PGR112 Rapport studentnummer 2050

Aug 8, 2023

Jeg har etter beste evne prøvd å implementere objektorienterte prinsipper i koden. Jeg bruke tid på å strukturere oppgaven hvor jeg har hatt fokus på krav og funksjonalitet :

Problem og formål: Målet med applikasjonen er å effektivt administrere nominasjon og stemmegivning for prisen "Årets student" under en universitetsavslutningsseremoni. Ved å tilby en brukervennlig plattform ønsker vi å muliggjøre en jevn og demokratisk prosess der studenter kan engasjere seg i å velge den mest verdige kandidaten.

Applikasjonens funksjonalitet:

- Tittel: Student valg
- Hovedaktør: Studenter /universitetsavslutningsseremoni
- Suksessscenario:

    1 Studenten logger seg inn i applikasjonen ved hjelp av gyldige påloggingsdetaljer.

    2 Etter pålogging kan studenten hente en liste over nominerte kandidater.

    3 Studenten kan avgjøre sin stemme og legge til en kommentar til sin stemme.

    4 Dersom studenten ønsker, kan de også foreslå en ny kandidat som ikke allerede er nominert.

    5 Listen over nåværende nominerte og det totale antallet stemmer for hver kandidat vil være tilgjengelig for studenten.

    6 Studenten kan navigere tilbake til hovedmenyen for å utføre andre handlinger.

Interaksjonsmønstre:
Interaksjon 1 (Innlogging):

1 Studenten logger inn i applikasjonen.

2 Studenten ser en oversikt over topp nominerte kandidater.

3 Studenten avslutter applikasjonen.

Interaksjon 2 (Nominasjon og stemmegivning):

1 Studenten ser en liste over alle nominerte kandidater.

2 Studenten foreslår en ny kandidat eller avgir sin stemme med kommentar.

3 Studenten går tilbake til hovedmenyen.

Interaksjon 3 (Utforsking av resultater):

1 Studenten ser alle nominerte kandidater og deres antall stemmer.

2 Studenten foreslår en ny kandidat eller endrer en kommentar.

3 Studenten ser hvilken nominert som har høyest antall stemmer.

4 Studenten ser stemmer for alle nominerte kandidater.

5 Studenten går tilbake til hovedmenyen.

I programmet har jeg seks klasser

I Main-klassen bruker jeg Singleton-mønsteret som er implementert ved bruk av en statisk blokk for initialisering av VoteSystem-instansen, og ved å ha en privat konstruktør. Dette sikrer at det kun eksisterer én instans av VoteSystem-objektet.

MainMenu-klassen har ansvar for å vise hovedmenyen, håndtere brukerens valg og kalle på de relevante funksjonene i VoteSystem-objektet og Nominees-klassen.

Nominees-klassen har ansvar for å vise valgmulighetene for nominerte, håndtere brukerens valg og kalle på relevante funksjoner i VoteSystem-objektet.

Nominees-klassen tar imot et VoteSystem-objekt i konstruktøren, og det er ikke avhengig av detaljene i implementasjonen av VoteSystem. Her har jeg brukt DIP prinsippet fra SOLID.

VoteSystem-klassen har ansvar for å håndtere stemme- og kommentarfunksjonalitet, samt håndtere databaseoperasjoner.

VoteSystem-klassen har en tydelig avgrenset rolle og tilbyr spesifikke metoder for å stemme, legge til kommentarer, hente resultater osv.

VoteResult-klassen er en enkel dataklasse som inneholder informasjon om en nominert og deres stemmer og kommentarer. Dette er en enkel klasse som inneholder data uten mye logikk.

Comment-klassen er en klasse jeg ikke klarte å mestre helt. Jeg fikk ikke til å implementere kommentarene til utskriften av top nominee resultatet. Men kommentarene ble lagret til databasen.

Når det gjelder basic OOP konsepter så har jeg foreksempel ikke brukt **Inheritance** noe jeg helt sikkert burde bruke for å arve og utvide fra andre klasser. **Polymorphism** har jeg brukt indirekte når jeg lagrer VoteResult-objekter i en liste, som kan inneholde objekter av forskjellige typer som implementerer VoteResult-grensesnittet. **Abstraction** er blitt brukt på metoder som **printNomineeInformation**, **printAllVotes, og getStudentOfTheYear** for å gi et høyere nivå med tilgang til informasjon om nominerte og stemmeresultater uten å vise interne detaljer om implementasjonen. **Encapsulation** er brukt i Feltene i klassene som ofte er private og tilgjengelige gjennom getter- og setter-metoder. Klassene har metodene for å utføre spesifikke handlinger uten å avsløre interne detaljer. Som i metoder som **addNominee, incrementVotes, addComment, getCommentsForNominee** i VoteSystem-klassen, og i noen andre klasser.

Det er flere ting jeg kunne jobbet mer med som å bruke grensesnitt for visse funksjonaliteter, f.eks. for stemme- og kommentarer. Dette ville gjort koden mer fleksibel og enklere å utvide.

Jeg kunne lagt til mer omfattende validering og feilhåndtering, spesielt siden det jobbes med brukerinput og databaseoperasjoner. Da ville jeg hatt et mer robust program.

Jeg liker å ha lite kommentarer i koden og heller ha selvforklarene klasse navn, metoder osv, dette er noe jeg lærte i python blant annet for å lage clean code.

Når det gjelder databasehåndtering kunne jeg også gjort mer avanserte spørringer for å få til mer kompliksitet.

Jeg vurderte også om noen av funksjonalitetene i VoteSystem skulle deles opp i separate klasser for å følge en enda renere ansvarsfordeling.

Ellers har jeg tatt i bruk HashMap, ArrayList, konstruktør, brukt if else, while loop, Scanner, lambda osv i java.