess, blir det mulig å utvikle flere områder på nettsiden samtidig uten at det nødvendigvis krasjer mellom de forskjellige funksjonene.

En iterativ metode gjør det mulig å splitte et større prosjekt i mindre, mer håndterlige biter. Utvikling av kode, planlegging og design blir gjennomført i gjenntatte sykluser [2]. Dermed kan man følge figur 1.1 for prosessen gjennom hele utviklingen.




FIGUR 1.1: Oversikt over trinnene per syklus i iterativ utvikling [3].

### 1.3.2 Prosjekt Plan

Det første som ble gjort mot denne oppgaven var å lage en plan over alle funksjoner og arbeidsoppgaver som må med i det endelige produktet, og legge disse inn i kategorier basert på viktighet. Kategoriene går fra 1 til 5, og man skal gå stegvis gjennom listen.

1. **Oppsett og Installasjon**
   (a) **Lokalt Utviklingsmiljø** - Installere Python, oppsett av virtualenv, Installere og integrere JetBrains PyCharm, installere Django.
   (b) **Lokal Database** - Installasjon og opprette kobling mellom Django prosjekt og lokal MySQL database.
   (c) **Google Cloud Platform** - Installering og oppsett av kobling mellom lokalt prosjekt og Google Cloud Platform. Dette inkluderer både hosting og database
2. **Første Steg i Utviklingen**
   (a) **Database Tabeller** - Planlegging og oppretting av database modeller og tabellene.
   (b) **Forside** - Enkel forside.
   (c) **Admin** - Mulighet til å legge til / slette biler fra opprettet database.
   (d) **Generere liste over biler fra database** - Basert på biltype.
   (e) **Bil Reservasjon** - Mulighet å legge inn er reservasjon av bil som går mellom to datoer.
3. **Videreutvikling**
   (a) **Bilder** - Mulighet for å kunne vise bilde av bilene. Både som thumbnail og alternativt galleri.

(b) **Kalender** - Gjør det mulig for kunde å ha visuell oversikt når den spesifikke bilen er ledig for utleie.

(c) **Epost Bekreftelse** - Send en email til kunde som bestiller. Denne skal inneholde informasjon om bestillingen.

(d) **Reservasjon Håndtering** - Legg til funksjonalitet som stopper en reservasjon fra å krasje med annen allerede eksisterende reservasjon.

(e) **Søkefunskjon** - Finne ledige biler på dato.

4. **Ekstra Funksjonalitet**

(a) **Pris Funksjon** - Det skal være en fordel å leie over lengre perioder. Derfor må en funksjon kunne dele ut riktig mengde rabatt over hvor mange dager som er i leieperioden. Videre skal denne kunne runde til nærmeste 5 for å slippe for mye småpenger.

(b) **PDF** - Mulighet å laste ned PDF med bestillingsdetaljer.

(c) **Filtrering av bil liste** - F.eks. kun biler som har automat.

(d) **Konfigurering av Admin** - Søk bil på skiltnr., oversikt over alle reservasjoner og kunde informasjon.

5. **Dersom tid**

(a) **Google Calendar** - Legg til informasjon i en Google Calendar for enkel oversikt over når biler hentes og leveres.

### 1.3.3 Workflow

#### 1.3.3.1 Kunde

Kunden skal kunne gå gjennom en enkel prosess for å få utført en bestiling på nettsiden. Man kan se på figur [FIG] fremgamgsmåten som skal benyttes. Man vil først møte en forside

#### 1.3.3.2 Administrasjon

## 1.4 Litteratur Studie

Det aller meste av studie mot denne oppgaven har foregått på internett. Dette inkluderer hovedsakelig bruk av dokumentasjonen til de forskjellige rammeverker og plattformer. Utover dette har det blitt benyttet noen bøker som har fungert som oppslagsverk gjennom hele prosessen.

**Bøker**

- Code Complete 2. Edition (Steve McConnel)
- Programming Google App Engine with Python (Dan Sanderson)
- HTML and CSS Design and Build Websites (Jon Duckett)

# Kapittel 2

# Oversikt over valgte Teknologier

## 2.1 Publisering

Ett av kravene til prosjektet var at nettsiden skal være så selvstendig som mulig. Det skal være minimalt med nedetid, serveren må kunne oppdateres uten manuelt inngrep og backup av database må foregå automatisk. Samtidig som å oppfylle dette burde selve hostingen være så billig som mulig. Derfor ble det noe moderne konseptet av Platform as a Service benyttet.

### 2.1.1 Platform as a Service

En Platform as a Service (PaaS) er et utviklings og distribusjons miljø basert i en skytjeneste [4]. En PaaS leverandør har som ansvar å levere og opprettholde en utviklings og distribusjons plattform for sine kunder. De har videre ansvaret på områdene rundt konfigurering, oppsett av servere og sikkerhet på server siden. Det er populært å referere til PaaS leverandører som IT-avdelingen til produktet, som gjør at utvikleren kan fokusere mer på selve utviklingen enn oppsett av infrastrukturen som ligger under [1, s. 10]

### 2.1.2 Fordeler ved PaaS

#### Spare Tid

Utviklingen foregår direkte mot arkitekturen til den valgte skytjenesten, derfor vil man unngå forarbeidet rundt oppsett og installasjon av servere. Dersom en server hos PaaS leverandøren skulle gå offline, vil dette bli ordnet fortløpende av leverandøren, uten at man som kunde må gripe inn [1, s. 9].

#### Billig

Samtidig som å levere mye i pakken, kan publisering vha. PaaS være billig. Dette kommer av at de aller fleste leverandørene bruker dynamisk ressursallokering. Det betyr at ressursene allokert til et prosjekt vil følge samme kurve som behovet. Ved lite eller ingen trafikk vil et prosjekt gå i dvalemodus og bli der helt til en forespørsel blir sendt til serveren. Man betaler kun for de ressurser som blir benyttet.

### 2.1.3 Valg av PaaS Leverandør

PaaS er fortsatt et moderne konsept, og man har mange muligheter ved valg av leverandør. Det er en enorm jobb å foreta en sammenlikning over de relevante mulighetene, derfor har jeg valgt å gå ut ifra et studie som jeg foretok under et bachelor prosjekt våren 2016 [1, s. 18]. Her ble det utført en sammenlikning av 3 av de største PaaS leverandørene: Google Cloud Platform, Microsoft Azure og Amazon Web Services.

#### Resultat

Alle de tre nevnte leverandørene har et bredt utvalgt av muligheter på sine løsninger. Alle tilbyr et løfte av server tilgjengelighet på 99,95% av tiden. De har gode tilkoblingsmuligheter i Europa og prisen er basert på tid og ressurser.

Men det er spesielt ett område de har unike løfter; Gratis Kvoter.
Microsoft Azure leverer kun gratis kvoter i form av trial account. Her får man ressurser for 200$ som man kan benytte som man selv bestemmer, i løpet av de første 30 dager etter Trial Account blir opprettet.
Amazon Web Services kommer utstyrt med 1 år 'free tier', som inneholder 5GB lagringsplass, 20 000 GET requests og 2 000 Put Requests.
Google Cloud Platform derimot tilbyr daglige gratis kvoter. Dette inkluderer bl.a. 28 Frontend Instance timer[1], 1GB plass for lagring av logger, 1GB data inn og ut.

Ved å bruke Google Cloud Platforms gratis kvoter, vil man kunne ha hosting av mindre trafikkerte nettsider bortimot gratis, og man behøver kun betale for databasen.
**Valg: Google Cloud Platform**

[1] Google Cloud Platforms 'frontend instance' blir opprettet ved at en applikasjon får en forespørsel, og denne instance vil være tilgjengelig for bruk de neste 15 minutter. Alle nye forespørsler i disse neste 15 minutter vil benytte samme frontend instance

## 2.2 Programmeringsspråk og Rammeverk

Med dette prosjektet ville jeg videreutvikle min kompetanse innen bruken av web-applikasjons rammeverket Django, basert på programmeringsspråket Python.

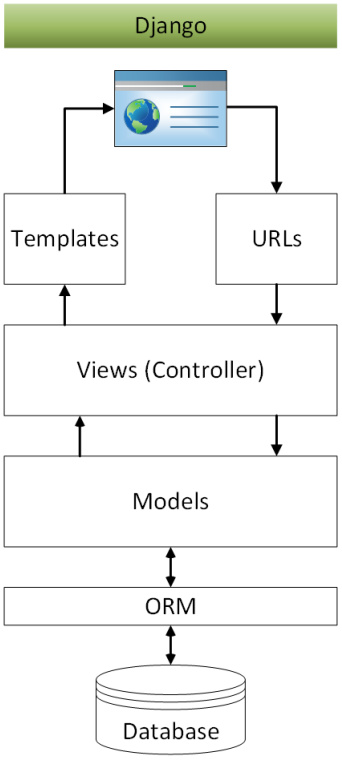
### 2.2.1 Python

Python er et skriptingspråk utviklet av nederlandske Guide van Rossum. Den tidligste versjonen kom frem i 1996 og det har stadig vært i utvikling siden. Python har et formål om å gjøre kode leselig og gjenbrukbar, og har et mindre fokus på ren hastighet. Utover dette har Python et fokus på å gjøre utviklingen, og debuggings fasene av prosjekter så raskt som mulig [1, s. 11].

### 2.2.2 Django

Django er et web-applikasjons rammeverk som kommer levert med en 'batterier inkuldert' filosofi. Dette betyr at vanlig funksjonalitet som ofte blir benyttet i en web sammenheng blir levert med i grunnpakken til Django. Dette inkluderer bruker-system, URL routing, template system, administrasjons system og database object-relational mapper (ORM) [5].

FIGUR 2.1: Stegene Django kjører gjennom ved et HTML request.




På figur 2.1 kan man se en oversikt over hvordan Django håndterer et HTML request.

Først vil den tyde hvilke URL som kommer inn. Django inneholder et register over tilgjengelige URL som kan benyttes vha. regular expressions. Den vil sjekke dette registeret og finne hvilken funksjon (view) som tilhører den innkommende URL.

Den korresponderende funksjon i view vil nå håndtere data fra HTML requestet, dette innebærer å sjekke om det er et POST eller GET request, og hente ut eventuell data med henyn til dette.

Data vil bli hentet vha. Djangos Modeller, som er et objekt orientert metode å fremstille den tilkoblede databasen. Når det er klart hvilke data som skal hentes, vil Djangos ORM konvertere Djangos Database API til SQL Query. Den data som resulterer fra spørringen vil bli returnert tilbake til view.

Til slutt vil view samle data og generere den tilhørende template (html), som videre sendes tilbake til browseren.

## 2.3 Frontend

På klientsiden har det i hovedsak blitt benyttet 2 større

## 2.4 Verktøy

**GitHub** ble benyttet som versjons kontroll underveis i utviklingen. For hver nye feature som skulle introduseres, ble dette utviklet på en egen branch. Når utviklingen av en ny feature fungerte som planlagt, skulle den testes grundig før den ble pushet over på Master branch.

**Google App Engine SDK.** Google Cloud Platform kommer utstyrt med en SDK når man skal arbeide mot Google App Engine. Denne SDK inneholder et lokalt utviklingsmiljø som kjører på samme infrastruktur som det benyttet ved hostingen. Man får også publisering og administrasjons muligheter direkte vha. enkel GUI. Denne har hovedsakelig blitt brukt som et test område underveis i utviklingen, samt. Publisering av nyere versjoner etter hvert som nettsiden har blitt oppdatert.

**JetBrains PyCharm** har blitt brukt som IDE. Denne inneholder en bred rekke hjelpemidler for å gjøre utviklingsprosessen smidigere. Den støtter både utvikling av Django prosjekter og html/css/javascript. I tillegg har den et godt system for autocomplete av kode, og vil generere hint for forbedringer av bl.a. navn på funksjoner, indentering og sørger for at all kode benytter samme navn konvensjon.

**LateX.** Under rapportskrivingen har typesettingssystemet LateX blitt brukt får å generere dokumentet. Dette systemet hjelper med å skape en god utforming av hele dokumentet, som gir bedre muligheter for å fokusere på selve skrivingen. LateX gir et såkalt WYSIWYM (What You See Is What You Mean), som beskriver en form for skriving hvor det resulterende dokument bedre representerer den faktiske informasjonen man arbeider med.

# Kapittel 3

# Løsning

## 3.1 Krav

### 3.1.1 Ikke-funksjonelle krav

## 3.2 Designspesifikasjoner

## 3.3 Implementasjon

## 3.4 Priser og Rabatt

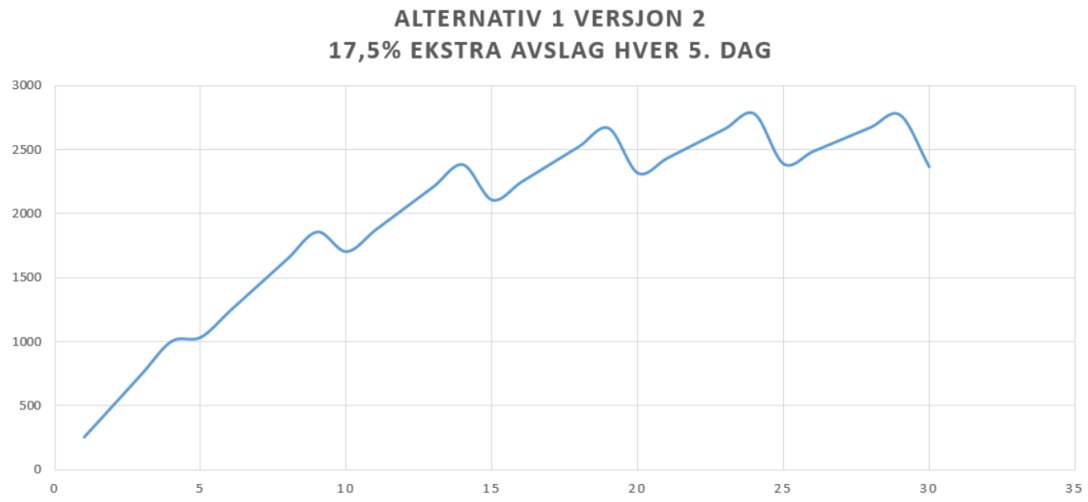
Rusta Vrak Bilutleige ønsker at når en kunde først skal leie en bil, skal det være så lenge som mulig for å slippe å hente og levere biler mer enn nødvendig. Derfor vil bedriften gi kunder gode rabatter som øker i forhold til lenge på leieperiode. Leieprisen av de aller fleste bilene ligger på 250kr pr. døgn, og i dag tilbyr bedriften utleie for 30 dager for kun 2500kr. Dette tilsvarer en prisreduksjon på 66.6%. For å kunne implementere et liknende system på nettsiden ble det laget en rekke løsningsforslag i Excel, dette finner man vedlagt i vedlegg OOAWNDO

### Valgte Løsning

Den valgte løsningen går ut på å gi kunden 17.5% ekstra avslag hver 5. dag i leieperioden. Se tabell 3.1 for en oversikt over hvordan pris avslaget stiger. Figur 3.1 viser hvordan prisantydningen blir på biler som har en døgnpris på 250kr. Dette vil kunne bidra til at en kunde leier litt ekstra kun for å få ekstra avslag på prisen.

Tabell 3.1: Rabatt man oppnår ved leieperioder

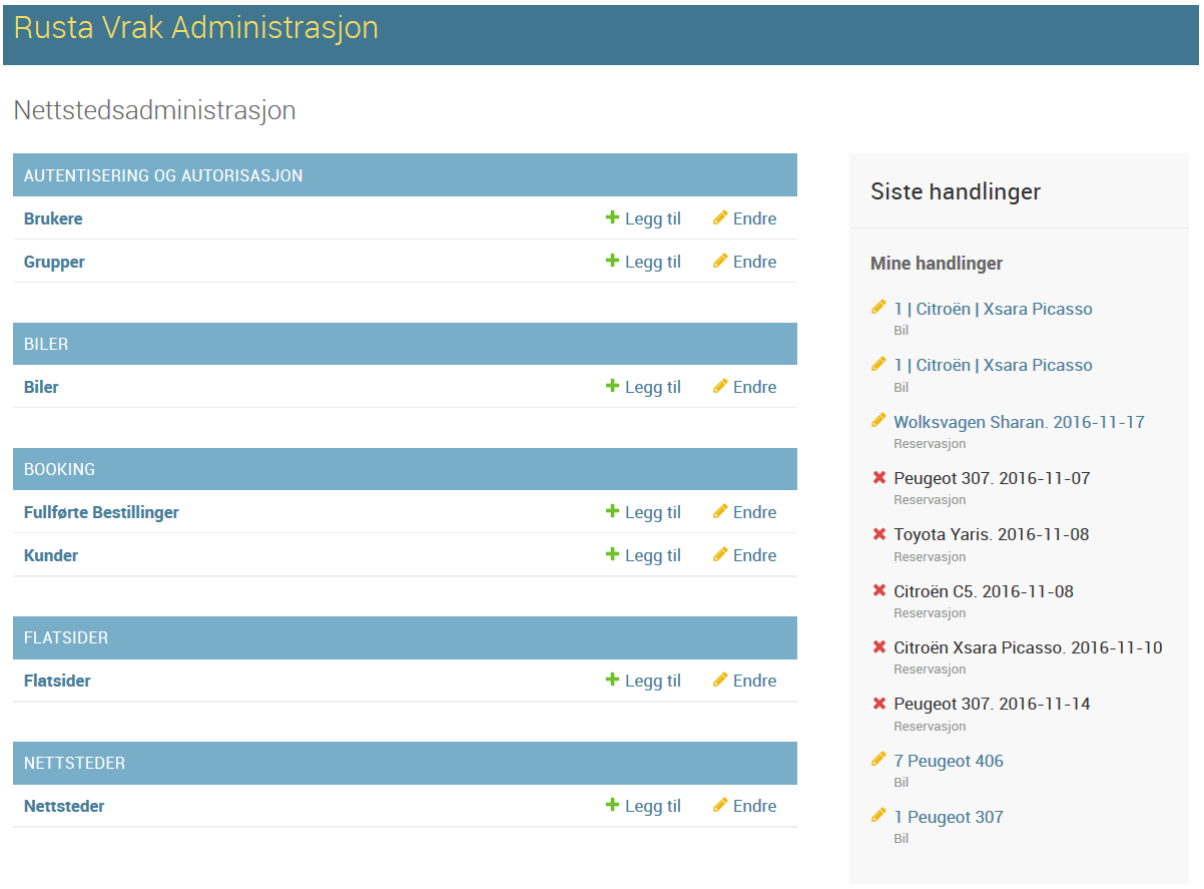
| Antall Dager | 0-4 | 5-9 | 10-14 | 15-19 | 20-24 | 25-29 | 30 |
|---|---|---|---|---|---|---|---|
| Rabatt | 100% | 82.5% | 68.0625% | 56.151% | 46.325% | 38.218% | 33.33% |




Figur 3.1: Oversikt over rabatt som tilføres i forhold til hvor lenge en bil blir utleid.

## 3.5 Administrasjon Side

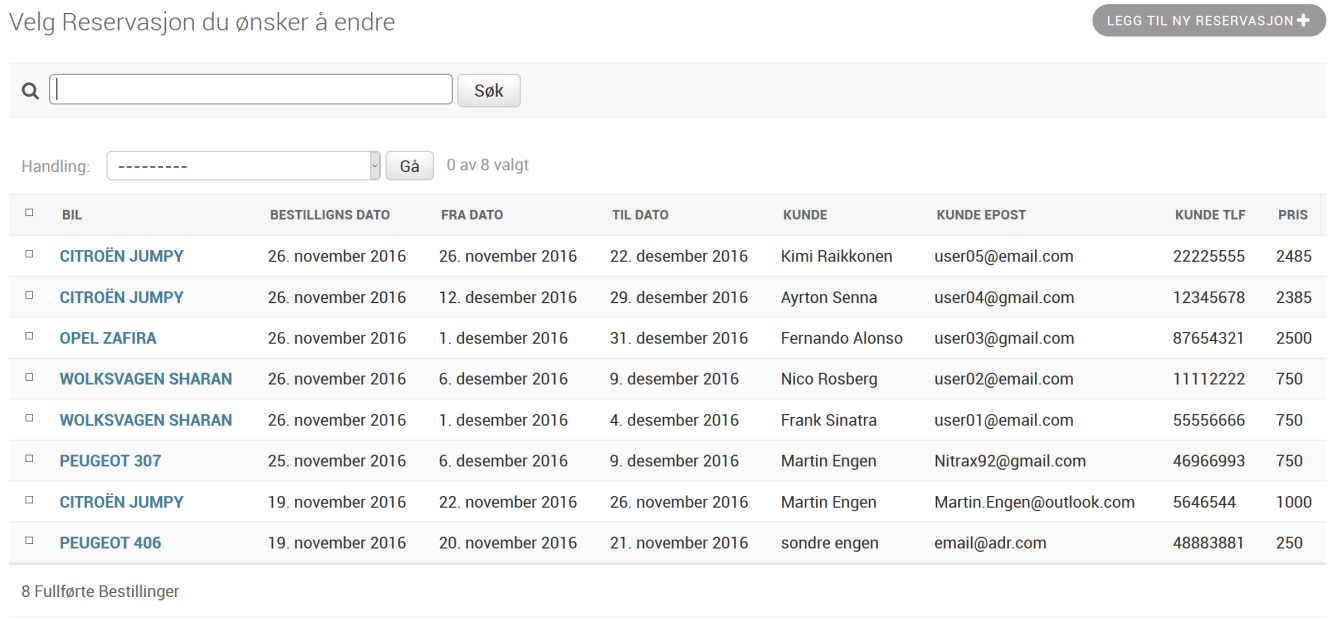
Administrasjonssiden skal hovedsakelig benyttes av bedriften for å kunne holde et styr over sine biler og bestillinger. På figur 3.2 kan man se et skjermbilde av forsiden av admin siden. Denne admin siden blir generert av Django og reflekterer de konfigurasjonene som blir gjort i koden. Man velger bl.a. hvilke tabeller som er interessante å ha i admin siden, hvilke elementer som skal ligge i listen, søke funksjoner osv. Her har man også muligheten til å opprette nye admin kontoer og velge hvilke rettigheter de skal kunne ha på nettsiden.




FIGUR 3.2: Hvordan administrasjonssiden ser ut. Denne har blitt generert og konfigurert med Django.

### 3.5.1 Reservasjoner i Administrasjonsside

På figur 3.3 kan man se listen over alle ferdig reservasjoner. Her kan man se informasjon om kunden, hvilken bil som er reservert, bestillings, hente og leveringsdato, og den totale prisen for reservasjonen. For å kunne filtrere ut spesifikke reservasjonen kan man bruke søke funksjonen. Denne godtar søk på skiltnummer, bilmerke og modell, kundens etternavn og epost. Søket blir gjort å en såkalt «Contains» metode, som betyr at dersom man f.eks. gjør et søk på 'TV', vil reservasjonen med en bil med skiltnummer 'TV65282' bli med i resultatet.

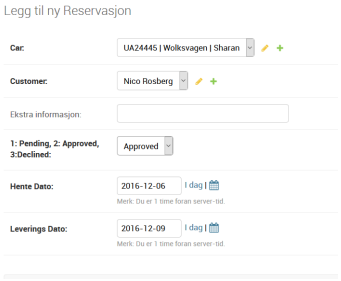



FIGUR 3.3: Liste over alle reservasjoner registert på nettsiden.

#### Legge til ny reservasjon

Dersom en kunde ikke ønsker å reservere gjennom nettsiden, kan ansatte også gjøre dette gjennom administrasjons siden. Som beskrevet i Workflow seksjonen, er det en enkel prosess. Man velger å legge til en ny reservasjon, fyller ut vinduet (Se Figur 3.4) og lagrer dette.

FIGUR 3.4: Stegene Django kjører gjennom ved et HTML request.



## 3.6 Database

Dette prosjektet benytter Django, og dermed vil dette rammeverket ta hånd om mye av arbeidet mot databasen. Django kommer utstyrt med en ORM (Object Relational Mapper), som håndterer overgangen fra Python klasseer til MySQL tabeller.

## 3.7 Testing og Validering

Det har blitt benyttet en iterativ arbeidsmetode som beskrevet tidligere: [REF BESKREVET METODE]. Denne metoden tilsier at testing skal og må foregå underveis i utviklingen i hver syklus. Dette har blitt gjennomført vha. pythons logging bibliotek og Djangos debuggings modus.

# Bibliografi


[1] Martin Engen. Bachelor rapport vår 2016. Technical report, Universitetet I Agder, 2016.

[2] URL `https://www.inflectra.com/GraphicsViewer.aspx?url=Methodologies/Waterfall.xml&name=wordml://03000005.png`. [Online; besøkt 14-November-2016].

[3] URL `https://www.inflectra.com/Methodologies/Waterfall.aspx`. [Online; besøkt 23-November-2016].

[4] "what is paas?". URL `https://azure.microsoft.com/nb-no/overview/what-is-paas/`. [Online; besøkt 23-November-2016].

[5] full stack python, django", . URL `https://www.fullstackpython.com/django.html`. [Online; besøkt 23-November-2016].

[6] system check framework". URL `https://docs.djangoproject.com/en/1.10/ref/checks/`. [Online; besøkt 22-November-2016].

[7] the django template language", . URL `https://docs.djangoproject.com/en/1.9/topics/templates/#the-django-template-language`. [Online; besøkt 22-November-2016].