TDT4145
Databaseprosjekt del 2

Gruppe 96: Lars Erik Vassbotn, Elias B. Vågan, Halvor Bakke

Om interfacet

Vi har et grensesnitt som er bygget ved hjelp av Scene Builder. Grensesnittet har en lang rekke tabs som lar deg gjøre oppgaver som tilfredsstiller de forskjellige use-casene.

Hver "scene" har ulike input-muligheter som endrer variabler som utfører enten "INSERT INTO" -kommandoer for å legge til data i databasen eller SELECT; FROM, WHERE-queries i SQL.

Filstruktur og klasser:

src/

Main.java ©
Dette er main-klassen. Her startes FXML-instansen med app.fxml.

App.fxml
Peker på AppController.java som kontroller-klasse, og har statiske attributter i tags for funksjonskall.

AppController.java ©
FXML-kontroller uten mye logikk, men tar hånd om alle input-felter og knapper for grensesnittet. Har også noen metoder som returnerer "Exceptions" tilbake til grensesnittet, slik at man ikke behøver en konsoll for å se feil.

TreningsApp.java ©
En objekt-instanse av denne klassen eksisterer i AppController.java.
Treningsapp-klassen tar hånd om all logikk utenom database-oppførsel, og eventuell logikk innenfor Okt- og Ovelse-klassene.

Okt.java ©
Brukes til å registrere økter i TreningsApp som objekter. Har toString()-metode og noe feilsjekking.

Ovelse.java ©
Det samme som Okt-klassen, men

Database.java ©
Utvider DBConn-klassen, slik at den kan instansieres som objekt.

DBConn.java ©
Abstrakt klasse, har statiske variabler som brukes til å koble til databasen på nett.

mysqlLoadDriver.java ©
Kan brukes til å teste mySQL-tilkoblingen. Denne har ingen direkte tilknytning til resten av prosjektet, men kan brukes til feilsøking under utvikling.

jars/

mysql-connector-java-8.0.15.jar:
En dependency for DBConn-klassen. Bruker til å koble til mySQL.
Er en JDBC-komponent.

setup.sql
En SQL-fil som definerer tabeller av hva som lagres og deres attributter. Denne trenger man ikke å bruke for å kjøre prosjektet. Alt er allerede ferdig satt opp i en remote phpMyAdmin-server med SQL, så filen er nå en export av denne databasen.

Om SQL og implementasjon

SQL-en er noe simplifisert fra DB1 for å gjøre det enklere å programmere systemet. Vi har fjernet noen små entiteter og gjort dem til variabler innad i andre, eksisterende entiteter. Dette er entitetene (tabellene) vi har i systemet vårt:

- Treningsøkt
- Øvelse
- Apparat
- Treningsøkt-Øvelse (relasjon)

Merk at øvelsesgrupper og den tilhørende funksjonaliteten til punkt 4 fra oppgavekravene har ikke blitt fullstendig implementert. Dette er stort sett en konsekvens av at vi underestimerte oppgavens takhøyde, og ble tvunget til å gjøre prioriteringer for arbeidet. Vi ville heller oppfylle så mange som mulig av kriteriene fullstendig, enn å prøve å oppfylle alle kriteriene enten minimalt eller på en ufullstendig måte.Gruppen vår har fortsatt brukt mye mer tid og krefter på prosjektet enn estimert, og det som er blitt implementert av funksjoner er i stor grad stabilt, og tar f.eks. høyde for feil i bruker-inputs.

Funksjonalitet og forandringer:

1. Treningsøkt, øvelse og apparat eksisterer som tabeller i SQL. Disse kan i prinsippet registreres hver for seg, men på grunn av ID-er må hver treningsøkt registreres sammen med 1 eller flere øvelser (en økt kan ikke inneholde 0 øvelser). En øvelse som registreres med apparat må kobles til et eksisterende apparat. Relasjoner mellom treningsøkt og øvelse registreres automatisk når du registrerer en treningsøkt.

2. Vi simplifiserte notatene slik at de er en tekst-attributt på treningsøkt-tabellen. Det betyr at en treningsøkt ikke kan ha flere notater. Denne måten å implementere øktnotatet på gjorde det mye lettere for oss å programmere, selv om vi skjønner at dette gjør at implementasjonen ikke helt stemmer med slik vi modellerte øktnotatene i DB1. Brukeren kan begrense **n** siste antall treningsøkter ved at vi bruker et LIMIT-clause i SQL-queriet, der **n** oppgis av brukeren.

3. Her må vi hente attributter fra både Treningsøkt og Øvelse. Disse to entitetene kobles sammen med relasjons-entiteten Treningsøkt-Øvelse. Vi bruker NATURAL JOIN til å koble disse sammen (attributtnavnene gjør det åpenbart hva som kobles sammen, som tillater bruk av NATURAL JOIN). Herfra bruker vi et SELECT-clause til å hente ut relevant informasjon, og et WHERE-clause til å hente ut en spesifikk øvelse og to datoverdier. Vi bruker INBETWEEN til å begrense utvalget til øvelser som ble utført mellom to datoverdier.

4. For å få tid til å implementere alle de andre funksjonene, implementerte vi ikke funksjonaliteten til #4. Dermed er øvelsesgrupper ikke implementert enda.

5. Selvbestemt use-case. Finne alle øvelser som tar i bruk et visst apparat. Her gjør et SQL-query slik at alle (unike) øvelser som tar i bruk det valgte apparatet vises ("DISTINCT navn" i SELECT-clauset).