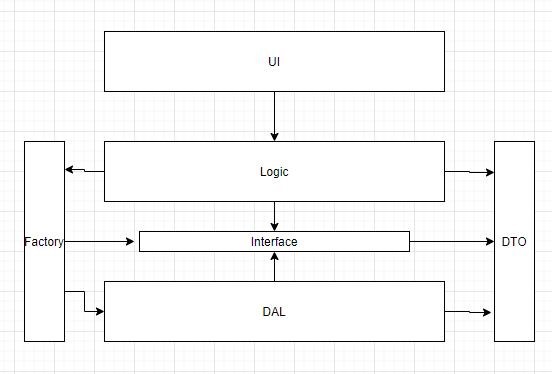
Ontwerp



Gemaakt door: Kevin Mout

Klas: DB02U.

# Architectuur(5 lagen)

Mijn architectuur bestaat uit de volgende componenten met hun afhankelijkheden.

* De UI (user interface), ook wel de presentatie laag of view genoemd.

Dit is de verbinding tussen de gebruiker en de applicatie. Via deze UI kan de gebruiker gegevens:

* Invoeren,
* Lezen,
* Aanpassen,
* Verwijderen.

In het Engels is dat create, read, update en read (CRUD).

De UI is bijvoorbeeld een webpagina, het is een informatie vertaler voor de gebruiker van de laag genaamd logic.

* De logic, ook wel de business laag genoemd.

Deze laag is het hart van de applicatie. Deze laag vervoert de opdrachten uit van de UI, neemt logische beslissingen, evalueert en voert berekeningen uit. Het verplaatst en verwerkt ook gegevens tussen de twee omringende lagen. Deze zijn:

* UI,
* En DAL (Data acces layer).

Tussen de Dal en de logic bevinden zich de:

* Factory,
* Interface,
* DTO (data transfer object).
* De Factory, ook wel Factory Method Pattern genoemd.

De Factory wordt in de logic met een interface aangeroepen. De Factory implementeert deze regels in de DAL.

* Interfaces.

In mijn applicatie maak ik gebruik van interfaces. Dit zijn contracten of regels die kan meegeven aan bijvoorbeeld een Factory. Een interface voert zelf niks uit. Hij zorgt er alleen voor dat bijvoorbeeld de juiste functies worden aangeroepen.

* DTO’s.

Dit is een object die data meeneemt tussen processen en opslaat. Het is “een containertje” met data.

* De Dal, ook wel de persistentie.

In de Dal vindt de connectiviteit tussen de applicatie en gekozen data opslag plaats. Hier roep je bijvoorbeeld een database aan. Een database is een digitale opslag van gegevens in een voor gedefinieerde vorm.

# Keuze van architectuur.

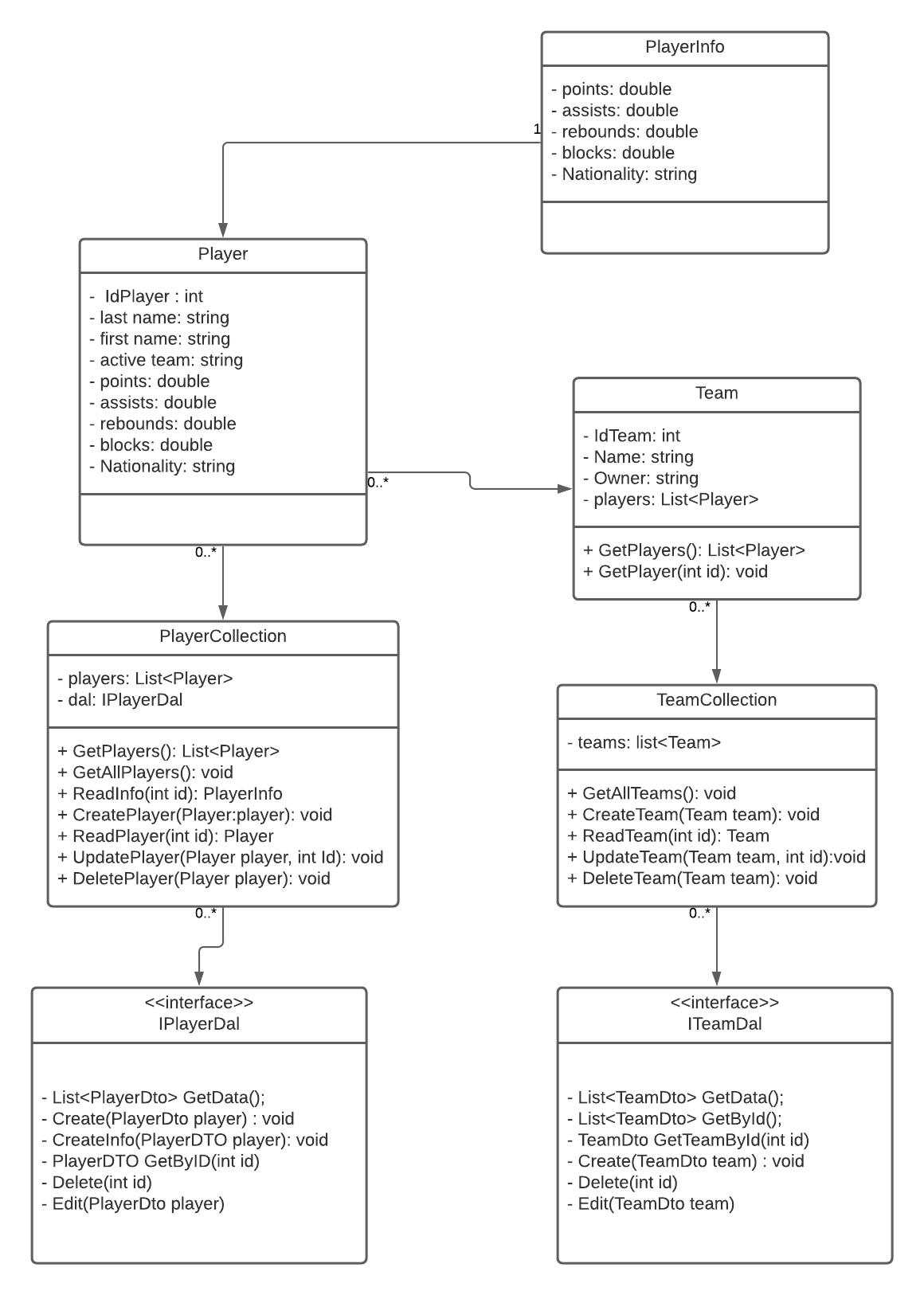
Ik heb voor deze architectuur gekozen, omdat de lagen niet afhankelijk zijn van concrete implementatie maar abstracte implementatie. De logic hangt niet af van de view. Dit wil zeggen dat het niet uit maakt welke soort UI (view) ik gebruik. We moeten gebruik van de SOLID principles en de D staat voor Dependency Inversion. Daarom heb ik een architectuur gekozen die hier gebruik van maakt. De Logic hangt niet direct of van de DAL. Er zit een abstractie tussen namelijk een interface, die wordt gebruikt door een Factory om DTO’s te vullen met gegevens van de DAL. Deze worden dan naar de Logic gestuurd. In theorie kan elke database gebruikt worden in mijn applicatie, zonder het hart van de applicatie opnieuw te programmeren. Met Dependency Injection en Inversion is het makkelijker om software aan te passen en uiteindelijk te testen. Om mijn PlayerCollection te unit testen, zou ik simpelweg een mockDal kunnen maken zonder de database aan te roepen.

# Klassendiagram

Dit is mijn klassendiagram. In de PlayerCollection class heb ik een aantal functies om data uit de database te halen. In die functies geef ik een regel van de IPlayerDal interface aan de factory. De factory haalt die gegevens op en geeft die terug aan de logic.

In mijn TeamCollection class gaat dit op dezelfde manier, maar dan geef ik de ITeamDal interface mee.

In de Player, Team en PlayerInfo class geef ik properties waar de PlayerCollection en TeamCollection class mee kunnen werken.



# Databaseontwerp

Dit is het databaseontwerp.

Met deze database kan ik de FirstName, LastName en ActiveTeam ophalen uit de player tabel. Ik kan ook met de PlayerId de informatie (PlayerInfo) ophalen.

Met Player Team Dal kan ik kijken welke spelers in welk team zit. Dit kan dan met TeamId en PlayerId.

In mijn applicatie is het mogelijk om spelers, spelerinfo’s en teams te maken.

