LOG YOUR WORKOUT WITH
LOGOUT

“Et spennende, nyskapende og revolusjonerende produkt for den jevne urbane innbygger”

Dokumentasjon for Treningsloggbok

Prosjektoppgave i TDT4145 - Innlevering 2

Av og med: Tobias Skjelvik, Henrik Syverinsen, Jesper Brynildsen og Magnus Rand

---

Forord:

Følgende dokumentasjon vil fungere som en beskrivelse av prosjektet, samt som en portal inn i dypere forståelse av programmet og dets struktur, komponenter og formål.

Innhold:

1.1 Beskrivelse av applikasjon:

Formål:

Formålet med applikasjonen er å tilby en måte for brukeren å loggføre sine treningsøkter i et system, slik at personen kan holde oversikt over fremgang i treningen.

1.2 Klasser:

1.2.1 DBconn:

Beskrivelse:

DBconn fungerer som en tolk mellom Java og SQL. Her opprettes en kobling til databaseserveren gjennom metoden `connect()`. Brukernavn, passord samt URI er her hardkodet inn, noe man i et reelt prosjekt ikke ville praktisert.

1.2.2 RegCtrl:

Beskrivelse:

RegCrtl fungere som en hjelpe eller «datainput»-klasse. Ved metodene navngitt på følgende måte: `regXXX()`, legger man inn data i databasen. Hver klasse er tilpasset etter hva den skal ta inn, slik at data blir riktig representert i databasen.

1.2.3 GetCtrl:

Beskrivelse:

GetCrtl fungere som en hjelpe eller «dataoutput»-klasse. Ved metodene sine sjekker den enten om data som taes inn eksisterer i databasen eller printer ut informasjon fra en spesifisert rad.

1.2.4 Main:

Beskrivelse:

main fungerer som en kobling mot «Terminalen» i operativsystemet. I praksis betyr dette at klassen vil fungere som vårt tekstbaserte brukergrensesnitt. Det er her en bruker registrerer seg, logger inn, legger inn og henter data og logger seg ut.

2.1 Implementert funksjonalitet (*Use cases*):

2.1.1 Liste over *Use cases*:

1	Registrere apparater, øvelser og treningsøkter med tilhørende data.
2	Få opp informasjon om et antall n sist gjennomførte treningsøkter med notater, der n spesifiseres av brukeren.
3	For hver enkelt øvelse skal det være mulig å se en resultatlogg i et gitt tidsintervall spesifisert av brukeren.
4	Lage øvelsegrupper og finne øvelser som er i samme gruppe.
5	Et valgfritt use case som dere selv bestemmer. For oss: *Person-entitet*

2.1.2 Hvilke:

Alle *Use cases* er implementert i systemet vårt.

2.1.3 Hvordan realisert:

Alle *Use cases* vil utføres gjennom «main»-klassen vår, men funksjonalitet foregår åpenbart i bakgrunnen.

2.1.3.1 *Use case* 1:

RegCtrl tar seg av all registrering, enten det gjenlder økter, øvelser eller apparater. Dette er, som nevnt i 1.2.2, gjennom metoder for de ulike entitetene . Data registreres gjennom `System.in`, slik at bruker enkelt kan legge inn data gjennom tekst-UI-en.

2.1.3.2 *Use case* 2:

Implementert gjennom metoden `load()` og `registrerOvelsegruppe()` i main. Her kan bruker spesifisere å laste inn de siste gjennomførte øvelsene ved kodeord '*N*' for n siste øvelser.

2.1.3.3 *Use case* 3:

Implementert gjennom metoden `load()` i main. Her kan bruker spesifisere å laste inn resultatlogg for i et gitt intervall ved kodeord '*resultatlogg*'. Bruker vil da videre kunne spesifisere et tidsintervall for hva en ønsker å se informasjon om.

2.1.3.4 *Use case* 4:

Implementert gjennom metodene `load()` i main. Her kan bruker spesifisere å laste inn de siste gjennomførte øvelsene ved kodeord '*N*' for n siste øvelser.

2.1.3.5 *Use case* 5:

Implementert ved at man kan logge inn som en spesifikk bruker. Dette gjøres ved å legge inn personnummeret til brukeren. Vi har valgt å ikke implementere passord-login, da dette ikke er en del av prosjektets og fagemnets læringsmål. Brukerinnloggingen er implementert enkelt ved å lagre personens ID midlertidig i en variabel.

3.1 Retroperspektiv

3.1.1 Databasen:

Det ble i etterkant endret litt på databasens oppsett i form av at flere ID-felit ble tillagt *constraint*-en «Auto increment». Dette ble gjort for å lettere kunne registrere nye brukere, apparater, øvelser o.l. uten ID-krasj.

3.1.2 Programmering:

Vi har brukt en del tid på å gjøre programmet '*fail proof*', slik at programmet ikke skal krasje, men heller gi en nyttig feilmelding. Det kan diskuteres om dette var et riktig valg da det førte til at det ble brukt mer tid enn kanskje nødvendig på Java-programmering kontra det å bruke en større del av tiden på SQL og databaselogikk. I lys av dette gjorde vi prioritering mot slutten hvor vi valgte å fokusere på å få fullstendig implementasjon av *use case*-ene på bekostning av mindre informativt feilmeldingssystem. Tross mangelen på gode feilmeldinger i deler av grensesnittet skal hele programmet likevel ikke krasje, bare enkelte metodekall hvor krasj vil resultere i å bli sendt tilbake til hovedmenyen.