

DANMARKS TEKNISKE UNIVERSITET

EKSAMINATIONS PROJEKT

Project Planner

Gruppemedlemmer:

Carsten Michael Rosenkilde NIELSEN s123062

Christian Mathias Rohde KIÆR s123812

Jonathan Binner BECKTOR s123094

12. maj 2013

INDLEDNING

Til opgave har vi fået stillet, at designe en planlægningsapplikation. Denne applikation skal skabes for et softwarehus, der mangler et system til at planlægge deres projekter. Projekter skal kunne oprettes efter de informationer, der haves på oprettelses tidspunktet. Dette skal kunne inkludere navn, start/sluttidspunkt og projektleder. Alle disse oplysninger skal også kunne tilføjes eller ændres senere hen. Projekter skal kunne tildeles aktiviteter, som tilføjes med et navn, en beskrivelse, en start/slut dato, antal allokerede timer og det aktuelle timeforbrug. Udviklere skal kunne tildeles aktiviteter, og derfra registrere deres timeforbrug på den pågældende aktivitet. Til løsning af problemerne, har vi opstillet 6 Use Cases som vi vil implementere testdrevent. Følgende Use Cases har vi valgt at kigge på:

- Create projects
- Manage projects
- Define Activities
- Manage developers
- Hours worked on project
- Seek Assistance

Vi har implemteret alle Use Cases test drevent, og det er gjort i forbindelse og uden konflikt med opgavebeskrivelsen.

INDHOLD

1	Kravspecifikation	3
1.1	Væsentlige begreber	3
1.2	Use cases	3
2	Programdesign	4
2.1	Klasser	4
3	Systematiske test	5
4	Konklusion	5

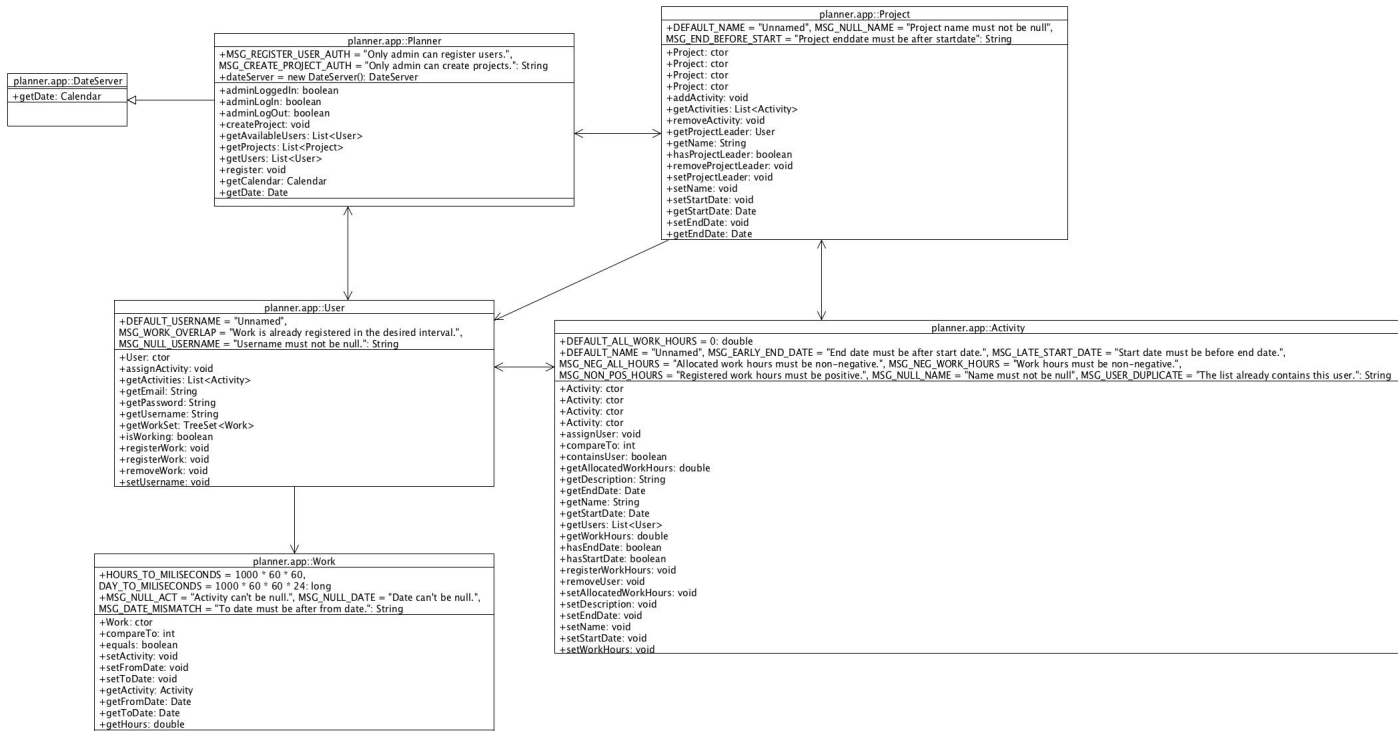
1 KRAVSPECIFIKATION

1.1 VÆSENTLIGE BEGREBER

1.2 USE CASES

2 PROGRAMDESIGN

2.1 KLASSER



Figur 1: Klassediagram

Vi som gruppe diskuterede vi hvad der skulle lægges fokus på. Til at starte med var vi enige om at der skulle være GUI, persistensy og god funktionalitet. Dog da vi var et godt stykke inde valgte vi at fokusere på programmets funktionalitet og de tilhørende tests for at få så god "code coverage" som muligt. Grundet det har vi kunnet løse alle vores usecases. Under konstruktionen af programmet og de tilhørende test var vi usikre på fx. hvordan programmet skulle håndtere hvad users skulle kunne ændre i om de skulle kunne ændre på activities osv. Derudover diskuterede vi også hvordan vi skulle håndtere fejl i programmet fx hvis der blev sat null værdier ind som activity navn. Vi blev enige i at håndtere næsten alle null værdier som en fejl og derfor skulle programmet altså kaste en fejl til useren.

3 SYSTEMATISKE TEST

Alle vores tests er i bund og grund lavet for at kunne få så høj "code coverage" som muligt. Med dette som baggrund har vi lavet tests til mere eller mindre alt i programmet. Vi har taget brug af EcEmma for at finde hvor vi manglede test og hvad der ikke var covered. Nogle af de oftest funde test er tests for getters & setters hvilket var forholdsvis let at få testet man laver en værdi som sættes i en setter hvorefter vi bruger assertEquals for at se om den værdi vi har sat i setteren er den samme vi får fra getteren når vi spørger efter den. En anden test der ofte skulle testes var hvis fx. en setter ikke kunne være null og hvis den var skulle kaste en fejl. Her bruger vi try/catch i testsne for at fange de fejl som programmet giver os og sammenligner dem med de ønskede fejlmeddelelser.

Hvis vi kigger på usecase1 først tester vi om admin er logget ind. Dette gør vi ved at have en boolean value som sættes til true når det rigtige password er blevet skrevet adminLogin() metoden. Derefter kan administratoren lave projekter. Hvis administratoren laver et projekt uden navn vil programmet tildele den et. Dette testes ved at lave et program uden navn og derefter teste ved getName metoden om navnet er blevet til "default name". Denne slags test bliver brugt flere gange i denne usecase hvis der ikke skrives start dato bliver den af default sat til dags dato. dette tjekkes på samme måde. Hvis projektet bliver lavet med end date før start date kaster programmet en fejl. Denne fanger vi med en try/catch som tjekker om fejlen og fejlmeddelelsen er det samme.

4 KONKLUSION