# Allgemeine Struktur

Die technische Struktur des Projektes orientiert sich stark an gängigen Strukturen aus dem Bereich der Microservices. Dies bietet den Vorteil, dass einzelne Teile beliebig skaliert werden können, um so die Anforderungen bestens umsetzen zu können.

Alle Nutzerinteraktionen mit dem Projekt laufen über das .NET Core Projekt. Dieses bietet Backend-Funktionen für ein zukünftig geplantes Frontent, worüber ein Nutzer alle Anfragen bequem über eine graphische Oberfläche stellen kann. Zudem frägt das .NET Core Projekt Daten von der OpenDota API ab und speichert diese in ein MongoDB Replica Set. Ebenso werden Nutzeranfragen an den KI-Bereich weiter gegeben.

Zum Trainieren unserer KI, sowie zum Auswerten von Spielen, welche ein Nutzer angeben kann, wird ein Jupyter Notebook Server verwendet, welcher über ein Jupyter Kernel Gateway per REST-Schnittstelle angesprochen werden kann. Dieser kann auf diese Weise mit Daten befüllt werden, welche in einem eigenen MongoDB Replica Set gespeichert werden, sowie über REST Befehle gesteuert werden, um z.B. anhand der Vorhandenen Daten einen Algorithmus zu trainieren, oder aber ein Spiel eines Nutzers auszuwerten.

Alle diese einzelnen Komponenten werden in eigenen Docker Dateien konfiguriert, welche über eine einzelne Docker-Compose Datei gestartet werden können. Dies bietet den Vorteil, dass die Laufzeitumgebung bei jedem Entwickler, sowie beim Deployen der Anwendung, bereits richtig konfiguriert ist und somit keine Probleme bereitet.

HIER BILD VON SYSTEMARCHITEKTUR EINFÜGEN (res/images)

# .NET Core

Für das zentrale Backend wurde das .NET Core Framework von Microsoft gewählt, da dieses in C# entwickelt wurde, womit die meisten Projektmitgliedern bereits Erfahrungen sammeln konnten. Zudem bietet Microsoft Docker-Images für .NET Core an.

Um die Grundidee von Microservices auch im Backend korrekt aufgreifen und umsetzen zu können wird das Domain-Driven Design bei der Entwicklung angewandt. Dieses sieht die Unterteilung des Projekts in die drei Komponenten „API“, „Data“ und „Domain“ vor. Die Komponente API beinhaltet alle Funktionalitäten zum Starten und Steuern des Projektes. In diesem befindet sich das .NET Core Framework mit seinen Controllern, welche die REST-Schnittstellen definieren, sowie Konfigurationen wie die „appsettings.json“ Datei für Umgebungsvariablen, oder die „Startup.cs“ Datei, in welcher der Dependency Injector konfiguriert wird, um Interfaces und dazugehörige Implementierungen an Konstruktoren verschiedener Klassen weiter geben zu können. Hierfür besitzt die API Komponente Abhängigkeiten zu sowohl der Data, als auch der Domain Komponente.  
Die Data Komponente ist von der Domain Komponente abhängig und implementiert die in der Domain Komponente definierten Repository Interfaces als vollständige Klassen. Dies bietet den Vorteil, dass die konkrete Implementierung der Datenhaltung unabhängig vom eigentlichen Code ist, was einen Austausch der verwendeten Datenbank, z.B. von MongoDB auf MSSQL, ohne Änderungen am eigentlichen Code der Domaine ermöglicht.  
Die Domain Komponente besitzt keine Abghängigkeiten auf eines der anderen Projekte und implementiert jegliche Logik, welche zur Umsetzung einer Aufgabe nötig ist. Sie implementiert alle verwendeten Models und definiert die Funktionen der Repositories in Interfaces.

Diese Aufteilung der Aufgabenbereiche ermöglicht zudem ein besseres Testen der einzelnen Komponenten.

HIER TEST VON FELIX EINFÜGEN