Københavns Universitet

Softwareudvikling - Deliverable 2

Af
Herluf Baggesen
Mads Buchmann
Philip Girelli

Marts 2016

1 Implementering

I denne iteration er alle de planlagte user-stories, fra forrige iteration, blevet implementeret og testet. Disse user-stories omhandlede, i store træk, skabelsen af en navigerbar hjemmeside som kan vise, slette og oprette data i en database.

De følgende to user-stories "En bruger skal kunne finde en liste over vejledere"og "En bruger skal kunne finde en liste over tilgængelige projekter"resulterede i at to liste oversigter blev udarbejdet med henholdsvis projekter og vejledere.
Ydermere haves der også en række user-stories som benævner muligheden for at se detaljer om projektet samt en række specificerede detaljer om vejlederen. Dette er gjort ved at gøre det muligt at trykke på et element på listen for at få detaljeret information om projektet/vejlederne man har trykket på.

For at løse den user-story'en "Det skal være muligt for en vejleder at verificere sig selv på siden for at kunne tilgå sine egne informationer og projekter og redigere i disse", er et login-system blevet implementeret. Dette giver en vejleder mulighed for at logge ind og ændre informationer vedrørende sin profil, eller de projekter vejlederen er tilknyttet.
Igennem udarbejdningen en user-story'en "En vejleder kan tilføje et projekt til listen over projekter", har vejlederen også mulighed for at tilføje et nyt projekt, hvis vejlederen vel og mærke er logget ind.
Login-systemet som anvendes i vores projekt, er en del af Django Frameworket der har funktionalitet til at oprette brugere med adgangskoder og brugernavne, logge dem ind, og forhindre adgang til visse sider hvis man ikke er logget ind. Ydermere er det muligt at associere disse brugere med oplysninger fra Counselor-modellen.

Enkelte user-stories handlede også om at skabe data-felter for at fastlægge et database design men også så mock data kunne benyttes til testing.

1.1 Planlægning og arbejdsvaner

Alle user-stories blev i starten af iterationen skrevet ind som issues på projektets github side. Dette gjorde det nemt for gruppen at markere user-stories som implementeret, kommentere med bugs og beskrive hvem der arbejdede på hvilken opgave.

Implementering af opgaver skete hovedsageligt ved hjælp af pair-programming metoden, hvor to personer assisterer hindanden i at skrive kode. Grundet denne metode var det optimalt at mødes i arbejdssessioner på 3-4 timer af gangen. Disse sessioner indeholdt, i slutningen, en kort orientering for at give alle gruppens medlemmer en klar fornemmelse af hvor projektet befandt sig efter dagens arbejde. Det var her user-stories kunne markeres som løste på github hvis gruppen mente at den var implementeret korrekt. Enkelte arbejdssessioner blev afholdt hjemmefra vha. Skype og Google Hangouts.

2 Afprøvning

De mål som blev sat til denne iteration har haft stor indflydelse på hvilken metode der blev valgt til testing. Der er blevet brugt unit-testing til "back-end"delen af denne iteration. Det er bla. data felterne og database designet, som er repræsenteret af python klasser. Men også de forskellige respons til anmodninger fra siden, repræsenteret med funktioner. Django, som er det framework der benyttes til projektet, er allerede unit-testet og har en unit-testing pakke til at teste de mere specifikke funktionaliteter. Unit-testing i Django bliver udført med en midlertidig og tom database.

2.1 Gennemgang af en unit-test

Der vil nu blive gennemgået et kort eksempel på en unit-test (Fig. 1) af en respons til en anmodning, også kaldt et "view"i Django.

```
class Test_project_detail_view(TestCase):
    def setUp(self):
        p = Project()
        p.save()

    def test_project_detail_response(self):
        request = self.client.get(reverse('project_detail', args=[1]))
        self.assertTrue(request.status_code == 200)
        self.assertEquals(request.resolver_match.func, project_detail)

    def test_project_detail_404(self):
        request = self.client.get(reverse('project_detail', args=[2]))
        self.assertTrue(request.status_code == 404)
```

Figur 1: En unit-test af et view

Denne test er til detalje-siden på et projekt. Testen skrives som en klasse der nedarver fra Djangos `TestCase`, hvilket giver adgang en række ekstra værktøjer. Herefter vil enhver metode som starter med "test..."automatisk blive aktiveret når projektet testes gennem Djangos "manage¨utility. Derudover kan en `setUp` metode defineres, som aktiveres inden resten af metoderne. Denne er i dette tilfælde benyttet til at indskrive et tomt projekt i den midlertidige database.

Den anden metode, den første test, er en best-case test. Den tester det scenarie hvor projektet der anmodes om findes i databasen. `TestCase` indeholder et `client` objekt beregnet til at imitere anmodninger. Denne benyttes til at gemme resultatet af en anmodning under navnet `request`. Herefter testes det om status koden er som forventet og at den rigtige funktion bliver kaldt via url-resolveren.

Hvis en enkelt assert i en test-metode returnerer false vil hele testen blive behandlet som fejlet og Django vil informere om hvor i testen det gik galt. Den sidste test-metode tester om siden kan håndtere anmodninger på projekter som ikke eksisterer. Siden skal returnere en 404-fejlkode i dette tilfælde.

Resultaterne af unit-tests kan findes i bilag.

2.2 Acceptance-testing

Det andet fokus i denne iteration var præsentation af data. Denne disciplin er langt mere subjektiv og kan derfor bedre testes med metoder som acceptance-testing.
Efter en use-case som omhandlede repræsentation af data var implementeret, ville de studerende i gruppen have en faglig samtale, på ca. 30 minutter om implementationen. Her var emner som overskuelighed, og fleksibilitet hyppige. Implementationen blev herefter ofte revurderet på grundlag disse samtaler.

Metoden kan anses som en mere uformel fortolkning af acceptance-testing. Dog var samtalerne yderst brugbare, da de hurtigt og nemt kunne foretages og derved kunne indgå i relativt små feedback-loops. Denne metode passede derudover godt til projektets størrelsesorden.

2.3 Test Driven Development

Opgavebeskrivelsen til denne rapport beskriver at der skal benyttes *Test Driven Development*, herefter omtalt TDD, i forbindelse med denne iteration af projektet. Gruppens medlemmer blev dog på baggrund af de user-stories som skulle implementeres enige om *ikke* at benytte TDD i denne

iteration. Dette skyldes at metoden ikke synes at understøtte de opgaver der var i fokus. Gruppen forudså bla. at udfordringen i udvalgte implementationer ikke ville være at implementere dem, men derimod at konstruere en test der gav mening. Denne problematik skyldes til dels at størstedelen af løsningerne til user-stories i denne iteration benyttede sig kraftigt af projektets framework.

Det blev derfor besluttet at TDD skulle udskydes til den kommende iteration hvor fokus vil være at skrive en web-scraper fra bunden i Python, en opgave som gruppen forudser vil have stor gavn af TTD.

3 Design

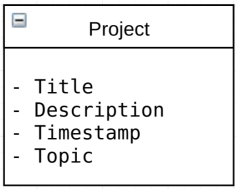
Dette afsnit vil forklare designet af systemet ved først at beskrive de konkrete modeller der er blevet anvendt i projektet og deres funktion. Herefter vil abstraktioner og designmønstre i systemet blive gennemgået med henblik på at give en forståelse af systemet og tankerne bag på et højere niveau.

3.1 Modeller/objekter

Til implementering af user-stories i denne iteration er to nye klasser blevet oprettet. Det er dog vigtigt at understrege at da arbejdet i denne iteration har fokuseret på at lave en webapplikation med Django-frameworket, er det ikke klasser i den oprindelige forstand. Det er derimod klasser med en række attributer der nedarver fra en klasse der allerede eksistere i Django-frameworket som bliver brugt af ORM'en til at lave database-tabeller. Disse kaldes i Django for modeller.
Så det er altså ikke klasser og objekter i den forstand at de indeholder metoder og direkte referencer til hinanden, men bliver derimod brugt til at udforme databasen.

Der er i denne iteration blevet udarbejdet to modeller som begge har til opgave at opbevare information, og bliver brugt til at indele disse.

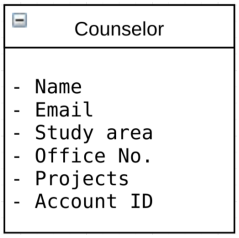
Project-modellen



Figur 2: Project-modellen samt dens indhold af data.

Navnene på attributerne er forholdvis selvforklarende og vil derfor ikke blive forklaret yderligere, det er dog nævneværdigt at deres eneste funktion er at skabe datafelter i data-basen

Counselor-modellen



Figur 3: Counselor-modellen samt dens indhold af data.

Heraf er *projects* og *account_id* attributer som indeholder referencer til andre objekter i databasen. *Projects* et mange-til-mange felt som holder referencer om hvilke projekter den givne vejleder er tilknyttet ved at forbinde id'et på vejlederen til id'et på projektet. På denne måde bliver der skabt en relation imellem Projects-modellen og Counselor-modellen. Dog indeholder Counselor modellen ikke direkte projects-objekter, da det blot er en reference til et project-entry i databasen via et unikt id.

Account_id indeholder en simpel integer, men denne fungere som reference til id'et på en *User* i Djangos indbyggede login-system, der gør det muligt at associere vejlederdata til en user i Djangos system. Ved ikke at knytte disse data direkte til kontoen i Django-systemet kan der komme opdateringer til denne del af Django, uden at det potentielt påvirker informationen knyttet til vejlederen, da den blot skal kende id'et på kontoen.

For at opsummere kan det siges at der ikke er blevet udarbejdet nogen synderligt komplekse klasser eller objekter. Målet med de to klasser der er blevet skrevet, er at organisere data og der har ikke været behov for mere komplicerede designmønstre. Dette skyldes at arbejdet i denne iteration hovedsageligt har været rettet imod at være i stand til at vise information ved hjælp af Django-frameworket.

3.2 Abstraktioner og designmønstre

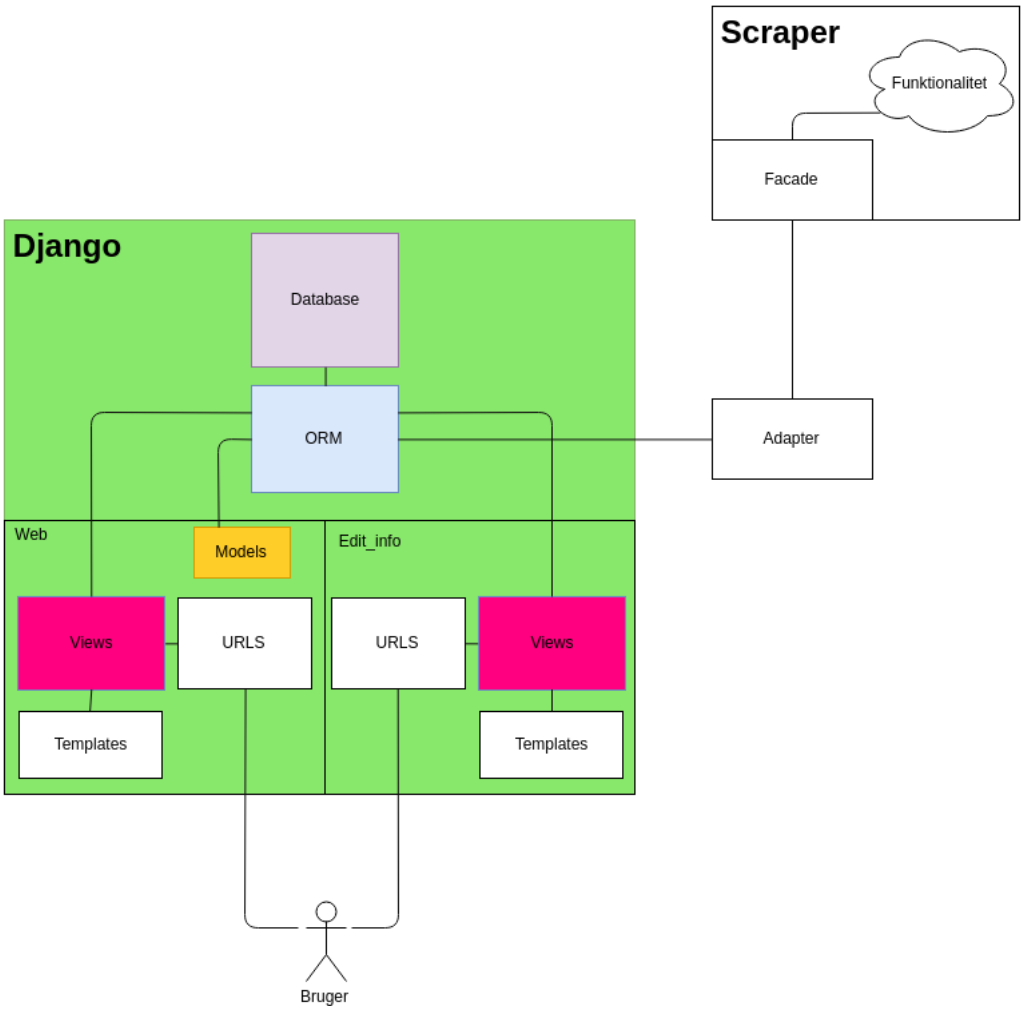
Højniveaus model

For at gøre klar til senere iterationer og for at skabe et fælles overblik over hvordan systemet skal hænge sammen på et højt niveau blev en model udarbejdet til at give et overblik (fig. 4). Denne er en forsimplet version da alle moduler i Django-delen af projektet ikke er synlige, men derimod kun de dele som der er blevet direkte arbejdet med. Den er ikke udarbejdet som et UML-diagram grundet at den hovedsageligt har været brugt som en overfladisk sketch og skal uddybes senere i processen, men har været behjælpelig ved intern kommunikation i gruppen.

Modellen (fig. 4) viser hvordan en bruger tilgår en URL som aktiverer et view, der derefter anvender ORM'en til at hente information fra databasen. Denne bliver derefter "serveret"til brugeren ved hjælp af en template som er html.

Ydermere kan scraperen også ses, og denne er sat på grundet der er blevet gjort tanker om hvordan denne skal kobles på systemet i senere iterationer.Det er blevet diskuteret at denne skal kobles til Django systemet ved at have en adapter som kommunikerer med ORM'en, hvor at adapteren bliver implementeret af en facade i scraperen. Dette tillader at scraperen kan eksistere uden Django projektet og ikke har nogle direkte afhængigheder til denne.

Funktionaliteten er tegnet som en tænkebobbel da det på nuværende tidspunkt ikke er fastlagt hvordan denne skal udarbejdes. Det er dog blevet afgjort at det skal designes i overensstemmelse med S.O.L.I.D. og andre agile principper (Agile, 2006). Dette skyldes at projektet bliver lavet i en lærings-sammenhæng hvor at der bliver opfordret til at designe og programmere systemet agilt. Dette har været en udfordring indtil nu, da der er blevet arbejdet i et framework hvor store dele af designet allerede er fastlagt. Derfor er Django også sat i en kasse for sig selv, da denne del af projektet også skal kunne skiftes ud. Dog indeholder Django-frameworket på nuværende tidspunkt også databasen hvilket måske ikke er helt korrekt. Dette kan være en overvejelse til senere iterationer.



Figur 4: Abstraktion af systemet

CRUD-modellen

Forklaring af CRUD-modellen Når der arbejdes med opbevaring af data som i dette projekt er der fire operationer man gerne vil være i stand til at udføre: *create, read, update og delete*. Forkortet bliver disse operationer til CRUD, som er en model for hvordan man arbejder med lagring af vedvarende data (persistent storage).
Disse operaioner kan direkte mappes til SQL eller HTTP som følger:

Operation	**SQL**	**HTTP**
Create	INSERT	POST
Read	SELECT	GET
Update	UPDATE	POST
Delete	DELETE	DELETE

Modellen kan anvendes til at give et overblik over hvad ens system skal være i stand til, hvilke brugere af systemet skal kunne udføre hvilke operationer og hvilke data skal en bruger kunne udføre hvilke operationer på.
Følgende afsnit vil uddybe hvorfor det er relevant for projektet, samt hvordan den har været anvendt. CRUD-modellen i projektet Det har været relevant at arbejde ud fra CRUD-modellen da data i systemet bliver opbevaret i en database, og dette skal præsenteres til en bruger som eventuelt skal være i stand til at gøre noget ved disse data. Dog er det ikke alle brugere som skal have adgang til alle operationer på al data. Her er CRUD-modellen blevet anvendt til udarbejdning af

følgende tabel, som giver en oversigt over hvilke brugere skal være i stand til hvad.

Bruger/Data	Project	Counselor
Studerende	R	R
Vejledere	C/R/U/D	R/U
Admin	C/R/U/D	C/R/U/D

Her ses brugere af systemet i kolonnen yderst til venstre, og data der haves i systemet ses i toppen.

Brugerne er udledt fra de udarbejdede user-stories som er dokumenteret i gruppens git-repository som issues på github. Her nævnes der blandt andet "En vejleder kan tilføje et projekt til listen over projekter", så altså er vejledere en bruger af systemet. Ligeledes nævnes følgende "En bruger skal kunne finde en liste over vejledere", i retrospekt er ordet bruger lidt vagt, men da systemet skal anvendes af studerende der skal finde et bachelor/kandidat-projekt siger vi at dette er en studerende.

Data i tabellen kommer fra de udarbejdede models i Django, som er abstraktioner af de data som systemet skal indeholde, og de kan findes direkte i filen "product/src/web/models.py".

Modellen viser at vejledere skal være i stand til at udføre alle operationer på Project-data samt read og update Counselor-data. Dog er det ikke al "Project-data"og "Counselor¨data en vejleder skal kunne rette i da følgende user-story skal opfyldes "Det skal være muligt for en vejleder at verificere sig selv på siden for at kunne tilgå sine egne informationer og projekter og redigere i disse".
Dette har ført til opdagelsen af at et form for login-system er nødvendigt, så data kan associeres til en vejleder med et login, og vejlederen vil kun være i stand til at redigere i disse igennem en lukket del af systemet, som kun vejledere har adgang til.

Ved at bruge CRUD-modellen i projektet er en ny forståelse af systemet blevet opnået. Det er blevet klart at user-stories kan kategoriseres efter operationer i CRUD-modellen, som herefter kan associeres til et view i selve Django-implementationen som tillader brugeren at udføre disse operationer på databasen. Det har ligeledes hjulpet med at forstå at den del af systemet som bliver udarbejdet ved hjælp af Django, er en grafisk brugergrænseflade til databasen, som tillader brugere let at udføre operationer på data i databasen. Alt den funktionalitet som er blevet lagt ud i views i Django-projektet haves allerede i en relationel database, dog er det ikke let tilgængeligt for en bruger.

4 Planlægning af næste iteration

I den kommende iteration af vores Projects in Stock vil der blive implementeret en web-scraper. Den grundlæggende opgave for scraperen er at indsamle ny information, så automatisk som muligt og derved minimere tid brugt på administration af information. Dog, vil det i nogen grad være nødvendigt med menneskelig administration af databasen.

Som udgangpunkt vil scraperen få et URL der peger mod en hjemmeside indeholdne relevant information omkring projekter, professorer, emner osv. Det er dog ikke blevet bestemt endnu hvilken (eller hvilke) URL scraperen skal gennemlæse fordi der ikke pt. eksisterer en samlet database over projekter, tilknyttede vejleder, kontakt oplysninger og lign. Som en løsning på dette har gruppen besluttet, at der findes sådan en kilde af information samlet på en hjemmeside, og skrive vores scraper ud fra denne imaginære størrelse.

Vores ræsonement bag denne beslutning går på, at hvis vi får vores egen scraper til at kunne hente information, formaterer det og skrive det ind i vores projekts database, har vi forstået de grundlæggende principper samt strukturer for en scraper. Denne viden kan vi så bruge til at tilpasse scraperen til andre hjemmesider samt udvide den om nødvendigt. Vi er klar over at det er et sats og arbejde ude fra ovennævnte grundlag, men vi synes at det giver god mening og web-scraperen

bliver gjort nem og tilpasse så den skulle kunne arbejde med forskellige hjemmesider.

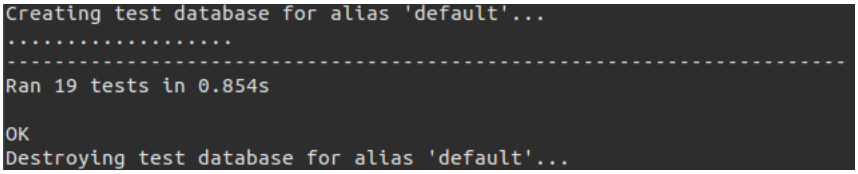
Da vi mere eller mindre har implementeret alle vores user-stories fra forrige iteration vil den kommende iteration næsten kun omhandle web-scraper funktionalitet.
På næste side har vi prøvet at indele web-scraperens funktioner i user-stories samt et forsøg på dets underopgaver.

1. **Scraper skal kunne indhente information fra en given webside**
   (a) Oprette forbindelse til en hjemmeside.
   (b) Læse hjemmesidens HTML-kode.
   (c) Skrive læst HTML-kode til en fil.
   (d) Genkende HTML tags på hjemmesiden.
   (e) Strukturet scraping af hjemmesider baseret på tid (automatik)
   (f) Formatering af parset tekst.
2. **Scraperen skal kunne tilgå databasen**
   (a) Skriveadgang til databasen.
   (b) Opdatering af allerede eksisterende information i databasen.
3. **Scraperen skal kunne tilpasses til indsamling af information fra andre websider**

Som sagt tidligere vil vores scrapers struktur tage udgangspunkt i en delvist imaginær hjemmeside, der præsenterer større mængder data, som f.eks Københavns Universitets hjemmeside over professorer på universitet. Vi forestiller os at det ville være et relevant sted for vores scraper at hente data. Igen vil et af de væsenligste punkter for udarbejdningen af vores web-scraper være evnen til nemt at kunne tilpasse den til scraping af andre hjemmesider.

5 Bilag

5.1 Testrapport



Figur 5: Testresultater af unit-tests

6 Bibliografi og kilder

- Martin, R. C., Martin, M. (2007). *Agile principles, patterns, and practices in C#*. Upper Saddle River, NJ: Prentice Hall.
- Cockburn, Alistair (1997, September/October). *Structuring Use Cases With Goals.* Tilgået Febuar 23, 2016, på `http://alistair.cockburn.us/Structuring+use+cases+with+goals`
- CRUD-modellen, Mullender 2004. Tilgået Febuar 23, 2016, på `https://msdn.microsoft.com/en-us/library/ms978509.aspx`
- Steve McConnell, S. (n.d.), 2004 *Code complete.*