

FA54 - Gruppe 3

AS-Projekt 3-Schichtenmodell

Projektdokumentation

Reich, Gilad / Krüger, Rico / Biller, Andreas

# Inhalt

[**Inhalt**](#_6q8kpg3zg06l) **2**

[**0. Einleitung**](#_49tbl9eb8i27) **3**

[**1. Projektaufgabe / -ziel**](#_21cut8b2xjty) **3**

[1.1. Die 3-Schichtenarchitektur](#_gai653s1cz8l) 3

[1.2. Unsere Design-Patterns](#_3kfumj7rbtu) 4

[1.2.1. Model-View-Presenter](#_vzmb1is99kpc) 4

[1.2.2. Strategy-Pattern](#_k0kyn1ihgkxb) 4

[1.3. Unser Anwendungsproblem](#_fx301vgnzdze) 5

[**2. Das Produkt**](#_jzafa3cvz1cn) **5**

[2.1. Klassendiagramm](#_2unpvq5cq2vm) 5

[2.1.1. Benutzungsoberflächen-Schicht](#_y4fe8vjhhq4t) 6

[2.1.2. Fachkonzept-Schicht (v1)](#_5dcz5hnaw5ou) 8

[2.1.3. Fachkonzept-Schicht (v2)](#_e2xtx1seaag) 9

[2.1.4. Datenhaltungs-Schicht](#_cptrqzgcvjf5) 10

[2.1.5. Sonstige Klassen](#_kn803jpy85gc) 11

[2.2. Benutzungsoberflächen](#_r5bbvf9mzf99) 12

[2.2.1. TUI](#_kkky9mizb903) 12

[2.2.2. GUI](#_lgjqnzbfavhj) 12

[2.3. Datenhaltung (ER-Modell, Datenbank-Schemas)](#_przrccu291dr) 13

[2.3.1. ER-Modell](#_1q3ubnendwjy) 13

[2.3.2. IDataLayer-Interface](#_bp1iplwnynwc) 13

[2.4. Realisierung des Design-Patterns](#_4ny6cv10trq4) 15

[2.4.1 Model-View-Presenter](#_568zysdzc6b9) 15

[2.4.2. Strategy-Pattern (v1)](#_chouy0vh9fzc) 16

[2.4.3. Strategy-Pattern (v2)](#_mosua5u5uvvi) 17

[2.5. Sonstige Funktionalität](#_tt2h2nxk1i7w) 20

[2.5.1. Search Functionality](#_60pxuostz4dc) 20

[2.5.2. Multi-Threaded Database Calls](#_mr6ajuenz2cd) 20

[2.5.3. Logging System](#_o49sde85dvbe) 20

[2.6. Sequenzdiagramm](#_i31fzjp63459) 21

[**Quellenangaben**](#_gfqn593j872e) **22**

# 0. Einleitung

In dieser Dokumentation stellen wir zwei verschiedene Versionen für die Implementierung der Fachkonzeptschicht.

Dabei ist die erste Version (v1), diejenige auf die sich der Großteil der Projektdokumentation stützt. Die zweite Version (v2) implementiert das Strategy-Pattern korrekt, wurde jedoch noch nicht umfassend getestet. Zudem existiert hierzu, aufgrund kurzfristiger Änderungen noch keine umfassende Code-Dokumentation.

# 1. Projektaufgabe / -ziel

Wir sollen auf Wunsch unseres Auftraggebers, Herrn Dr. Ehlert, das Entwickeln und Bereitstellen von Anwendungssystemen üben und ein Softwaresystem in C# oder Java als 3-Schichten-Architektur (mit Benutzungsoberfläche, Fachkonzept und Datenhaltung) erstellen. Diese sollen je zwei verschiedene Formen erhalten und jeweils untereinander Austauschbar sein. Als Pflichtenheft soll eine kurze Projektdokumentation mit ersten Skizzen, Modellen und Meilensteinen erstellt werden[[1]](#footnote-0). Zusätzlich zum fertigen Produkt soll eine ausführliche Projektdokumentation erstellt werden.

## 1.1. Die 3-Schichtenarchitektur

Die 3-Schichtenarchitektur.besteht in Ihrer Grundidee der Schichtenarchitektur aus verschiedenen horizontalen Schichten, in denen Subsysteme beliebig aufeinander zugreifen können, während zwischen den Schichten strengere Zugriffsregeln gelten. Hierzu stellen die Schichten den darüberliegenden Schichten eine klar definierte Menge an Diensten zur Verfügung, während die internen Strukturen der Schichten, wie bei einer *Black box,* nach außen hin in der Regel nicht sichtbar sind.

Somit können dank dieser Architektur spätere Änderungen der Anforderungen leichter lokal an einzelnen Komponenten vorgenommen werden. Somit fällt es auch leichter die einzelnen Komponenten getrennt von den anderen Systemen zu entwickeln und die einzelnen Subsysteme der Schichten sind dank der definierten Standardschnittstelle gegeneinander austauschbar, ohne die Funktionalität des restlichen Systems zu beeinflussen.

Auch die Fehlerbehandlung kann in dieser linearen Ordnung der Schichten flexibler erfolgen, da uns hier die Wahl bleibt, die Fehler auf der Schicht abzufangen, auf der sie passieren, oder diese bei der Entstehung entsprechend zu Beschreiben (oder besser noch diese zu gruppieren) und dann so an die höheren Schichten weiterzugeben. Hier können eventuell auch später noch zusätzliche Überwachungsschichten implementiert werden, um entsprechend die Sicherheit oder Systemstabilität zu erhöhen.

Auch sind wir so in unserem Projekt dank der Minimierung der Abhängigkeiten zwischen den einzelnen Schichten selbst beim Entwurf relativ flexibel und unabhängig voneinander in der Lage, an unseren Implementierungen der Aufgabenstellung zu arbeiten. Auch die Wiederverwertbarkeit der einzelnen von uns entwickelten Komponenten steigt, und bei einer möglichen Portierung wird unter Umständen nur die untere Schicht benötigt.

Mögliche Effizienzverluste, da die Daten in unserer linearen Form der Schichtenarchitektur über alle Schichten weitergegeben werden müssen kann ggf. später durch das überspringen dazwischenliegender Schichten behoben werden, wenn die Anforderungen diese zusätzlichen Abhängigkeiten erlauben.

## 1.2. Unsere Design-Patterns

### 1.2.1. Model-View-Presenter

Wir benutzen das MVP-Muster, da es das zumeist empfohlene Design-Pattern für das Arbeiten mit Windows Forms ist. Die Aufgabenbereiche der einzelnen Bereiche des Model-View-Presenter-Patterns gliedern sich wie im folgenden beschrieben:

* **Model** - Das Model enthält die Fachkonzeptschicht mit der Klasse *Strategy* um die Daten aus der in der Datenbankschicht verwendeten Datenbank mit der in der in der Klasse *Strategy* definierten Sortierungsstrategie zu holen. Über das Model läuft somit die gesamte Kommunikation mit der Datenbank ab.
* **View** - Der View enthält alles, was der Benutzer im Endeffekt zu sehen bekommt, allem voran die graphische Benutzeroberfläche oder das Text-User-Interface und die angezeigten Ladeanimationen. Er kommuniziert nur mit dem Presenter über bestimmte Aktionen. Jedes Fenster hat seinen eigenen Presenter.
* **Presenter** - Der Presenter enthält die gesamte Business-Logik, man kann ihn als Mittelmann zwischen View und Model verstehen. Hier wird ein Großteil der benötigten Aufgaben erledigt. Er holt sowohl Daten vom Model und reicht diese an den View zur Darstellung weiter, als auch in umgekehrter Reihenfolge bei einer Eingabe durch den Benutzer.

### 1.2.2. Strategy-Pattern

“Das Strategie-Muster definiert eine Familie von Algorithmen, kapselt sie einzeln und macht sie austauschbar. Das Strategy-Muster ermöglicht es, den Algorithmus unabhängig von den Clients, die ihn einsetzen, variieren zu lassen.”[[2]](#footnote-1)

“Wenn es für Aufgaben unterschiedliche Algorithmen gibt, [...] dann ermöglicht es das Strategie-Muster über eine einheitliche Schnittstelle zur Laufzeit den Algorithmus zu wählen, der im aktuellen Kontext am besten geeignet ist.”[[3]](#footnote-2)

## 1.3. Unser Anwendungsproblem

Wir haben uns bei der Umsetzung der Aufgabenstellung für die Erstellung eines Teammanagers für Fußballvereine und deren Spieler entschieden. In diesem sollen einzelne Teams und deren Spieler verwaltet werden. Über zwei verschiedene Eingabemasken (GUI und TUI) sollen Vereine und Spieler erstellt, verändert und gelöscht werden können. Die Daten sollen durch zwei verschiedene Datenbanksysteme (Postgresql und MongoDB) gespeichert werden. In zwei technischen Konzepten soll die Sortierung der Daten auf unterschiedliche Weise erfolgen. Die Systeme der Schichten sollen dank der verwendeten 3-Schichtenarchitektur jeweils beliebig untereinander austauschbar sein.

# 2. Das Produkt

Unser fertiges *Softwareprodukt*, dessen *Quellcode* sowie eine ausführliche *Dokumentation des Quellcodes* und größere *Diagramme der verwendeten Klassen* sind unter den folgenden Links zu finden[[4]](#footnote-3):

* **Produkt / Quellcode:**

https://github.com/ndbiller/fa54-projekt-as

* **Quellcode-Dokumentation:**

https://ndbiller.github.io/fa54-projekt-as

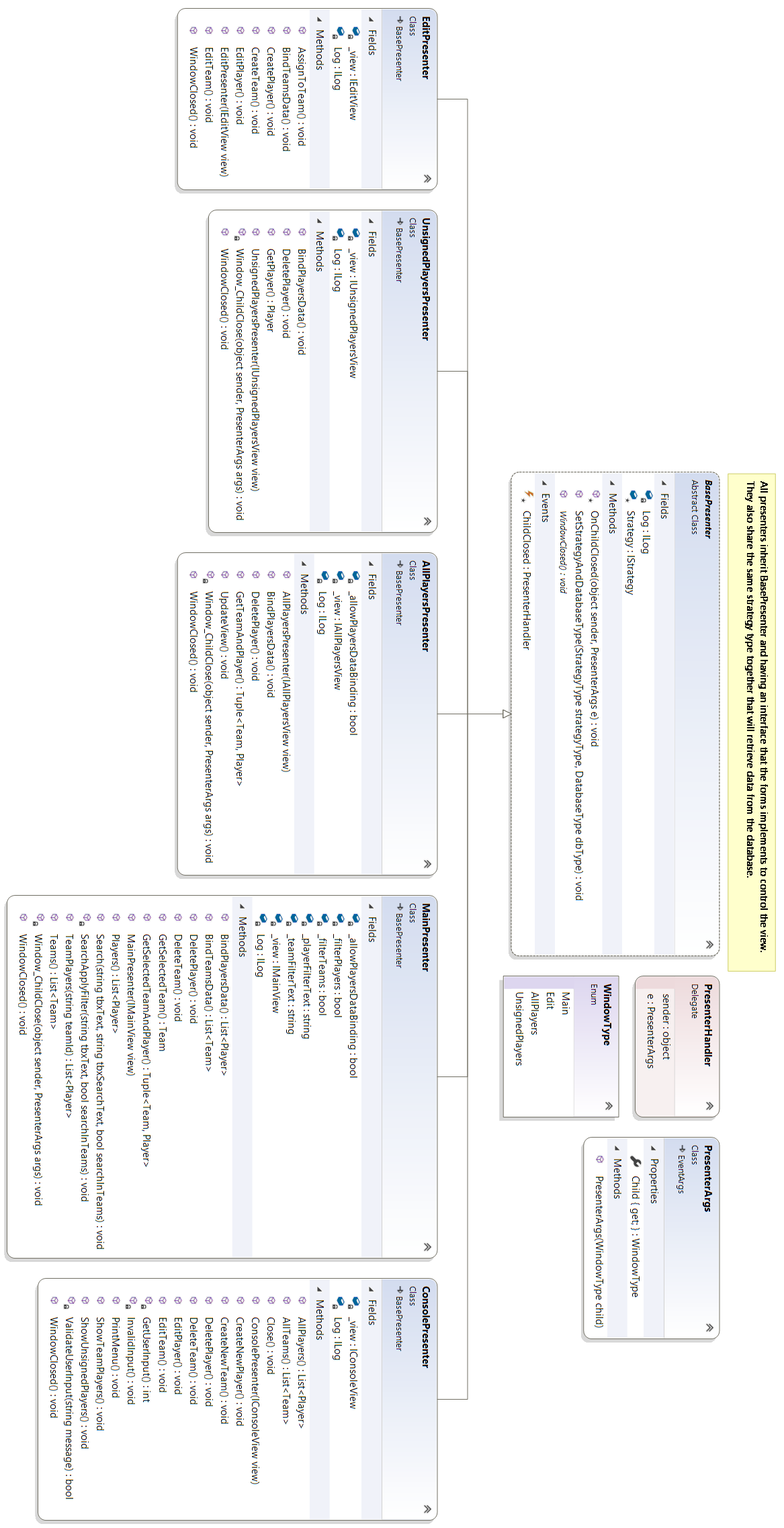
* **Klassendiagramm:**

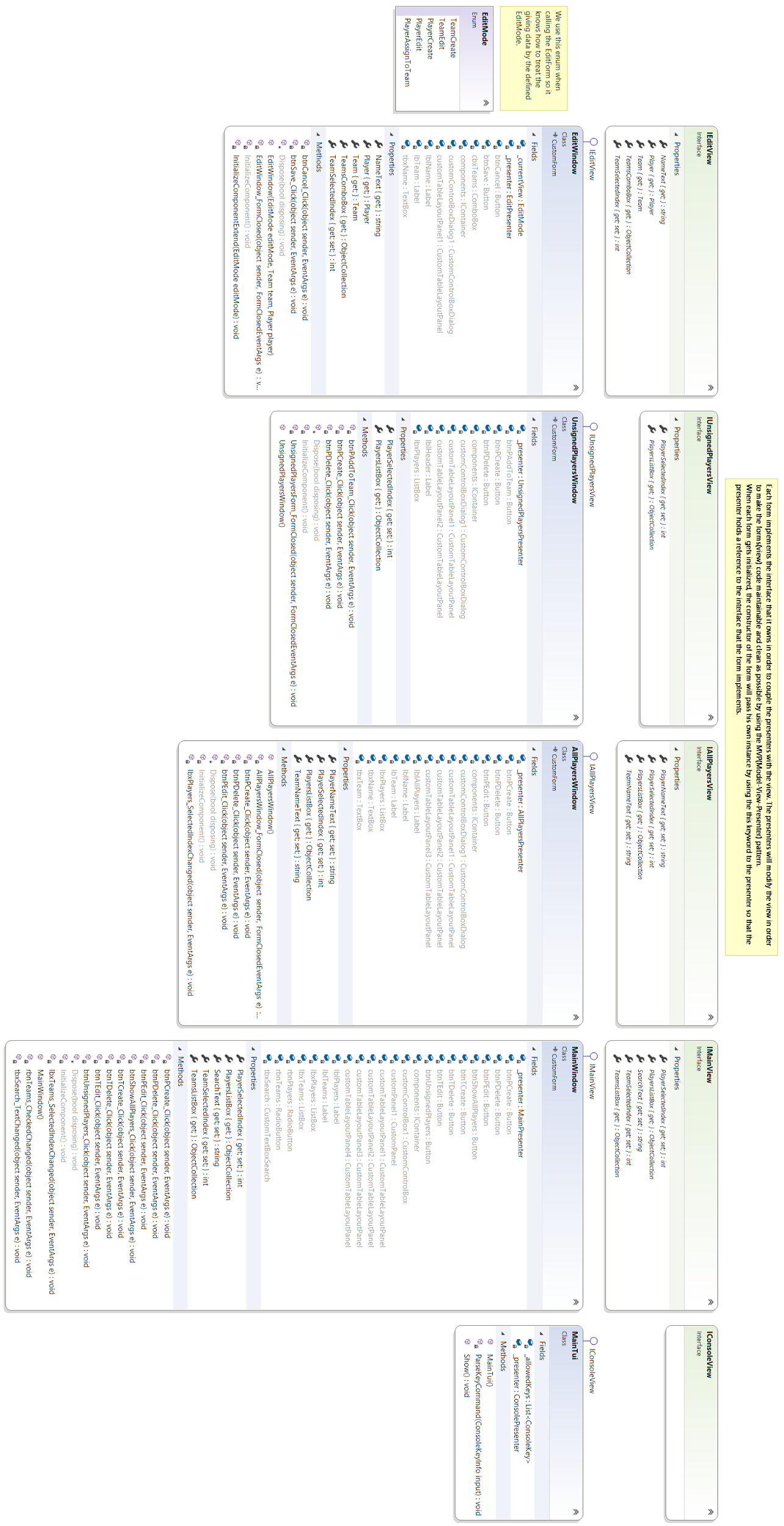
https://github.com/ndbiller/fa54-projekt-as/blob/master/pictures/ClassDiagram.png

## 2.1. Klassendiagramm

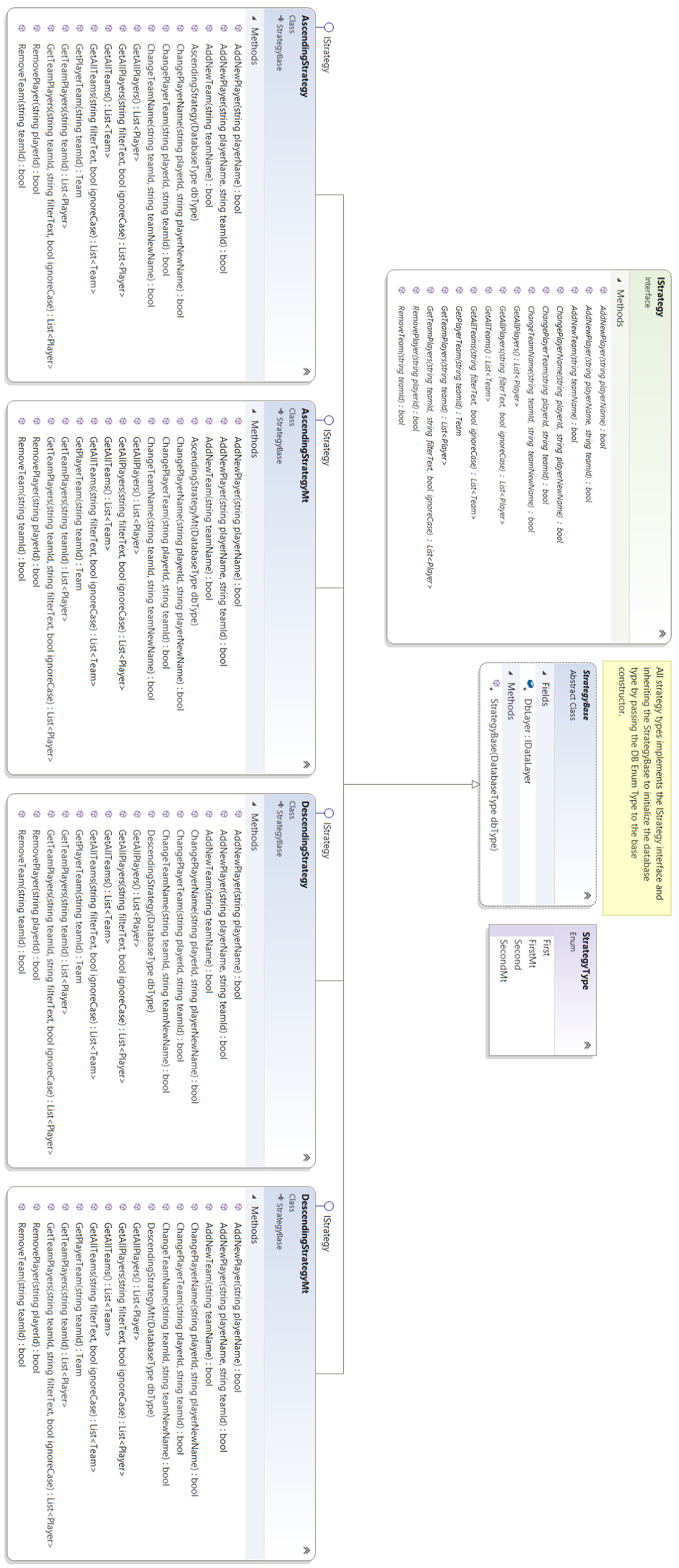
Nachfolgend finden sich die Bilder zu den verwendeten Klassen der Formulare und des Presenters in der Benutzungsoberflächen-Schicht, der Datenhaltung, des Fachkonzeptes sowie der sonstigen verwendeten Klassen. Eine besser lesbare Version ist unter der oben genannten URL zu finden.

### 2.1.1. Benutzungsoberflächen-Schicht

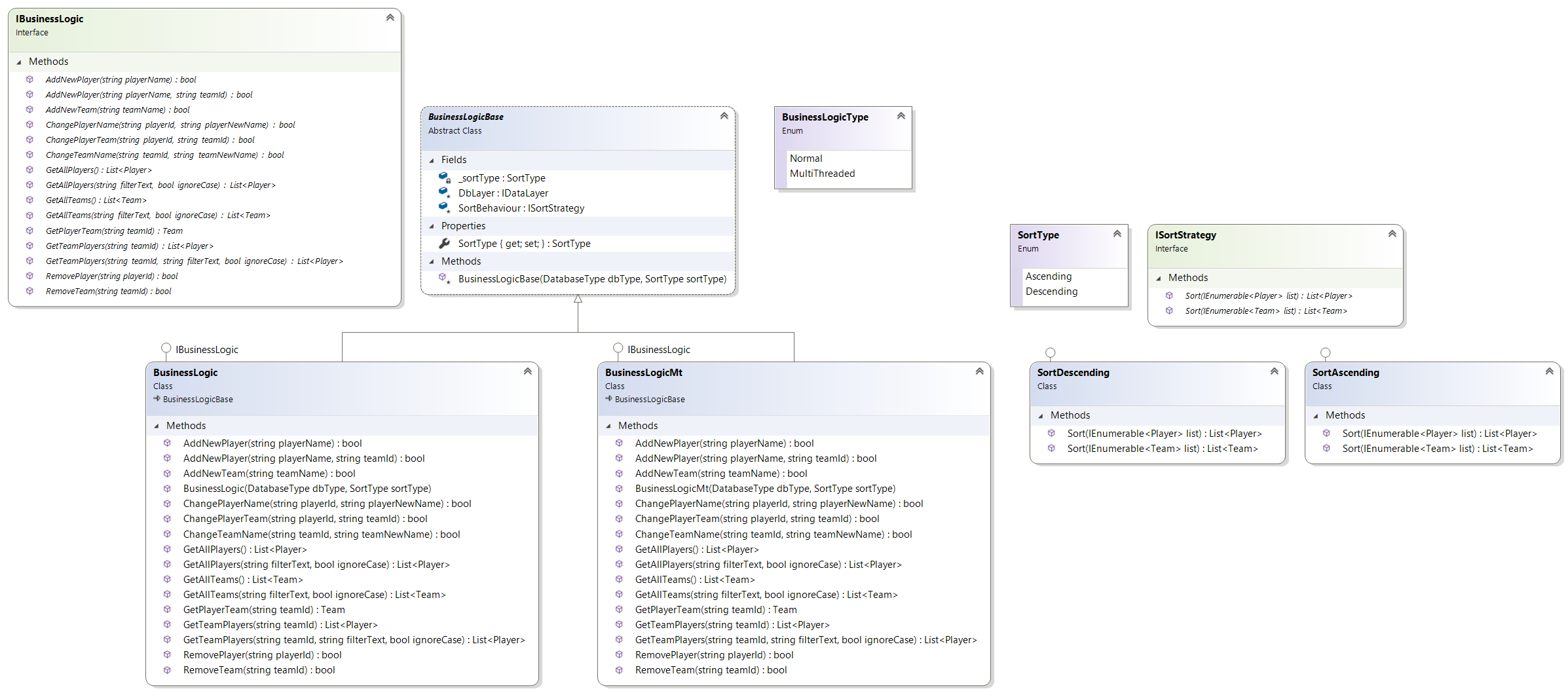




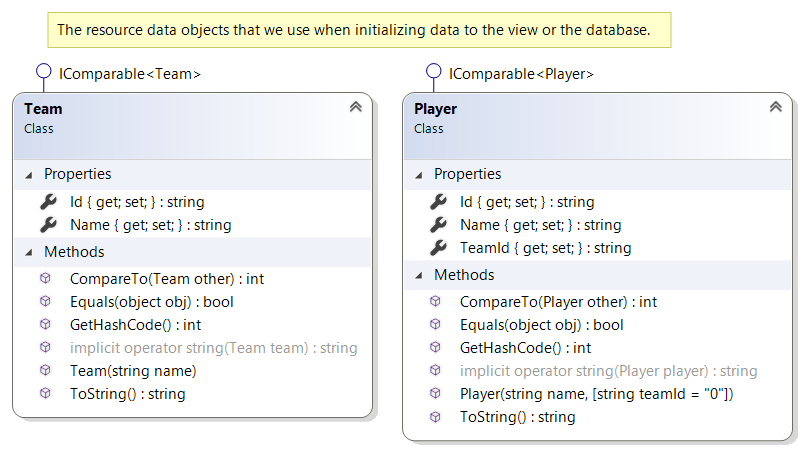
### 2.1.2. Fachkonzept-Schicht (v1)



### 2.1.3. Fachkonzept-Schicht (v2)



### 2.1.4. Datenhaltungs-Schicht



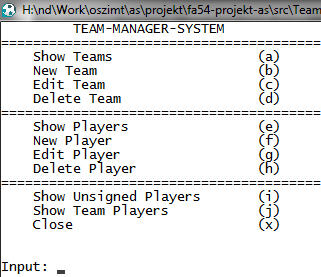
### 2.1.5. Sonstige Klassen

### 

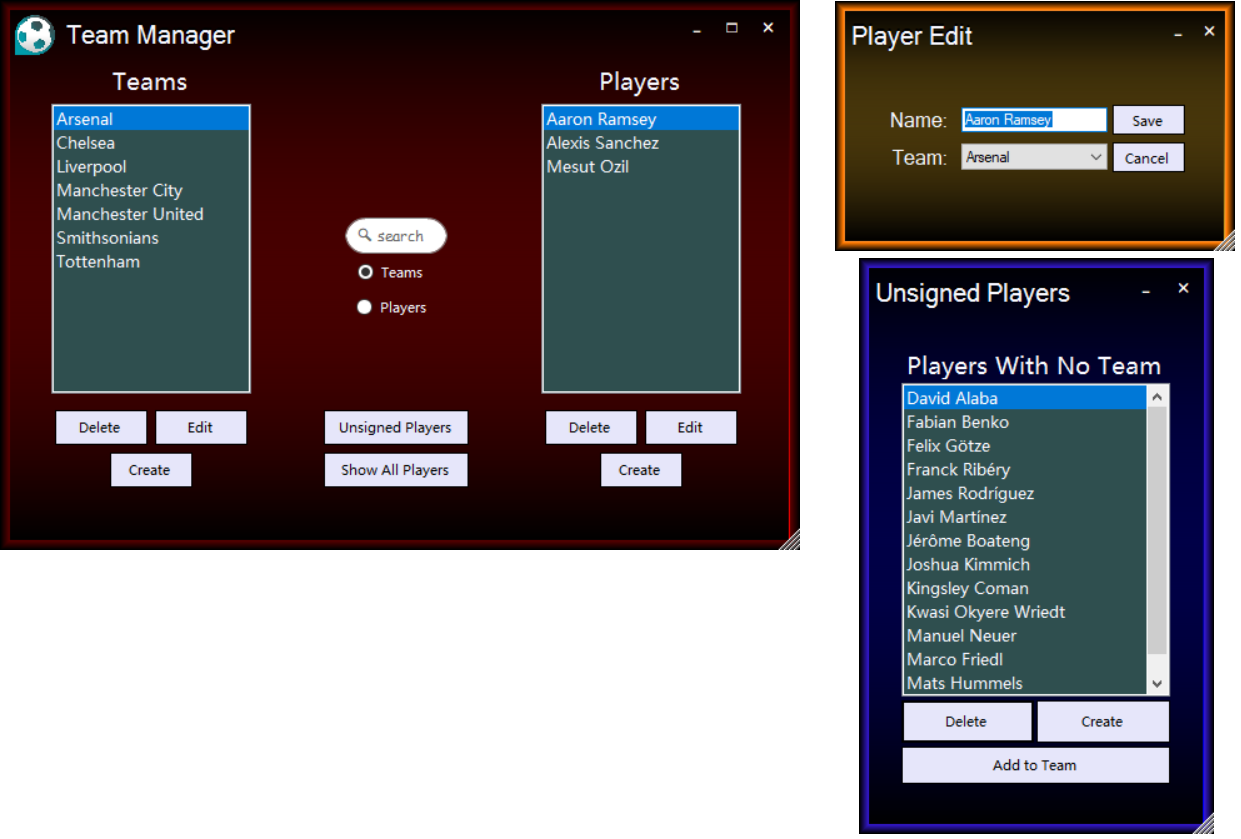
## 2.2. Benutzungsoberflächen

Das Text-User-Interface (TUI) soll die gleichen Bedienfunktionen ermöglichen, wie die grafische Oberfläche (GUI). Von uns erstellte Mock-Ups von TUI und GUI sind in unserer ersten Pflichtenheft-Dokumentation zu finden[[5]](#footnote-4).

### 2.2.1. TUI



### 2.2.2. GUI

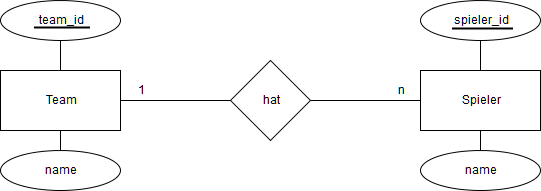


## 2.3. Datenhaltung (ER-Modell, Datenbank-Schemas)

Die Datenhaltung erfolgt in unserem Projekt über zwei verschiedene Datenbanksysteme, zum einen MongoDB, zum anderen über eine SQL-Datenbank. MongoDB und die benutzte Postgresql-Datenbank können im Projekt sowohl lokal wie auch in der Cloud betrieben werden. Beide implementieren ein Interface um die Datenzugriffe zu realisieren und sind somit im Hintergrund unabhängig von den anderen benutzten Schichten austauschbar.

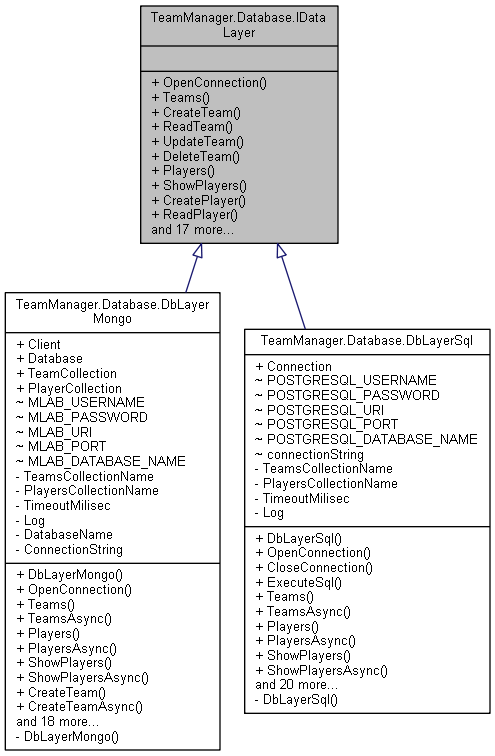
### 2.3.1. ER-Modell

Die Beziehung der Datenobjekte Team und Spieler wurden in den Tabellen der beiden Datenbanken nach folgendem ER-Modell modelliert. Der Datentyp Primärschlüssel *spieler\_id* und *team\_id* ist in MongoDB eine Zeichenfolge mit einer GUID, in Postgresql werden serielle Ganzzahlen als Zeichenfolge benutzt.



### 2.3.2. IDataLayer-Interface

Die Anbindung der beiden Datenbanken zu den übergeordneten Schichten findet über die Schnittstelle *IDataLayer* statt, welche von beiden implementiert wird. Das Setzen der GUID erfolgt hierbei durch den im Projekt benutzten C# MongoDB Driver. Für die Postgresql-Datenbank wird im Projekt zur Datenbankverbindung Npgsql eingesetzt. Die externen Datenbanken werden als kostenloses Postgresql Database Add-On bei Heroku und als kostenlose MongoDB Datenbank bei mLab gehostet und die Verbindungsdaten dieser Testdatenbanken müssen als Umgebungsvariablen gesetzt werden, um eine erfolgreiche Verbindung herstellen zu können und gleichzeitig den Quellcode sauber zu halten. Im folgenden Diagramm wird sowohl das Interface und seine Implementierung wie auch beispielhaft das verwendete Datenmodell bei MongoDB erläutert:



## 2.4. Realisierung des Design-Patterns

### 2.4.1 Model-View-Presenter

Jedes Fenster der Ansicht implementiert das Interface *IWindow*, um spezifische Komponenten der Benutzungsoberfläche für den Presenter sichtbar zu machen. Wenn eines der Fenster initialisiert wird, reicht der Konstruktor die Fensterinstanz an den zugehörigen Presenter weiter. Wenn diese Beziehung hergestellt ist, kann der View die Methoden des Presenters aufrufen, ohne selbst die Business-Logik zu implementieren. Auf diese Weise erfolgt eine klare Trennung zwischen den Aufgabenbereichen der einzelnen Komponenten des Model-View-Presenter Design-Patterns. Alle Presenter erben von der Klasse *BasePresenter* und können somit alle das selbe Interface *IFachkonzept* einsetzen und der Benutzer kann bei der Ausführung des Programmes durch Parameter wählen, welche Datenbank und welches Fachkonzept benutzt werden soll. Das Design-Pattern wird hier exemplarisch an der Klasse *MainWindow* dargestellt:

## 

## 

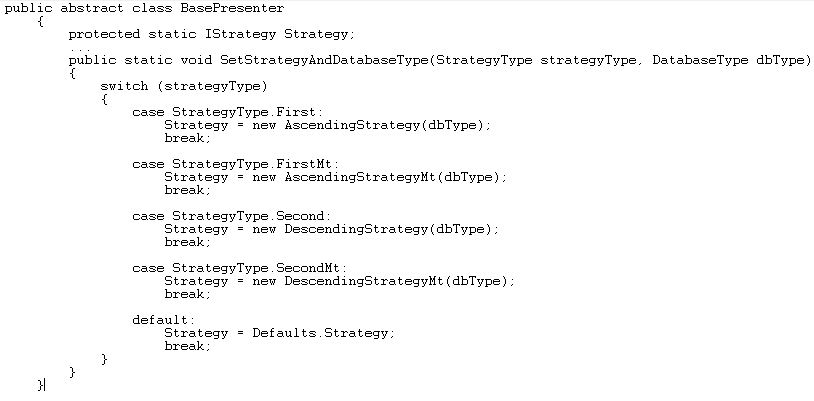
## 

## 

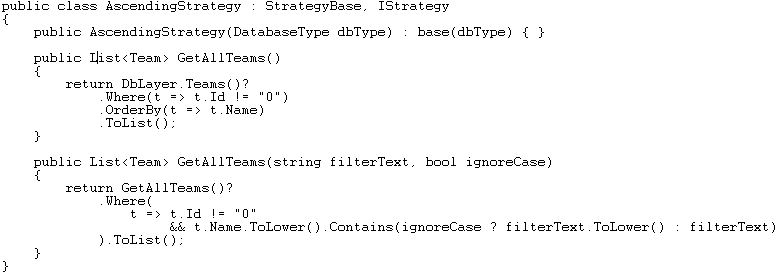
### 2.4.2. Strategy-Pattern (v1)

Jede Strategie wird durch eine eigene Klasse, welche von einer Basisklasse erbt, implementiert. Die benötigten Funktionalitäten werden von einem Interface vorgegeben. So wird die jeweilige Klasse der Strategie im Presenter gespeichert. Um die (Sortier-) Strategie und/oder die Art der Datenbankabfrage (Multithreading) zu ändern, wird dem Presenter der entsprechende Strategietyp mitgeteilt. So sind wir in der Lage sowohl die Sortierreihenfolge als auch die Art des Methodenaufrufs zu ändern.

## 



## 

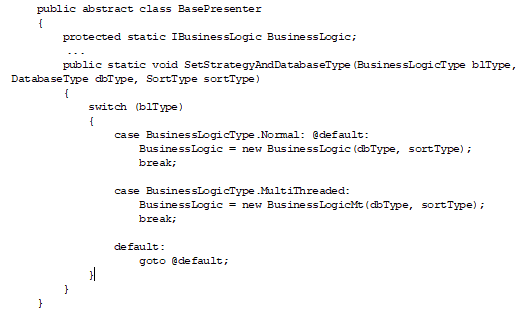


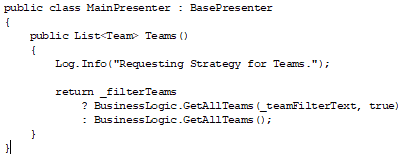
### 

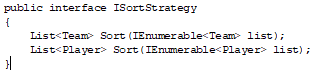
### 

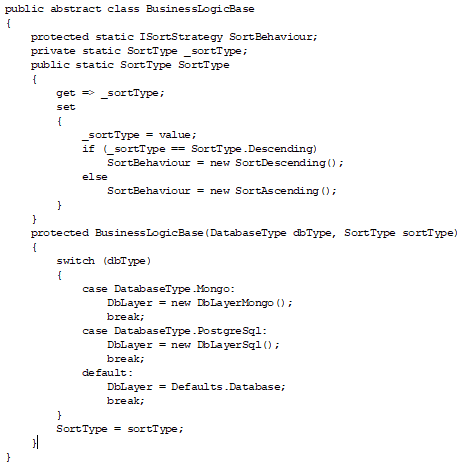
### 2.4.3. Strategy-Pattern (v2)

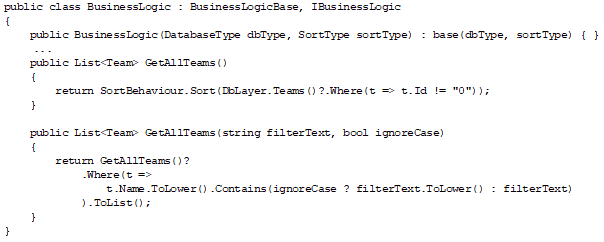
Bei dieser Implementierung wurde für das Sortierverhalten das Interface *ISortStrategy* erstellt und von den Klassen *AscendingSort* bzw. *DescendingSort* implementiert. Die Fachkonzeptklasse *BusinessLogic* implementiert eine dieser Klassen und ruft die Sortiermethode der jeweiligen implementierten Strategie auf. So lässt sich zur Laufzeit die entsprechende Strategie für das Sortieren jederzeit ändern und beliebig weitere Sortierstrategien hinzufügen.

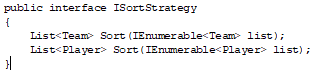












## 2.5. Sonstige Funktionalität

There are also custom features that are implemented that weren’t part of the original task:

### 2.5.1. Search Functionality

We created a custom control that kind of simplifies the usage of the search behavior and brings a user interactive gui to the user. but the whole logic is implemented in the MainPresenter. This basically allows us to filter the data that we see in the PLB(PlayersListBox) and TLB(TeamsListBox) and since the PLB depends on TLB, we can search for specific player in a team after the filter applied behind the scenes or for a team of course.

Our IFachkonzept/IStrategy contains also methods that can apply this filter for the view when the view requests for data.

### 2.5.2. Multi-Threaded Database Calls

The Database contains the async methods and the normal methods. Async is just a build in keyword in C# that uses the TPL(Task Parallel Library) and advanced managed threads contexting behind the scenes, so with our implementation, during run-time, without the need for recompiling the whole project, we can specify in the parameters that we send to the executable before the application starts that we want to use the multi-threaded calls to the database instead.

That basically helps us free-up the main UI thread so the application doesn’t freeze when there is for example a connection problem or a long operation task.

### 2.5.3. Logging System

The logging system that is implemented in the project uses the apache log4net framework. It basically helps us get an idea of what’s happening behind the scene when the application doing certain operations, so if anything goes wrong, all errors are logged into the catch block or we watch the debug/info logs before to get an idea of what exactly happened in a situation of crash for example.

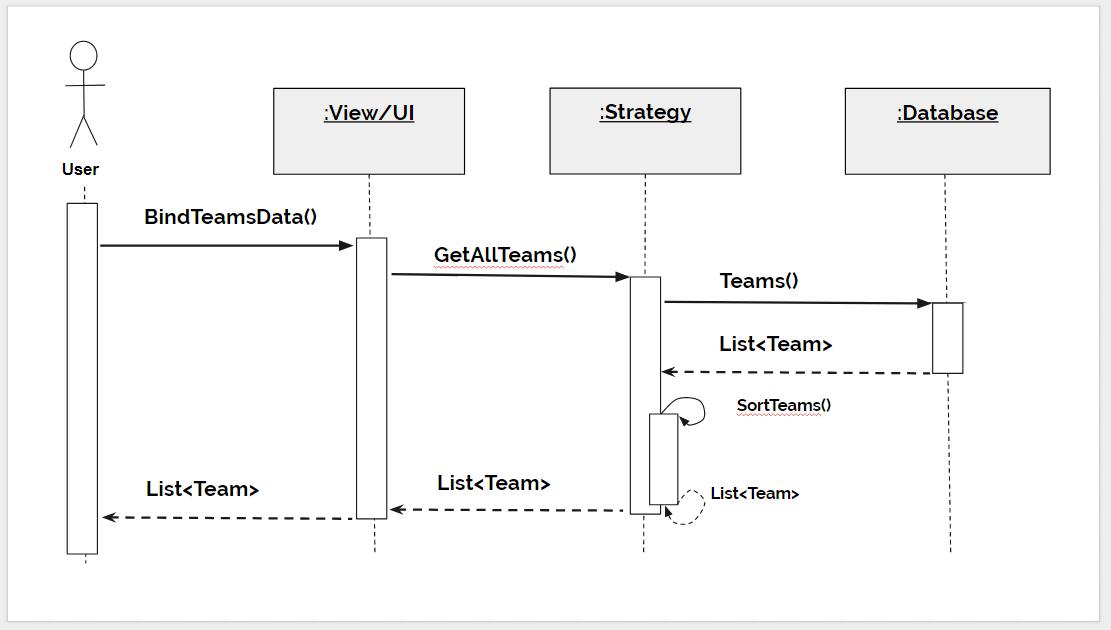
There are also 3rd party tools to use as Logger Viewer for viewing the logs that are appended to the xml log format during the usage of the application, so It’s much easier and convenient to debug things that way.

This is an open source project for example that does the job:

* <https://drive.google.com/open?id=1sN-NdMF_kZyEeCpSiwI9aKtdGogXeyyU>
* <https://github.com/zarunbal/LogExpert>

## 2.6. Sequenzdiagramm

Nachfolgendes Sequenzdiagramm erläutert beispielhaft den zeitlichen Ablauf der Kommunikation der Klassen zwischen den einzelnen Schichten bei einem Aufruf der GUI durch den Benutzer zur Anzeige aller Teams:



# 

# 

# Quellenangaben

Balzert, Helmut. Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb. Spektrum, Akad. Verlag, 2011., S. 46 ff.

Balzert, Helmut. Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering, Akad. Verlag, 2009., S. 333 ff.

Freeman, Eric & Freeman, Elisabeth. Head First Design Patterns. Aus dem Englischen von Schultern, Lars & Buch, Elke. O’Reilly Media, Inc. & O’Reilly Verlag GmbH & Co. KG, 2008., S. 22 ff.

Npgsql API Documentation. N.p., The Npgsql Development Team. Web. 03 Jan. 2018. <<http://www.npgsql.org/api/Npgsql.html>>.

C# and .NET MongoDB Driver. N.p., MongoDB, Inc. Web. 03 Jan. 2018. <<https://docs.mongodb.com/ecosystem/drivers/csharp/>>.

PostgreSQL Tutorial Website. N.p., PostgreSQL Tutorial Website. Web. 03 Jan. 2018. <<https://docs.mongodb.com/ecosystem/drivers/csharp/>>.

Model–view–presenter. N.p., n.p. Web. 03 Jan. 2018. <<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter>>.

Package Repositories:

* <https://github.com/distantcam/ConfigureAwait>
* <https://github.com/Fody/Costura>
* <https://logging.apache.org/log4net/>
* <https://github.com/mongodb/mongo-csharp-driver>
* <https://github.com/npgsql/npgsql>
* <https://github.com/dotnet/corefx>

Bildquellen:

Die verwendeten Bilder wurden mit Hilfe von Microsoft Visual Studio 2017 Professional, PlantUML und anderen Programmen von den Autoren selbst erstellt.

Application Icon:

<https://twitter.com/forzafootball>

1. Erste Dokumentation: https://github.com/ndbiller/fa54-projekt-as [↑](#footnote-ref-0)
2. Entwurfsmuster von Kopf bis Fuß (2008), S. 24 [↑](#footnote-ref-1)
3. Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering (2009), S. 96 [↑](#footnote-ref-2)
4. Zugangsdaten der Testdatenbanken gerne auf Anfrage an: andie.biller@doctena.com [↑](#footnote-ref-3)
5. Mock-Ups: https://github.com/ndbiller/fa54-projekt-as#mock-ups [↑](#footnote-ref-4)