1. Semesterprojekt forår 2019

Svømmeklubben Delfinen

Gruppe 8
Alexander Sarson
Benjamin Paepke
Mads Brandt
Tobias Boldt-Jørgensen

Indhold

Indledning

Rapporten er udarbejdet i forbindelse med 1. Semester projektet: Svømmeklubben Delfinen. Projektet gik ud på at lave en 'konsol' baseret program der skal afhjælpe en række problemstillinger og krav svømmeklubben ønsker løst.

Gruppen tilgang, til at skabe et funktionelt program der gavne svømmeklubben, var ved at skabe overblik over forretning strukturen og forretningens fremgangsmetoder. Fra starten af projektet blev der fremstillet flere forskellige diagrammer for at hjælpe med forståelsen af svømmeklubbens organisering. De forskellige diagrammer bliver diskuteret i afsnittet *Kravspecifikation.*

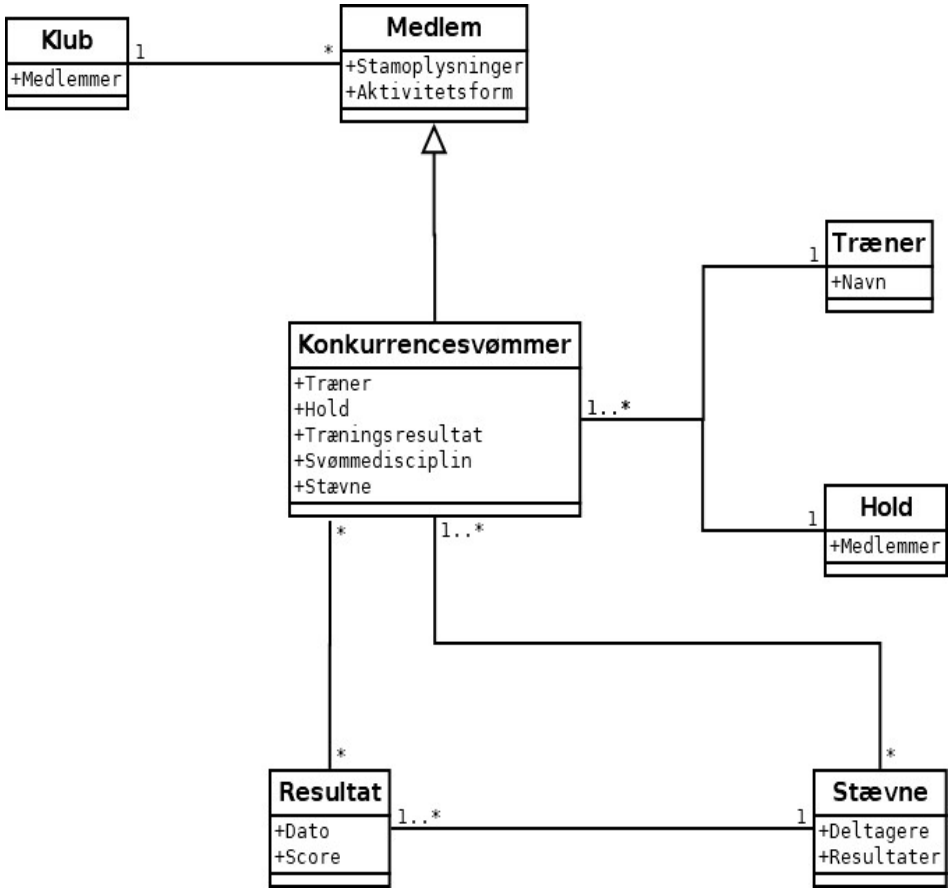
Der er blevet, under udviklingen af programmet, gjort brug af Pair-Programming og Test-driven Development. Pair-Programming, var med til at sikre bedre kodekvalitet ved at vi havde flere øjne på det givne problem. Vi brugte TDD for at skabe bevidsthed omkring hvad de forskellige metoder og klasse skal gøre og arbejder med. Derudover gav TDD også større sikkerhed omkring refactorering af koden, hvilke betød at vi altid kunne være sikre på at vi stadig udførte det samme arbejde, bare pænere eller hurtigere.

Kravspecifikation

Domænemodel

I starten af projektet udarbejdede vi en domænemodel for svømmeklubben Delfinen. Det blev klart for os under arbejdet med modellen, at det vigtigste element var konceptet *Konkurrencesvømmer* da det er dette element der har de fleste interaktioner og indflydelse på andre dele af projektet.

I vores model er *Træner* trukket ud som værende en speciel entitet, hvor i vores endelige projekt så er træner bare en speciel type af medlem. Ligeledes for *Hold*, i projektet fremtræder hold ikke som værende en reel klasse men derimod som en implicit del af medlem i form at valget af discipliner.

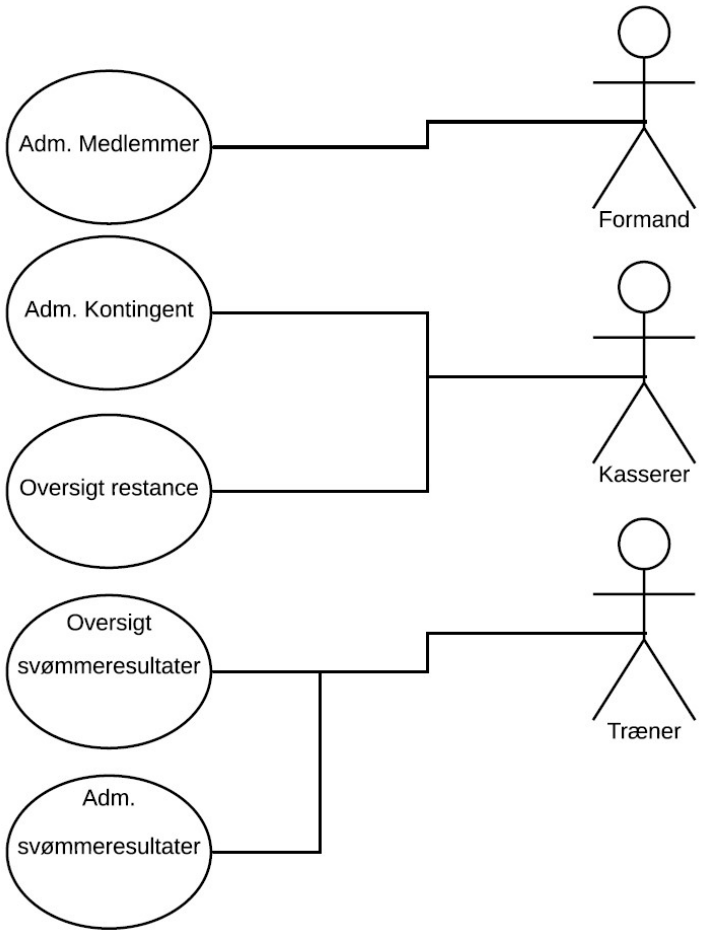


Figur 1 Domænemodel

Use-case diagram

Efterfulgt arbejdet med domænemodellen, udarbejdede vi også et use-case diagram der illustrerer hvilke aktører og use-cases der indgår i forretnings struktureren.

I vores projekt har vi ikke arbejdet med at adskille arbejdsopgaverne. For vores program betyder det at alle der har adgang til programmet at varetage alle use-cases vist i det følgende diagram: *Figur 2 Use-case diagram*.



Figur 2 Use-case diagram

Ikke-funktionelle krav

Fra starten af projektet var der stillet en liste af krav, listen er som følgende:

1. Flere klasser, indholdene indkapsling
2. Logisk opdeling af relaterede klasser i packages
3. Relevante datastrukturer såsom Array/ArrayList
4. Nedarvning og/eller interfaces
5. Gemme data i en database (persistens), således, at data bevares/hentes.

6. Simpel tekstbaseret brugergrænseflade
7. Relevante unit tests

De er ingen af de opstillede krav, der ikke er blevet udført. Vi har sikret at alle klasser ikke har unødvendige offentlige metoder, det vil altså sige at vores interne metoder ikke kan tilgås af andre end klassen selv.

Alle vores klasse er logisk inddelt i forskellige packages, for at sikre et bedre kontrol-flow. Vi har gjort brug af følgende packages for inddeling: *core, storage,* og *ui*.

Core: Denne package blev brugt til al forretningslogik, det vil sige vores klasse der omhandler medlemmer, resultater.

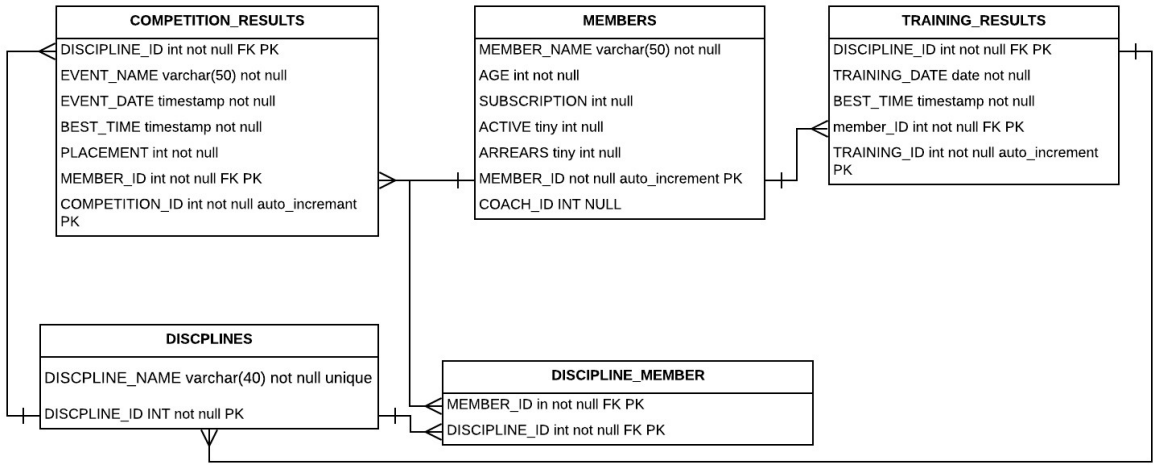
UI: Alle klasse der arbejdet med brugergrænseflader.

Storage: Alt der gør brug af vores lagersystem, i dette projekt MySQL database, er blevet isoleret i denne package, hvor det primære bindeled med andre packages er interfacet *Storage.*

ER-diagram

Et af kravene for projektet var at der skulle gøres brug af en MySQL database for at gemme persistent data. Digrammet *Figur 3 Entity Relation diagram*, viser vores tabeller over de data vi ønsker skal gemmes fra en session til en anden. Derudover viser tabellen også hvilke relationer der er i mellem alle vores entiteter.

For at kunne håndterer at et medlem kan have flere forskellige discipliner og at flere discipliner kan være tilsluttet flere medlemmer, dannede vi et bindeled *DISCIPLINE_MEMBER* som afhjalp problematikken.



Figur 3 Entity Relation diagram

Afgrænsning

En afgræsning vi havde for vores implementering af projektet, var at kunne skifte fra en konkurrencesvømmer til almindelig svømmer, og ligeledes den anden vej. I vores implementering er det muligt at skifte navn, alder, medlemskabs status, men ikke discipliner og træner. Med yderligere arbejde ville det også være muligt at implementere denne komponent.

Status

Projektet er færdiggjort og det resulterende system er klar til brug.

Det udarbejdede system kan håndtere de ønskede funktioner, og er robust så der ikke forekommer programstoppende fejl. Systemfejl bliver håndteret på en måde der gør at brugeren ikke skal genstarte systemet for at kunne fortsætte. Systemet gør brug af diverse fejlmeddelelser, der giver brugeren feedback omkring problemet.

Sammen med projektet er der også tilknyttet to MySQL script filer, en der sætter database op og en der fylder databasen med data. Ved at kører de to scripts kan alle der vil gøre brug af det udviklede system, danne en database der stemmer overens med hvad systemet forventer. Det forventes dog at enhver bruger af programmet selv udfylder informationer *user* og *password* i klassen *SQLConnector*, for at kunne forbinde til databasen.

Integrationstests

For at kunne teste om vores system pålideligt vil kunne integrere korrekt med en MySQL database gjorde vi brug af integrationstest. Disse tests blev udarbejdet ved at prøve at sende SQL-querys til databasen og derefter tjekke om query'en blev udført. For at sikre at vores database altid startede fra et samme udgangspunkt, gjorde vi brug af vores to SQL-scripts. Vi læste filerne ind i vores program og linje for linje sendte disse linjer afsend som querys. Ved brug af denne tilgang blev det muligt at starte fra det samme startpunkt over flere forskellige tests.

Commits

Da vi har arbejdet primært med Pair-Programming i forbindelse med udviklingen af systemet, er det ikke ligetil at se hvor meget hvert medlem har tilføjet til projektet. Derfor vil det ikke være repræsentativt at se på vores Github Repository commits. Ligeledes har det også haft indflydelse på statistikken hvordan commits er blevet lavet, commits lavet igennem en IDE bliver ikke altid registreret hos den rigtige bruger. I vores tilfælde har vi f.eks. fem *contributors*, på trods af vi kun er en gruppe på fire.