Dokumentation der Software-Architektur

Prettner, Stephan
s.prettner@oth-aw.de

Rube, Alexander
a.rube@oth-aw.de

Schafberger, Tobias
t.schafberger@oth-aw.de

Schneider, Oliver
o.schneider1@oth-aw.de

Schuster, Felix
f.schuster@oth-aw.de

Stangl, Dennis
d.stangl@oth-aw.de

Zuber, Maximilian
m.zuber@oth-aw.de

***Abstract*— In diesem Technical Report wird die Architektur vorgestellt, die für die Projektarbeit im Fach Big Data und Cloud-basiertes Computing (BDCC) eingesetzt wird. Ziel dieser Projektarbeit ist es, die Themengebiete Big Data, Cloud Computing und Künstliche Intelligenz zu verknüpfen.**

I. Allgemeine Struktur

Die technische Struktur des Projekts orientiert sich stark an gängigen Strukturen aus dem Bereich der Microservices. Dies bietet den Vorteil beliebiger Skalierbarkeit einzelner Komponenten, um dadurch die Anforderungen bestens umsetzen zu können.
Alle Nutzerinteraktionen mit dem Projekt erfolgen über die Frontend-Komponente, welche mit React.js implementiert wird. Über eine REST-Schnittstelle kann diese mit der .NET Core-Komponente kommunizieren, um Informationen über Matches abrufen zu können. Sollte die optional angedachte Frontend-Komponente nicht wie geplant fertiggestellt werden können, ist ein direkter Zugriff des Benutzers auf die .NET Core-Komponente vorgesehen. Zudem fragt das .NET Core Projekt Daten von der OpenDota API ab und speichert diese in ein MongoDB Replica Set. Ebenso werden Nutzeranfragen an den Matchanalyse-Bereich weitergegeben.
Zum Trainieren der künstlichen Intelligenz, sowie zum Auswerten von Spielen, welche ein Nutzer angeben kann, wird ein Jupyter-Notebook-Server verwendet. Dieser kann über einen Jupyter-Kernel-Gateway per REST-Schnittstelle angesprochen werden. Auf diese Weise werden bei der Anfrage Daten übermittelt, welche in einem eigenen MongoDB Replica Set gespeichert werden. Außerdem kann er über REST-API-Befehle gesteuert werden, um zum Beispiel anhand von vorhandenen Daten einen Algorithmus zu trainieren oder ein Spiel eines Nutzers auszuwerten.
Die einzelnen Komponenten werden in einer YAML-Datei konfiguriert, um die Umgebungsvariablen sowie die unterschiedlichen Abhängigkeiten zwischen diesen Containern zu definieren. Mithilfe von Docker-Compose können diese dann auf dem Host-System gestartet werden. Dies bietet den Vorteil, dass die Laufzeitumgebung bei jedem Entwickler sowie beim Deployen der Anwendung bereits richtig konfiguriert ist und somit keine Probleme bereitet.
Figure 1 stellt die beschriebene Struktur in grafischer Form dar.

II. Matchverwaltung (.NET Core)

Für das zentrale Backend wurde das .NET Core Framework von Microsoft gewählt, da dieses in C# entwickelt wurde, womit die meisten Projektmitglieder bereits Erfahrungen sammeln konnten. Zudem bietet Microsoft eine leichte Integration von Docker-Images für .NET Core an.
Um die Grundidee von Microservices auch im Backend korrekt aufgreifen und umsetzen zu können, kommt bei der Entwicklung das Domain-driven Design zum Einsatz. 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 Projekts. In diesem befindet sich das .NET Core Framework mit seinen Controllern. Diese definieren die REST-Schnittstellen sowie Konfigurationen wie die „appsettings.json“ Datei für Umgebungsvariablen oder die „Startup.cs“ Datei. In dieser wird der Dependency Injector konfiguriert, um Interfaces und dazugehörige Implementierungen an Konstruktoren verschiedener Klassen weitergeben 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, zum Beispiel von MongoDB auf PostgreSQL oder Apache Cassandra, ohne Änderungen am eigentlichen Code der Domain ermöglicht.
Die Domain-Komponente besitzt keine Abhängigkeiten auf eines der anderen Projekte und implementiert jegliche Logik, welche zur Umsetzung einer Aufgabe nötig ist. Sie bettet alle verwendeten Modelle ein und definiert die Funktionen der Repositories in Interfaces.
Diese Aufteilung der Aufgabenbereiche ermöglicht zudem ein besseres Testen der einzelnen Komponenten.

A. Tests für die Matchverwaltung

Für die Umsetzung eines Testprojekts zur Matchverwaltung, die in einer .NET Core-Umgebung realisiert ist,

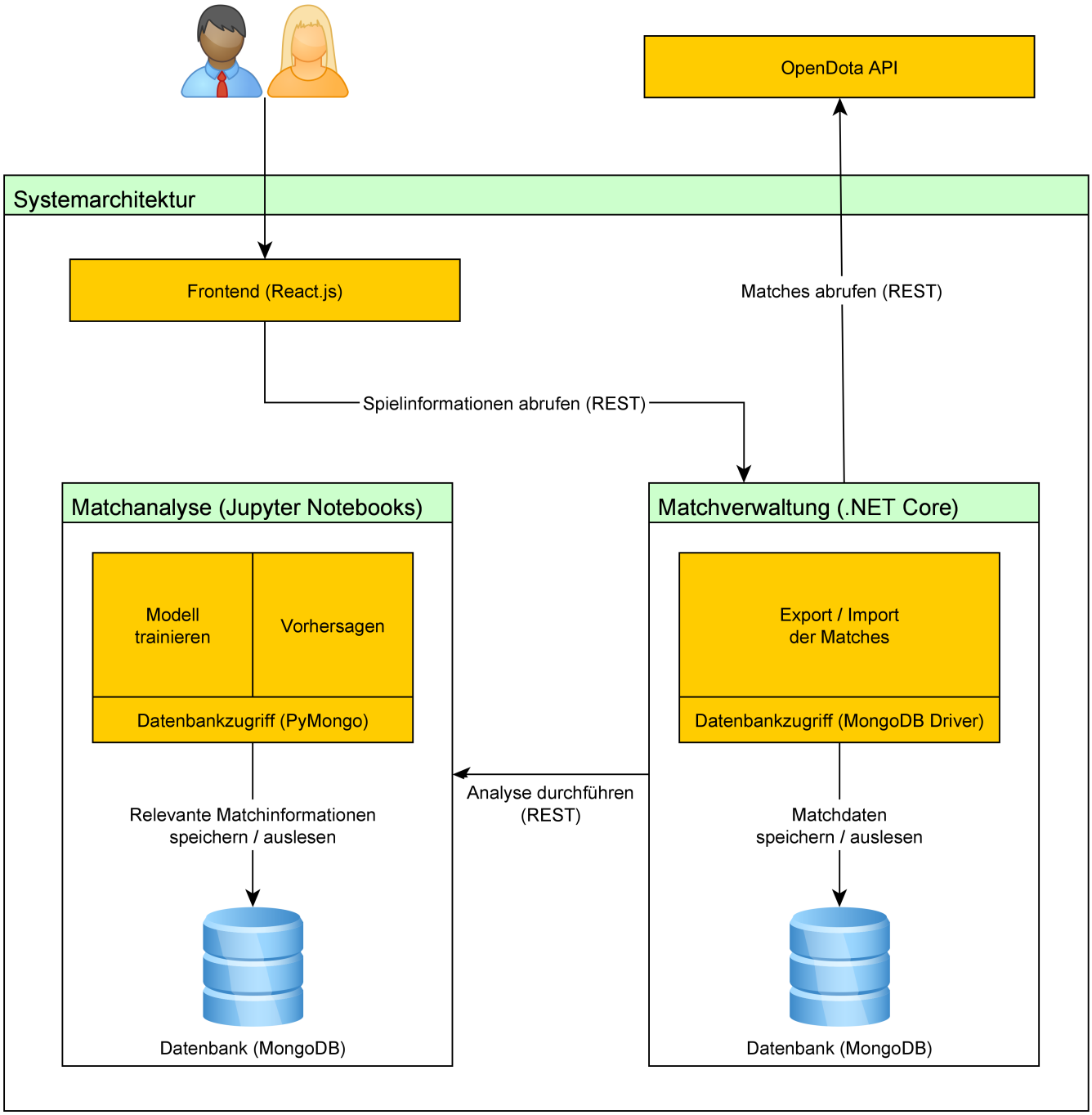

Fig. 1. Grafische Darstellung der verwendeten Architektur

kommt das unter der MIT-Lizenz stehende Open-Source Test-Framework NUnit [1] zum Einsatz. Mit diesem lassen sich Testmethoden erstellen, die beliebige Methoden des Backends für die Matchverwaltung unter vorher festgelegten Voraussetzungen ausführen. Durch Assert-Anweisungen lässt sich prüfen, ob das System wie erwartet reagiert.

Das Testprojekt, das auf die Funktionalität von NUnit zurückgreift, bündelt dabei sowohl Unit-Tests zur Sicherstellung der Funktionstüchtigkeit der in der Matchverwaltung implementierten Methoden als auch diverse Integrationstests. Letztere prüfen die Lauffähigkeit und Korrektheit externer Systeme und den dazugehörigen Schnittstellen. Dazu gehören die Aufrufe der verwendeten MongoDB-Datenbank zur Verwaltung darin gespeicherter Matchdaten, die Nutzung der REST-Schnittstelle, die OpenDota zur Sammlung von Spieldaten zu „Dota 2“ zur Verfügung stellt, als auch die Steuerung der Matchanalyse, die separat in einem Jupyter-Notebook-Server realisiert ist.

Die Ausführung aller in dem Projekt vorhandenen Tests ermöglicht dabei einen sofortigen Überblick über die korrekte

Implementierung der Komponenten und zeigt potentiellen Handlungsbedarf an.

B. OpenData

Für die Beschaffung der benötigten Informationen aus den Spieldaten wird die Programmierschnittstelle (API) des Open Source Projekts OpenDota [2] verwendet. In der hier verwendeten kostenlosen Variante können bis zu 60 Anfragen pro Minute an die Webseite gestellt werden. Ein „TimeLimiter“ nach dem „Token bucket"-Prinzip sorgt für die Einhaltung dieser Begrenzung.
Jede Anfrage an die API verbraucht dabei einen Token. Der "TimeLimiter" fügt jede Minute 60 Token zu einem Kontingent (dem sogenannten Bucket) mit einer maximalen Größe von 60 Tokens hinzu. Dadurch wird das vorgegebene Limit der Abfragen nie überschritten, da keine weiteren Abfragen stattfinden, solange der Bucket leer ist. Durch die asynchrone Auffüllung der Token können auch bis zu 60 Abfragen auf einen Schlag (Burst) ausgeführt werden, ohne eine Wartezeit zwischen den Aktionen zu benötigen.
Die Informationsbeschaffung erfolgt durch die 100 neusten abgeschlossenen öffentlichen Spiele oder den Spielen eines gewünschten Spielers. Dazu werden wiederholt Anfragen an den OpenDota-Server gesendet, um mehrere eindeutige Match-IDs zu bekommen und alle Attribute der jeweiligen Spiele anfordern zu können. Mittels der Steam32-ID eines Spieleraccounts können so ebenfalls gezielt Spiele des jeweiligen Spielers betrachtet werden.
Durch eine verringerte Datenstruktur, in der nur die nötigsten Attribute enthalten sind, wird die Speicherbelastung von circa 215 kB pro Spiel auf ungefähr 15 kB reduziert. Jedes angeforderte Spiel wird aufgrund der Anfragenbegrenzung in der MongoDB-Datenbank gespeichert.

III. Matchanalyse (Jupyter Notebooks)

A. Interface zwischen .NET Core- und Python-Backend

Das Interface für die Interaktion zwischen dem .NET Core- und Python-Backend wird mit einem Jupyter-Server realisiert, welcher als sogenannter Jupyter-Kernel-Gateway aufgesetzt wird. Dadurch ist es möglich, verschiedene REST-Schnittstellen über ein Jupyter-Notebook zu definieren, mithilfe dessen HTTP-Anfragen gestellt werden können. In diesem Projekt werden ausschließlich POST-Anfragen an das Python-Backend gesendet. Die Rückgabe des Aufrufs sind HTTP Status Codes.
Das Gateway ist über die lokale URL über den Port „8898“ erreichbar und bietet folgende Schnittstellen:

- /writematch: schreibt übergebene Daten eines einzelnen Matches in die MongoDB/Matchanalyse
  - Parameter: Daten des zu übertragenden Matches im JSON-Format
  - Rückgabe: Keine
- /deletematch: löscht ein Match aus der MongoDB/Matchanalyse
  - Parameter: Match-ID des zu löschenden Matches
  - Rückgabe: Keine
- /trainmodel: startet das Training des Matchanalyse-Modells
  - Parameter: Name des zu trainierenden Modells
  - Rückgabe: Keine
- /predict: startet die Vorhersage des Matchanalyse-Modells
  - Parameter:
    - Modellname: Name des gewünschten Modells für die Vorhersage im JSON-Format
    - Matches: Daten der zu analysierenden Matches im JSON-Format
  - Rückgabe: Ergebnisse der Vorhersage

B. Matchanalyse-Modell

Als Matchanalyse-Modell wurden sieben Klassifizierer evaluiert. Zur Auswahl standen:

- MLPClassifier
- RandomForestClassifier
- SupportVectorMachines (SVC)
- SGDClassifier
- DecisionTreeClassifier
- KNeighborsClassifier
- GaussianNB

Evaluiert wurde über mehrere Datensätze und mit verschiedenen RandomStates. Durch das Setzen eines RandomStates können Ergebnisse reproduziert werden. Dies ermöglicht einen Vergleich zwischen den Klassifikatoren, da die Initialbelegungen, sowohl für den Datensatz als auch für die Klassifizierer, konstant sind.
Die besten Ergebnisse erzielte hierbei der MLPClassifier. Dieser war, unabhängig vom benutzten Datensatz und den verwendeten RandomStates, immer einer der besten und unterlag keinen großen Schwankungen in seiner Genauigkeit. Vom Multilayer-Perzeptron (MLP) werden zwei Varianten trainiert und zur Vorhersage eingesetzt. Das erste Modell mit dem Namen "no_kda" benutzt nur Pings, um vorherzusagen, ob das analysierte Match ein Sieg ist und wie sicher sich das Modell dabei ist. Beim zweiten Modell mit dem Namen "kda" werden zusätzlich zu den Pings auch die Features Assists, Deaths und Kills zur Vorhersage verwendet.
Das Training beider Modelle wird mit dem kompletten Datensatz vorgenommen, welcher in der MongoDB/Matchanalyse gespeichert ist. Damit die beiden Modelle für jede Vorhersage nicht neu trainiert werden müssen, werden diese nach dem Training in die MongoDB/Matchanalyse gespeichert. Für die Vorhersage wird das benötigte Modell aus der MongoDB/Matchanalyse geladen und der übergebene Datensatz anhand der Features bewertet. Die Bewertung wird zurück an das .NET Core-Backend gegeben, welches die Ergebnisse an den Nutzer weiterleitet.

References

[1] NUnit, https://nunit.org/, Abgerufen: Juni 2021.
[2] OpenDota API, https://docs.opendota.com/, Abgerufen: Juni 2021.