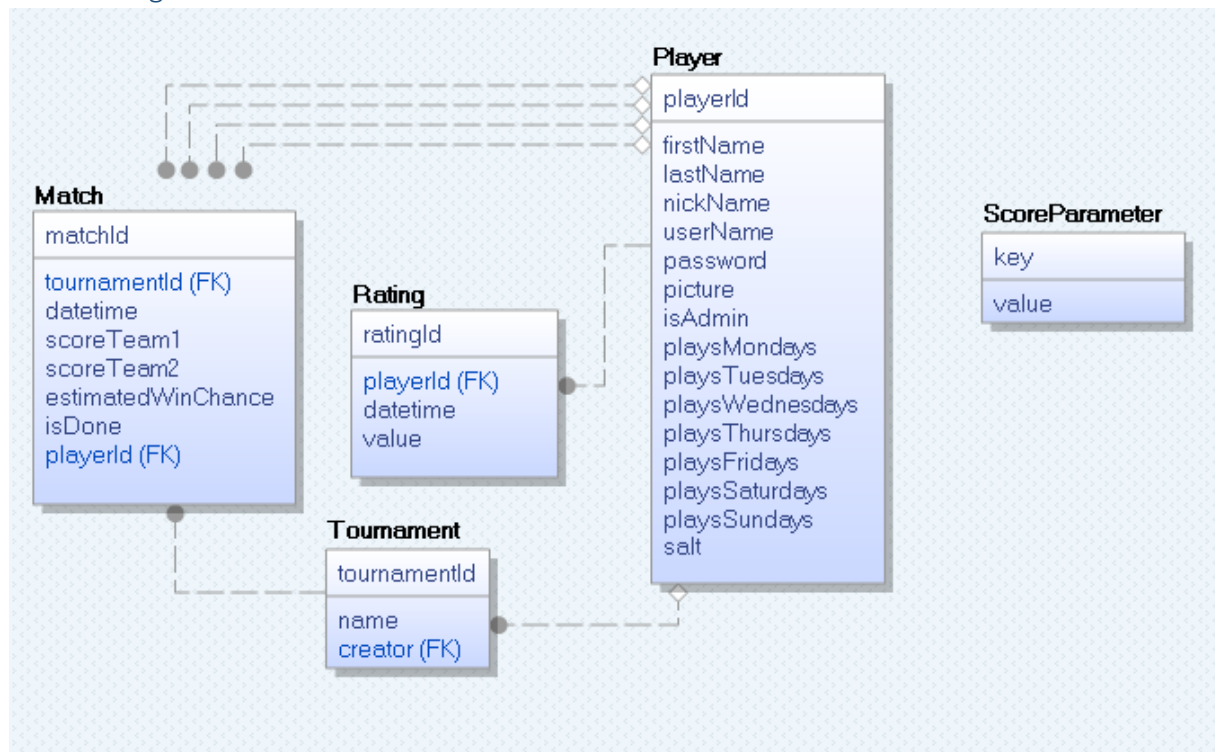


WuHu – WuzzelHub

Erste Ausbaustufe

Datenbank

Modell Diagramm



Beschreibung

Als Datenbank verwende ich ein MySQL Localdb File wie in der Übung.

Achtung! Vor Ausführung der Tests sollten die App Settings (App.config in WuHu.Test bzw. WuHu.Server) richtig konfiguriert sein (ConnectionString, DbPath und SqlPath!)

Das Datenmodell ist auf möglichst wenige Entitäten beschränkt, ohne Normalformen zu verletzen. Es deckt dabei trotzdem die Problemstellung ausreichend ab.

Ein *Player* enthält alle Informationen zu einem Spieler. Darunter fallen alle Namen, sein gehashtes Passwort und das zugehörige generierte Salz, um den Passwortstring vor dem Hashes zu salzen. Weiters ist sein Bild binär, sowie sein Adminstatus als boolean gespeichert, und an welchen Tagen er spielt ebenfalls als booleans. Dies ermöglicht ein schnelles Abfragen von Spielern an einem gewissen Tag, ohne aufwendige Joins.

Im *Rating* werden alle Wertungen von Spielern gespeichert. Diese Wertungen sind mit einem Datum versehen, sodass man nacher eine Statistik der Spielerwertungen über einen längeren Zeitraum anzeigen kann. Jede Wertung ist einem gewissen Spieler zugeordnet.

Tournament fasst eine Menge von Spielen zu einem Turnier zusammen. Diesem kann ein Name gegeben werden, und es wird auch der Ersteller des Turniers als creator gespeichert.

Im *Match* werden alle relevanten Informationen eines Spiels gespeichert. Dazu gehören die vier Mitspieler, wobei Spieler1 und Spieler2 als Team1 gegen Spieler3 und Spieler4 als Team2 spielen. *scoreTeam1* und *scoreTeam2* speichern den derzeitigen Punktestand des Spiels. Dieser kann sich im Verlaufe der Zeit ändern, sollte jedoch fix sein, sobald der *isDone* boolean auf True gesetzt wird. Dieses zeigt an, wenn ein Spiel vorbei ist. *EstimatedWinChance* wird beim Erzeugen des Spiels von der Anwendung berechnet. Es wird aus den aktuellen (also zuletzt eingetragenen) Ratings der Teams kalkuliert. Wenn beide Teams das gleiche durchschnittliche Rating haben, ist die *EstimatedWinChance* gleich null. Desto höher die Differenz vom Team1 gegenüber Team2 zu Gunsten vom Team1 ist, desto höher ist die *EstimatedWinChance*. Aus dieser Chance kann man sich dann bei Abschluss des Spiels den Punkte-Gewinn bzw. Verlust der zwei Teams errechnen.

Berechnet wird die Winchance mit der Formel

$$E_A = \frac{1}{1 + 10^{(R_B - R_A)/400}}$$

berechnet, wobei R_A und R_B das Durchschnittsrating von Team 1 bzw. Team 2 sind.

Die Punkte, die die Spieler des Team 1 dazu (oder abgezogen) bekommen, errechnet man mit der Formel

$$\text{delta_points}_A = k\text{-rating} \times (\text{Won}_A - E_A),$$

wobei *k-rating* ein beliebiger konstanter Faktor (meist ca. 15 – 35) ist, und $\text{Won}_A = 0$, wenn Team 1 verloren hat, und 1, wenn sie gewonnen haben.

Dasselbe gilt für Team 2, wobei $E_B = 1 - E_A$ ist.

In der letzten Tabelle *ScoreParameter* können alle notwendigen Parameter zur Berechnung gespeichert werden. Diese könnte man auch in einer eigenen Konfigurationsdatei speichern, aber ich habe mich für diese Version entschieden, da Datenbankzugriffe geregelter, sicherer und weniger störanfällig als Filezugriffe sind. So kann ein beliebiger User nicht so einfach auf die Parameter zugreifen oder sie verändern.

Die Parameter werden dabei einfach als key-value Stringpaare gespeichert. In der *WuHu.BL.Impl.DefaultParameter.cs* sind außerdem Default-Werte eingetragen, falls sie nicht in der Datenbank eingetragen sind.

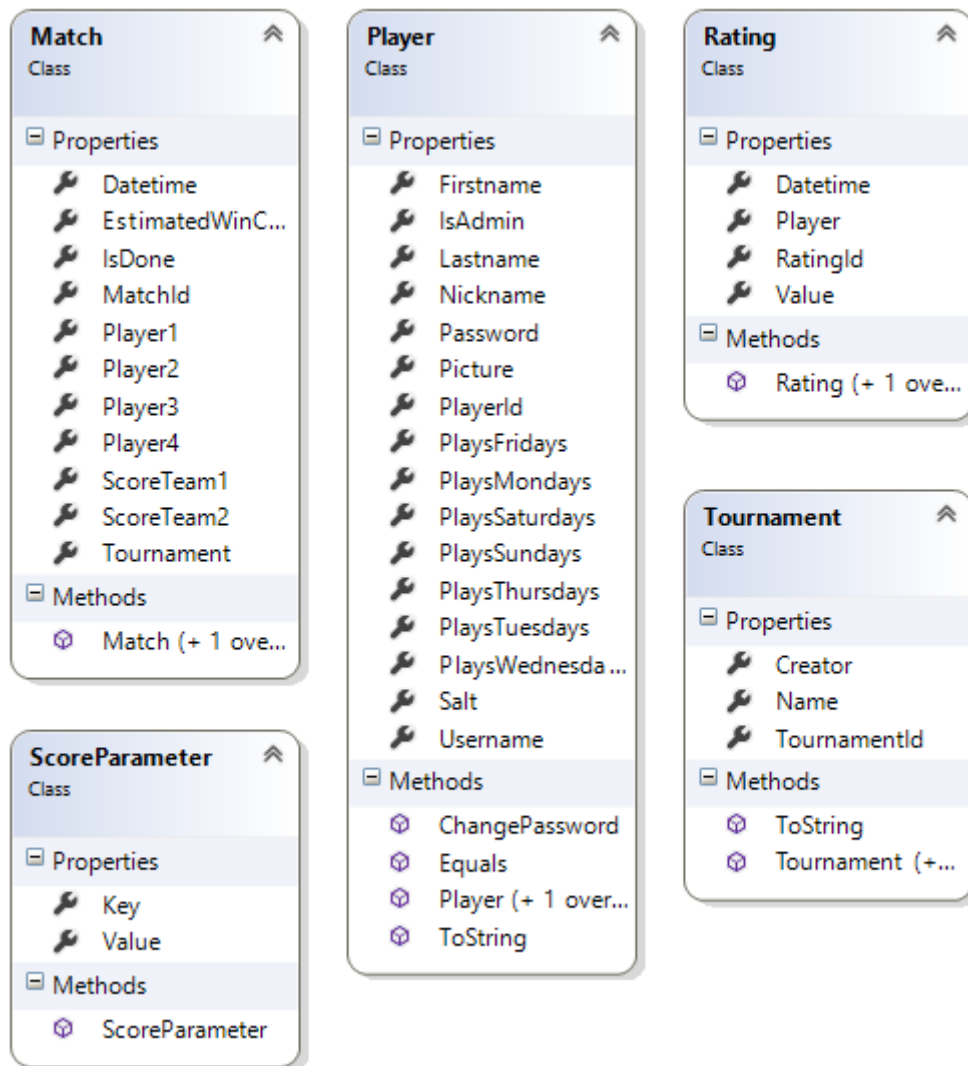
Mögliche Parameter sind:

- *initalscore*: Besagt, wieviele Punkte standardmäßig ein neuer Spieler haben soll.
- *scoredMatches*: Wieviele der letzten Matches zur Punkteberechnung hergenommen werden sollen. Zur Scoreberechnung werden dann nur die letzten *n* Matches gezählt.
- *halflife*: Damit frühere Spiele weniger in die Wertungsberechnung einfließen als weiter erst vor kurzem gespielte, kann man ein *halflife* festlegen. Spiele, die so lange zurückliegen, fließen nur mehr halb so stark in die Wertung ein usw.
- *k-rating*: Wird zur Berechnung der *delta_points* hergenommen. Ein höherer Wert bewirkt stärker fluktuierende Ratings.
- *timepenalty*: Dieser Wert bestimmt, wieviele Punkte einem Spieler (pro Monat/Woche/Tag) abgezogen werden, wenn ein Spieler länger nicht spielt. Seine Score kann jedoch nicht unter *initalscore* sinken.

Domainklassen

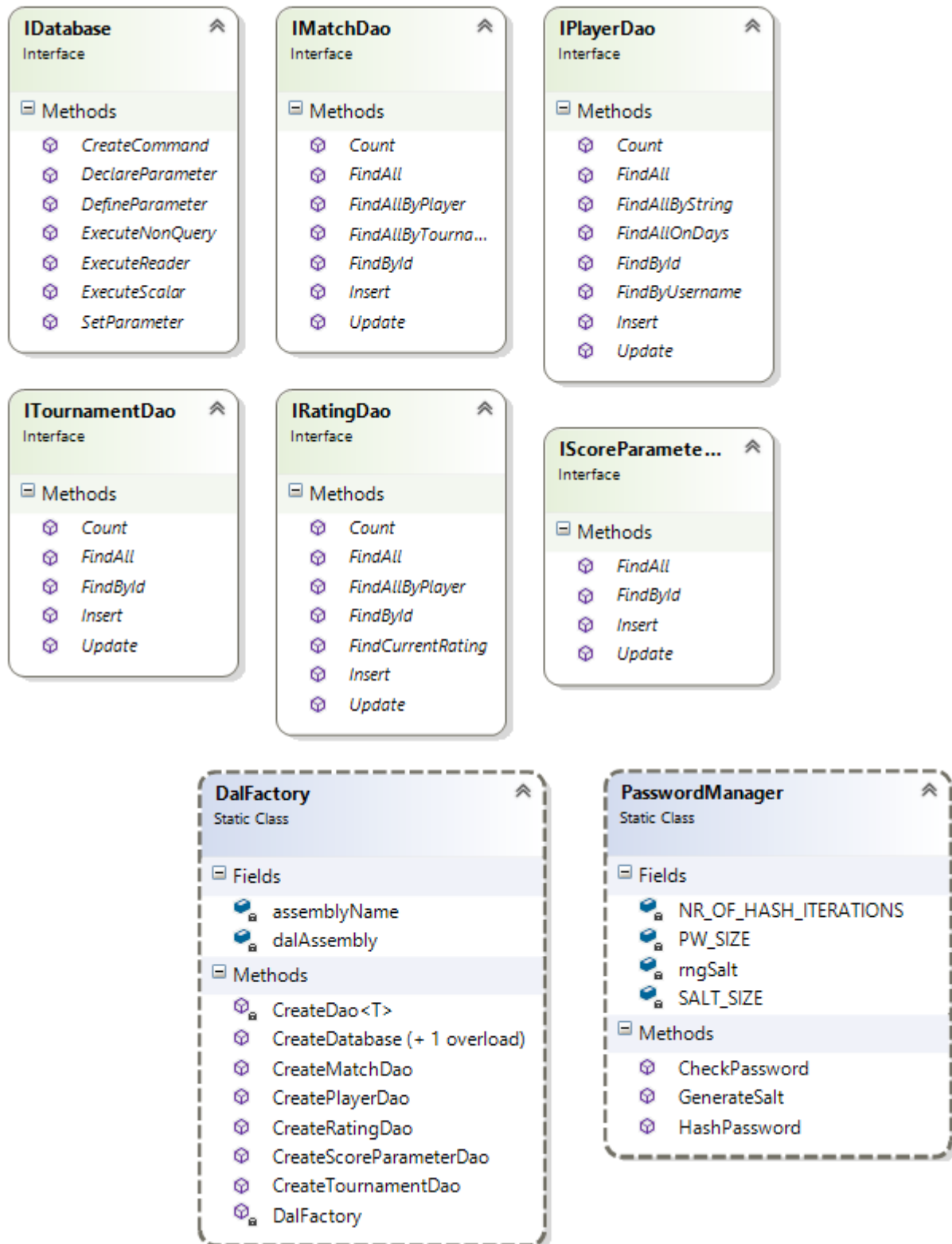
Die Domainklassen bilden die Datenbank-Entitäten im Code ab.

In der Player Klasse erstellt der Konstruktor bzw. die *ChangePassword* Methode automatisch ein Salt und hasht das gegebene Passwort mit diesem, bevor es beides als Property speichert.



Daos

Modell



DAOs Beschreibung

Die Datenzugriffsschicht ist ähnlich wie in der Übung als Interfacesammlung und dazugehörige Implementierungen für die konkrete Datenbank gegliedert. Wie man oben im Klassendiagramm sehen kann, gibt es für jede Entität ein zugehöriges DAO. Diese definieren alle Zugriffe auf die Datenbank über das `IDatabase` interface. Die konkreten DAOs und die Database selbst wird dann von der `DalFactory` zur Verfügung gestellt.

Die Methoden sind dabei für alle Klassen größtenteils die gleichen: *Insert*, *Update*, *FindAll*, *FindById* und *Count* werden von fast allen implementiert und sind selbstredend.

Dazu kommt:

Rating implementiert *FindAllByPlayer*, um den Verlauf der Wertung eines bestimmten Players anzeigen zu können, sowie *FindCurrentRating*, um das aktuellste Rating eines Spielers abzufragen.

Match implementiert zwei Methoden, um alle Matches eines Spielers oder eines Turnieres abfragen zu können.

Player implementiert zudem Methoden zur Suche eines Spielers entweder nach seinem Namen (hier wird im Vor-, Nach- und Spitznamen gesucht) sowie eine Methode zur Suche nach seinem Usernamen (für den Login). Außerdem kann man alle Spieler, die an einem (oder mehreren) gewissen Tag(en) spielen, suchen lassen.

Zuletzt gibt es eine statische Klasse `PasswortManager`. Diese enthält Funktionen zum Generieren eines kryptografisch sicheren zufälligen „Salz“ mit der Klasse `RNGCryptoServiceProvider`, sowie einen ebenso sicheren Password Hasher (`Rfc2898DeriveBytes`) zum Hashen bzw. Überprüfen eines Passwortes.

Tests

Für die Tests werden normale Microsoft Unit Tests verwendet. Diese decken in 52 einzelnen Tests den kompletten DAO und Domainklassen Code ab. Die Tests sind sinnvoll auf mehrere Testklassen aufgeteilt und testen den Datenbankzugriff (Einfügen, Updaten und Abfragen), die Erzeugung der Domainobjekte, und die Funktionen des Passwortmanagers.

Ausführung

Alle 52 Tests können erfolgreich ausgeführt werden.

▷ ✓ DatabaseTests (3 tests)	[0:00.964] Success
▷ ✓ MatchTests (12 tests)	[0:11.328] Success
▷ ✓ PasswordManagerTests (2 tests)	[0:00.547] Success
▷ ✓ PlayerTests (12 tests)	[0:06.753] Success
▷ ✓ RatingTests (10 tests)	[0:01.222] Success
▷ ✓ ScoreParameterTests (5 tests)	[0:01.685] Success
▷ ✓ TournamentTests (8 tests)	[0:00.260] Success

Code Coverage

Die oben angeführten Tests decken den gesamten relevanten Code ab.

Symbol ▼	Coverage (%)	Uncovered/Total Stmts.
▲ Total	100%	0/837
▲ WuHu.Domain	100%	0/190
▲ () WuHu.Domain	100%	0/190
▷ Tournament	100%	0/20
▷ ScoreParameter	100%	0/9
▷ Rating	100%	0/21
▷ Player	100%	0/79
▷ Match	100%	0/61
▲ WuHu.Dal.SqlServer	100%	0/596
▲ () WuHu.Dal.SqlServer	100%	0/596
▷ TournamentDao	100%	0/73
▷ ScoreParameterDao	100%	0/55
▷ RatingDao	100%	0/110
▷ PlayerDao	100%	0/143
▷ MatchDao	100%	0/145
▷ Database	100%	0/70
▲ WuHu.Dal.Common	100%	0/31
▲ () WuHu.Dal.Common	100%	0/31
▷ DalFactory	100%	0/31
▲ WuHu.Common	100%	0/20
▲ () WuHu.Common	100%	0/20
▷ PasswordManager	100%	0/20

Testdaten

Die Testdaten (2 Jahre normale Nutzung lt. Angabe) werden mit dem Pythonscript „data_generator.py“ generiert. Diese erstellt ein sql script mit Inserts, die direkt ausgeführt werden können. Die statische Klasse *TestHelper* enthält außerdem die Methode *InsertTestData*, die genau auf dieses Sql-Skript zugreift und ausführt. Dieses kann einige Zeit dauern.

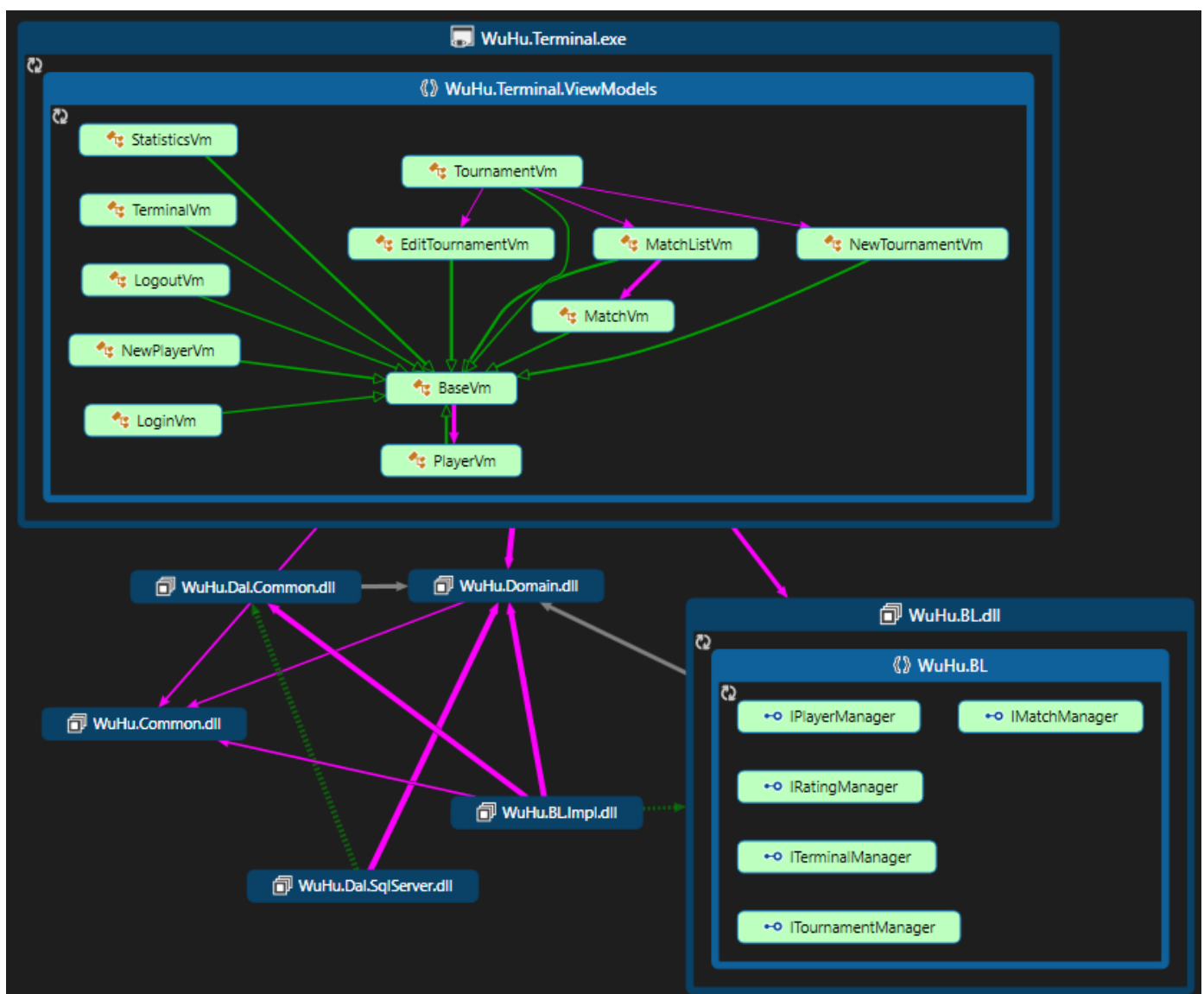
Program.cs im Startupprojekt WuHu.Server ist bereits so konfiguriert, dass es die tables erstellt und die Testdaten einfügt. Dies setzt voraus, dass die App.config Dateien richtig eingestellt sind.

Beim Ausführen der Unit-Tests wird automatisch ein Backup der Datenbank unter dem Namen „WuHuDB.mdf.bak“ bzw. „WuHuDB_log.ldf.bak“ erstellt, falls es noch nicht existiert. Falls das Backup bereits existiert, wird vor Ausführung der Tests die derzeitige Datenbank mit dem Backup überschrieben. So wird eine konsistente Testumgebung gewährleistet.

Zweite Ausbaustufe

In dieser Ausbaustufe wurde die gesamte Businesslogik sowie das Nutzerinterface als WPF-Applikation implementiert. Aus folgender Codemap kann man sehr gut erkennen, wie die einzelnen Komponenten voneinander abhängen. Die grünen (durchgehenden sowie gestrichelten) Pfeile bezeichnen dabei Ableitungen, die rosanen zeigen Verwendungen. Zum Beispiel leitet TournamentVm von BaseVm ab, aber verwendet MatchListVm, EditTournamentVm und NewTournamentVm in seiner Implementierung. MatchListVm wiederum verwendet und verwaltet MatchVms.

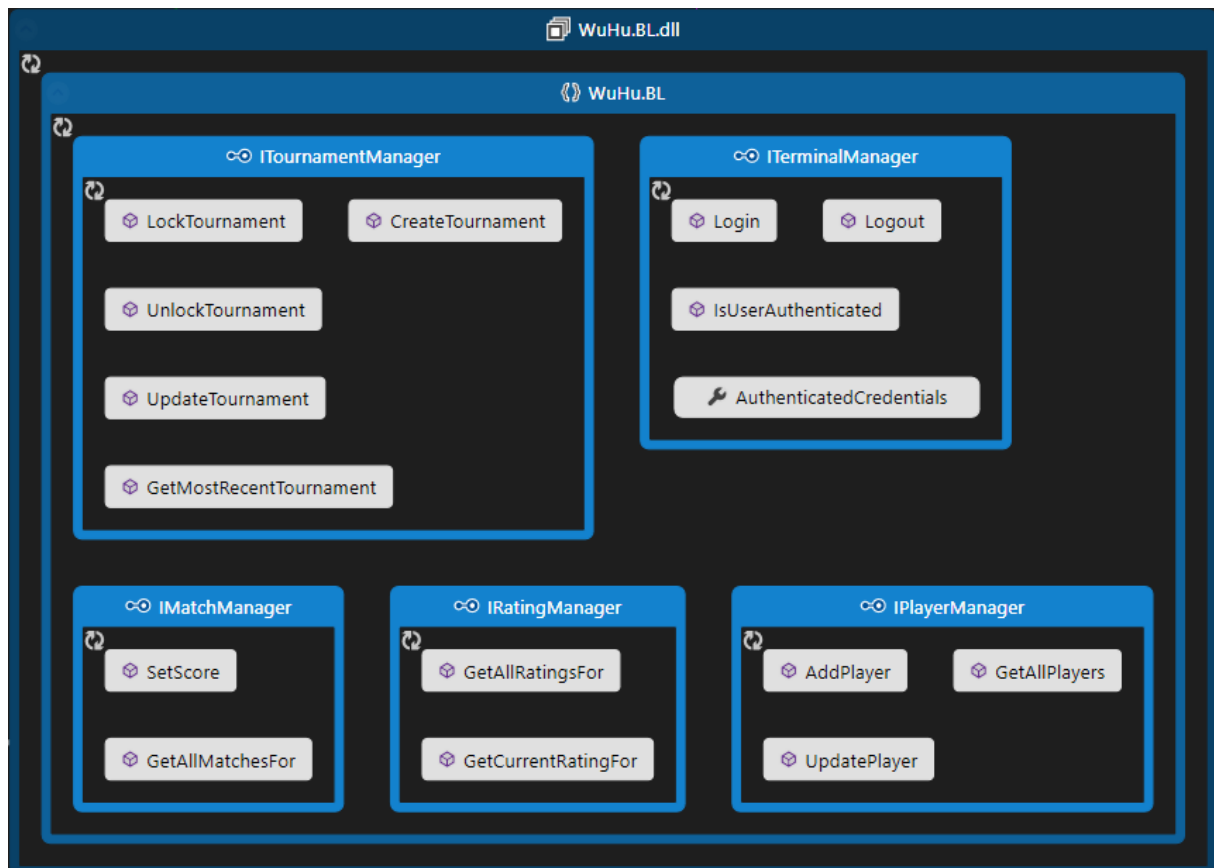
Gesamt Architektur



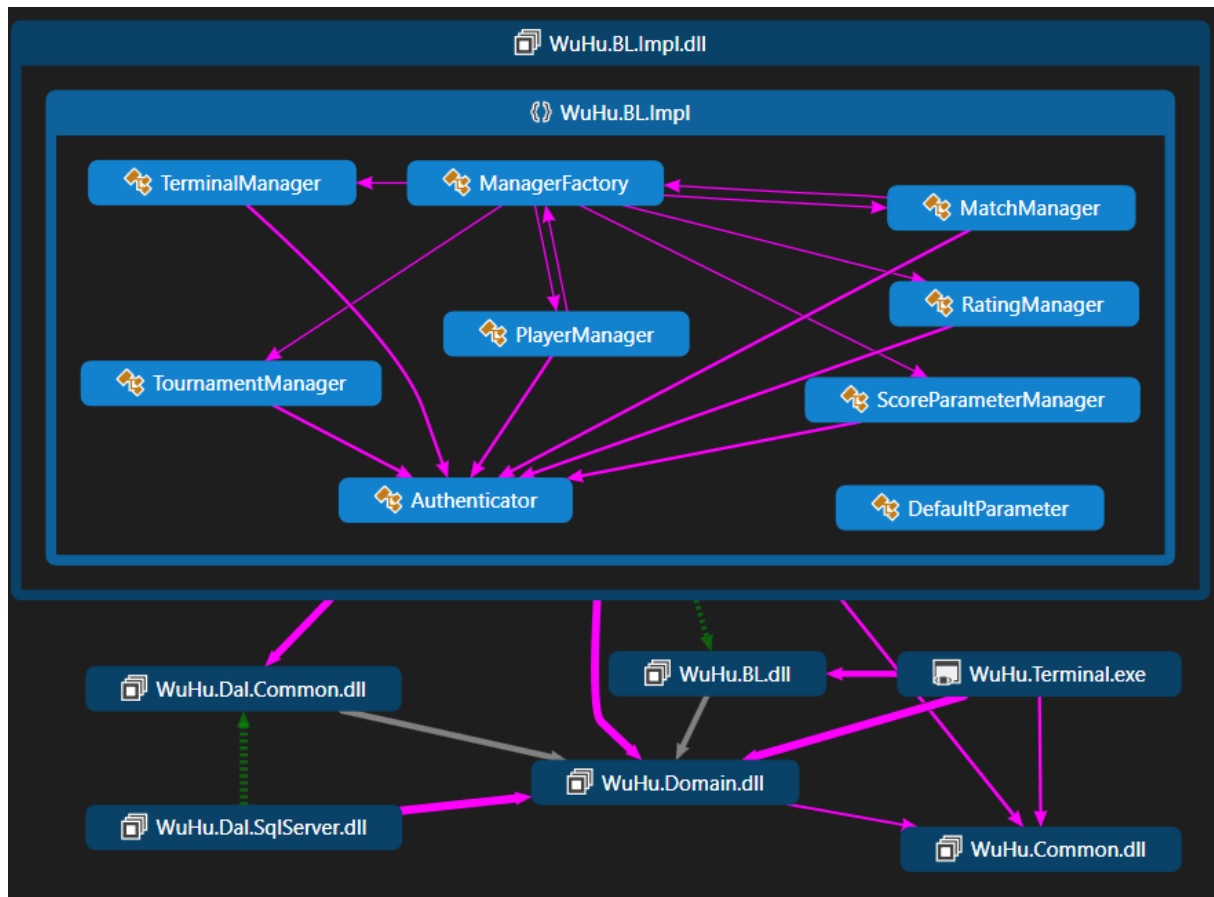
Auf dieser Map erkennt man, dass die Domainklassen eine zentrale Rolle spielen. Sie werden von allen Komponenten verwendet.

Weiterhin sieht man, dass das Terminal-Projekt (also die WPF-Applikation) in keinem Fall direkt auf die DAL zugreift. Sie verwendet ausschließlich die Business Logic, die Domainenklasse, sowie in einem Fall einen CryptoService im „Common“ Projekt. Nur die Business Logik selbst greift auf die Datenbank zu. Dies geschieht auch nur indirekt: Die WPF App verwendet nur die BL Interfaces, und die Implementierung (WuHu.BL.Impl) greift dann auf die Interfaces in der WuHu.Dal.Common zu. Diese wiederum werden von WuHu.Dal.SqlServer implementiert.

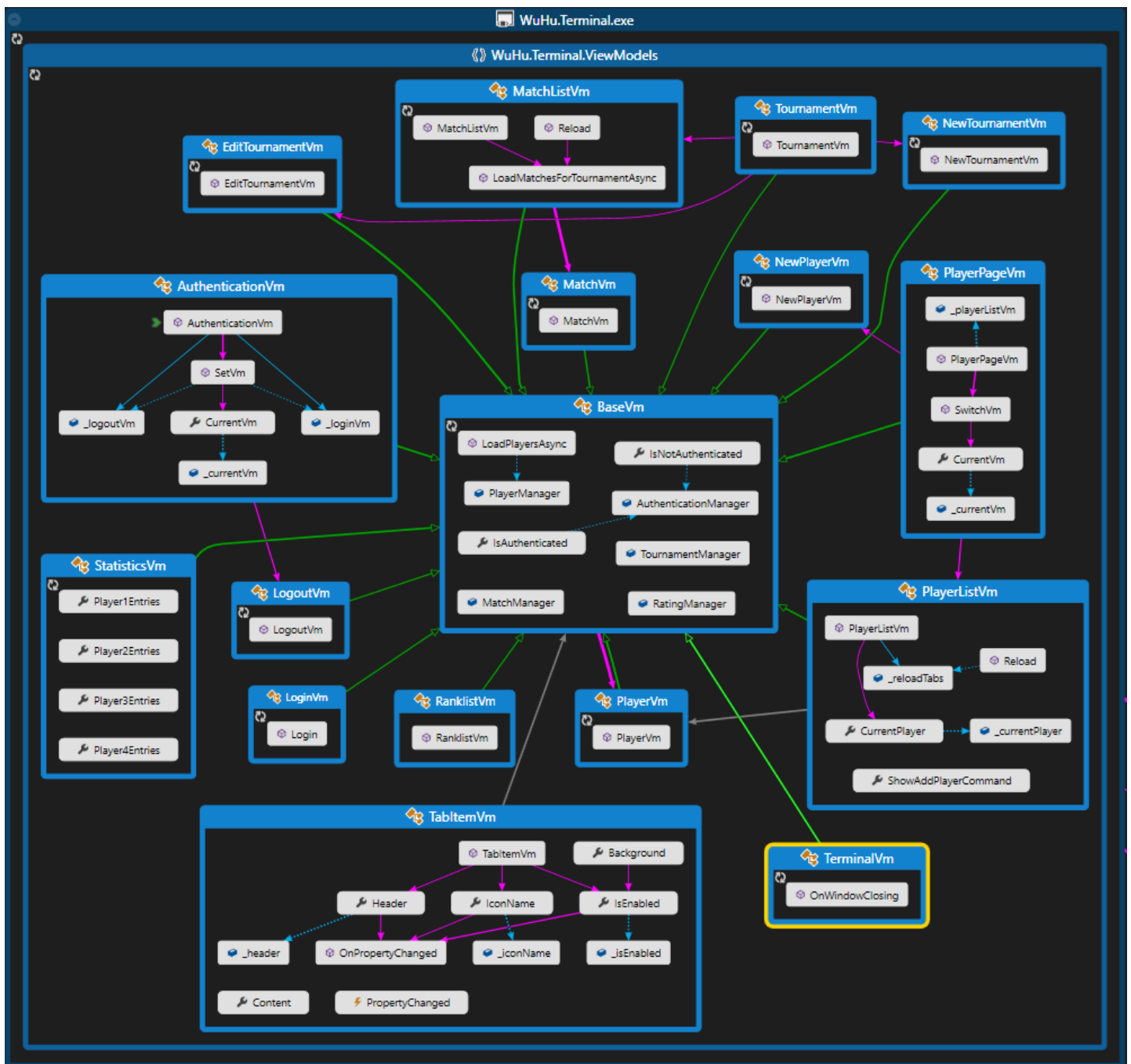
Business Logik



Hier sieht man die gesamte Funktionalität der Business Logik anhand der Interfaces. Sie decken die Funktion der DAL ab und sind ausreichend, um die Funktion der gesamten Applikation abdecken zu können.



Hier sieht man Details zur BL Implementation. Die Manager sind in logische Gruppen aufgeteilt. Sie werden von der ManagerFactory verwaltet und können so (als Interfaces) von der WPF App verwendet werden.

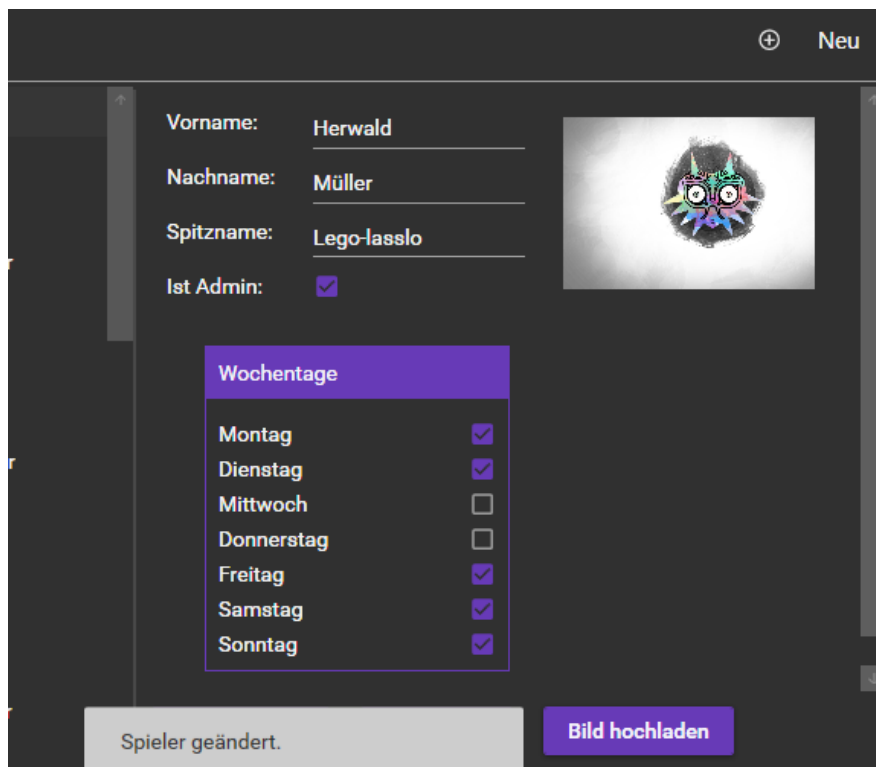
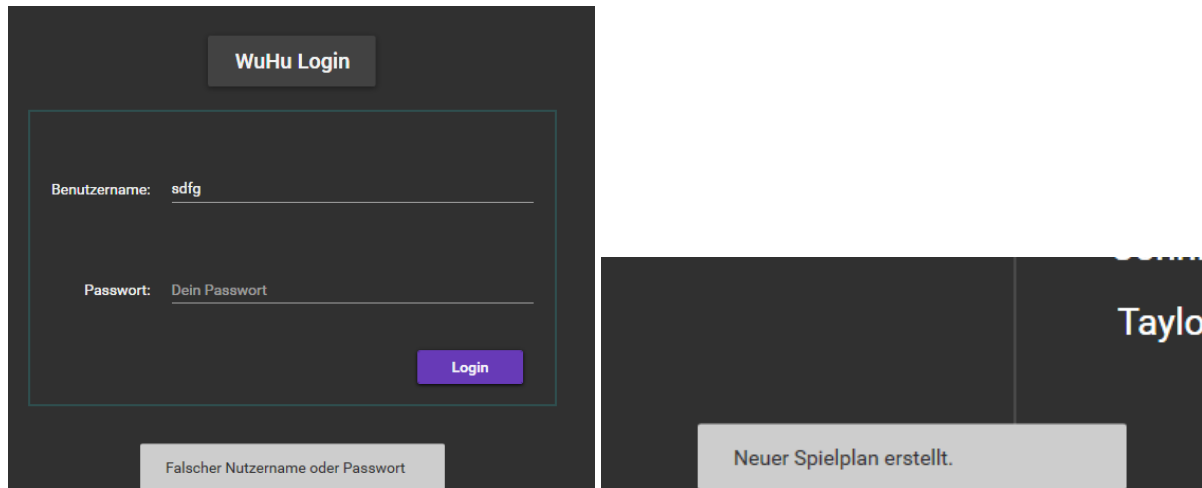


Hier sieht man den Aufbau der Implementierung des WPF Klienten im Detail. Es gibt ein Base-Viewmodel, von dem alle anderen ableiten. In diesem werden die Manager erstellt und verwaltet, sowie eine Methode zum Laden der aktuellen Spielerliste, die in einigen Klassen verwendet wird implementiert.

MessageQueue

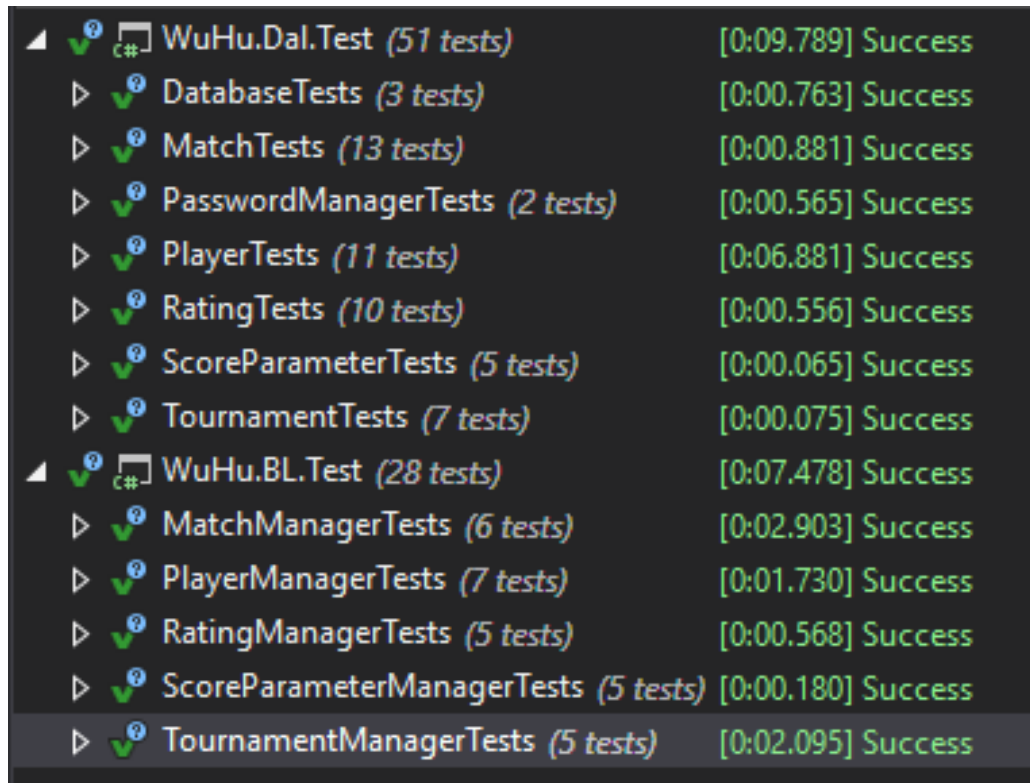
Meine Implementierung beinhaltet ein relativ kompliziertes Messaging System. Nachrichten werden über Delegates von der auslösenden View über Delegates Stück für Stück nach oben propagiert und am Ende vom Fester selbst als „Snackbar“ angezeigt. Diese benachrichtigen den User über wichtige Ereignisse, zum Beispiel ob der Login fehlgeschlagen ist oder nicht, ob ein Spieler oder ein Spielplan angelegt werden konnte oder nicht, und, falls dem so ist, dass ein Turnier gerade in Bearbeitung ist.

Beispiele:



Tests

Eine Reihe an neuen Tests decken die Funktionalität der Business Logik ab. Diese durchlaufen alle positiv. Desweiteren wurden auch die alten DAL Tests teilweise angepasst, um auf dem aktuellen Stand zu sein (da die DAL auch zum Teil leicht angepasst werden musste, um den Anforderungen des Terminals zu entsprechen).



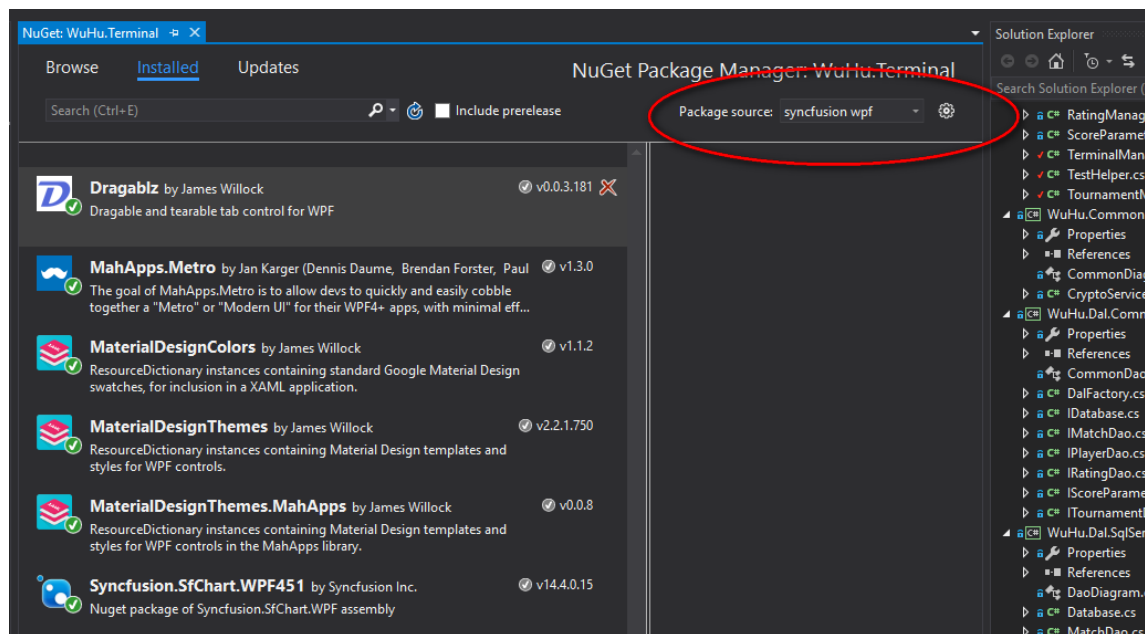
▲	✓ ⓘ	WuHu.Dal.Test (51 tests)	[0:09.789] Success
▷	✓ ⓘ	DatabaseTests (3 tests)	[0:00.763] Success
▷	✓ ⓘ	MatchTests (13 tests)	[0:00.881] Success
▷	✓ ⓘ	PasswordManagerTests (2 tests)	[0:00.565] Success
▷	✓ ⓘ	PlayerTests (11 tests)	[0:06.881] Success
▷	✓ ⓘ	RatingTests (10 tests)	[0:00.556] Success
▷	✓ ⓘ	ScoreParameterTests (5 tests)	[0:00.065] Success
▷	✓ ⓘ	TournamentTests (7 tests)	[0:00.075] Success
▲	✓ ⓘ	WuHu.BL.Test (28 tests)	[0:07.478] Success
▷	✓ ⓘ	MatchManagerTests (6 tests)	[0:02.903] Success
▷	✓ ⓘ	PlayerManagerTests (7 tests)	[0:01.730] Success
▷	✓ ⓘ	RatingManagerTests (5 tests)	[0:00.568] Success
▷	✓ ⓘ	ScoreParameterManagerTests (5 tests)	[0:00.180] Success
▷	✓ ⓘ	TournamentManagerTests (5 tests)	[0:02.095] Success

Externe Ressourcen

Ich habe in dieser Ausbaustufen ein paar Packages verwendet. Folgende können ganz einfach von Nuget geladen werden:

- MahApps.Metro
- Dragablz
- MaterialDesignColors
- MaterialDesignThemes
- MaterialDesignThemes.MahApps

Folgendes Package kann nur über eine alternative „Package source“ geladen werden. Dazu muss man sie zunächst im Nuget Package Manager rechts oben umstellen.



Danach kann über den „Browse“-Tab folgendes Package geladen und installiert werden:

- Syncfusion.Sfchart.Wpf451

Dritte Ausbaustufe

Änderungen DB, DAL und BL

Einige kleinere Änderungen mussten durchgeführt werden, um die volle Funktionalität des Webservices und Webclients zu gewährleisten.

In der BL wurde ein UserManager implementiert, der eine einzige Methode zur Authentifizierung einer Benutzername + Passwort Kombination bietet. Diese wird vom Webservice sowie im Authentifizierungs-Service des Terminals verwendet.

Außerdem wurde ein Statistik Manager in der BL implementiert, der Daten über die Spieler (wie zum Beispiel ihre Gewinnrate) am Server auswertet, damit nicht so viele Daten an den Client übermittelt werden müssen.

Die Tests wurden an alle Änderungen angepasst und laufen immer noch alle ohne Probleme durch.

Webservice

Das Webservice ist in vier Teile aufgeteilt. Jeder Teil kommuniziert nur mit den anderen Teilen oder der Businesslogik, sowie die Controller mit potentiellen Clients über ihre REST-Schnittstellen.

Authentifizierung

Der erste kümmert sich rein um das Startup und die Authentifizierung über einen OAuthProvider. Dieser stellt über die REST-Schnittstelle */api/token* einen Weg zu Verfügung, sich als User oder Admin anzumelden. Dabei muss ein POST request an diese Schnittstelle getätigt werden, wobei im Body des Requests Username und Passwort geschickt werden. Der Provider durchsucht dann die Player-Datenbank nach dem Nutzer und vergleicht die (gehashten) Passwörter. Bei erfolgreicher Anmeldung schickt er dann ein token zurück, dass der Web-Client speichern, und bei subsequenten Requests, die eine Authentifizierung benötigen, in einem Request-Header mitschicken kann. Die einzelnen Controller-Methoden, die bestimmte Rechte benötigen, sind über das [Authorize]-Attribut geschützt.

Modelle

Der zweite Teil sind bestimmte Modelle, die die Kommunikation zwischen Web-Client und Web-Service erleichtern sollen. Wenn also die Domainen-Modelle der DAL und BL nicht ausreichen, habe ich hier neue Modelle erstellt. Das sind im speziellen Modelle für die Rangliste, Statistiken über einen Spieler (Gewinnrate, aktuelle Punktwertung...)

Controller

Der dritte Teil sind die Controller, die die eigentliche Schnittstelle für externe Clients anbieten. Sie dienen als Anbindung von externen Clients an die Business Logik. Wie oben geschrieben, sind sie teilweise durch Authentifizierungsmaßnahmen geschützt, zum Teil aber auch für Gäste offen. So können z.B. nur Administratoren neue Spieler anlegen oder Turniere verwalten, während jeder die aktuellen Ranglistendaten abfragen kann.

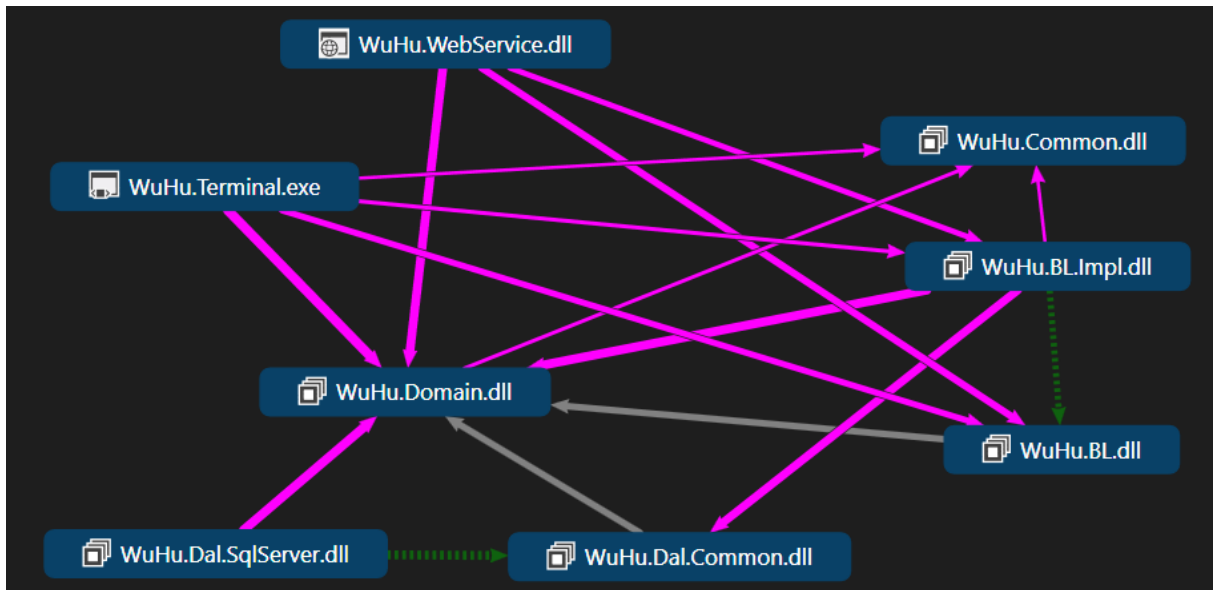
Websockets (SignalR)

Zu guter Letzt gibt es einen sogenannten „Hub“. Dieser dient als Anbindung für SignalR, um Live Spiele über Websockets an die Web-Clients vermitteln zu können bzw. von den Clients Live-Daten zu erhalten.

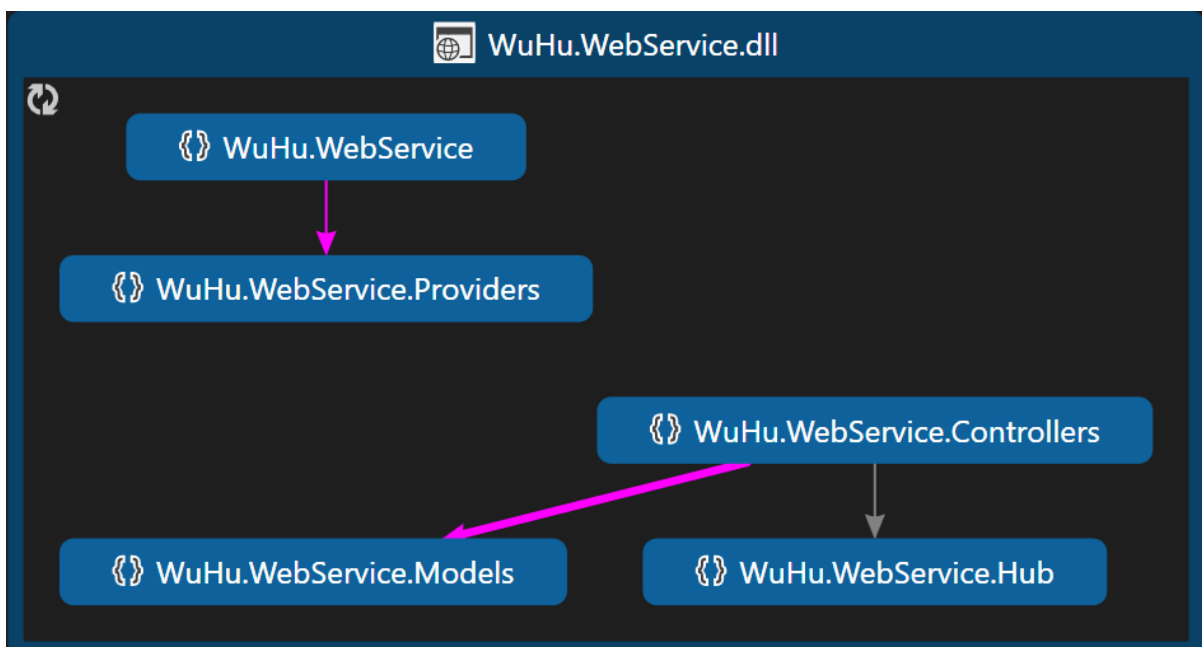
Die Web-Clients können sich zum Beispiel mithilfe der JQuery SignalR library über die */signalr* REST-Schnittstelle mit dem Hub verbinden. Danach kann am Client ein Proxy des Hubs erstellt werden, über das der Client dann mit dem Server kommuniziert. Dabei ist Kommunikation in beide Richtungen möglich, dh. es kann sowohl der Client Events auf dem Server auslösen als auch umgekehrt. Im Gegensatz zum Polling-Ansatz ist keine strikte Client-Request → Server-Response Kommunikation wie über http notwendig. So kann der Client jederzeit neue Spiel-Daten an den Server übermitteln, der diese sofort an die anderen Web-Clients weiterleitet, sowie die Daten in der Datenbank persistiert. Beim `OnConnected()` event, das bei Verbindungsaufbau ausgelöst wird, erhält der sich verbindende Client außerdem sofort die aktuellen Spiel-Daten.

Auch, wenn der Client einen neuen Spielplan anlegt oder den alten abändert, wird kurzzeitig eine Verbindung vom Client mit dem Hub aufgebaut, um die anderen Spieler von dieser Änderung zu informieren.

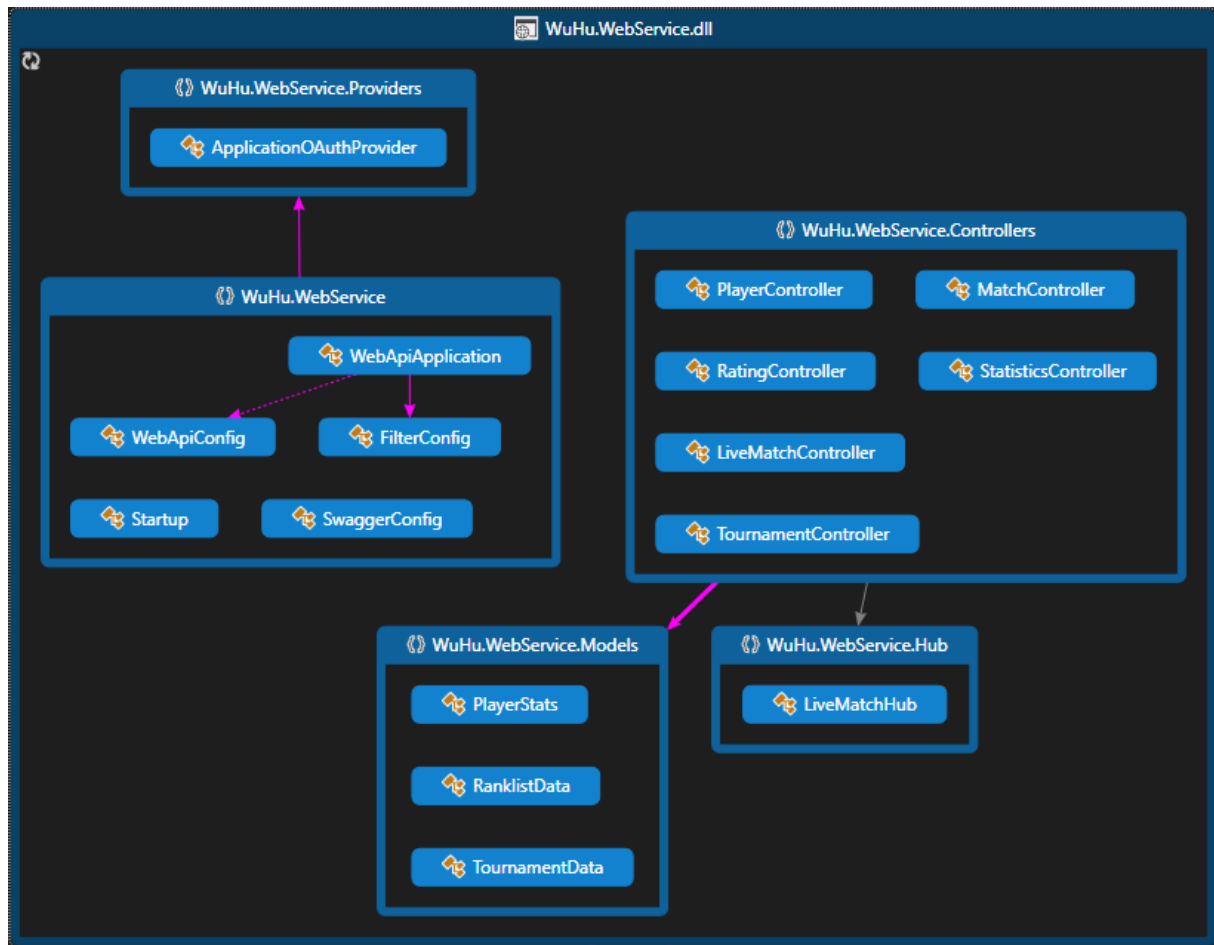
Architektur



In dieser Codemap sieht man wieder genau, dass die Datenzugriffsschicht strikt von der Präsentationsschicht getrennt ist. Dies gilt sowohl für das Terminal, als auch für den Webservice. Jeder Zugriff auf die Daten erfolgt nur über die Businesslogik. Die Businesslogik Implementation ist die einzige Schicht, die direkt auf die DAL zugreift.



Intern sieht das Webservice so aus. WuHu.WebService und Providers ist nur für die Initialisierung zuständig, die anderen Namespaces für die Websockets und die Schnittstellen.

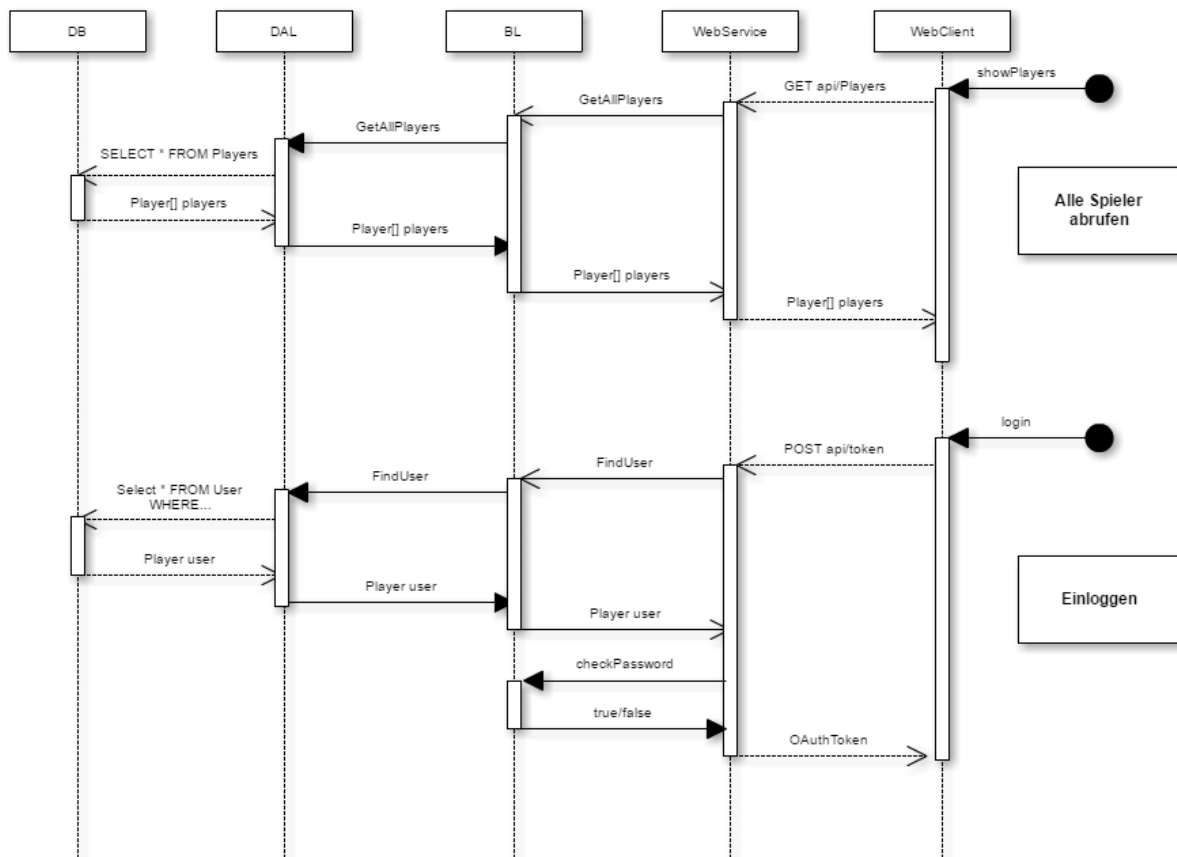


Hier noch einmal das WebService Assembly, aber ausgeklappt. Man sieht, dass es für jeden Manager in der Businesslogik einen zugehörigen Controller gibt (außer den UserManager, der vom `ApplicationOAuthProvider` für die Authentifizierung genutzt wird), sowie einen zusätzlichen `StatisticsController`, der statistische Auswertungen aufbereitet überträgt. So wird verhindert, dass der Server große Datenmengen versenden und der Client die Statistiken selbst berechnen muss.

Sequenz Diagramm

Im ersten Fall wird der Ablauf bei einem simplen Request eines Webclients auf alle Spieler abgegangen. Man sieht, wie es ganz einfach durch die Schichten kaskadiert.

Im zweiten Fall will sich ein Nutzer einloggen. Der Webservice holt sich dabei zunächst den Nutzer samt korrektem (gehashtem) Passwort aus der DB, und lässt dann die BL die Korrektheit überprüfen (über eine eigenen CryptoService Klasse).



Web Client

Der Web Client wurde mit Angular2 implementiert. Zur Unterstützung beim Design verwende ich außerdem Bootstrap, sowie ein *Theme* von bootswatch.com namens *Flatly* (<https://bootswatch.com/flatly/>).

Der Client trennt strikt die Logik von der HTML-Präsentation sowie dem Styling (SASS). Er wurde mit der *Angular-CLI* aufgebaut und ist – wie mit Angular2 üblich – in einzelne Komponente aufgeteilt. Die Navigation erfolgt über einen Router, der gewisse Routen blockiert, wenn der Benutzer nicht authentifiziert ist. Die Authentifikation erfolgt über einen eigenen Service, der mit der *api/token* Schnittstelle des Servers kommuniziert. Bei erfolgreichem Login über die Login-Seite erhält der Service einen Token und speichert diesen im Localstorage. So ist der Benutzer auch nach Verlassen und späterer Wiederkehr zur Seite eingeloggt, solange der Token nicht abgelaufen ist.

Die Website ist aufgeteilt in insgesamt acht Teile:

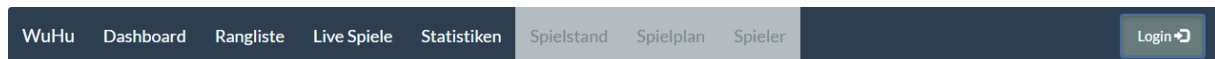
- **Login** für Spieler und Administratoren
- **Dashboard (alle)**, nützliche Infos für alle plus zusätzliche Infos für angemeldete Benutzer/Admins.
- **Rangliste (alle)**, hier wird die aktuelle Rangliste angezeigt, sortiert nach Punktwertung der Spieler. Zusätzlich wird die Gewinnrate jedes Spielers angezeigt.
- **Live Spiele (alle)**, hier werden die Spiele des heutigen Spielplans und die aktuellen Punkte der Spiele angezeigt, sowie ob sie fertig sind oder nicht.
- **Statistiken (alle)**, hier wird – wie im Terminal – der Punktestand aller Spieler über Zeit dargestellt. Es kann der Graph von bis zu fünf Spielern gleichzeitig angezeigt und verglichen werden.
- **Spielstand (Benutzer)**, diese Ansicht zeigt dem Benutzer seine derzeit offenen Spiele an, und lässt ihm den Punktestand mitführen. Die Punkte werden direkt an die Live-Ansicht übertragen.
- **Spielplan (Admins)**, in dieser Ansicht können Administratoren neue Spielpläne anlegen und den aktuellen abändern, also neue Spieler hinzufügen oder entfernen.
- **Spieler (Admins)**, hier können alle Spieler verwaltet werden, oder neue hinzugefügt werden. Es kann auch ein Bild für den Nutzer hochgeladen werden.

Ich habe sehr viel Wert auf Nutzerfreundlichkeit gelegt. Dem Benutzer wird immer mitgeteilt, wenn etwas schief läuft, zum Beispiel wenn die Verbindung zum Server abbricht. Formulare zeigen dem Nutzer direkt an, wenn ein Feld falsche oder keine Daten enthält, wenn richtige benötigt werden. Knöpfe werden deaktiviert und ausgeblendet, wenn der Nutzer nicht die Rechte hat, sie zu benutzen, oder wenn zum Beispiel ein Formular noch nicht vollständig ausgefüllt ist.

Des Weiteren ist alles möglichst klar beschrieben, Formulare sind sowohl mit Labels als auch mit Platzhaltern beschriftet, Knöpfe und Tabs sind möglichst selbstverständlich. Die einzelnen Tabs sind durch große Überschriften klar erkenntlich, damit der User immer erkennt, wo er sich befindet.

Die Routen werden durch sogenannte Guards geschützt, die der Router verwendet, um dem Nutzer nur auf die Seiten navigieren zu lassen, für die er berechtigt ist. Falls z.B. ein Gast versucht, unerlaubt auf die /Spieler Route zu navigieren, wird er auf den Login umgeleitet.

Login

A light gray rectangular box containing a login form. It has two white input fields: the top one is labeled 'Benutzername' and the bottom one is labeled 'Passwort'. Below the fields is a dark gray button with the text 'ANMELDEN' in white capital letters.

In dieser Ansicht kann sich ein Benutzer oder ein Administrator anmelden. Bei einem fehlerhaften Login wird natürlich eine Warnung angezeigt.

A light gray rectangular box containing a login form. The top input field contains the text 'asdf'. The bottom input field contains four dots '....'. Below the fields is a dark gray button with the text 'ANMELDEN' in white capital letters. Below the button, the text 'ACHTUNG! FALSCHER NUTZERNAME ODER PASSWORT' is displayed in red capital letters.

Dashboard




Das Dashboard in der Gast-Ansicht. Es wird kurz erläutert, was der Gast auf der Seite anschauen kann.



Das Dashboard für einen angemeldeten Administrator. Hier werden zusätzliche Infos angezeigt, wie die eigene Gewinnrate, die derzeitigen Punkte, sowie Hilfe zur Seite und Navigation.

Rangliste

WuHu	Dashboard	Rangliste	Live Spiele	Statistiken	Spielstand	Spielplan	Spieler	Login 
------	-----------	-----------	-------------	-------------	------------	-----------	---------	---

Rangliste

Platz	Name	Gewinnanteil	Wertung
#1	Tanner 'laborum' Hancock	47.88%	4292
#2	Ross 'aliqua' Henson	51.74%	3417
#3	Heinz 'Javanda' Janda	50.92%	2844
#4	Stefan 'VoMe' Vordermetaller	46.88%	2769
#5	Celina 'ut' Cote	49.76%	2497
#6	Mark 'Deutscher' Spinner	51.90%	2406
#7	Jewel 'enim' Dennis	51.89%	2345
#8	Hilary 'elusmod' Boyd	50.77%	2249
#9	Herwig 'Herbi' Maurer	45.98%	2246
#10	Boone 'elit' Coleman	50.39%	2228
#11	Bernard 'voluptate' Cummings	50.00%	2097
#12	Geneva 'anim' Morin	53.03%	2094
#13	Warner 'laborum' Summers	46.32%	2034
#14	Birgit 'der Hai' Hai	47.42%	2029
#15	Andrea 'Hobli' Hobelmayer	51.94%	1817
#16	Trinke 'Sai Kamhucha' Fanta	51.20%	1712

Hier wird die aktuelle Rangliste inklusive der Gewinnrate und der Wertung der Spieler angezeigt.

Live Spiele

[WuHu](#) [Dashboard](#) [Rangliste](#) [Live Spiele](#) [Statistiken](#) [Spielstand](#) [Spielplan](#) [Spieler](#) [Login](#)

Live

Chicopee_Barrett

Spiel 1 - Team 1 hat gewonnen!

Team 1		Team 2
Mark Spinner	10	Rojas Burks
Dobрила Reiter		Herwald Müller
		2

Spiel 2 - Team 1 hat gewonnen!

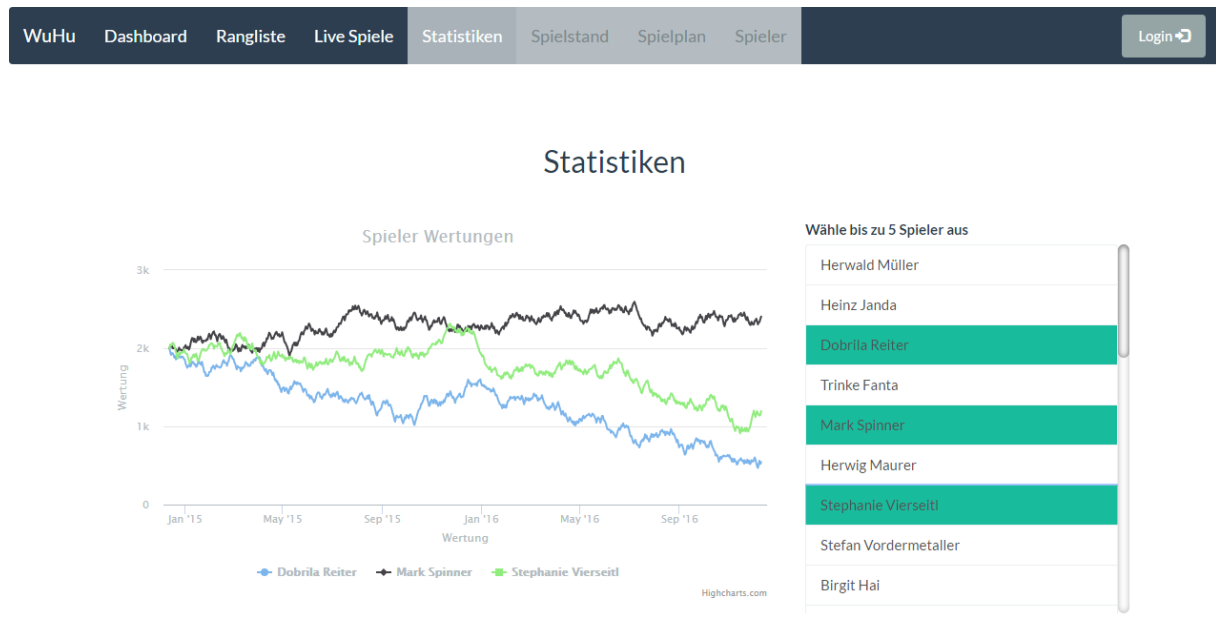
Team 1		Team 2
Oliver Miranda	10	Stefan Vordermetaller
Maura Sweet		Celina Cote
		7

Spiel 3 - Team 1 hat gewonnen!

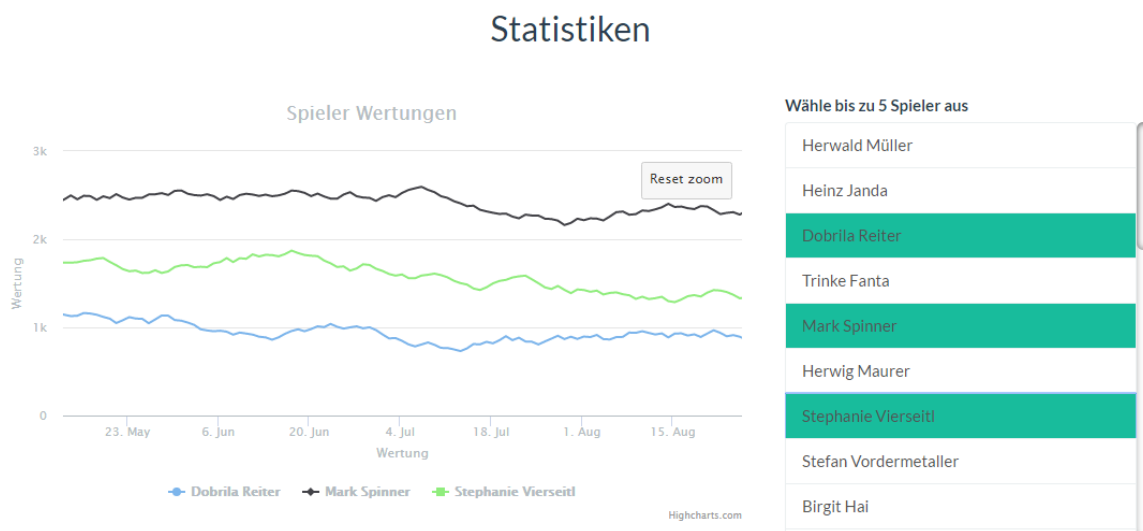
Team 1		Team 2
---------------	--	---------------

In dieser Ansicht sieht der Spieler alle Spiele, die im heutigen Spielplan sind, sowie den aktuellen Punktestand. Diese werden laufend per Websocket aktualisiert.

Statistiken



Hier kann die Wertung über Zeit der Spieler angezeigt. In der Liste rechts wählt man mit der Maus die Spieler aus (maximal 5 auf einmal), und links sieht man ihre Punkte im gesamten Zeitraum. Mit der Maus kann der Zeitraum auch eingeschränkt werden, um eine detailliertere Ansicht zu erhalten.



Über den Reset Zoom Knopf rechts oben kommt man dann wieder in die Gesamt-Ansicht.

Spielstand



Spielstand

Spiel 1 - Team 1 hat gewonnen!

Team 1		Team 2
Mark Spinner	10 +	Rojas Burks
Dobrila Reiter		Herwald Müller
		2 +

In dieser Ansicht kann ein eingeloggter Benutzer seine eigenen Spiele des heutigen Spielplans verfolgen und seine Punkte mitführen. Diese werden direkt live an den *Live Spiele* Tab geleitet. Das beenden eines Spieles kann jedoch, wie in den Anforderungen beschrieben, immer noch nur von einem Administrator über das Terminal erfolgen.

Spielplan

[WuHu](#) [Dashboard](#) [Rangliste](#) [Live Spiele](#) [Statistiken](#) [Spielstand](#) [Spielplan](#) [Spieler](#) [Logout](#)

Spielplan

[Neuer Spielplan](#) [Spielplan ändern](#)

Spieler

Herwald Müller
Heinz Janda
Dobrila Reiter
Trinke Fanta
Mark Spinner
Herwig Maurer
Stephanie Viersehl
Stefan Vordermetaller
Birgit Hai
Markus Spinner
Andrea Hobelmayer
Connie Cole
Sheena Schneider
Tanner Hancock
Oliver Miranda
Pauline Terry
Blackburn Hoffman
Ross Henson

Name
Anzahl an Spielen
[Absenden](#) [Abbrechen](#)

In dieser Ansicht kann ein neuer Spielplan erstellt, oder der heutige abgeändert werden. Dies geschieht über die zwei Knöpfe direkt unter der *Spielplan* Überschrift.

In der linken Spielerliste wählt man diejenigen Spieler aus, die am Turnier teilhaben sollen. Wie im Terminal sind die Spieler, die am heutigen Wochentag planmäßig dabei sind, bereits aktiviert. Man kann diese aber natürlich auch wieder deaktivieren.

Rechts kann man außerdem den Namen des Spielplans (dieser wird auch in der Live Ansicht neben dem Titel angezeigt), sowie die Anzahl der Spiele angeben.

Spieler

WuHu
Dashboard
Rangliste
Live Spiele
Statistiken
Spielstand
Spielplan
Spieler
Logout

Spieler

+

Herwald "Lego-las" Müller
Heinz "Javanda" Janda
Dobrila "Pascal" Reiter
Trinke "Sei Kambucha" Fanta
Mark "Deitscher" Spinner
Herwig "Herbi" Maurer
Stephanie "Steffi" Vierseitl
Stefan "VoMe" Vordermetaller
Birgit "der Hai" Hai
Markus "Springl" Spinner
Andrea "Hobli" Hobelmayer
Connie "reprehenderit" Cole
Sheena "cupidatat" Schneider
Tanner "laborum" Hancock
Oliver "tempor" Miranda
Pauline "veniam" Terry
Blackburn "consequat" Hoffman
Ross "aliqua" Henson

Vorname
Stephanie

Nachname
Vierseitl

Spitzname
Steffi

☒ Admin

Wochentage
☒ Montag ☐ Dienstag ☒ Mittwoch
☒ Donnerstag ☐ Freitag ☒ Samstag
☒ Sonntag

Änderungen speichern

Änderungen verwerfen




Bild hochladen (max. 1.5Mb!)

Datei auswählen Fish_Titlepic.jpg

In dieser Ansicht kann ein Administrator die Spieler verwalten, sowie neue Spieler hinzufügen. Das verwalten eines existierenden Spielers erfolgt so, dass man ihn zunächst links in der Liste anklickt, und dann rechts seine Daten ändert. Nun kann man entweder die Änderungen verwerfen und von neu anfangen, oder auf den „Änderungen speichern“ Knopf drücken, und somit den Spieler an den Server schicken.

Über den grünen Plus Knopf links oben können außerdem neue Spieler hinzugefügt werden. Hier muss man natürlich auch Username und Passwort mit angeben.

Vorname	Bild hochladen (max. 1.5Mb!)
<input type="text" value="Vorname"/>	<input type="button" value="Datei auswählen"/> Keine ausgewählt
Nachname	
<input type="text" value="Nachname"/>	
Spitzname	
<input type="text" value="Spitzname"/>	
Benutzername	
<input type="text" value="Benutzername"/>	
Passwort	
<input type="text" value="Passwort"/>	
<input type="checkbox"/> Admin	
Wochentage	
<input type="checkbox"/> Montag <input type="checkbox"/> Dienstag <input type="checkbox"/> Mittwoch	
<input type="checkbox"/> Donnerstag <input type="checkbox"/> Freitag <input type="checkbox"/> Samstag	
<input type="checkbox"/> Sonntag	
<input type="button" value="Spieler erstellen"/>	<input type="button" value="Abbrechen"/>