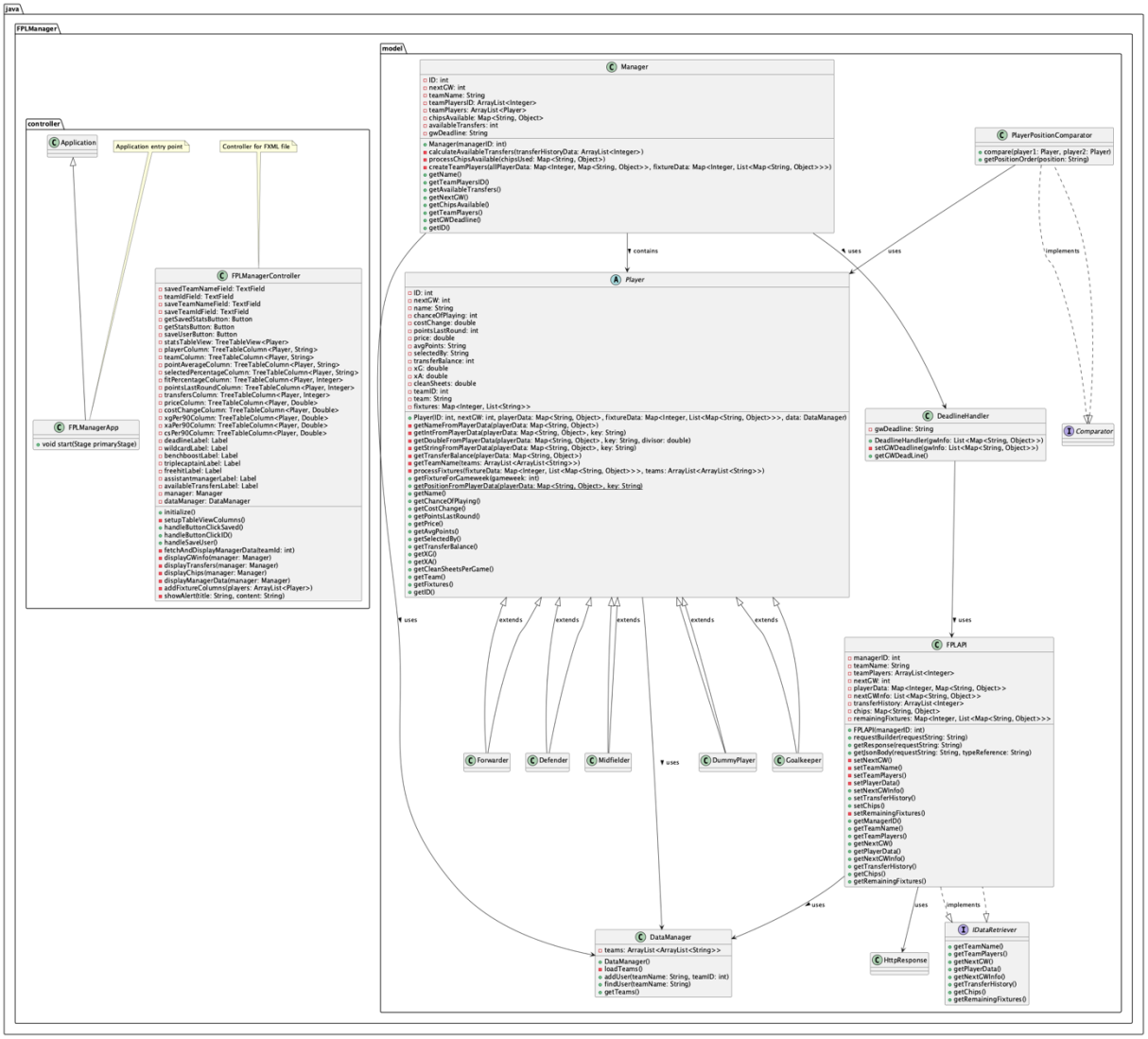
Dokumentasjon

Beskrivelse av appen

Appen jeg har laget lar brukere hente sine valgte spillere i spillet Fantasy Premier League og vise relevant statistikk for hver spiller. For å få en oversikt over spillerne sine må man enten skrive inn sin Manager ID, eller lagre sin ID med tilhørende navn, og deretter bruke brukernavnet i stedet for Manager ID-en. Dataen og statistikken hentes ved hjelp av API til Fantasy Premier League. I tillegg til statistikk for spillere, hentes oversikt over chips, tilgjengelige overganger og deadline for neste runde.

Diagram



Klassediagrammet viser en oversikt over strukturen til klassen. Metodene og attributtene til subklassene til superklassen «Player» er skjult for å spare plass. Disse subklassene arver alt

fra superklassen og har ingen ekstra metoder eller attributter. I tillegg finnes det en «resource»-mappe sidestilt med «java»-mappen. Denne mappen inneholder FXML-filen til appen og filene til å lese fra/til, men fikk dessverre ikke plass på diagrammet.

Spørsmål

Prosjektet bruker flere deler av pensum på forskjellige måter. Den abstrakte klassen «Player» blir arvet av «Goalkeeper», «Defender», «Midfielder» og «Forward». I Fantasy Premier League tilhører enhver spiller en av subklassene, men har fortsatt lik statistikk i dataen API-en tilbyr. Det er også mulig å liste opp relevant statistikk for hver posisjon ved hjelp av metoder i subklassene. Superklassen «Player» er abstrakt, ettersom enhver spiller har en posisjon, så det gir dermed ikke mening å instansiere en «Player» uten posisjon. I tillegg til å kunne kalle på subklassene om hverandre med superklassen «Player», blir også listen til en Manager over valgte spillere sortert på bakgrunn av posisjon på banen. Dette blir gjort ved hjelp av «Comparator»-grensesnittet. Grensesnitt blir også brukt i form av «IDataRetriever», som definerer en mengde metoder en klasse som tilbyr data skal ha. «Manager»-klassen henter data på bakgrunn av dette grensesnittet, som gjør det mulig å hente data på andre måter enn gjennom «FPLAPI»-klassen, som for eksempel ved hjelp av lesing fra fil. Delegering blir brukt til en viss grad, ved at «Manager»-klassen delegerer oppgaven med å hente data (til «FPLAPI»-klassen) og kalkulere deadline-tidspunktet (til «DeadlineHandler»-klassen). Lesing av fil blir dekket ved hjelp av InputStream/OutputStream og Scanner/Writer. Ellers blir mindre deler av pensum som Exception, Collection-rammeverket, synlighetsmodifikatorer, bruk av this osv. også flittig brukt i prosjektet.

Observatør-observert-teknikken blir ikke brukt i prosjektet, men kunne blitt brukt til for eksempel å legge merke til når en spillers «Chance-of-Playing» blir redusert, for å varsle om skader som har forekommet. Delegering kunne blitt brukt mer aktivt ved å sende bort oppgaver som å kalkulere tilgjengelige chips, kalkulere tilgjengelige overganger og lage spillerlisten til andre klasser enn «Manager»-klassen.

Programmet forholder seg godt til Model-View-Controller-prinsippet. Filstrukturen forholder seg til prinsippet ved å skille kategoriene og inneholder egnede filer. Det er minimalt med logikk i «Controller»-klassen som binder View og Controller-delen, samt at applikasjonen skiller data, presentasjon og logikk tydelig.

Applikasjonen har hovedsakelig blitt testet ved at jeg har sammenlignet dataen som blir listet opp for min bruker i appen med den faktiske Fantasy Premier League-applikasjonen. I tillegg

har dataen blitt sammenlignet med JSON-dataen som blir listet opp når du skriver de forskjellige URL-ene i en nettleser. Det har også blitt laget tre testklasser som tester de tre viktigste klassene i prosjektet, «Manager», «Player» og «DataManager». Klassen som tester fillagring er den mest omfattende, for å sørge for at forskjellige utfall blir håndtert på riktig måte. Det er laget relativt få JUnit 5-tester i forhold til hvor mange klasser prosjektet har, ettersom ønsket tilstand i applikasjonen sammenlignes med dynamisk data som endrer seg hele tiden, blir det vanskelig å lage tester som treffer uavhengig av tiden de blir testet i.