KØBENHAVNS UNIVERSITET

Softwareudvikling - Deliverable 4

Af

Herluf Baggesen

Mads Buchmann

Philip Girelli

Maj 2016

Indhold

1 Ændrede krav og implementering af opgaver

I denne iteration har der været fokus på de emner der blev omtalt som "Forbedringer og Mangler" i den forrige rapport. Dette inkluderer refaktoreringen af projektets kodebase, mere om dette i afsnit 3, og unit- samt acceptancetesting af eksisterende funktionalitet, se afsnit 2. Derudover er en implementation af de fem userstories omhandlende et kontrolpanel til database opdatering for administratorer blevet påbegyndt, dog ikke færdiggjordt endnu. Planen for denne iteration blev på grundet en opfordring i forbindelse med kravene til denne rapport revurderet men der blev ikke fortaget ændringer på baggrund af dette.

2 Afprøvning

I denne iteration har der været stor fokus på at teste programmet grundigt for at opnå et stabilt program igennem unit- og integrationstest, og undersøge om projektet lever op til kravene ved acceptancetesting. Derudover er en make-fil *maketest* blevet skrevet for at kunne køre projektets forskellige test suites nemt. En guide til hvordan denne bruges findes på projektets GitHub.

2.1 Unittests

For at se om de udarbejdede moduler fungere som forventet er der blevet udført unittest på samtlige moduler. Til udarbejningen af unittestsne er Pythons PyUnit unittestest-framework blevet anvendt. Dette har tilladt os at skrive unittest forholdsvis hurtigt og det har dikteret standadiseret testformat hvilket har gjort sig til udtryk for både fejlmeddelser såvel som koden for testsne.

Selve unittesten er udført ved at teste hver klasse i alle moduler (med undtagelse af Django) hvor klassen bliver testet ved at der laves faktiske test på metoderne. Dog er det ikke alle metoder der er blevet testet, da flere af klasserne har metoder som er relaterede til implementeringen af klassens funktionalitet og ikke skal bruges uden for klassen.

I andre sprog som for eksempel Java ville man have markeret disse metoder med `private` så de ikke kan tilgås uden for klassen. Men da Python ikke tillader brugen af `private` og `public` metoder, er det kun de metoder der normalt ville have været public/dem der skal kunne tilgås af andre klasser, der er blevet testet. Man ville også godt kunne have testet de resterende metoder, men da de er afhængige af den nuværende implementation ville de potentielt skulle skrives om hvis implementationen af klassen og dens funktionalitet ændres.

Metoderne der derimod skulle have været `public`, er lukkede for forandring. Det skal forstås som at den forventede opførsel altid skal være den samme, hvordan denne opførsel derimod opnås er mindre relevant hvis det blot er det samme input og output. Test-suiten fungerer dermed som en oversigt over hvilke metoder i hvilke klasser der bliver anvendt uden for klassen. De testcases som bliver udført har fokuseret på om den forventede opførsel bliver udført, baseret på et givent input, et eksempel på dette kunne være følgende test fra CommandManager-testen:

```
def test_que_excludes_command(self):
    #setup
    self.instance = CommandManager()
    command = ICommand(datetime(2017,1,5))
    self.instance.enqueue_command(command)

    #save values and execute command
    startContains = (command in self.instance.commandQueue)
    self.instance.delete_command(0)
    endContains  = (command in self.instance.commandQueue)

    self.assertFalse(endContains,startContains)
    self.instance.myTimer.cancel()
```

En stor del af testene fokusere på edge-cases og dette har medført at en række bugs er blevet opdaget og efterfølgende rettet i koden. Et eksempel på dette kunne være en bug der medførte, at når *delete_command* fra *CommandManager* blev kørt, når der haves en tom kø, ville en *IndexError* blive kastet, og programmet ville lukke ned. Altså har unittesting ført til et mere stabilt program, men den nuværende test-suite udgør også et godt værtkøj til videre udvilking når der opstår fejl, og denne skal findes.

2.2 Integrationstest

For at finde ud af om de forskellige moduler kan arbejde sammen er der blevet udført integrationstest. Dette er blevet gjort ved at teste klassen *Adapter* da det er denne der forbinder de udarbejdede moduler, bruger dem til at opnå den ønskede funktionalitet, og anvendes af front-end delen af projektet til at udføre handlinger såsom at opdatere en vejleders information. Integrationstesten er blevet udført efter at unittesting af de moduler som adapteren bruger er blevet udført. Dette er da hvis der haves fejl i modulerne kan der opstå fejl i adapteren som følge af disse. Så opstår der en fejl i integrationstesten kan det være fordelagtigt at køre prjektets unittests for at tjekke at de ikke er opstået i modulerne.

Selve integrationstesten af *Adapter*-klassen er blevet udført i samme stil som de førnævnte unittest i det at kun metoder der skal bruges uden for klassen der bliver testet, da klassen er åben for udvidelse men lukket for forandring hvad angår dens signatur og opførsel.

Selve integrationstesten har bekræftet at modulerne arbejder sammen som de skal, da den er i stand til at udføre den ønskede funktionalitet uden problemer. Dog udgør de udførte test også et værdifuldt værktøj til videre fejlfinding når programmet skal udvikles yderligere.

2.3 Acceptancetesting med Robot Framework

Frameworket *Robot Framework* er et Python modul der i stor grad fokuserer på automatiseret navigation i applikationer, især i webbrowsere. Det er derfor ofte brugt, og yderst egnet til, at automatisere acceptancetesting i en forholdsvis stor skala. Robot Framework giver en række værktøjer til at simulere inputs, såsom klik og tekst indtastning, og til at verificere responser og inholdet af disse. Robot Framework benytter en yderst letlæselig syntaks til at formulere tests og efter en test er gennemført genererer Robot Framework en testrapport på almindeligt læseligt engelsk. Disse features gør det langt nemmere at inkludere personer uden programmeringsviden i testprocessen.

I alt fire acceptance tests er blevet implementeret med frameworket Robot. To af disse tests adresserer hver for sig en userstory fra tidligere iterationer af projektet. De udvalgte userstories lyder som følger:

- En bruger skal kunne finde en liste over vejledere

- En bruger skal kunne finde en liste over tilgængelige projekter

Testene simulerer en bruger der klikker sig fra projektets startside hen til de respektive oversigter, og sikrer sig at den korrekte respons blev modtaget fra serveren.

De resterende to acceptancetests omhandler loginsystemet for vejledere, et system der er en naturlig konsekvens af en række af userstories omtalt i forrige rapporter. Heriblandt er:

- En vejleder kan tilføje et projekt til listen over projekter
- En vejleder kan markere sig selv som utilgængelig eller tilgængelig

Disse tests simulerer to scenarier; en bruger med et ugyldigt login og en bruger med et gyldigt. Her testes der om det kun er brugere med det gyldige login, som kommer videre i systemet.

Intentionen med disse test var at integrere et acceptancetest framework i projektet og demonstrere at det derved er muligt at automatisere acceptancetests udfra på userstories. Det kan derfor siges at den egentlige værdi af denne proces er at der nu er tilrettelangt et godt fundament for videre acceptancetesting. Resultaterne af at køre hele testsuiten på en testserver lokalt kan ses i bilag under Testrapport: Acceptancetests

3 Refaktorering

En komplet gennemgang af projektets Python kode er blevet gennemført. Undervejs er koden blevet formateret efter de retningslinjer som blev omtalt i den forrige rapport. Disse retningslinjer handlede hovedsageligt om at rette navngivninger og tilføje beskrivende kommentarer hvor nødvendigt. Resultatet af denne process er at projektet kodebase nu er mere letlæselig og nemmere at overskue, både for gruppens egne medlemmer men også for udefra kommende. Kodens struktur, dvs. klassers fordeling og ansvar mm. blev diskuteret i forbindelse med refaktoreringen, men der var enighed om at der ikke var behov for at foretage ændringer på dette område.

En anden konsekvens af refaktoreringen er at projektet nu er skiftet fra Python 3.4 til Python 2.7. Et minimalt antal syntaktiske ændringer er blevet foretaget og kodens opførsel er uændret. Grundlaget for dette skift er et nyt krav om at implementere automatiserede acceptancetests vha. et framework. Gruppen har i denne forbindelse valgt Robot frameworket. Dog var der op imod fire forskellige frameworks på tale. Fælles for alle disse frameworks var at ingen understøttede Python 3.x. Gruppen vurderede derfor at det mest fordelagtige var at skifte til en ældre version af Python, især fordi dette skift var en langt mindre arbejdsbyrde end at automatisere disse tests på egen hånd. Gruppen ser skiftet som en positiv udvikling da det nu er endnu nemmere at integrere udefrakommende værktøjer i projektet da Python 2.7 har et større antal understøttede pakker end Python 3.4.

4 Fremtidige planer

På nuværende tidspunkt mener gruppen, at projektet har et forholdsvis tilfredsstillende indhold af funktionalitet og features. Helt grundlæggende kan hjemmesiden vise aktuelle projekter som er oprettet af deres tilknyttede professorer samt deres kontaktoplysninger. Dernæst kan der automatisk indhentes information om professorne som bliver vist på hjemmesiden i henhold til den relevante professor. Processen, for den studerende som skal vælge projekt, skulle menes og være mindsket kraftigt og præsentationen af nødvendige information er tilgængelig på engelsk alle døgnets timer. I store træk dækker det over projektets grundlæggende krav - opret en platform så bachelor- og kandidatstuderende kan undgå 'projekt-shopping' når de skal finde emner til deres projekter og gør denne platform let tilgængelig for alle dets relevante brugere.

Med hensyn til automatik er det muligt og sætte scaperen til at indhente information fra en given side, på et specificeret tidspunkt, hver dag. For at tage et eksempel kan det i praksis betyde at sætter scaperen til og hente information hver dag kl.04:00. Det vil scaperen så fortsætte med indtil den stoppes. Der kan også foretages

manuelle scrapes, skulle det være nødvendigt. Automatisk indhentning af information menes at være opfyldt på et tilfredstillende niveau og projektkravet om automatik burde være opfyldt.

Der vil herunder følge tre områder der hver indeholder punkter, inden for den givne kategori, som kan forbedres og finpudses. Punkterne skal ikke ses som en præcis arbejdsplan men nærmere et forsøg på og skabe overblik over projektets stadie.

4.1 Funktionalitet

Det er tidligere blevet defineret, at der for gruppens projekt, eksisterer en imaginær database som skulle indeholde al relevant information: projektnavn, projektejer, projektområde osv. Denne afhænghighed er scraperen også bygget op efter og fordi gruppen har været opmærksom på denne afhængighed er scraperen forholds vis enkel og tilpasse, til andre hjemmesider eller databaser.

I fremtiden ses det gerne at denne imaginære database bliver en realitet, da det synes og være punkt der kræver lidt energi og tid, i forhold til hvad den ville bringe. Hvis denne database oprettes vil man kunne centralisere information og måske mindske tiden brugt på koordinering og udveksling af information professorer, vejleder og sekræterer i blandt. Der ville altså kun eksistere ét sted hvor denne form for information skulle indtastes og indhentes. På længere sigt synes det og være en investering værd.

Et andet område hvor der gerne ses en ny implementering er omkring `CommandManageren`. Her kunne et interface, som ligger før `CommandManageren`, være meget brugbart da det ville simplificere processen og gøre interaktionen mellem de forskellige dele af projektet nemmere og forstå fra en udefra kommende part. Processen ville være mere gennemsigtig, hvilket er og stræbe efter (når og) hvis gruppens projekt skulle tages i brug i den virkelig verden.

4.2 Finpudsning

Selve hjemmesiden der kommunikere den scrapede information til brugeren kan godt gøres simplere og mere direkte i dens præsentation. Der er internt i gruppen blevet diskuteret forskellige specifikke justeringer men forslagene kan mere eller mindre koges ned til en mere direkte præsentation. For at se den relevante information skal antallet af skridt gerne være på et minimum. Situationen er ikke slem og informationen menes ikke at være gemt væk på hjemmesiden, men det anerkendes dog, at der kan ske en forbedring. Det skulle ikke være en større omstrukturering af hele websiden men blot en omrokering af nogle blokke med information så f.eks. aktuelle projekter samt tilgængelige vejleder og professorer kan ses så snart du lander på forsiden.

4.3 Testing

Som med et hvert stykke software der involverer menneskelig interaktion, skal gruppens hjemmeside køres igennem en række forskellige test. Rent funktionelt er der er blevet udført unittest, acceptancetest og integrationstest men det er ikke fyldestgørende. Før hjemmesiden potentielt kan lanceres skal der udføres adskillge userexperience- og usabillitytest for at teste hjemmesiden brugervenlighed. Dettes menes og være af essentiel karakter (selvfølgelig først efter funktionalitet og stabillitet er testet og godkendt) da mange forskellige mennesker potentielt kan komme til og benytte hjemmesiden.

5 Bilag: Testrapport

5.1 Unittests:

```
python Scraper/request_tests.py
.....
----------------------------------------------------------------------
Ran 5 tests in 1.186s

OK
python Scraper/parser_tests.py
.......
----------------------------------------------------------------------
Ran 7 tests in 0.000s

OK
python Scraper/scraper_test.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
python CommandManager/CommandManager_test.py
..................
----------------------------------------------------------------------
Ran 17 tests in 0.006s

OK
python Website/manage.py test Website
Creating test database for alias 'default'...
....................
----------------------------------------------------------------------
Ran 19 tests in 1.996s

OK
Destroying test database for alias 'default'...
```

5.2 Integrationstests:

```
python Website/database_manager/adapter_test.py
...............
----------------------------------------------------------------------
Ran 15 tests in 0.307s

OK
```

5.3 Acceptancetests:

```
robot robottest
==============================================================================
Robottest
==============================================================================
Robottest.Logintest :: A test suite for valid and invalid login scenarios
==============================================================================
Invalid Login                                                         | PASS |
------------------------------------------------------------------------------
Valid Login                                                           | PASS |
------------------------------------------------------------------------------
Robottest.Logintest :: A test suite for valid and invalid login sc... | PASS |
2 critical tests, 2 passed, 0 failed
2 tests total, 2 passed, 0 failed
==============================================================================
Robottest.Navigationtest :: A test suite for basic site navigation
==============================================================================
Navigate To Counserlor Overview                                       | PASS |
------------------------------------------------------------------------------
Navigate To Project Overview                                          | PASS |
------------------------------------------------------------------------------
Robottest.Navigationtest :: A test suite for basic site navigation    | PASS |
2 critical tests, 2 passed, 0 failed
2 tests total, 2 passed, 0 failed
==============================================================================
Robottest                                                             | PASS |
4 critical tests, 4 passed, 0 failed
4 tests total, 4 passed, 0 failed
==============================================================================
Output:  /home/herluf/Python2.7/Projects-in-Stock/Product/output.xml
Log:     /home/herluf/Python2.7/Projects-in-Stock/Product/log.html
Report:  /home/herluf/Python2.7/Projects-in-Stock/Product/report.html
```