DHBW
Duale Hochschule
Baden-Württemberg

Entwicklung einer Schach-Trainingsanwendung zur Erlernung von Eröffnungen unter Verwendung von Schach-APIs

Studienarbeit (Modul T3_3101)

des Studiengangs Informatik

an der DHBW Stuttgart Campus Horb

von

Marvin Schlegel

25. Mai 2025

Matrikelnummer, Kurs	1490481, INF2022
Betreuer	Daniel Schaber

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich meine Arbeit mit dem Thema *Entwicklung einer Schach-Trainingsanwendung zur Erlernung von Eröffnungen unter Verwendung von Schach-APIs* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

ORT, DATUM

_____

AUTOR

Abstract

Im Rahmen dieser Arbeit wurde untersucht, wie Menschen allgemein lernen und wie sie Schach lernen. Dabei stellte sich heraus, dass es besonders wichtig ist Inhalte zu wiederholen. Im Optimalfall wiederholt man sie mithilfe von Spaced Repetition in exponentiell größer werdenden Abständen. Das aktuelle Angebot an Lernmedien, bietet diese Funktionalität aber selten. In dieser Arbeit wurde eine Webseite erstellt, auf der Schachspieler Eröffnungen interaktiv betrachten und üben können. Den Nutzern wird auch eine Empfehlung gegeben, welche Eröffnung als nächstes wiederholt werden soll, auf Basis der genannten wissenschaftlichen Erkenntnissen. Das entstandene Produkt lässt sich gut in Verbindung mit bestehenden Lernmedien verwenden.

Inhaltsverzeichnis

Abkürzungsverzeichnis

MCTS	Monte Carlo Tree Search
UCI	Universal Chess Interface
API	Application Programming Interface
REST	REpresentational State Transfer
TCEC	Top Chess Engine Championship
NNUE	Efficiently Updatable Neural Network
PGN	Portable Game Notation
CCRL	Computer Chess Rating Lists
URI	Unique Resource Identifier
ORM	Object Relational Mapping
MAC	Message Authentication Code
HMAC	Hash-based MAC
JWT	JSON Web Tokens

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

Schach ist ein weit verbreitetes und beliebtes Spiel. In den letzten Jahren gab es einen großen Anstieg an Beliebtheit, verursacht durch die Pandemie und Online-Schachplattformen, wie Chess.com. Auch die Serie Queens Gambit und die vergrößerte Präsenz von Schach in sozialen Medien haben stark dazu beigetragen. Im Jahr 2022 schrieb Chess.com „In [den letzten] 18 Monaten haben wir mehr neue Mitglieder gewonnen, als in den 13 Jahren zuvor zusammen.“ [Che22] Aus diesem Grund gibt es auch immer mehr Personen, die ihr Schachspiel verbessern und Strategien lernen wollen.

1.1 Motivation

Ein wichtiger Teil der Strategien sind die Eröffnungen. Die ersten Züge eines Spiels entscheiden oft in welche Richtung sich das Spiel entwickelt. Durch eine gut geplante Eröffnung kann ein Spieler frühzeitig die Kontrolle über das Zentrum gewinnen, die Figuren optimal platzieren und Schwachstellen minimieren. Wer hingegen nicht vertraut mit Eröffnungen ist, wird schnell zum Opfer von Fallen und taktischen Manövern. Im schlimmsten Fall verliert man ein Spiel, noch bevor es richtig begonnen hat. Mit einem fundierten Wissen über Eröffnungen können auch Schwachstellen in typischen Stellungen schneller erkannt und ausgenutzt werden. Auf diese Weise kann man sich frühzeitig einen Vorteil verschaffen und das Spiel zu seinen Gunsten gestalten. Das Erlernen von Schacheröffnungen gestaltet sich häufig anspruchsvoll, da sie nicht nur komplex sind, sondern auch schnell wieder in Vergessenheit geraten können. Daher ist es essenziell, Eröffnungen auf eine systematische und regelmäßige Weise zu trainieren.

Zum Erlernen von Schacheröffnungen existieren viele Bücher Eröffnungen und weitere Medien, wie Videos und Online-Kurse. Diese Materialien eignen sich gut um ein Verständnis von Eröffnungen aufzubauen und die zentralen Ideen und Ziele dahinter zu verstehen. Sie sind jedoch oft wenig interaktiv, wodurch das Lernen häufig auf reines Auswendiglernen reduziert wird. Für viele Lernende ist dadurch das Lernen erschwert und Eröffnungen werden schnell wieder vergessen. Durch eine Computeranwendung kann der Prozess interaktiver gestaltet werden. Chess.com bietet ein interaktives Training für Eröffnungen an. Diese Interaktivität beschränkt sich allerdings darauf, dass der Spieler vorgeschriebene Züge nachspielen soll. Ein weiterer Nachteil ist, dass ein monatliches Abonnement benötigt um Zugriff auf das Training zu bekommen. Für Spieler, die nicht monatlich Geld ausgeben

wollen gibt es kaum Alternativen. Es existiert zu diesem Zeitpunkt kein Programm, das ein individuell zugeschnittenes und personalisiertes Lernen von Schacheröffnungen ermöglicht, bei dem spezifische Bedürfnisse, Stärken und Schwächen der Lernenden gezielt berücksichtigt werden.

1.2 Ziel der Arbeit

Ziel dieser Arbeit ist es eine Schach-Trainingsanwendung zu entwickeln, welche es Spielern ermöglicht Eröffnungen interaktiv zu erlernen. Dabei soll die Anwendung nicht nur die grundlegenden Prinzipien und Variationen der Eröffnungen vermitteln, sondern auch eine personalisierte Lernerfahrung bieten, die sich an den individuellen Fortschritt und die spezifischen Bedürfnisse der Spieler anpasst.

1.3 Geplantes Vorgehen

Zu Beginn wird untersucht, wie das menschliche Gehirn lernt und wie es dabei unterstützt werden kann. Darauffolgend wird erläutert, wie Schachcomputer funktionieren und wie diese mithilfe von Schnittstellen in ein Programm eingebunden werden können. In diesem Zusammenhang werden auch Schnittstellen im Allgemeinen betrachtet.

In Kapitel 3 wird der aktuelle Markt darauf untersucht, welche Lernmaterialien und Tools es bereits gibt. Es wird auch verglichen, welche Schachengines und Datenbanken sich für das geplante Programm eignen.

Anschließend wird die Architektur der Anwendung konzipiert unter Berücksichtigung der technischen und funktionalen Anforderungen. Auf dieser Grundlage wird ein Prototyp entwickelt, der die zentralen Funktionen der Schach-Trainingsanwendung demonstriert.

2 Theoretische Grundlagen

In den folgenden Kapiteln sollen die theoretischen Grundlagen erläutert werden. Zunächst wird untersucht, wie das Lernen im Allgemeinen funktioniert und wie diese Erkenntnisse auf das Lernen von Eröffnungen angewendet werden können. Danach wird gezeigt, wie die Lerntechnik Spaced Repetition beim Lernen helfen kann. Anschließend wird auf die Erkenntnisse im Bereich der Schachpsychologie eingegangen. Dabei wird betrachtet, wie Schachspieler nach heutigem Wissensstand lernen und spielen. Schließlich werden Schnittstellen in Computerprogrammen besprochen und welche Schnittstellen bei diesem Projekt nötig sind.

2.1 Allgemeines Lernen

Über das Thema Lernen gibt es mehrere Theorien und Modelle. Aus diesen Theorien können teilweise Schlüsse gezogen werden, wie das Erlernen von Schacheröffnungen gestaltet werden kann. Im Folgenden werden die Theorie des Behaviorismus, des Lernen am Modell und die strukturgenetische Lerntheorie kurz beschrieben. Danach wird die Funktionsweise unseres Gedächtnisses betrachtet.

2.1.1 Behaviorismus

Der Behaviorismus basiert auf tatsächlich beobachtbarem Verhalten. Er besagt, dass jeder Mensch durch seine Umwelt beeinflusst wird. Das bedeutet, dass man das Verhalten auch verändern kann durch Stimulation, primär durch positive Verstärkung und negative Verstärkung. Bei gutem Verhalten ist also Belohnung und Lob sinnvoll und bei schlechtem Verhalten können Strafen eingesetzt werden. Vor allem die positive Verstärkung sorgt dafür, dass eine Person extrinsische Motivation bekommt und auch gerne weiterlernt. [KJS24]

Bezogen auf eine Schachanwendung kann man bei einem guten Zug positives Feedback geben zum Beispiel durch Lob oder durch positive Symbole, Farben und Animationen. Man könnte auch Belohnungen einführen, wenn der Spieler mehrere aufeinanderfolgende Tage Übungen durchgeführt hat, oder sich verbessert.

2.1.2 Lernen am Modell

Eine weitere Theorie ist, dass Menschen an Modellen lernen. Das bedeutet, dass sie durch Beobachtung eines Modells neue Dinge erlernen können. Dieses Modell kann zum Beispiel eine andere Person sein. Dieser Effekt tritt insbesondere dann auf, wenn die beobachtete Aktion in einem positiven Ergebnis resultiert. Dann ist die Wahrscheinlichkeit am höchsten, dass diese Aktion in einer ähnlichen Situation nachgeahmt wird. [KJS24]

In einer Schachanwendung kann man dieses Wissen so anwenden, dass man dem Spieler einen Zug vorführt und ihm dadurch zeigt, wie dieser einen Vorteil bringt. In einer ähnlichen Position kann der Spieler sich dann eher an den Zug erinnern und ihn durchführen.

2.1.3 Strukturgenetische Lerntheorie

Die strukturgenetische Lerntheorie geht davon aus, dass ein Mensch am besten lernt, wenn er selbst verschiedene Handlungswege ausprobieren kann. Es geht darum, Personen in Herausforderungen zu versetzen und sie durch selbstständige Entdeckung etwas lernen zu lassen. Voraussetzung dafür ist, dass die Aktionen im Nachhinein reflektiert werden und dass die Person auch die notwendigen Kenntnisse und Mittel dafür hat. Dadurch kann sich Erfahrung bilden, denn „Erfahrung ist [...] immer reflektiertes oder durch Reflexion bestimmtes Tun.“[KJS24]

In einer Schachanwendung kann das durch die Implementierung von Rätseln geschehen. Man kann dem Spieler eine bestimmte Anfangsposition geben in welcher nur eine Auswahl an Zügen zu einer guten Position führen. Es wäre auch möglich ein Analysetool zur Verfügung zu stellen, mit welchem ein vergangenes Spiel reflektiert werden kann. So können gute und schlechte Züge identifiziert werden und der Spieler kann lernen, welche Züge gut funktioniert haben und welche nicht.

2.1.4 Funktionsweise des Gedächtnis

Für das Erlernen von Schacheröffnungen ist es auch sinnvoll zu verstehen, wie das menschliche Gedächtnis funktioniert. Das Gedächtnis bildet die Grundlage des Lernens. Es wird als Netzwerk verstanden, das mit den Wahrnehmungsprozessen verbunden ist. Eine Besonderheit des menschlichen Gedächtnisses ist, dass sich Erinnerungen durch wiederholtes Erinnern verändern. Jedes mal, wenn man sich an etwas erinnert verändern sich die Erinnerungen ein wenig und es wird eventuell mehr Kontext hinzugefügt. Das bedeutet, dass eine Erinnerung, immer subjektiver wird und somit auch verfälscht werden kann. Außerdem kann es auch zum Vergessen kommen, indem vorhandene Erinnerungen überschrieben

werden. Das Abspeichern von Informationen kann unterstützt werden durch verschiedene Lerntechniken. Dazu gehört zum Beispiel das Wiedergeben von Inhalten, Gruppieren von Inhalten oder Herausfiltern von Hauptideen. Gespeichert werden Inhalte auf drei unterschiedliche Arten. Das sensorische Gedächtnis ist für Inhalte zuständig, die jetzt im Moment wahrgenommen werden. Damit sind äußere Reize und innere Zustände gemeint. Das Kurzzeitgedächtnis verarbeitet aktuelle Informationen zu Wissen. Seine Kapazität ist allerdings stark begrenzt. Deshalb können Inhalte nicht lange im Kurzzeitgedächtnis behalten werden, sie werden durch neue Inhalte ersetzt und verdrängt. Im Langzeitgedächtnis werden Inhalte des Kurzzeitgedächtnisses übernommen. Dieser Prozess kann durch Lerntechniken unterstützt werden. Das Langzeitgedächtnis hat eine sehr große Kapazität und behält Informationen lange um später darauf zurückgreifen zu können. Hier wird wiederum zwischen dem deklarativen und dem prozeduralen Gedächtnis unterschieden. Das deklarative Gedächtnis ist zuständig für bewusst abrufbare Inhalte, wie persönliche Erlebnisse, Fakten und allgemein Inhalte die mit Worten kommuniziert werden können. Das prozedurale Gedächtnis enthält auch Wissen, das nicht unbedingt bewusst wahrgenommen wird. Dort sind auch zum Beispiel motorische Kenntnisse gespeichert. [KJS24]

Aus diesen Erkenntnissen kann man schlussfolgern, dass es besonders wichtig ist Inhalte zu wiederholen, damit sich Gelerntes nicht verfälscht und es nicht vergessen wird. Im Schach sollte man um Eröffnungen zu erlernen diese also häufig betrachten. Durch das eigene Spielen und Erinnern kann das Übergehen dieser Eröffnungen in das Langzeitgedächtnis unterstützt werden. Bestimmte Variationen, welche seltener vorkommen sollten auch wiederholt werden, um ein Vergessen, Überschreiben oder Verfälschen zu verhindern.

2.2 Lernen durch Spaced Repetition

Der Zeitpunkt, wann etwas wiederholt werden sollte, ist sehr wichtig und wirkt sich deutlich darauf aus, wie gut etwas im Langzeitgedächtnis behalten wird. Auch dazu gibt es bereits wissenschaftliche Erkenntnisse. Zwei grundsätzlich unterschiedliche Herangehensweisen, die untersucht wurden, sind das Spacing und das Cramming.

2.2.1 Cramming

Beim Cramming lernt man sehr viele Inhalte in einem kurzen Zeitraum. Dieses Vorgehen wird oft von Schülern verwendet, wenn sie sich auf Klausuren vorbereiten, da es sehr zeiteffizient wirkt. Auf kurze Sicht kann man dadurch mit relativ geringem Aufwand erfolgreich sein. Auf lange Sicht ist allerdings das Spacing effektiver. [SMS17]

2.2.2 Spacing

Das Spacing basiert auf einer Erkenntnis von Ebbinghaus [Her85], der in einem Sebstexperiment feststellte, dass er sich erfundene Wörter besser merken kann, wenn er beim Lernen Pausen einlegt. Insgesamt hatte er also beim Lernen mit Spacing weniger Zeitaufwand als beim Lernen mit Cramming. Ebbinghaus endteckte außerdem die sogenannte Vergessenskurve. Sie wird durch folgende Formel beschrieben:

$$R = e^{(-t/S)}$$

R steht hier für die Behaltensfähigkeit, S für die relative Erinnerungsstärke und t für die Zeit. An dieser Funktion kann man auch erkennen, dass die Erinnerung, wie bei einer inversen Exponentialfunktion, zuerst stark fällt und dann langsamer. Darin begründet sich auch der Lag Effect, welcher beschreibt, dass das Lernen noch effizienter ist, wenn man mit größer werdenden Pausen arbeitet. Am Anfang sollte man einen Inhalt also am häufigsten wiederholen und danach in immer größer werdenden Abständen. Allgemein kann auch gesagt werden, dass man Inhalte genau dann wiederholen sollte, wenn man kurz davor ist sie zu vergessen. Am nächsten kommt man an diese Grenze, indem man Spacing verwendet und den Lag Effect ausnutzt. [SMS17]

2.2.3 Umsetzung von Spacing

Das Verwenden von Spacing um Inhalte zu lernen wird auch Spaced Repetition genannt. In der Vergangenheit wurden einige Techniken entwickelt um dieses Vorgehen umzusetzen.

Statische Modelle

Eine davon ist die Pimsleur Methode. Hierbei werden Inhalte in exponentiell größer werdenden Intervallen abgefragt. Kritik an diesem Vorgehen ist, dass die Intervalle im Vorhinein festgelegt werden und nicht an das Können der Lernenden angepasst werden kann.

Leitner [Lei73] entwickelte ein System, dass primär für Karteikarten gedacht war. Bei diesem Vorgehen verwendet man mehrere Kästen, in die man Karteikarten einordnen kann. Jeder Kasten steht dabei für ein anderes Wiederholungsintervall. Neue Karteikarten starten im ersten Kasten und werden am häufigsten wiederholt. Wenn man sich korrekt an eine Karte erinnert hat, wandert diese in den nächsten Kasten mit einem längeren Wiederholungsintervall. Wenn man sich falsch an eine Karte erinnert hat, wandert die Karte zurück in den vorherigen Kasten mit einem höheren Wiederholungsintervall. Dieses

System ist gut geeignet für Karteikarten, doch durch die Verwendung von Computern ist es möglich genauere Methoden zu entwickeln.

Halbwertszeitregression

Eine neuere Herangehensweise ist die Halbwertszeitregression. Als Basis dafür wird die Vergessenskurve von Ebbinghaus aus Unterabschnitt 2.2.2 verwendet. R ist dann die Wahrscheinlichkeit p, dass ein Inhalt richtig wiederrufen wird, t die Zeit seit der letzten Trainingseinheit Δ und S die Halbwertszeit h und die Basis e wird durch 2 ersetzt. Dadurch erhält man die Formel:

$$p = 2^{-\Delta/h}$$

Die Funktion $\hat{h}_\Theta = 2^{\Theta * x}$ beschreibt nun die geschätzte Halbwertszeit, wobei x ein Merkmalsvektor ist, der die vorherige Erfahrung des Lernenden beschreibt und Θ ein Parametervektor, der jeder Eigenschaft aus x eine Gewichtung zuordnet. Als Merkmale kann zum Beispiel die Anzahl der erfolgreichen Trainingseinheiten, fehlgeschlagenen Trainingseinheiten und Anzahl Trainingseinheiten insgesamt verwendet werden. Die Algorithmen von Pimsleur und Leitner können auch durch eine Ausprägung des Parametervektors in dieser Funktion dargestellt werden. Ziel der Halbwertszeitregression ist es, ein Θ zu finden, das möglichst genau die tatsächliche Vergessenskurve beschreibt. Dafür werden vergangene Trainingsergebnisse als Trainingsdaten verwendet. Über eine Verlustfunktion kann erkannt werden, wie gut das aktuelle Modell ist und mithilfe eines Gradientenabstiegverfahrens kann es angepasst werden. Dieses Modell wurde bereits erfolgreich in der Sprachlernanwendung Duolingo verwendet, mit einer reduzierten Fehlerrate von 45% im Vergleich zum Leitner-System. Da das Lernen von Schacheröffnungen, genau wie das Lernen von Sprachen, zu einem großen Teil aus Auswendiglernen besteht, ist zu erwarten, dass das Modell beim Lernen von Schacheröffnungen ähnliche Vorteile bringt. [SM16]

2.3 Schachpsychologie

In der Schachpsychologie wurde von unterschiedlichen Personen bereits einige gemeinsame Beobachtungen gemacht. Gobet und Jansen haben in [GJ06] die folgende Kernaussagen gesammelt:

1. Ein Schachspieler hat einen hocheffizienten Überblick. Er erkennt zentrale Elemente einer Position sehr schnell.
2. Schachspieler können sich Schachpositionen und Spiele außergewöhnlich gut merken. Diese Fähigkeit ist außerhalb von Schach nicht erkennbar.

3. Ihr Wissen besteht aus mehreren Ebenen, im speziellen die niedrige Ebene, welche aus Muster von Figuren besteht und eine hohe konzeptionelle Ebene, welche sich mit Plänen und Bewertungen von Positionen befasst.

4. Die Spieler suchen sehr selektiv nach guten Zügen. Es werden nur ganz bestimmte Pfade durchdacht und andere sehr schnell verworfen.

5. Es gibt keinen Unterschied in dem Suchalgorithmus eines Expertenspielers und dem eines Großmeisters.

6. Meister verlieren in simultanen Spielen relativ wenig ihrer Fähigkeiten.

2.3.1 Die Template Theorie

Aufschluss über diese Beobachtungen soll die Template Theorie geben, welche von Gobet und Simon in [GS96] beschrieben wird. Die Template Theorie berücksichtigt, dass das kognitive System der Menschen aus drei Hauptbestandteilen besteht, wie in Unterabschnitt 2.1.4 beschrieben. Das sensoriesche Gedächtnis wird hier räumlich-zeitlicher Speicher genannt und es wird auch das stark begrenzte Kurzzeitgedächtnis und das Langzeitgedächtnis näher betrachtet. Bei dem Langzeitgedächtnis wird wieder zwischen deklarativem und prozeduralem Wissen unterschieden. [GJ06]

Bei dem deklarativen Wissen in Bezug auf Schach wird vermutet, dass Chunks eine große Rolle spielen. Das sind kleine Muster, also Anordnungen von einigen Spielfiguren, die öfter bei einem Schachspiel vorkommen. Ein Beispiel für einen Chunk könnte die Aufstellung in Abbildung 2.1 sein. Dort ist eine Position zu sehen, die häufig nach einer Rochade entsteht. Ein geübter Spieler erkennt dieses Muster sofort und weiß, dass der König relativ sicher ist. Es wird geschätzt, dass das Kurzzeitgedächtnis Platz für ungefähr sieben Chunks hat. Ein großer Unterschied zwischen geübten und weniger geübten Schachspielern ist also, wie viele sie in ihrem Langzeitgedächtnis haben. Je besser ein Schachspieler ist, desto

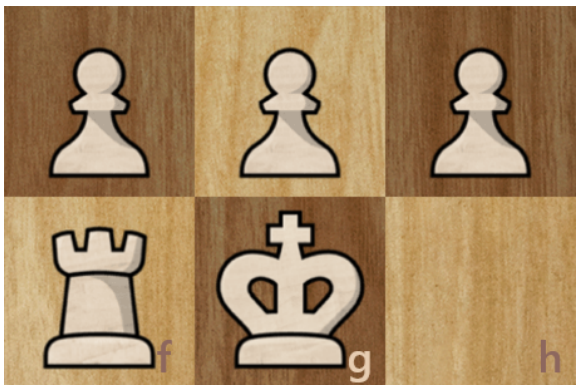

Abbildung 2.1: Beispiel eines Chunks

mehr und desto größere befinden sich in seinem Langzeitgedächtnis. Es wird vermutet, dass ein geübter Spieler ungefähr 50.000 in seinem Langzeitgedächtnis hat. Wenn ein Muster erkannt wird, muss im Kurzzeitgedächtnis nur ein Zeiger auf den Chunk im Langzeitgedächtnis behalten werden. Auf diese Weise können geübte Schachspieler sich sehr effizient Schachpositionen merken. Das erklärt auch, warum sich diese Fähigkeit nicht auf Bereiche außerhalb des Schachs auswirkt. Selbst zufällig angeordnete Positionen, die in regulären Spielen selten vorkommen, können sich gute Spieler schlecht merken. Erweitert wird diese Theorie durch Templates, also Vorlagen. Templates sind spezielle Chunks, welche Platzhalter für bestimmte Figuren beinhalten. Eine Template kann also auch angewendet werden, wenn sich vereinzelte Figuren an einer anderen Position befinden. So können durch Templates noch mehr Positionen abgedeckt werden, als es durch einfache Chunks möglich ist. Viele Chunks und Templates reduzieren die Notwendigkeit voraus zu schauen. Wenn bekannten Chunks begegnet wird, ist auch klar, wie gut diese Position zu bewerten ist und welche Züge in Frage kommen. [GS96]

Dieses Wissen, was als nächstes getan werden kann, wird von Gobet und Jansen unter dem prozedurales Wissen eingeordnet. Es wird durch sogenannte Produktionen abgespeichert. Produktionen sind Wissenseinheiten, welche aus Bedingung und Aktion bestehen. Ein Beispiel wäre, „Wenn es eine Linie ohne Figuren gibt und du einen Turm besitzt, dann platziere den Turm auf dieser Reihe.“ Durch diese Produktionen können Spieler schnell Entscheidungen treffen. Dieser Mechanismus kann bewusst oder unbewusst stattfinden und wird oft als Intuition verstanden. [GJ06]

2.3.2 Erlernen von Eröffnungen

Das Lernen von Eröffnungen ist sehr zeitaufwendig. Es wird vermutet, dass sieben bis 10 Sekunden benötigt werden um einen Chunk im Langzeitgedächtnis zu lernen. Außerdem wird es passieren, dass man Dinge wieder vergisst. Deshalb wird empfohlen sich auf wenige Eröffnungen zu konzentrieren. Für diese Eröffnungen kann man dann die häufigen Variationen und Positionen lernen. Man sollte so viele Variationen lernen, dass man auf die typischen Züge des Gegners reagieren kann. Indem man die Anzahl der Eröffnungen limitiert, erhöht man die Wahrscheinlichkeit, dass gelernten Chunks und Templates in einem richtigen Spiel begegnet wird. Beim Erlernen von Eröffnungen ist es außerdem wichtig eine Balance zwischen Auswendiglernen und Verständnis zu finden. Um sich die Abfolge der Züge zu merken muss man selbstverständlich viel Auswendiglernen, allerdings ist das Verständnis der Hauptideen nützlich um auch in unbekannten Situationen sinnvolle Züge zu finden. Dazu kommt, dass das Herausfiltern von Hauptideen, wie in Unterabschnitt 2.1.4 beschrieben auch eine Lerntechnik ist, die beim Übertragen von Inhalten in das Langzeitgedächtnis helfen kann. Man sollte die Eröffnungen auch aus unterschiedlichen Perspektiven betrachten

und Verknüpfungen zu Mittelspielen und Endspielen herstellen. Das soll die Erstellung von Templates beschleunigen. Ein zentraler Ort um die Eröffnungen nachzuschauen ist auch sinnvoll. Damit kann man sein Wissen einfach erneuern und verhindert, dass die Erinnerungen verblassen oder sich verfälschen, wie in Unterabschnitt 2.1.4 beschrieben. [GJ06]

Für eine Schachanwendung bedeutet das, dass man einen Spieler dazu ermutigen sollte eine Eröffnung und ihre Variationen im Detail zu erlernen. Man sollte die Spieler auch dazu bringen die Eröffnungen oft zu betrachten. Hier kann die Schachanwendung selbst der zentrale Ort werden um Eröffnungen nachzuschauen und seine Erinnerung aufzufrischen. Um die Verknüpfungen zwischen Eröffnungen und Mittelspielen herzustellen kann man dem Spieler die Möglichkeit geben, von einer bestimmten Eröffnung als Startpunkt gegen einen Computergegner zu spielen. Für die Verknüpfung von Eröffnungen und Endspielen kann die von Gobet und Jansen in [GJ06] vorgeschlagene Dekompositionsmethode verwendet werden. Hierbei werden nach der Eröffnung fast alle Figuren außer den Bauern und Königen vom Spielfeld entfernt. Dadurch bekommt man ein besseres Gefühlt dafür, welche Seite eine bessere Position hat. Man kann nach der Dekomposition den Spieler auch von dieser Position aus gegen einen Computer spielen lassen. Das Verstehen der Hauptideen von Eröffnungen kann durch beschreibende Infotexte erreicht werden. Auf diese Weise werden alle zuvor beschriebenen Bereiche abgedeckt.

2.3.3 Lernunterstützende Medien

Es gibt viele Möglichkeiten das Lernen von Schacheröffnungen unterstützen. Heutzutage hat man viele Medien zur Auswahl, wie Bücher, Online Kurse, Videos oder Computerprogramme. Gobet und Jansen schreiben, dass die Reihenfolge und Art, wie Lernmaterial aufgeteilt wird essentiell für effektives Training ist. Sie empfehlen, dass eine lernende Person das nicht selbst auswählen sollte. Es sei wichtig, einen guten Trainer, gute Computerprogramme oder ein gutes Buch zu haben [GJ06].

Bücher waren lange Zeit das einzige und damit das Hauptmedium, über das Schachwissen übertragen wurde. Gobet und Jansen kritisieren, dass viele Bücher keinen psychologischen und pädagogischen Prinzipien folgen. Die meisten Bücher würden Schema nur anhand einer kleinen Anzahl an Positionen präsentieren. Außerdem seien Diagramme zu selten vorhanden [GJ06]. Für Anfänger kann es schwierig sein anhand der textuellen Notation der Züge dem Text zu folgen. Ein Vorteil von Büchern ist, dass sie viel Hintergrundinformationen geben können. Sie können die langfristigen Ziele gut beschreiben und Kernideen erklären.

Ein Trainer kann einen genauen Trainingsplan für Lernende vorbereiten. Seine Erfahrung hilft ihm dabei, das richtige Material auszuwählen und zu wissen welche Methoden

effektiv sind. Er kann die Schwächen eines Spielers erkennen und zielgerichtet Übungen vorbereiten um diese auszubessern. Auch das persönliches Feedback und Ratschläge können sehr hilfreich sein. Die Notwendigkeit eines Trainers ist umstritten, doch wenn man die Möglichkeit hat mit einem Trainer zu lernen, ist es wahrscheinlich die beste Option. [GJ06]

Es existiert bereits eine beachtliche Anzahl an Computerprogrammen, die einem Schachspieler eine umfangreiche Lernerfahrung bieten, oder ihn beim Lernen unterstützen können. Es gibt Plattformen mit Erklärvideos, interaktiven Lektionen und anschließenden Übungsaufgaben. Viele Webseiten bieten Puzzles und andere Taktikaufgaben an um sein Können zu trainieren. Auf Online Plattformen kann man auch über das Internet gegen andere Spieler spielen, die ein ähnliches Können haben. Computergegner bieten eine gute Möglichkeit neue Ideen auszuprobieren und seine Taktik zu verbessern. Mithilfe von Computeranalysen können Schwächen im eigenen Spiel ausgemacht werden und mithilfe von Spieldatenbanken können häufig gespielte Variationen genauer analysiert werden.

Auch Gobet und Jansen sehen das Potential für computergestütztes Lernen. Sie erwähnen die Nützlichkeit von Spieldatenbanken, Computergegnern und Analysen. Sie beschreiben auch, dass zukünftige Computerprogramme noch weitergehen können, indem sie wie ein Trainer personalisiert Übungsmaterial heraussuchen basierend auf den Stärken und Schwächen des Spielers. [GJ06]

Gusev verwendet auch Computer um auf Schacheröffnungen vorzubereiten. In [Gus21] zählt er verschiedene Softwareprodukte auf, die beim Erlernen von Schacheröffnungen helfen können. Darunter sind Schachengines, Spieldatenbanken, Cloud Services und Puzzles. Er präsentiert eine computerbasierte Herangehensweise um Eröffnungen zu trainieren. Dafür hat er mithilfe von Schachengines, Spieldatenbanken und einem Cloud Service beliebte Eröffnungsvarianten identifiziert und sortiert. Mögliche Fortführungen wurden durch eine Schachengine mit Eröffnungsbuch herausgefunden. Das Resultat war ein nützliches Werkzeug um Studenten Eröffnungen zu zeigen.

2.4 Funktionsweise von Schachengines

Seit langer Zeit wird die Fähigkeit Schach zu spielen mit Intelligenz verbunden. Daher gab es auch großes Interesse daran Computern das Spielen von Schach beizubringen. Es wurden Preise ausgeschrieben für die Programme, die als erstes eine bestimmte Wertung erreichen oder den Weltmeister besiegen. Das erste Mal, dass ein Computer einen Weltmeister besiegte war im Jahr 1997. Das von IBM entwickelte Programm namens „Deep Blue“ besiegte den aktuellen Weltmeister Garry Kasparov. Seitdem ist die Lücke zwischen

Menschen und Computern immer größer geworden und heutzutage kann kein Mensch mehr gegen die besten Schachcomputer gewinnen. [Vje19]

Computerprogramme, die Schach spielen werden typischerweise Schachengines genannt. Es gibt unterschiedliche Formen von Schachengines. Die einen nutzen den Minimax-Algorithmus in Verbindung mit Alpha-Beta-Pruning. Das ist auch die Art von Schachengines, die zuerst entwickelt wurden. Eine weitere Form wurde in den letzten Jahren entwickelt. Diese arbeiten mit reinem Reinforcement Learning und erlernen das Spiel, indem sie zahlreiche Spiele gegen sich selbst spielen. In den Folgenden zwei Kapiteln werden beide Herangehensweisen näher erläutert.

2.4.1 Alpha-Beta-Pruning

Der Alpha-Beta-Pruning Algorithmus kann genutzt werden um ein Spiel mit zwei Spielern, die gegensätzliche Ziele verfolgen, zu spielen. Der Algorithmus baut auf dem Minimax Algorithmus auf und optimiert diesen durch Kürzen (Pruning) des Suchbaums.

Für den Minimax Algorithmus wird ein Spiel anhand von Positionen p definiert und anhand von Regeln, wie von einer Position in eine andere übergegangen werden kann. Wenn es keine weiteren legalen Züge gibt, dann ist eine Terminalposition erreicht. Diese Struktur kann als sogenannter Spielbaum dargestellt werden. Die Positionen bilden die Knoten und wenn ein Übergang von einer Position in eine andere existiert, wird eine Kante eingezeichnet. Falls eine Position schon einmal erreicht wurde darf allerdings keine Kante hinzugefügt werden um Zyklen zu verhindern. Bei diesem Baum ist h die Tiefe des Baumes und d ist die Anzahl an Verzweigungen an einem Knoten. Die Terminalpositionen bilden die Blattknoten. [KM75]

Jeder Terminalposition wird ein Wert zugewiesen durch die Evaluationsfunktion $f(p)$. Für einen Spieler ist der Wert des Spiels $f(p)$ und für den anderen der Wert $-f(p)$. Beide Spieler möchten jeweils ihren Wert maximieren. Alternativ kann man auch formulieren, dass ein Spieler den Wert von $f(p)$ maximieren will und der Gegner will ihn minimieren. Beide Herangehensweisen sind äquivalent. Wenn p eine Position ist, bei der es es d legale Züge $p_1, \ldots, p_d$ mit $d > 1$ gibt, dann ist es die Aufgabe des Algorithmus, den besten Zug auszuwählen. Der beste Zug ist der Zug, bei dem der beste Wert für den Spieler am Zug erreicht wird. Es wird davon ausgegangen, dass der Gegner auch jeweils die Züge auswählt, die für ihn zu dem besten Wert führen. Es sei $F(p)$ der größtmögliche Wert, der von Position p aus erreichbar ist gegen einen optimal spielenden Gegner. Die Funktion $F(p)$

wird definiert durch:

$$F(p) = \begin{cases} f(p) & \text{für } d = 0 \\ max(-F(p_1), \ldots, -F(p_d)) & \text{für } d > 0 \end{cases} \tag{2.1}$$

Mit Pseudocode kann diese Funktion folgendermaßen abgebildet werden:

```
function F(p: Position): Int {
    ps = findSuccessors(p)
    d = ps.length
    if (d == 0) {
        return f(p)
    }
    m = -infinity
    for (pi in ps) {
        t = -F(pi)
        m = max(t, m)
    }
    return m
}
```

Listing 2.1: Minimax-Algorithmus

Dieser Algorithmus durchsucht alle Möglichen Fortsetzungen von p und findet den Zug mit dem besten Wert. [KM75]

Es ist allerdings nicht immer notwendig alle Fortsetzungen zu zu überprüfen. Oft kann erkannt werden, dass eine bestimmte Zugfolge keinen besseren Wert mehr erreichen kann und daher ignoriert werden kann. Wenn man zum Beispiel eine Zugfolge herausgefunden hat, die mit dem Wert 3 bewertet wird und man über eine andere Zugfolge sagen kann, dass diese nicht besser als 3 wird, muss man diese nicht weiter nachverfolgen. Wenn also $-F(p_1) = 3$ ist, dann ist $F(p) \geq 3$ und jeder Pfad für den $-F(p_i) \leq 3$ gilt, kann ignoriert werden. Der Gegner kann in diesem Fall den Zug so wählen, dass entweder derselbe Wert oder ein für uns kleinerer Wert erzielt wird. Indem man den aktuell höchsten Wert zwischenspeichert, ist es möglich bestimmte Suchzweige auszulassen. [KM75]

Dieses Vorgehen kann weiter verbessert werden indem man den höchsten und den kleinsten Wert zwischenspeichert. Das wird Alpha-Beta-Pruning genannt. Knuth und Moore definieren die Funktion folgendermaßen:

$$
\begin{aligned}
F2(p, \alpha, \beta) &\leq \alpha \qquad \text{für } F(p) \leq \alpha \\
F2(p, \alpha, \beta) &= F(p) \quad \text{für } \alpha < F(p) < \beta \\
F2(p, \alpha, \beta) &\geq \beta \qquad \text{für } F(p) \geq \beta
\end{aligned}
\tag{2.2}
$$

In Pseudocode kann der Algorithmus folgendermaßen umgesetzt werden:

```
function F2(p: Position, alpha: Int, beta: Int): Int {
    ps = findSuccessors(p)
    d = ps.length
    if (d == 0) {
        return f(p)
    }
    m = alpha
    for (pi in ps) {
        t = -F2(pi, -beta, -m)
        m = max(t, m)
        if (m >= beta) break
    }
    return m
}
```

Listing 2.2: Alpha-Beta-Pruning

[KM75]

Für diesen Algorithmus können weitere Verbesserungen vorgenommen werden. Einen großen Faktor spielt die Reihenfolge, in der die Züge untersucht werden. Bei einer optimalen Reihenfolge muss der Alpha-Beta-Algorithmus $d^{\lceil h/2 \rceil} + d^{\lfloor h/2 \rfloor} - 1$ Terminalpositionen bewerten. [KM75] Das ist dann der Fall, wenn beim Durchsuchen des Baumes immer der richtige Zug ausgewählt wird, sodass er maximal gekürzt werden kann. Der Algorithmus kann dahingehend verbessert werden, dass Züge, die mit hoher Wahrscheinlichkeit einen Vorteil bringen zuerst betrachtet werden. Im Schach können zum Beispiel Züge in welchen eine Figur geschlagen wird zuerst betrachtet werden, da diese oft einen Vorteil bringen. Andere Züge, wie zum Beispiel Damenopfer können später betrachtet werden. Um die Reihenfolge der Züge für eine tiefere Suche zu verbessern, wird häufig zunächst eine flachere Suche durchgeführt. Dieses Verfahren wird als iterative Vertiefung (iterative deepening) bezeichnet. Eine optimale Reihenfolge kann man dadurch jedoch nicht garantieren. Wie in der Gleichung zu sehen ist steigt die Anzahl der zu besuchenden Knoten selbst im Optimalfall exponentiell. Es kann also nur eine bestimmte Anzahl an Zügen in die Zukunft geschaut werden. Für viele Spiele, einschließlich Schach, ist die Gesamtzahl der möglichen

Züge in einem Spiel jedoch so hoch, dass es unmöglich ist, alle Szenarien vollständig zu analysieren. Aus diesem Grund wird eine bestimmte Suchtiefe festgelegt, ab der die Positionen wie Terminalpositionen behandelt werden.[KM75] Es muss also für diese Position auch eine Evaluationsfunktion f existieren. In der naivsten Form kann dafür die Anzahl der Figuren der Spieler genommen werden. In komplexeren Schachengines werden Neurale Netze verwendet um Positionen zu bewerten. Eine Tabelle mit bereits berechneten Positionen kann Berechnungen auch beschleunigen. Für Eröffnungen und Endspiele können Eröffnungsbücher und vorgerechnete Endspiel Tabellen verwendet werden.[Sil+17a]

2.4.2 Reinforcement Learning

Der klassische Alpha-Beta-Pruning Algorithmus kann nur durch viel spielbezogenes Wissen verbessert werden. Im Gegensatz dazu kann im Reinforcement Learning ein Computer ein Spiel ohne Vorwissen erlernen, indem er wiederholt Spiele gegen sich selbst simuliert. Das bekannteste Beispiel dafür ist AlphaZero, ein Algorithmus, der nach wenigen Stunden Training die besten Programme für Schach, Shogi und Go schlagen konnte. [Sil+17a] In diesem Kapitel wird die Funktionsweise von AlphaZero näher erklärt.

AlphaZero basiert auf dem Algorithmus AlphaZero Go, welcher der erste ist, der Go auf einem „superhuman level“[Sil+17b] spielen kann. Einige Verbesserungen, die speziell für Go waren, wurden dafür weggelassen.[Sil+17a] Beide Algorithmen verwenden ein Deep Neural Network im Zusammenhang mit der Monte Carlo Tree Search (MCTS).

Bei MCTS wird ein Suchbaum inkrementell aufgebaut bis ein Rechenlimit erreicht wird. Dieses Rechenlimit wird meistens durch eine Zeitbegrenzung realisiert. Es wird also für eine bestimmte Zeit ein Suchbaum aufgebaut, dann abgebrochen und der bis dahin beste gefundene Wert ausgegeben. Dieses Verfahren ist sehr effizient für Bäume mit hoher Verzweigung. In diesem Verfahren werden bereits bekannte Knoten durch eine Baumstrategie (Tree Policy) ausgewählt und sobald ein Knoten mit unbekannten Kindesknoten erreicht wird, wird die Standardstrategie (Default Policy) angewendet. Bereits besuchte Knoten besitzen eine Bewertung und die Anzahl, wie oft sie besucht wurden. Eine Tree Policy muss die Knoten so auswählen, dass vielversprechende Knoten besucht werden und auch neue Pfade erkundet werden. Es gilt also eine Balance zu finden zwischen gut bewerteten Knoten und selten besuchten Knoten. Der Erfolgt von MCTS hängt zu großen Teilen von dieser Tree Policy ab. Ein Verfahren, was sich als erfolgreich herausgestellt hat ist der „upper confidence bound for trees (UCT)“ Algorithmus, welcher hier nicht näher erläutert wird. Bei einem Knoten mit unbekannten Kindesknoten wird ein Spiel mithilfe der Default Policy simuliert, bis ein Terminalknoten erreicht wird. Der Terminalknoten kann mit einem Belohnungswert bewertet werden. Dieser Belohnungswert wird zurückpropagiert und die

Bewertungen der vorhergegangen Knoten werden angepasst. Im einfachsten Fall kann die Default Policy eine gleichmäßige Zufallsverteilung sein. Die Züge werden also zufällig ausgewählt. [Bro+12]

Bei AlphaZero wird MCTS mit einem tiefen Neuralen Netz f_θ mit den Paramtern θ verwendet. Dieses Netz bekommt die aktuelle Position s als Eingabe und gibt einen Vektor p und einen Skalar v aus $(p, v) = f_\theta(s)$. Der Vektor p gibt die Wahrscheinlichkeit an, dass ein bestimmter Zug ausgewählt wird. Der Skalar v schätzt die Wahrscheinlichkeit, dass der Spieler am Zug mit der aktuellen Position gewinnen wird. AlphaZero wird durch Reinforcement Learning trainiert indem es wiederholt gegen sich selbst spielt. Für jede Position s wird eine MCTS Suche durchgeführt, welche durch das Neurale Netz f_θ geführt wird. Die Ausgabe dieser Suche sind die Zugwahrscheinlichkeiten π und der Gewinner z. Diese Wahrscheinlichkeiten sind deutlich besser, als die Ausgaben des Neuralen Netzes. Die Parameter des neuralen Netzes werden angepasst, sodass der Unterschied zwischen $(p, v) = f_\theta(s)$ und (π, z) möglichst gering wird. Konkret werden die Parameter θ durch den Gradientenabstieg auf einer typischen Verlustfunktion angepasst. [Sil+17b][Sil+17a]

Der beschriebene Algorithmus wurde jeweils für Schach, Shogi und Go trainiert. In Schach und Shogi spielte AlphaZero nach weniger als einem Tag Training besser als die bis dahin führenden Programme. Im Vergleich zu Alpha-Beta-Pruning Algorithmen, wie zum Beispiel Stockfish durchsucht AlphaZero deutlich weniger Pfade pro Sekunde. Das ganze wird dadurch kompensiert, dass AlphaZero bei der Auswahl der zu untersuchenden Pfade selektiver vorgeht. Es wurde auch gezeigt, dass AlphaZero effizienter skaliert mit mehr Bedenkzeit. Des weiteren werden Fehler in der Schätzung von Positionen durch MCTS geschwächt, während sie bei Alpha-Beta-Pruning Algorithmen bis zur Wurzel wandern. [Sil+17a]

2.5 Schnittstellen

Damit Schachengines in andere Programme eingebunden werden können sind Schnittstellen notwendig. Das zu entwickelnde Programm soll ein Backend für eine Schachlernanwendung im Web bilden. Dafür muss es zum einen eine Schachengine einbinden und ihre Schnittstelle ansteuern und zum anderen Schnittstellen, für eine Weboberfläche zur Verfügung stellen. Schnittstellen zwischen Programmen werden Application Programming Interface (API) genannt. Sie erlauben es unterschiedlichen Programmen miteinander zu kommunizieren und interagieren. Die Programme können dabei mit unterschiedlichen Technologien erstellt word sein und auch auf verschiedenen Rechnern ausgeführt werden, solange sie sich durch eine gemeinsame API verständigen können. In den folgenden Kapiteln wird beschrieben,

wie die Schnittstelle zu einer Schachengine meistens gestaltet wird und wie Schnittstellen zwischen einer Weboberfläche und dem Backend gebildet werden können.

2.5.1 Schachengine Schnittstellen

Heutzutage unterstützen die meisten Schachprogramme das Universal Chess Interface (UCI). Es hat fast vollständig das ältere Protokoll XBoard ersetzt. Der primäre Zweck des Protokolls ist es eine Engine mit einer GUI zu verbinden. Die Aufgabe der GUI ist es dabei nicht nur eine Benutzeroberfläche dazustellen, sondern auch den Zustand zu speichern. Das ermöglicht es, dass die Engine zustandslos ist und reduziert somit ihre Komplexität. Die GUI kann auch Züge von einem Eröffnungsbuch oder einer Endspieltabelle auswählen. Ein Format für die Eröffnungsbücher wird nicht vorgeschrieben. Sie können je nach GUI Programm variieren. Diese Aufgabe muss nicht in der Engine erledigt werden. [Wik24c]

Die Kommunikation findet bei diesem Protokoll über die Standardeingabe und Standardausgabe der Konsole statt. In dem Protokoll sind einige Kommandos definiert, über die eine GUI Daten abfragen und setzen kann und Aktionen starten. Ein typischer Ablauf kann folgendermaßen aussehen:

1. Die GUI schickt zur Initialisierung das Kommando `uci`.
2. Die Engine antwortet mit ihrem Namen, Autor und den Optionen, die eingestellt werden können.
3. Wenn alle Informationen gesendet wurden schickt sie `uciok`
4. Jetzt kann die GUI mit dem Kommando `setoption` Optionen setzen.
5. Mit dem Kommando `isready` kann überprüft werden, ob die Engine bereit ist für den nächsten Schritt.
6. `ucinewgame` startet ein neues Spiel.
7. Die Position wird mit dem Kommando Position gesetzt, z. B. `position startpos moves e2e4 e7e5`. Statt `startpos` kann auch ein FEN-String übergeben werden. Das ist eine String Darstellung der aktuellen Position. Nach `moves` können die gespielten Züge in algebraischer Notation dargestellt werden. Dabei beschreiben die ersten zwei Zeichen die ursprüngliche Position der Figur und letzten zwei den Zielort. e2e4 bewegt also eine Figur von der Position e2 zur Position e4. Da die Position davor die Startposition war, muss es ein Bauer gewesen sein.

8. Mit dem Kommando `go` startet die Engine die Berechnung. Diesem Kommando können auch Begrenzungen mitgegeben werden, wie zum Beispiel die maximale Tiefe oder maximale Anzahl an Knoten, die untersucht werden sollen.
9. Während der Berechnung gibt die Engine kontinuierlich Zwischeninformationen über die aktuelle Tiefe untersuchten Züge und weitere Parameter.
10. Die GUI kann die Berechnung jederzeit beenden mit dem Kommando `stop`.
11. Wenn die Berechnung beendet ist oder durch `stop` abgebrochen wurde, sendet die Engine den besten herausgefundenen Zug mit dem Kommando `bestmove`.

[Ste25]

2.5.2 Web-Schnittstellen

Die Benutzerschnittstelle für dieses Projekt soll als Weboberfläche realisiert werden. Dafür wird es in zwei Bestandteile aufgeteilt, das Backend und das Frontend. Damit diese Bestandteile miteinander kommunizieren können, ist es notwendig eine API zur Kommunikation zu entwickeln.

Für diesen Anwendungsfall werden häufig REpresentational State Transfer (REST)ful APIs erstellt. REST ist eine Architektur, die HTTP verwendet, um zwei Systeme miteinader kommunizieren zu lassen. Damit eine API als RESTful bezeichnet werden kann, müssen einige Prinzipien oder auch Einschränkungen eingehalten werden.

REST-Prinzipien

REST-APIs sind Client-Server-Architekturen. Dahinter steckt das Prinzip der Separierung von Anliegen (separation of concerns). Das hat den Vorteil, dass der Client und Server gegenseitig nichts über ihre Technologien und Implementierungen wissen müssen und separat entwickelt werden können. Auch bei diesem Projekt werden Client und Server separat entwickelt.

Ein Kernaspekt von REST-APIs ist, dass sie zustandslos sind. Der Server muss sich den Zustand des Clients nicht merken. Stattdessen liegt es in der Verantwortung des Clients, bei jeder Anfrage den aktuellen Kontext mitzusenden. Das ist auch der Fall bei mehrteiligen Anfragen. Die Zuverlässigkeit wird dadurch verbessert, da es einfacher ist, Fehlerzustände zu beheben. Die Skalierbarkeit ist auch verbessert, da die Aufgaben des Servers vereinfacht sind und Load Balancer einfacher zu implementieren sind.

Ein weiteres Rest-Prinzip ist das Caching. Daten einer Antwort werden implizit oder explizit als chachebar oder nicht cachebar markiert. Ein Client kann dann cachebare Antworten wiederverwenden für spätere äquivalente Anfragen. Das ermöglicht schnellere Antwortzeiten und kürzere Latenzen. Es kann allerdings die Zuverlässigkeit beeinträchtigen.

REST-APIs sind in hierarchischen Schichten organisiert. Eine einzelne Schichten sind voneinander gekapselt. Das reduziert die Komplexität des Systems, da verschiedene Bestandteile separat betrachtet werden können. Die Schichten können auch auf verschiedene Prozesse oder Systeme verteilt sein. An den Schichtgrenzen können gemeinsame Caches genutzt werden und es können Sicherheitsregeln zwischen den Schichten definiert werden. Das kann die Sicherheit des Systems erhöhen. Auf diese Weise Rate Limiting und Load Balancing implementiert werden. Ein Nachteil ist, dass die Schichtenarchitektur Overhead hinzufügt und die Latenz erhöhen kann. Dass kann allerdings durch die Verwendung von Chaches ausgeglichen werden.

Eine REST-API hat eine einheitliche Schnittstelle. Das hilft dabei die Kommunikation zwischen den Komponenten zu definieren. Es entkoppelt die Kommunikation von der Architektur. Jede Ressource hat eine eindeutige URI über die man auf sie zugreifen kann. Das Frontend muss nicht wissen, wie die Daten abgespeichert sind. Sie werden mit einem portablen Format, wie XML oder JSON verschickt.

Eine optionale Anforderung an REST-APIs ist code-on-demand. Das beschreibt die Möglichkeit den Client durch das Herunterladen von Applets oder Skripten zu erweitern. [FT00][De17]

URI Design

In REST-APIs werden Ressourcen über Unique Resource Identifier (URI)s identifiziert. Sie bestehen aus dem Protokoll, der Webadresse und darauffolgend einem Pfad, z. B. `https://api.example.com/path`. Für REST-APIs gibt es einige Regeln und Best-Practices, wie der Pfad von URIs gestaltet wird.

Die erste Regel ist, dass der Schrägstrich dazu genutzt werden soll, eine hierarchische Struktur darzustellen. Zum Beispiel könnten Eröffnungen folgendermaßen angordnet sein: `/openings/queens-gambit`. Das Queen's Gambit ist eine Eröffnung, deshalb ist sie unter der Ressource `openings` platziert.

In dem vorherigen Beispiel sind auch weitere Regeln zu sehen. Um die Lesbarkeit zu erhöhen, werden Bindestriche verwendet. Unterstriche werden vermieden, da sie in Links oft schlecht oder gar nicht sichtbar sind. Außerdem sollen Pfade ausschließlich Kleinbuchstaben enthalten, um Verwirrung zu vermeiden.

In REST-APIs gibt es verschiedene vier verschiedene Arten von Daten, die ihre eigenen Namenskonventionen besitzen.

Dokumente (documents) sind einzelne Ressourcen, wie zum Beispiel eine einzelne Objektinstanz oder ein Datenbankeintrag. Diese werden mit einem Substantiv oder einer Nominalphrase benannt. Auch das ist in dem Queen's Gambit Beispiel zu sehen. `queens-gambit` ist hier das Substantiv.

Sammlungen (collections) sind Verzeichnisse von Ressourcen, die durch den Server verwaltet werden. Clients können neue Elemente für diese Verzeichnisse vorschlagen, der Server entscheidet, ob das Element hinzugefügt wird. Eine Sammlung wird mit einem Substantiv im Plural oder einer Nominalphrase benannt. Ein Beispiel ist der Pfad `/openings`, welcher eine Sammlung von Eröffnungen enthält.

Ein Speicher (store) ist ein Speicherort, der vom Client verwaltet wird. Ein Anwendungsfall kann ein Client sein, der die Dokumentenressource `queens-gambit` in seinen Speicher `favorites` speichert. `PUT /users/1234/favorites/queens-gambit`

Controller stellen ausführbare Funktionen mit Parametern und Rückgabewerten dar. Darunter fallen Aktionen, die sich nicht eindeutig den klassischen CRUD-Operationen – also Erstellen (Create), Empfangen (Read), Aktualisieren (Update) oder Löschen (Delete) – zuordnen lassen. Cotroller sind meistens das letzte Element in der Hierarchie und besitzen keine Kindelemente. Benannt werden sie mit einem Verb oder einer Verbalphrase. Ein Beispiel wäre eine Controllerressource, die es einem Client erlaubt eine Meldung erneut zu senden: `/alerts/2345/resend`.

URIs sollten keine CRUD-Funktionen in ihren Namen verwenden. Stattdessen sollten HTTP Methoden zeigen, welche CRUD-Operation ausgeführt wird. Die CRUD-Funktionen einer Userressource können folgendermaßen aussehen:

- Create user → `POST oder PUT /users`
- Receive user → `GET /users`
- Update user → `PATCH /users/2345`
- Delete user → `DELETE /users/3423`

In URIs kann es Pfadsegmente geben die statisch oder dynamisch sind. Statische Pfadsegmente sind unveränderliche Namen, die durch einen Designer im Voraus festgelegt werden. Dynamische Pfadsegmente sind veränderlich und erfüllen meist den Zweck eines Identifikators, der aus einer Sammlung oder einem Speicher ein Element identifiziert. Im folgenden Beispiel ist das Pfadsegment `openings` statisch und das Segment `{id}` ist dynamisch und wird vor der Ausführung durch eine konkrete Eröffnungs-ID ersetzt: `/openings/{id}`

Nach dem Pfad können noch weitere Query-Parameter definiert werden. Diese sind dazu da um Sammlungen oder Speicher zu filtern. Folgende Query könnte genutzt werden um alle Eröffnungen zu bekommen, die zur Familie Queen's Gambit gehören: `GET /openings?family=queens-gambit`.

[Mas12]

Authentifizierung

Authentifizierung ist der Prozess, bei dem die Identität eines Nutzers verifiziert wird. Dieser Prozess ist in vielen APIs wichtig und muss sicher implementiert werden. Eine unsichere Authentifizierung ist eine große Sicherheitslücke, die es Dritten ermöglicht Zugriff zu Ressourcen zu erlangen, die ihnen nicht zustehen. Es gibt viele Gründe, warum Authentifizierung benötigt wird. Einer davon ist, um nutzerbezogene Daten zu sammeln. Auch in der Schachanwendung soll das geschehen, um eine personalisierte Erfahrung zu ermöglichen. Einen Nutzer könnte zum Beispiel interessieren, welche Schacheröffnungen er bereits oft geübt hat. Außerdem ist es sinnvoll rechenintensive Operationen nur authentifizierten Nutzern zur Verfügung zu stellen. Das reduziert die Wahrscheinlichkeit einer DoS Attacke zu erliegen, bei der durch viele unauthentifizierte Anfragen der Server überlastet wird. Das Verwenden einer Schachengine ist zum Beispiel sehr rechenintensiv und sollte nur authentifizierten Nutzern erlaubt sein.

Authentifizierung lässt sich in drei Kategorien einteilen:

- Wissensbasierte Authentifizierung erfolgt durch etwas, dass man weiß, wie zum Beispiel ein Passwort.
- Besitzbasierte Authentifizierung erfolgt durch etwas, dass man besitzt, wie zum Beispiel einen Schlüssel.
- Biometrische Authentifizierung erfolgt durch etwas, dass man ist, wie zum Beispiel wie der Fingerabdruck aussieht.

Auf Webseiten findet die erste Authentifizierung meistens wissensbasiert mithilfe eines Passworts statt. Bei der Erstellung des Accounts muss ein Nutzer ein Passwort angeben, das er bei erneutem Anmelden wieder angeben muss. Um die Sicherheit dieses Verfahrens zu gewährleisten, darf die Übertragung des Passworts ausschließlich über eine verschlüsselte Verbindung (z. B. HTTPS mit TLS) erfolgen. Außerdem dürfen Passwörter nicht in Klartext gespeichert werden, sondern müssen durch sichere Hashfunktionen umgewandelt werden. Diese Funktionen produzieren bei gleicher Eingabe die gleiche Ausgabe, es kann allerdings nicht durch die Ausgabe auf das Passwort zurückgeschlossen werden. Noch

sicherer werden diese Funktionen, wenn vor der Anwendung ein Salt hinzugefügt wird, also eine zufällige Kombination von Zeichen.

Passwörter sollten nicht bei jeder Anfrage geschickt werden, da jede Anfrage, in der ein Passwort geschickt wird, auch ein gewisses Risiko birgt. Das Überprüfen von Passwörtern mit einer Hashfunktion ist auch aufwendig und würde den Webserver verlangsamen. Deshalb existiert die tokenbasierte Authentifizierung. Bei ihr wird am Anfang ein Login mit Passwort durchgeführt und der Client bekommt daraufhin ein Token. Dieses Token ist ein zufällig generierter String, den nur der Server und der Client kennt. Der Client kann sich damit also wissensbasiert authentifizieren.

Für die genaue Verwendung von Tokens gibt es mehrere Möglichkeiten. Eine Möglichkeit ist das Token auf dem Server in einer Datenbank abzulegen, sodass dieser einen Client wiedererkennen kann. Um ein Token zu deaktivieren kann man es wieder aus der Datenbank entfernen. Auf dem Client kann man das Token in einem Cookie speichern und ihn auffordern das Token bei zukünftigen Anfragen zu verwenden. Die Technologie ist in den aktuellen Browsern gut unterstützt und daher auch leicht zu implementieren. Nachteilhaft bei diesem Vorgehen ist, dass der Server in diesem Fall Zustände speichern muss. Das wiederspricht dem Prinzip von REST-APIs, die per Definition zustandslos sein sollen. Die Überpüfung der Token mit einer Datenbank ist skaliert bei sehr vielen Nutzern auch nicht so gut und ist anfällig für Timing Attacken. Das bedeutet, dass ein Angreifer Token herausfinden kann anhand dessen, wie lange eine Anfrage benötigt.

Das Problem der Timing Attacken kann gelöst werden, indem die Nachrichten durch einen Message Authentication Code (MAC) ergänzt werden. Das Token erhält also zusätzlich einen Code, der bestätigt, dass er von dem Server erstellt wurde. Dafür kann die Hash-based MAC (HMAC)-Funktion genutzt werden. Diese Funktion erstellt unter Verwendung der Daten und eines geheimen Schlüssels den MAC. Jede Veränderung der Daten führt zur Ungültigkeit des MAC. Die Funktion kann nicht nur auf Tokens angewendet werden, sondern auf beliebige Daten. Im Endeffekt benötigt man auch keine Tokens mehr und die Informationen über die Berechtigungen können direkt an den Client übergeben werden, da sie vor Bearbeitung geschützt sind. Auf diese Weise funktionieren auch JSON Web Tokens (JWT). Ein JSON-Objekt wird in Base64 umgewandelt und mithilfe einer HMAC-Funktion signiert. Mit der selben Funktion kann bei späteren Anfragen getestet werden, ob die Daten unverändert sind. Wenn das der Fall ist muss der neu generierte MAC mit dem vorhandenen übereinstimmen. [Mad20][SK23]

3 Marktanalyse

Wie bereits erwähnt gibt es viele Programme, die einen Schachspieler beim Erlernen von Schacheröffnungen unterstützen kann. In diesem Kapitel wird zuerst betrachtet, welche Lernplattformen es gibt und ob es in dem aktuellen Angebot Defizite gibt, die durch ein neues Programm ausgeglichen werden können. Anschließend findet ein Vergleich von Schachengines und von Datenbanken statt unter der Betrachtung, welche am besten in dieses Projekt eingebunden werden können.

3.1 Lernplattformen

Wenn man bereit ist Geld auszugeben, kann man sich zwischen einigen unterschiedlichen online Lernplattformen entscheiden. Bei kostenlosen Plattformen ist die Auswahl kleiner. Die Plattformen haben einige Gemeinsamkeiten, aber auch zum Teil größere Unterschiede. In diesem Abschnitt werden die kostenpflichtigen Plattformen Chess.com und Chessable und die kostenlose Plattform Lichess verglichen.

3.1.1 Chess.com

Chess.com ist mit seinen über 100 Millionen Mitgliedern eine der größten Schachplattformen im Internet. [Che22] Neben der Möglichkeit gegen andere Mitspieler oder Computergegner zu spielen gibt es auch interaktive Lektionen, um Schach zu erlernen. Einige darunter sind speziell, um Eröffnungen zu lernen. Um Zugriff auf die Lektionen zu bekommen, muss man allerdings ein kostenpflichtiges Abonnement abschließen für aktuell 4,17€ pro Monat. Die Lektionen sind immer ähnlich aufgebaut. Am Anfang kann man ein erklärendes Video anschauen. Anschließend werden bestimmte Teile der Eröffnung abgefragt. Dadurch, dass in einem Video mehrere Varianten einer Eröffnung erklärt werden, können diese relativ lang werden. Teilweise sind sie über 30 Minuten und das kann es erschweren alle Informationen auf einmal aufzunehmen. Außerdem kann es auch nachteilhaft sein, dass nicht die gesamte Eröffnung abgefragt wird, sondern nur bestimmte Ausschnitte. Auf der Seite gibt es auch die Möglichkeit seine vergangenen Spiele zu analysieren. Ein Teil der Analyse ist auch kostenfrei zur Verfügung. Insgesamt ist die Interaktivität der Kurse etwas begrenzt.

3.1.2 Lichess

Lichess ist eine vollständig kostenlose Plattform. Auf der Seite gibt es offizielle interaktive Kurse um Grundlagen oder fortgeschrittenere Themen zu lernen. Eröffnungen kann man über die sogenannten Studien lernen. Das sind Kurse, die von anderen Benutzern erstellt wurden. Dadurch variieren sie auch in der Qualität und es ist nicht für jede Eröffnung ein guter Kurs vorhanden. Auch bei Lichess hat man die Möglichkeit seine vergangenen Partien zu analysieren.

3.1.3 Chessable

Chessable ist eine Seite, die betont, dass sie durch wissenschaftliche Erkenntnisse das Lernen so einfach wie möglich gestalten wollen. [Pro] Im Zentrum steht dabei Spaced Repetition. Die Seite besitzt auch spielerische Elemente, wie zum Beispiel Belohnungen in Form von Punkten und Abzeichen. Dadurch soll Dopamin ausgeschüttet werden, was den Lerneffekt erhöht. Die Kosten von Chessable sind allerdings weniger Grund zur Dopaminausschüttung. Ohne Geld auszugeben, hat man nur eine kleine Auswahl an Kursen und Funktionen. Für 11,99$ kann man ein monatliches Abonnement abschließen, das einem Zugriff auf viele Funktionen und Kurse gibt. Des weiteren gibt es die Möglichkeit einzelne Kurse zu kaufen, darunter einige mit Kosten von über 100$. Sie sind meistens so gestaltet, dass ein Zug vorgeführt und erklärt wird. Anschließend spielt man selbst den Zug nach. Am Ende einer Variante wird die gesamte Zugabfolge abgefragt. Es wird also gelernt durch die Imitation des Gesehenen und durch Wiederholung. Bei Chessable ist positiv hervorzuheben, dass sie einen Fokus darauf gesetzt haben wissenschaftliche Erkenntnisse mit einzubeziehen, um das Lernen so einfach wie möglich zu gestalten. Die Plattform ist für Einsteiger allerdings verhältnismäßig teuer, vor allem wenn man neben dem Abonnement zusätzliche Kurse kaufen will.

3.1.4 Ergebnis

Die betrachteten Plattformen bieten verschiedene Vor- und Nachteile. Lichess ist weniger personalisiert, hat ein begrenztes Angebot an Kursen, ist aber im Gegensatz zu Chess.com und Chessable kostenfrei. Chess.com ist eine etablierte Marke und bietet einige interaktive Kurse, die das Lernen erleichtern. Personalisierung ist allerdings begrenzt. Chessable hat am meisten Personalisierung und auch am meisten Anreize das Lernen attraktiv zu machen durch Punkte, Abzeichen und Spaced-Repetition. Es ist allerdings auch die Seite mit den höchsten Preisen. Unser Programm kann hierbei eine kostenlose Alternative bieten, die auch personalisiert den Lernfortschritt in unterschiedlichen Eröffnungen zeigt.

3.2 Schachengines

Mittlerweile existieren sehr viele Schachengines, die leicht unterschiedliche Algorithmen implementieren. Einen Startpunkt für den Vergleich der Engines bietet der Top Chess Engine Championship (TCEC). In diesem Turnier treten unterschiedliche Schachengines gegeneinander an in mehreren Spielen. Der Sieger einer Saison wird TCEC Champion. In der Vergangenheit waren die Engines Stockfish, LCZero, Komodo und Houdini Champions. [TCE25]

Ein weiterer Anhaltspunkt sind die Computer Chess Rating Lists (CCRL), eine Sammlung an Statistiken, welche die meisten Schachengines abdeckt. [CCR25] Tabelle 3.1 zeigt vier Engines mit ihrer Anzahl an TCEC-Siegen und ihrer Elo-Bewertung laut CCRL.

Engine	Algorithmus	Elo	Siege	kostenfrei
Stockfish	Alpha-Beta + NNUE	3642	13	ja
Komodo	MCTS + NNUE	3626	4	nein
LCZero	MCTS	3443	2	ja
AlphaZero	MCTS	-	-	nein

Tabelle 3.1: Schachengines

3.2.1 AlphaZero

AlphaZero war ein Forschungsprojekt, dass in Unterabschnitt 2.4.2 näher beschrieben wurde. Die Engine wurde hauptsächlich für Forschungszwecke entwickelt und ist nicht frei zur Verwendung verfügbar. Es haben sich allerdings einige andere Entwickler an den Forschungsergebnissen der Engine orientiert und ähnliche Engines entwickelt.

3.2.2 LCZero

LCZero ist ausgeschrieben Leela Chess Zero und wird auch mit dem Namen lc0 bezeichnet. Diese Engine verwendet den gleichen Ansatz, wie AlphaZero mit ein paar Anpassungen. Sie arbeitet also mit einem Neuralen Netzwerk um die besten Züge herauszufinden. Im Gegensatz zu AlphaZero kann jeder LCZero mit einem vortrainierten Neuralen Netz herunterladen und über UCI verwenden. [20]

3.2.3 Komodo

Komodo gewann vor ein paar Jahren einige TCEC. Die Engine wird von Chess.com verwendet für Computergegner mit unterschiedlicher Stärke. Seit Version 12 ist in Komodo eine MCTS-Engine integriert. Die neuesten Versionen heißen Komodo Dragon und verwenden zusätzlich ein Efficiently Updatable Neural Network (NNUE). Die Engine wird über UCI angesteuert. Die neuesten Versionen sind allerdings kostenpflichtig. [Wik24b]

3.2.4 Stockfish

Stockfish verwendet Alpha-Beta-Pruning um die besten Züge herauszufinden. Früher wurden die Schachpositionen durch eine handgeschriebene Evaluationsfunktion bewertet. Mittlerweile wurde diese Funktion durch ein NNUE ersetzt, was Stockfish deutlich verbesserte. Die Engine wird oft als die stärkste Schachengine angesehen. Sie hat auch die meisten TCEC gewonnen. Die letzten sechs Saisons blieb sie ungeschlagen. Stockfish ist kostenlos verfügbar und kann über UCI verwendet werden. [Wik25b]

3.2.5 Auswahl

Für dieses eignet sich Stockfish am besten. Sie ist aktuell die stärkste Schachengine und kostenlos verwendbar. Sie kann von der Stärke auch angepasst werden und über UCI angesteuert werden. Das macht sie geeignet um sie in ein anderes Programm einzubinden.

3.3 Eröffnungssammlungen

Damit die Eröffnungen in unserem Programm geübt werden können, müssen sie auch dort hinterlegt sein. Welche Eröffnungssammlungen existieren und wie gut sie für eine Lernanwendung verwendet werden können, wird in diesem Kapitel beschrieben.

3.3.1 Encyclopedia of Chess Openings

Die Encyclopedia of Chess Openings ist eine der bekanntesten Sammlungen von Schacheröffnungen. Sie wurde erstellt, um den aktuellen Stand der Eröffnungstheorie festzuhalten. Die Editoren, von denen die meisten Großmeister sind, wählten aus den Partien von Schachmeistern die Züge aus, die sie für am bedeutendsten hielten. Die Eröffnungen wurden in die fünf Kategorien A bis E aufgeteilt, für die jeweils ein Buch veröffentlicht

wurde. Die Kategorien sind wiederum unterteilt in hundert nummerierte Unterkategorien. Die Kategorien und ihre Nummern werden auch ECO Codes genannt. [Wik24a]

Mit dieser Sammlung wären die wichtigsten Eröffnungen abgedeckt. Allerdings ist die Encyclopedia of Chess Openings in erster Linie eine Buchreihe, in der die Eröffnungen in Tabellenform dargestellt wird mit textuellen Beschreibungen in natürlicher Sprache. Um die Eröffnungen in einem Programm zu verwenden, müssen sie in maschinenlesbarer Form vorliegen.

3.3.2 Eröffnungsdatenbanken

Eröffnungsdatenbanken werden auch Opening Explorer genannt. Sie besitzen Datensätze von vielen vergangenen Schachspartien. In der Benutzeroberfläche kann man betrachten, welche Züge wie häufig in diesem Datensatz gespielt wurden und wie oft schwarz oder weiß nach dieser Kombination gewonnen hat.

Eröffnungsdatenbanken sind ein nützliches Werkzeug, um sein Wissen über bekannte Eröffnungen zu vertiefen und zu erfahren, welche Varianten häufig gespielt werden. Sie sind allerdings nicht dafür gedacht, neue Schacheröffnungen zu erlernen. Sie können auch nicht in ein anderes Programm eingebunden werden, da sie meist nur eine grafische Schnittstelle besitzen und keine API.

3.3.3 Eröffnungsbücher für Computer

Die Datensätze hinter den Eröffnungsdatenbanken werden als Eröffnungsbücher bezeichnet. Sie werden oft von Schachengines genutzt, um in den Anfangszügen Rechenleistung und Zeit zu sparen. Es gibt sie in Binär- und Textformaten. Die Binärformate sind häufig nicht vollständig dokumentiert, mit der Ausnahme von Polyglott. Die Textformate sind meist in Portable Game Notation (PGN). Bei dieser Notation werden die Züge in der verkürzten algebraischen Notation festgehalten. Die algebraische Notation ist die Notation, die auch in Eröffnungsbüchern für Menschen genutzt werden. [Wik25a]

Im Internet kann man einige kostenlose Polyglott- oder PGN-Eröffnungsbücher finden. Diese haben nicht unbedingt den selben Umfang, wie in kostenpflichtigen Datenbanken, können aber dennoch verwendet werden um selber eine Eröffnungsdatenbank zu implementieren. Für ein Programm zum Erlernen von Schacheröffnungen ist es allerdings nicht notwendig mehrere Millionen von vergangenen Partien zu speichern und zu analysieren. Eine Liste mit den häufigsten Eröffnungen und ihren Varianten wäre bereits ausreichen.

3.3.4 Eröffnungslisten

Es existieren auch einige Eröffnungslisten, die keine vollständigen Partien enthalten sondern nur die Eröffnungen. [ÖJ25] ist zum Beispiel eine Sammlung an Eröffnungen im JSON-Format. Sie enthält Eröffnungen aus der Encyclopedia of Chess Openings, der SCID Sammlung und von Wikipedia. Als Basis dient wiederum die Sammlung [lic25] von Lichess. Diese enthält die Eröffnungen der Encyclopedia of Chess Openings in TSV-Dateien, die den ECO-Code, den englischen Namen, die Zugfolge in PGN und UCI-Notation und die Endposition enthält.

Die Sammlung von Lichess ist in einem Format, das relativ gut für eine Lernanwendung geeignet ist. Sie ist mit 3514 Eröffnungen allerdings sehr umfangreich und sollte noch gefiltert werden. Es gibt in der Datenstruktur auch keine direkte Gruppierung für die Eröffnungsfamilien. Die Familien können aber durch den englischen Namen erkannt werden. Er besitzt die Struktur `Familie: Variation, Untervariation, ...`. Ein weiteres Problem ist, dass es keine eindeutigen IDs gibt. Es wird zwar zu jeder Eröffnung der ECO-Code angegeben, dieser ist aber nicht eindeutig. Auch der englische Name erlaubt keine eindeutige Identifikation.

3.3.5 Angepasste Eröffnungsliste

Wie im vorigen Kapitel gesehen wurde, können die gefundenen Eröffnungslisten nicht sofort verwendet werden. Sie müssen erst gefiltert werden, damit die Anwendung nicht unübersichtlich wird und es müssen Daten ergänzt werden, wie die Eröffnungsfamilie und die ID.

Die Zuordnung der Eröffnungsfamilien kann über den englischen Namen der Eröffnung gelöst werden. Alle, die zur selben Familie gehören, beginnen mit dem selben Namen. Die Eröffnungen können also danach gruppiert werden.

Für die anderen beiden Probleme gibt es mehrere Lösungsmöglichkeiten. Um beide Probleme gleichzeitig zu lösen, könnte man pro ECO-Code eine Eröffnung auswählen. Danach hat man noch 500 Eröffnungen, was immer noch eine große Auswahl ist. Um herauszufinden welche Eröffnung am relevantesten ist könnte man entweder vergleichen welche Eröffnungen in Büchern typischerweise beschrieben werden oder in einer Eröffnungsdatenbank nachschauen, welche Eröffnungen am häufigsten gespielt werden. Diese Optionen sind allerdings beide sehr zeitaufwendig und bei der Anzahl an Eröffnungen würde es mehrere Monate dauern alle Eröffnungen zu untersuchen. Für den ersten Prototyp könnte man stattdessen als Auswahlkriterium bei gleichem ECO-Code immer die erste Variation wählen, eine

zufällige oder die Variation mit den meisten Zügen. Falls nicht so viele Eröffnungsvariationen herausfiltern werden sollen kann man auch den Hash des englischen Namens als ID verwenden und pro Name eine Eröffnung auswählen. Hier hat man die gleichen Auswahlkriterien zur Auswahl. Man kann die große Anzahl an Eröffnungen auch als Vorteil sehen und die Liste nicht verkleinern. In diesem Fall könnte man die Eröffnungen der Reihe nach durchnummerieren um eine eindeutige IDs zu bekommen.

4 Entwurf

In diesem Kapitel wird der Entwurf und die Architektur des Programms beschrieben. Zuerst wird beschrieben, welche Anforderungen es an das Programm gibt. Danach werden die verwendeten Architekturpatterns erläutert. Darauf folgt ein Kapitel zur Architektur des Backends in dem beschrieben wird, wie die Endpunkte der API gestaltet sind, welche Technologien verwendet werden und wie der innere Aufbau der API aussieht. Anschließend werden die wichtigsten Prozesseabläufe dargestellt und schließlich, wie das Projekt auf einem Server aufgesetzt wird.

4.1 Anforderungen

Das Programm soll eine Webanwendung sein auf der Nutzer mit einer benutzerfreundlichen Oberfläche verschiedene Eröffnungen betrachten und trainieren können. Es soll dabei als zentraler Ort dienen um Eröffnungen nachzuschauen und sein Gedächtnis aufzufrischen. Die Anwendung unterstützt Lernende, indem sie personalisiert Vorschläge gibt, welche Eröffnungen erneut geübt werden sollen. Folgende Features sollen implementiert werden:

- Login: Ein Nutzer kann sich bei der Anwendung anmelden.
- Registrieren: Ein Nutzer kann einen Account anlegen.
- Spielumfang: Die Anwendung bietet zu Beginn jeweils mindestens eine Eröffnung für die weißen und für die schwarzen Figuren an.
- Lernmodus: Ein Nutzer kann eine Eröffnung auswählen und anschließend verschiedene Zugfolgen und Varianten der Eröffnung betrachten.
- Übungsmodus: Ein Nutzer kann eine Eröffnung auswählen und muss dann die Figuren einer Farbe nach der Zugfolge der Eröffnung bewegen. Der Computer spielt dabei die Züge des Gegners.
- Übungsmodus Hinweise: Während dem Üben hat ein Nutzer die Möglichkeit sich Hinweise geben zu lassen.
- Übungsmodus Erweiterung: Ein Nutzer hat die Möglichkeit nach einer Eröffnung mit oder ohne Dekomposition das Spiel zu Ende zu spielen gegen einen Computergegner.

- Vorschläge: Ein Nutzer bekommt Vorschläge welche Eröffnung trainiert werden soll.
- Statistik: Ein Nutzer kann sich Statistiken zu den Eröffnungen anschauen. Dazu zählt die Anzahl an richtigen und falschen Übungsdurchläufen. Diese Statistik wird auch zur Erstellung der Vorschläge verwendet.

4.2 Patterns

In diesem Kapitel werden architekturübergreifende Patterns vorgestellt, die unabhängig vom gewählten Architekturstil eingesetzt werden können. Sie tragen dazu bei, den Code besser strukturiert, wartbar und erweiterbar zu gestalten. Dazu gehören das Repository Pattern und Dependency Injection. Beide Patterns werden im Folgenden erläutert.

4.2.1 Dependency Injection

Das Ziel von Dependency Injection ist es, Softwarekomponenten lose zu verbinden. Klassen sollen, anstatt selbst Objekte zu erstellen, beschreiben, welche Funktionalität sie benötigen. Diese Beschreibung findet über Interfaces (Schnittstellen) statt. Diese definieren, welche Funktionen benötigt werden. In der einfachsten Form, der puren Dependency Injection, werden die Objekte über den Konstruktor übergeben. Häufig wird allerdings ein Dependency Injection Container verwendet um die benötigten Objekte in eine Klasse zu „injizieren“.

Das Verwenden von Dependency Injection bietet einige Vorteile. Zum einen ermöglicht es Late Binding, also das späte definieren von Verbindungen. So kann zum Beispiel über eine Konfigurationsdatei im Nachhinein geändert werden, welche Klasse für die Instanzierung einer bestimmten Abhängigkeit verwendet wird. Zum anderen erleichtert es auch die Erweiterung eines Programms. Zusätzliche Features können implementiert werden, indem einzelne Abhängigkeiten angepasst oder ersetzt werden, ohne dass sich das Interface verändert. Auch für Tests ist es dadurch einfach, die tatsächlichen Abhängigkeiten durch Testdoubles zu ersetzen und Klassen isoliert zu testen. Das ist vor allem bei Unittests nützlich. Durch die Definierung von klaren Schnittstellen, ist auch die parallele Entwicklung an unterschiedlichen Modulen möglich. Wenn man sich zusätzlich an die Regel hält, dass jede Klasse nur einen Zweck hat, verbessert Dependency Injection auch die Wartbarkeit. Es ist dann oft möglich neue Features umzusetzen, indem vorhandene Klassen durch Einbindung neuer Klassen erweitert werden. [See19]

4.2.2 Repository Pattern

Um die Datenspeicherung von dem Rest des Backends zu trennen wird das Repository Pattern angewendet. Bei diesem Pattern wird eine Repository-Klasse erstellt, welche die Abfragelogik enthält. Nach außen werden nur abstrakte domänenspezifische Funktionen exponiert, wie zum Beispiel `GetUser(id: int): User` oder `CreateUser(name: string, password : string)`. Die Implementierung dieser Klasse sorgt dafür, dass die entsprechenden Daten gefunden, aktualisiert oder hinzugefügt werden. Diese Abfragen können zum Beispiel komplexe SQL-Abfragen sein oder auch das Lesen einer Datei. In diesem Projekt wird jede Repository durch ein Interface definiert. Das ermöglicht es die jeweiligen Implementierungen mittels Dependency Injection zu ersetzen. [Eva04]

4.3 Architektur

Die Architektur ist in mehrere verschiedene Bestandteile aufgeteilt. Die zentrale Aufteilung besteht zwischen Backend und Frontend. In dieser Arbeit wird nur das Backend betrachtet. In Abbildung 4.1 ist ein Überblick über die Architektur zu sehen. Das Frontend greift auf eine REST-API zu. Die API ist wiederum aufgeteilt in die vier Bestandteile Eröffnungen, Nutzerdaten, Statistiken und die Schachengine. Die Bestandteile werden in den folgenden Unterkapiteln näher erklärt.

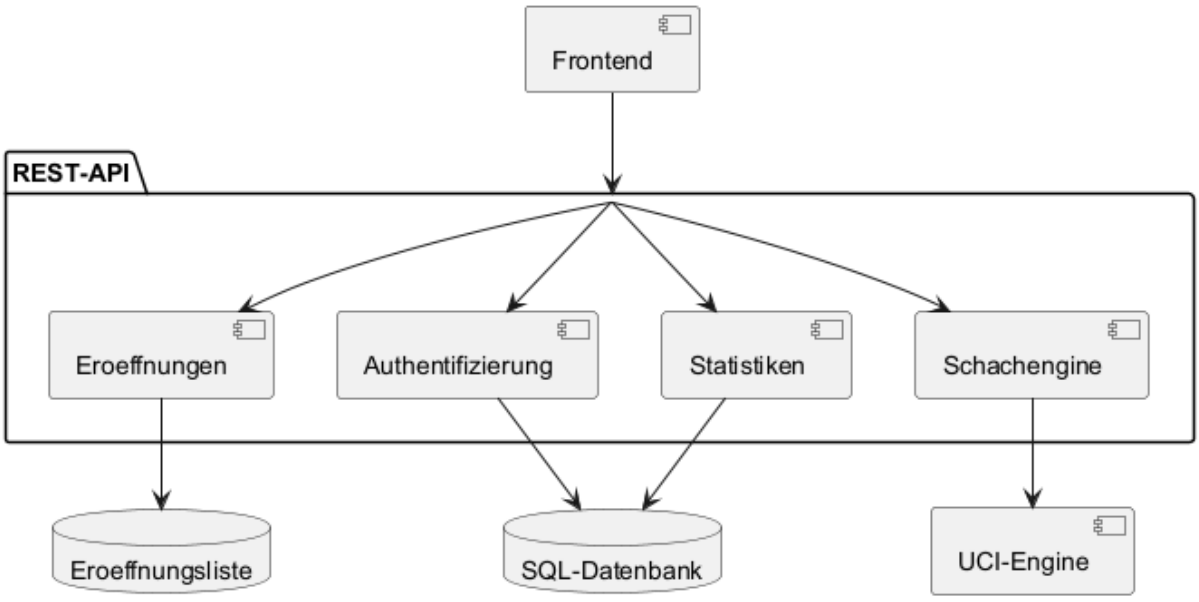

Abbildung 4.1: Komponentendiagramm

4.3.1 REST-API

Die API wird in C# realisiert. Mit dem ASP.NET Core Framework wird eine controllerbasierte Web-API erstellt. Das bedeutet, dass die Controller und die Daten voneinander getrennt werden. Die Architektur ist ähnlich, wie eine Model View Controller Architektur, mit dem Unterschied, dass es keine View gibt. Die View wird separat im Frontend entwickelt. Die Controller und die Models sind im Backend vorhanden. Die Controller offenbaren die API-Endpunkte nach außen und die Models modellieren die Daten. Für die vier unterschiedlichen Bereiche des Backends existiert jeweils ein dazugehöriger Controller. Jeder Controller repräsentiert dabei einen Wurzelendpunkt in der REST-API. Damit gibt es die vier Wurzelendpunkte `/engine`, `/openings`, `/users` und `/stats`. Ein Controller-Endpunkt gibt immer ein `IActionResult` zurück. Wenn in der Implementierung ein anderer Datentyp zurückgegeben wird, kümmert sich das Framework darum, dass er umgewandelt wird. Das resultiert in einem `OkObjektResult` mit dem Statuscode 200 und dem zurückgegebenen Objekt als Wert. Dieses Objekt wird vor der Antwort in JSON umgewandelt. Um Verständlichkeit der nachfolgenden Klassendiagramme zu erhöhen wird nur der Datentyp dieses Objekts dargestellt. In manchen Fällen ist es auch möglich, dass statt des Datentyps eine Antwort mit Fehlercode und anderen Daten übergeben wird. Das wird an den entsprechenden Stellen beschrieben und ist auch in der OpenAPI Dokumentation zu sehen.

4.3.2 Eröffnungen

Der Bestandteil Eröffnungen ist für alles zuständig, wofür die Daten der Eröffnungen benötigt werden. Dazu gehört also das Lernen und Üben von Eröffnungen. Abbildung 4.2 zeigt die dazugehörigen Klassen.

OpeningController

Der OpeningController definiert alle Endpunkte unter der URI `/openings`. Jede public Funktion der Klasse ist in diesem Fall für einen Endpunkt zuständig. Die Zuteilung ist folgendermaßen gestaltet:

- `GET /openings` → GetOpenings
- `GET /openings/{id}/variant` → GetVariants
- `GET /openings/{id}/variant/next-moves` → GetMoveVariants
- `GET /openings/{id}/next-move` → GetNextMove

Abbildung 4.2: Klassendiagramm Eröffnungen

Der Endpunkt `GET /openings` liefert eine Liste mit allen Wurzeleröffnungen in Form von OpeningOverview-Objekten, die jeweils den Namen und die ID einer Eröffnung enthalten. Eine Eröffnung wird in dem verwendeten Datensatz dann als Wurzeleröffnung kategorisiert, wenn der Name keinen Doppelpunkt enthält.

Der Endpunkt `GET /openings/{id}/variant` ist dazu da alle Eröffnungen einer Eröffnungsfamilie zu bekommen. Eine Eröffnung gilt als zugehörig zu dieser Familie, wenn sie mit dem selben Namen beginnt, wie die im Pfad angegebene Eröffnung. Zum Beispiel besitzt die Eröffnung mit der ID D20 den Namen „Queen's Gambit Accepted". Das bedeutet alle Eröffnungen, die mit „Queen's Gambit Accepted" beginnen gehören zu dieser Familie.

Der Endpunkt `GET /openings/{id}/next-move` gibt ein OpeningMove-Objekt zurück, dass den Namen der Eröffnung und den nächsten Zug enthält. Bei diesem Endpunkt kann zusätzlich das Query-Parameter played übergeben werden mit den bisher gespielten Zügen im UCI-Format. Wenn alle Züge gespielt wurden, wird ein leerer String als Zug übergeben.

Der Endpunkt `GET /openings/{id}/variant/next-moves` dient dazu alle möglichen nächsten Züge in einer Eröffnungsfamilie herauszufinden. Er gibt alle OpeningMove-Objekte der gegebenen Eröffnungsfamilie zurück. Auch hier kann man das Query-Parameter played mitgeben. Die Liste enthält dann nur noch die Eröffnungen, die auch mit den angegebenen Zügen beginnen. Beispielsweise werden bei der Anfrage `GET /openings/D20/variant?played =d2d4` nur Eröffnungen zurückgegeben, die mit dem Name „Queen's Gambit Accepted" beginnen und mit dem Zug `d2d4` starten.

OpeningRepository

Opening ist die Model Klasse. Sei enthält alle benötigten Daten zu einer Eröffnung. Die API gibt bei einer Anfrage entweder Instanzen der Klassen OpeningOverview oder OpeningMove zurück, die einen Teil der Informationen von Opening besitzen. Dadurch wird Bandbreite gespart und der Client bekommt nur die Daten, die er auch benötigt. Der Datenspeicher ist eine TSV-Datei, die von dem Lichess Datensatz abgeleitet wurde. Die ID wurde erstellt aus dem ECO-Code ergänzt durch einen Hash des englischen Namens. Bei identischer ID wurde immer die erste Variante ausgewählt. Der Datensatz wird über das Repository Pattern abgefragt. IOpeningRepository enthält also alle Funtionen, um die Daten abzufragen und zu strukturieren, wie sie der OpeningController benötigt. Die OpeningRepository verwendet wiederum den IOpeningReader, welcher das Lesen der TSV-Datei abstrahiert. Auf diese Weise kann die TSV-Datei ohne viel Aufwand durch eine CSV- oder JSON-Datei ausgetauscht werden. Dafür muss lediglich eine entsprechende Implementierung für den IOpeningReader erstellt werden.

4.3.3 SQL-Datenbank

Die dynamischen Daten werden in einer relationalen SQL-Datenbank gespeichert. Dazu zählen die Logindaten der Nutzer und die Nutzerstatistiken zu den Eröffnungen. Dafür existieren die zwei Tabellen User und TrainingResult, welche nachfolgend in Relationenschreibweise notiert sind, wobei der Primärschlüssel fett gedruckt ist und der Fremdschlüssel kursiv:

- User(**Id**, Name, PasswordHash)
- TrainingResult(**Id**, *UserId*, OpeningId, Errors, Hints, DateTime)

Die Usertabelle existiert um das Einloggen und Registrieren zu ermöglichen. Mit einem Benutzernamen und einem Passwort kann sich ein Nutzer anmelden. Der Name besitzt aus diesem Grund einen Unique Constriant. Das Passwort wird in gehashter Form gespeichert.

Für die Statistiken werden die Ergebnisse der Trainings gespeichert. Es wird gespeichert, wer das Training ausgeführt hat, für welche Eröffnung das Training ist, wie viele Fehler gemacht wurden, wie viele Hinweise benötigt wurden und zu welchem Zeitpunkt das Training ausgeführt wurde. OpeningId ist in diesem Fall kein Fremdschlüssel, da die Eröffnungen nicht in der relationalen Datenbank abgespeichert sind, sondern in einer separaten TSV-Datei. Mithilfe dieser Daten können durch SQL-Abfragen weitere Statistiken erstellt werden.

Für den Datenbankzugriff wird ein Object Relational Mapping (ORM) verwendet. Microsoft bietet in C# dafür das Entity Framework Core an. Der Vorteil eines ORMs ist, dass es den Zugriff auf die Datenbank abstrahiert. Zwischen verschiedenen SQL-Datenbanken gibt es einige Unterschiede, auf die geachtet werden muss, beim Schreiben von reinem SQL. Ein ORM abstrahiert diese Unterschiede, indem der Zugriff und die Tabellenstruktur in der Sprache des ORM-Frameworks beschrieben wird, in diesem Fall C#. Ein weiterer Vorteil ist, dass ORMs vor einigen Fehlern, wie zum Beispiel Query-Injection schützen. Das Entity Framework Core besitzt zudem ein robustes System für Datenbankmigrationen. So ist es möglich Tabellenstrukturen auch im Nachhinein zu ändern und die Datenbanken mit geringem Risiko zu migrieren. Mit dem Entity Framework Core kann man zwischen vielen SQL-Datenbanken wählen. Für dieses Projekt wird PostgreSQL verwendet, da sie weit verbreitet ist und bekannt für ihre Stabilität.

4.3.4 Authentifizierung

Wie bereits in Unterunterabschnitt 2.5.2 beschrieben, wird auch in diesem Projekt eine Authentifizierung der Nutzer benötigt, um nutzerbezogene Statistiken zu sammeln und eine personalisierte Erfahrung zu bieten, und um den Zugriff auf die Schachengine zu begrenzen. Abbildung 4.3 beschreibt die Klassen, die bei der Authentifizierung verwendet werden. Der `UserController` implementiert die folgende Endpunkte:

- `POST /users` → CreateUser
- `POST /users/login` → Login

Über den ersten Endpunkt geschieht die Registrierung. Das Frontend schickt mit einer POST Anfrage UserParams mit dem Nutzername und Passwort das verwendet werden soll. Durch den Unique Index auf dem Nutzernamen ist es später möglich sich mit Name und Passwort zu authentifizieren. Wenn der angegebene Nutzername bereits existiert, wird eine Fehlermeldung mit dem Code 409 (Conflict) geschickt, da es sich um einen semantischen Fehler handelt und die Anfrage im Konflikt mit dem Zustand des Servers steht. Wenn der Nutzer noch nicht existiert, wird er mithilfe der Klasse UserRepository angelegt.

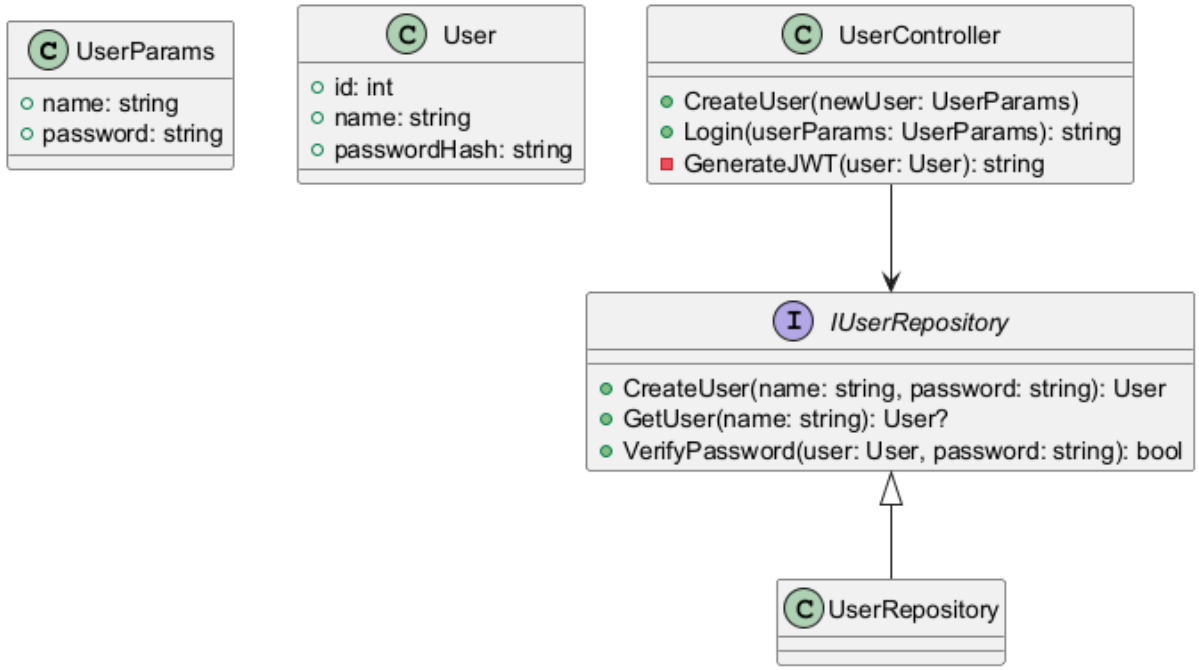

Abbildung 4.3: Klassendiagramm Authentifizierung

Zur sicheren Speicherung der Passwörter wird der Passwort-Hasher von ASP.NET Core verwendet, der eine sichere Hashfunktion mit einem zufällig generierten Salt verwendet.

Die erste Authentifizierung wird über den Controller-Endpunkt `POST /users/login` durchgeführt. Dafür muss der eindeutige Nutzername und das Passwort angegeben werden. Der UserController kann mithilfe des UserRepositorys überprüfen, ob der Nutzer existiert und ob das angegebene Passwort richtig ist. Falls der Nutzer nicht existiert wird der Fehlercode NotFound (404) zurückgegeben und falls das Passwort nicht stimmt wird Unauthorized (401) zurückgegeben. Andernfalls wird über die Funktion GenerateJWT ein JWT erstellt. Dieses Token enthält die User ID und den Ablaufzeitpunkt, welcher initial zwei Stunden in der Zukunft liegt. Der Ablaufzeitpunkt ist wichtig, damit das Token nicht auf unbestimmte Zeit gültig bleibt und somit ein Sicherheitsrisiko birgt. Weitere Informationen und komplexere Nutzerrollen sind in dieser Anwendung nicht notwendig. Mithilfe von diesem Token kann der Nutzer die weiteren Endpunkte verwenden, die auch eine Authentifizierung benötigen. Das ist bei allen der Fall, außer bei den Endpunkten des UserControllers, um die Angriffsfläche zu minimieren.

4.3.5 Statistiken

Um die Benutzererfahrung zu personalisieren werden Statistiken in Form von Trainingsergebnissen gesammelt. Die Trainingsergebnisse können verwendet werden, um verschiedene Kennwerte zu berechnen, zum Beispiel, wie viele Fehler der Nutzer im Durchschnitt bei

einer Eröffnung macht. Die Kennwerte können dem Nutzer angezeigt werden, damit dieser einschätzen kann, wie gut er eine Eröffnung beherrscht. Zusätzlich dazu werden die Kennwerte verwendet, um einen Wert zu berechnen, der angeben soll, wie gut der Spieler eine Eröffnung kann. Dieser Wert wird Expertise genannt und gibt an, mit welcher Wahrscheinlichkeit ein Nutzer die Eröffnung fehlerfrei nachspielt. Um diese Funktionalitäten bereitzustellen, werden die Klassen aus Abbildung 4.4 implementiert.

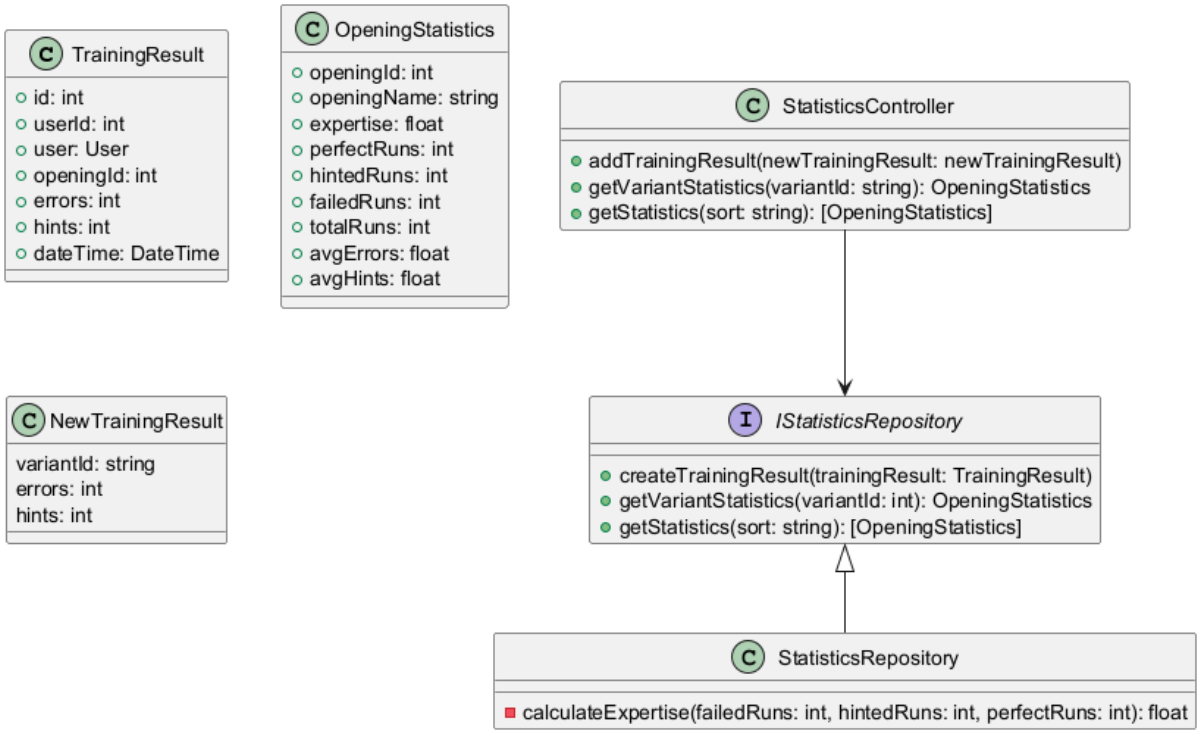

Abbildung 4.4: Klassendiagramm Statistiken

StatisticsController

Der StatisticsController übernimmt die Bereitstellung folgender Endpunkte:

- `POST /stats/variants` → AddTrainingResult
- `GET /stats/variants/{id}` → GetVariantStatistics
- `GET /stats/variants` → GetMoveVariants

Während dem Übungsmodus zählt das Frontend mit, wie viele Fehler der Spieler macht und wie viele Tipps er verwendet. Sobald der Nutzer das Training beendet, werden diese Daten über den Endpunkt `POST /stats/variants` an das Backend geschickt. Das erkennt durch das JWT den Nutzer und ergänzt die Daten durch die UserID und den aktuellen Zeitstempel, bevor sie über das IStatisticsRepository in die Tabelle TrainingResult geschrieben werden.

Eine Ausgabe der TrainingResults gibt es nicht, da es alleinstehend nicht sonderlich interessant ist. Stattdessen können berechnete Statistiken abgefragt werden. Über `GET /stats/variants/{id}` werden die Statistiken einer einzelnen Eröffnungsvariante abgerufen und über `GET /stats/variants` bekommt man eine Liste aller bisher geübten Eröffnungen mit ihren Statistiken. Um Nutzern eine Empfehlung zu geben, welche Eröffnungen als nächstes geübt werden sollen, kann man die Eröffnungen nach der Expertise sortieren. Je höher eine Eröffnung in dieser Liste ist, umso mehr sollte sie wieder geübt werden. Die Berechnungen für die Expertise werden im StatisticsRepository durchgeführt.

StatisticsRepository

Die Klasse OpeningStatistics enthält alle statistischen Werte, die das StatisticsRepository berechnet. Besonders interessant ist hier der Wert „expertise“, welcher angeben soll, mit welcher Wahrscheinlichkeit ein Nutzer die Eröffnung fehlerfrei wiedergeben kann. Die Berechnung dieses Wertes basiert auf der Halbwertszeitregression aus Unterabschnitt 2.2.3.

Wie bei der Halbwertszeitregression ist die Vergessenskurve $p = 2^{-\Delta/h}$ die Basis, wobei p als der Expertisenwert zu verstehen ist. Für Δ kann die Dauer eingesetzt werden, seitdem das letzte Training durchgeführt wurde. Die Halbwertszeit h wird auch durch die Funktion $\hat{h}_\Theta = 2^{\Theta * x}$ bestimmt. Die Bestandteile des Merkmalsvektors x und seinen Gewichtungen Θ sind in Tabelle 4.1 zu sehen. In den TrainingResults werden noch weitere Daten gesammelt, damit man später die Flexibilität hat, andere Merkmale zu verwenden, wie zum Beispiel die durchschnittlichen Fehler pro Trainingsdurchlauf. Im Gegensatz zur Halbwertszeitregression wird keine Regression durchgeführt, um den Gewichtsvektor Θ zu bestimmen, weil das ein erheblicher Zusatzaufwand wäre und Datensätze benötigt werden, die beschreiben, wie schnell Schacheröffnungen gelernt und vergessen werden. Die Gewichte werden stattdessen im Vorhinein festgelegt. Der Nachteil davon ist, dass sie nicht getestet sind und unklar ist, wie gut sie zu der tatsächlichen Vergessenskurve passen. Eine Festlegung durch empirisch erfasste Werte oder durch Regression bestimmte Werte könnten den Algorithmus deutlich verbessern.

θ	x	Beschreibung
-0,5	x_f	Anzahl der Trainingsdurchläufe mit Fehlern
0,2	x_t	Anzahl der fehlerfreien Trainingsdurchläufen mit Tipps
0,6	x_p	Anzahl der perfekten Trainingsdurchläufe ohne Tipps und Fehler
0,1	x_n	Anzahl der gesamten Trainingsdurchläufe

Tabelle 4.1: Merkmalsvektor und Gewichte

Mit den aktuellen Werten muss Δ in Tagen angegeben werden. In Abbildung 4.5 sind Vergessenskurven mit den angegebenen Gewichten und Beispielvektoren x zu sehen. Bei vier fehlgeschlagenen und vier perfekten Trainingsdurchläufen fällt die Kurve in den ersten zehn Tagen stark ab und ist nach 8 Tagen bereits unter 10%, wie in der unteren Kurve zu sehen. Nach einigen weiteren perfekten Durchläufen fällt die Kurve flacher ab. So ist nach diesem Modell 12 Tage später die Wahrscheinlichkeit noch über 80%, dass der Nutzer die Eröffnung wieder fehlerfrei durchführt, was bei zehn perfekten Trainingsdurchläufen durchaus plausibel ist.

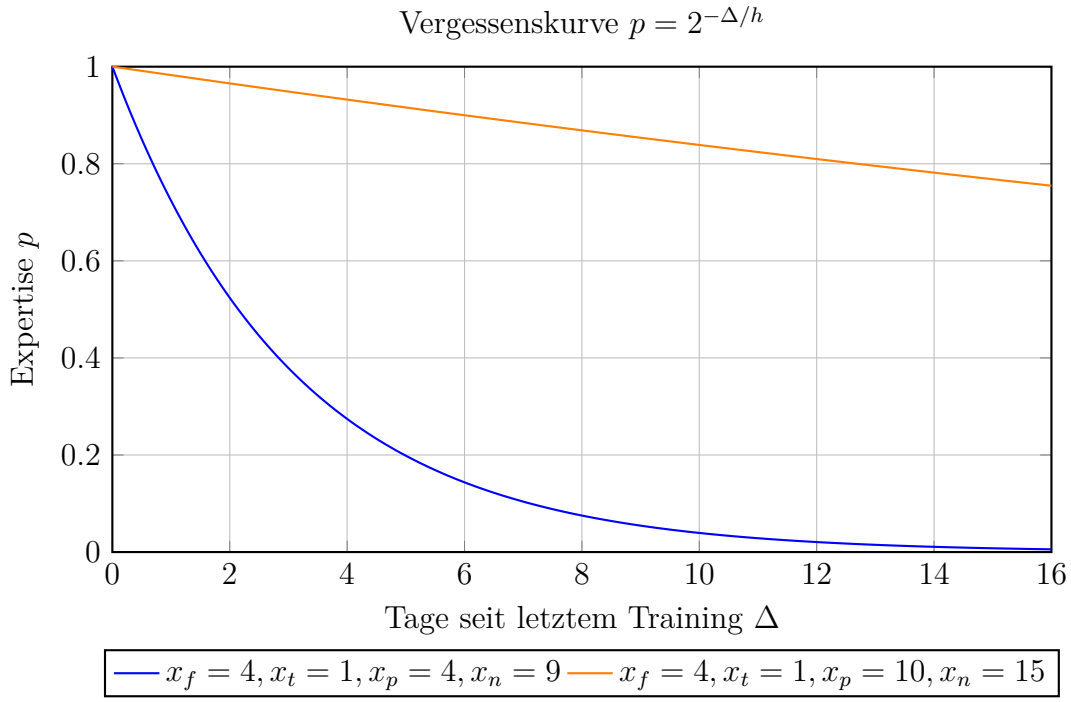

Abbildung 4.5: Beispielhafte Vergessenskurve

4.3.6 Schachengine

Eine Schachengine mit UCI-Protokoll kann eingebunden werden, indem ein Adapter geschrieben wird, der REST-Anfragen für die Engine übersetzt. Das wird dadruch ermöglicht, dass REST und UCI keinen Zustand besitzen. Über einen Endpunkt, wie zum Beispiel `GET /engine` könnte ein Computerzug angefragt werden. Über Query-Parameter played und level müsste das Frontend die bisher gespielten Züge übertragen und die Stärke der Engine. Aus Zeitgründen wurde im finalen Programm keine Schachengine eingebunden.

4.4 Laufzeitsicht

In diesem Kapitel wird der Ablauf von den komplexeren Funktionen der Anwendung beschrieben. Dabei wird erklärt, welche Anfragen das Frontend schickt und wie die REST-Endpunkte dabei verwendet werden.

4.4.1 Lernmodus

Für den Lernmodus werden die Funktionen unter dem Endpunkt `/openings` aus Unterabschnitt 4.3.2 verwendet. Das Sequenzdiagramm in Abbildung 4.6 zeigt den Ablauf des Lernmodus. Er kann in den folgenden drei Schritten beschrieben werden.

1. Als erstes wählt der Nutzer den Lernmodus aus. Das Frontend schickt dann eine GET Anfrage an den Endpunkt `/openings`. Dieser liefert eine Liste von OpeningOverviews, die die Namen und IDs der Wurzeleröffnungen enthalten. Die Namen werden dem Nutzer zur Auswahl gezeigt.

2. Angenommen der Nutzer wählt die Eröffnung mit der ID D20 aus, dann schickt das Frontend eine GET Anfragen an den Endpunkt `/openings/D20/variants/next-moves`. Daraufhin liefert die API ein OpeningMove-Objekt für jede Eröffnung, die mit dem selben Namen anfangen, wie die Eröffnung mit der ID D20. Die OpeningMove-Objekte enthalten den Namen und den ersten Zug der jeweiligen Eröffnung. Diese werden dem Nutzer angezeigt.

3. Der Nutzer kann anschließend auswählen, welchen Zug er ausführen möchte, zum Beispiel 1. d4. Die darauffolgenden Züge können wiederum mit der folgenden GET Anfrage herausgefunden werden: `/openings/D20/variants/next-moves?played=d2d4`. Hier werden die bereits ausgeführten Züge im Query-Parameter `played` mit UCI-Format übergeben. Die API liefert dann nur noch die Eröffnungen, die auch mit den selben Zügen beginnen. Dieser Schritt wird so oft wiederholt, bis keine weiteren Züge mehr vorhanden sind, also die Liste mit OpeningMoves leer ist.

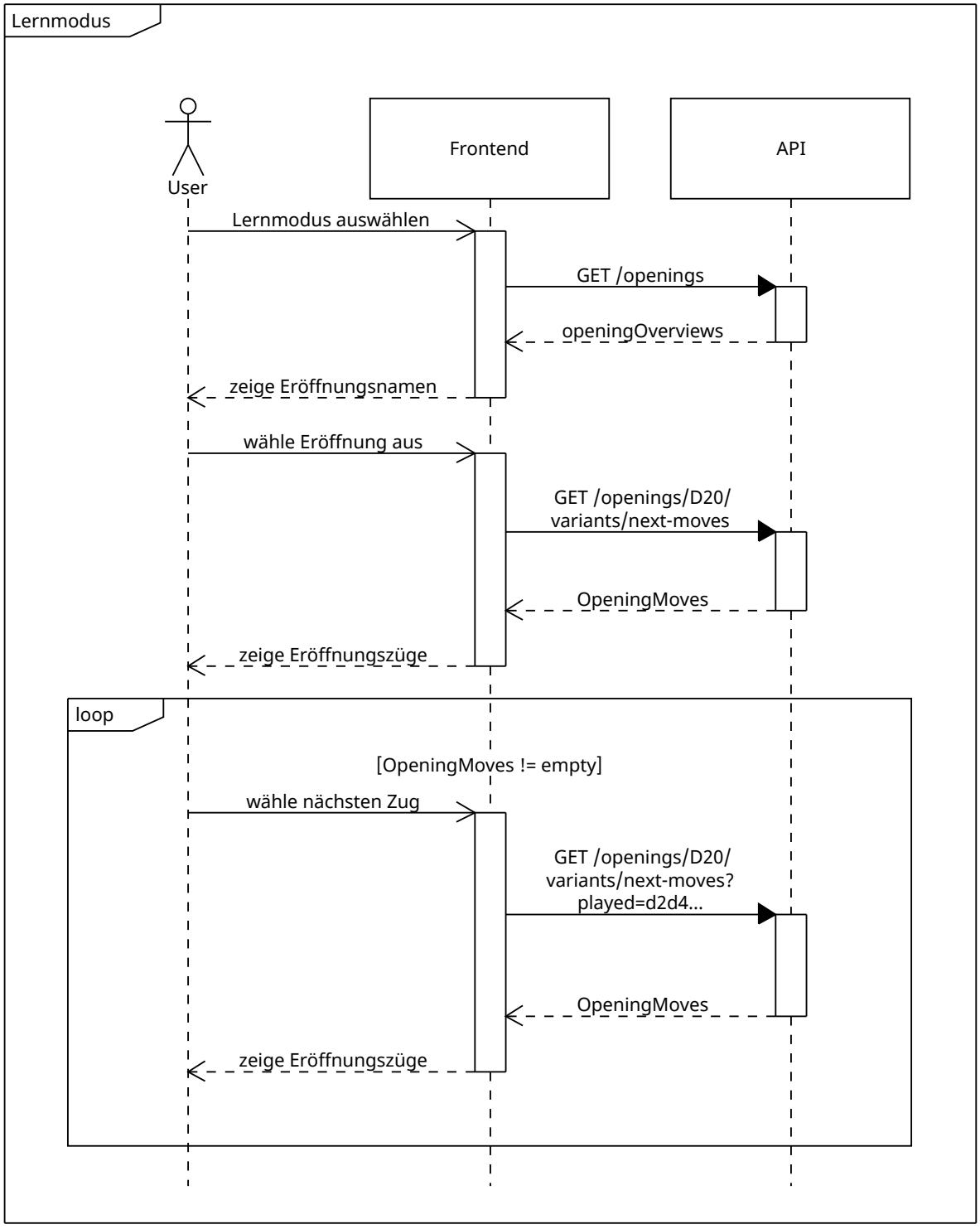

Abbildung 4.6: Ablauf Übungsmodus

4.4.2 Übungsmodus

Der Übungsmodus verwendet ähnliche Endpunkte, wie der Lernmodus. Statt dem Endpunkt `/openings/{id}/variants/next-moves` wird hier der Endpunkt `/openings/{id}/next-move` verwendet, da die unterschiedlichen Varianten nicht betrachtet werden müssen. Der Ablauf wird in dem Zustandsdiagramm in Abbildung 4.7 dargestellt, da dieser Ablauf etwas komplexer ist. Es ist jedoch zu erwähnen, dass der Zustand sich ausschließlich im Frontend befindet, um der REST Architektur treu zu bleiben. Alle benötigten Zustandsinformationen werden mit jeder Anfrage an das Backend mitgegeben.

1. **Eröffnungswahl:** Als erstes wählt der Nutzer den Übungsmodus aus und das Frontend holt sich wieder über `GET /openings` die Liste der Wurzeleröffnungen und zeigt diese mit Namen an.

2. **Variantenwahl:** Nachdem der Nutzer eine ausgewählt hat, muss er eine der Varianten auswählen, die über `GET /openings/{id}/variants` zurückgegeben werden und entscheiden ob er die schwarzen Figuren oder die weißen spielen will. Bei Auswahl der weißen Figuren geht es mit dem Spielerzug weiter, bei Auswahl der schwarzen mit dem Computerzug.

3. **Spielerzug:** Für den nächsten Zug wird der Nutzer aufgefordert, ihn auszuführen. Das Frontend fragt daraufhin mit dem Endpunkt `GET /openings/{id}/next-move` ab, was der richtige nächste Zug für diese Eröffnung ist. Wenn die Züge nicht übereinstimmen, wird er aufgefordert es erneut zu versuchen. An dieser Stelle kann auch die Hinweisfunktion implementiert werden. Wenn der Nutzer nach einem Hinweis fragt, kann das Frontend über den selben Endpunkt den richtigen Zug abfragen und einen Hinweis geben, zum Beispiel welche Figur bewegt werden muss. Dafür ist kein zusätzlicher Endpunkt in der API notwendig. Wenn der Nutzer den richtigen Zug gefunden hat und es noch weitere Züge gibt, geht es mit dem Computerzug weiter. Falls es keine weiteren Züge mehr gibt folgt der Schritt Training speichern.

4. **Computerzug:** Der nächste Zug wird vom Computer ausgeführt. Dafür holt sich das Frontend den Zug über den Endpunkt `GET /openings/{id}/next-move` und führt ihn aus. Wenn das der letzte Zug war geht es mit dem Training speichern weiter, ansonsten ist der Nutzer wieder an der Reihe und es geht mit dem Spielerzug weiter.

5. **Training speichern:** Nach dem letzten Zug wird ein TrainingResult angelegt mit der Anzahl an Fehlern und Hinweisen, über den Endpunkt `POST /stats/variants`.

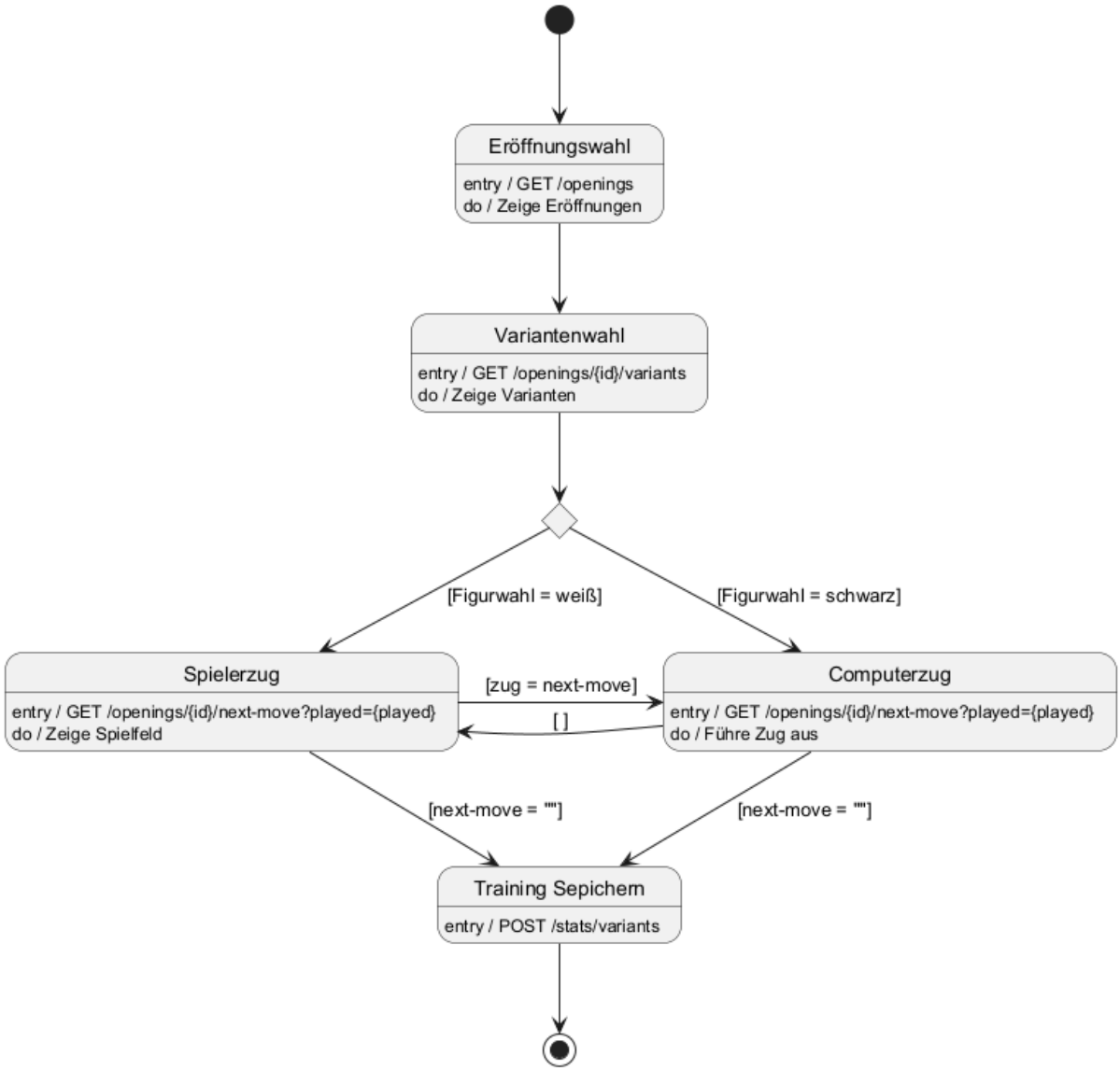

Abbildung 4.7: Ablauf Trainingsmodus

4.5 Verteilungssicht (Deployment)

Um das Programm auf einem Server bereitzustellen werden die einzelnen Bestandteile in Docker Images eingebettet. Das hat vor allem den Vorteil, dass die Programme auf allen Plattformen, die Docker unterstützen, unkompliziert ausgeführt werden können. Die einzelnen Images können mittels Docker-Compose gestarten und verbunden werden. Diese Anwendung ist in drei Images aufgeteilt, die auch in Abbildung 4.8 zu sehen sind. Ein Image enthält das Frontend, eins die Rest-API und eins die Postgres Datenbank. Die Datei mit den Eröffnungen wird in das Image der Rest-API integriert und ist daher nicht auf dem Diagramm zu sehen. Das Frontend und die API ist durch einen Reverse Proxy über das Internet erreichbar. Das ermöglicht es, dass die Komponenten auch auf unterschiedlichen Servern ausgeführt werden können.

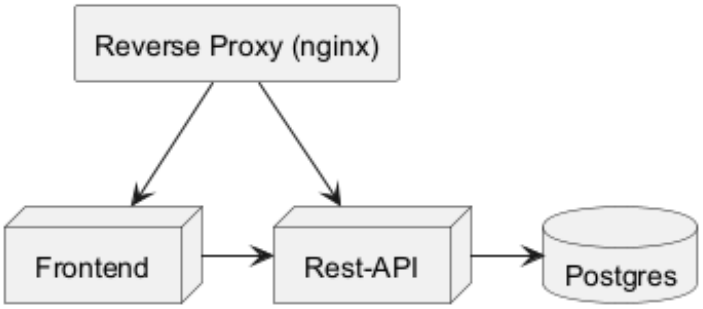

Abbildung 4.8: Verteilungssicht

5 Fazit

Im Rahmen dieser Arbeit wurde untersucht, wie Menschen im Allgemeinen lernen und wie Schachspieler lernen. Dabei zeigte sich, dass es für ein Thema wie Schacheröffnungen wichtig ist, die Inhalte oft zu wiederholen, um zu verhindern, dass sich die Erinnerungen verfälschen oder in Vergessenheit geraten. Im Optimalfall wiederholt man die Inhalte in exponentiell größer werdenden Intervallen, mithife von Spaced Repetition. In der Marktanalyse wurde herausgefunden, dass es einige Medien gibt, die einen Schachspieler beim Erlernen von Eröffnungen unterstützen. Diese sind aber oft nicht kostenlos oder bieten wenig Personalisierung. Daher wurde ein neues Programm erstellt, um diese Lücke zu schließen. Im Rahmen der Recherche wurden dafür Datensätze und Schachengines gefunden, die für die Erstellung einer solchen Schachlernanwendung hilfreich sind.

5.1 Implementierung

In dem implementierten Programm wurden die meisten, der im Voraus definierten Anforderungen, erfüllt. Mit ihm kann man Eröffnungen betrachten und auch üben. Es kann als zentrales Nachschlagewerk genutzt werden, um sich Eröffnungen nochmal ins Gedächtnis zu rufen. Außerdem wird dem Nutzer vorgeschlagen, welche Eröffnung er als nächstes wiederholen soll, aufgrund eines berechneten Expertisenwertes. Durch die Einbindung eines öffentlichen Datensatzes stehen sehr viele Eröffnungen zur Auswahl. Aus zeitlichen Gründen war es nicht mehr möglich die Übungsmodus Erweiterung umzusetzen. Das Programm kann dennoch gut als Ergänzung zu bestehenden Lernmedien verwendet werden. So kann man zum Beispiel einen Trainer oder ein Buch verwenden, um die Kernideen einer Eröffnung zu verstehen und das Programm verwenden, um die Eröffnung zu wiederholen und sein Wissen zu festigen.

5.2 Ausblick

Das entwickelte Programm kann noch in einigen Aspekten verbessert werden. Ein Punkt wurde bereits bei den Statistiken genannt. Die Vergessenskurve und damit auch der Expertisenwert, der ausschlaggebend für die Empfehlungen ist, wird aufgrund von geratenen Parametern berechnet. Es wäre möglich durch die Statistiken der Anwendung Daten zu

sammeln, um die Parameter durch Regression zu bestimmen. Das würde die Berechnung des Expertisenwertes deutlich verbessern und die Vorschläge, welche Eröffnung als nächstes wiederholt werden soll, wären akkurater. Durch positives Feedback könnte man Nutzer dazu motivieren häufiger zu trainieren. Dafür kann man positive Farben und Animationen verwenden, oder man orientiert sich an den Belohnungen von Chessable und bindet auch ein Punktesystem oder Belohnungen ein. Eine nützliche Erweiterung wären auch erklärende Infotexte, um das tiefere Verständnis von Eröffnungen verstärken. Ein Nachteil der Anwendung ist, dass sich in dem verwendeten Datensatz sehr viele Eröffnungen befinden. Das kann vor allem für Anfänger überfordernd sein. Empfehlungen für ungeübte Eröffnungen wären auch eine Verbesserung, denn die Vorschläge in der aktuellen Implementierung begrenzen sich auf Eröffnungen, die bereits geübt wurden. Es wäre auch denkbar eine Eröffnungsdatenbank einzubinden, um Nutzern zu zeigen, wie beliebt verschiedene Eröffnungen sind. Zu guter letzt kann man durch Einbindung einer Schachengine die Übungsmodus Erweiterung ergänzen. Dann könnten Nutzer nach einer Eröffnung gegen einen Computergegner weiter spielen. Das verbessert das Verständnis zwischen Eröffnungen und Mittelspiel. Die Anwendung könnte auch erweitert werden, indem Übungen für das Mittel- und Endspiel angeboten werden. Das kann zum Beispiel durch Puzzles umgesetzt werden.

Literatur

[20] „Neural network topology“. In: (Nov. 2020). URL: https://lczero.org/dev/backend/nn/ (besucht am 27. 01. 2025).

[Bro+12] Cameron B. Browne u. a. „A Survey of Monte Carlo Tree Search Methods“. In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), S. 1–43. DOI: 10.1109/TCIAIG.2012.2186810.

[CCR25] CCRL team. „CCRL 40/15“. In: (Jan. 2025). URL: https://computerchess.org.uk/ccrl/4040/ (besucht am 31. 01. 2025).

[Che22] Chess.com. „Chess.com hat 100 Millionen Mitglieder“. In: (Dez. 2022). URL: https://www.chess.com/de/article/view/chess-com-hat-100-millionen-mitglieder (besucht am 03. 11. 2024).

[De17] Brajesh De. *API Management*. en. Berkeley, CA: Apress, 2017. ISBN: 978-1-4842-1306-3 978-1-4842-1305-6. DOI: 10.1007/978-1-4842-1305-6. URL: http://link.springer.com/10.1007/978-1-4842-1305-6 (besucht am 17. 01. 2025).

[Eva04] Eric Evans. *Domain-driven design: tackling complexity in the heart of software*. en. Boston: Addison-Wesley, 2004. ISBN: 978-0-321-12521-7.

[FT00] Roy Thomas Fielding und Richard N. Taylor. „Architectural styles and the design of network-based software architectures“. ISBN: 0599871180. PhD Thesis. University of California, Irvine, 2000.

[GJ06] Fernand Gobet und Peter Jansen. „Training in chess: A scientific approach“. In: *Chess and education* (Jan. 2006).

[GS96] Fernand Gobet und Herbert A. Simon. „Templates in Chess Memory: A Mechanism for Recalling Several Boards“. In: *Cognitive Psychology* 31.1 (1996), S. 1–40. ISSN: 0010-0285. DOI: https://doi.org/10.1006/cogp.1996.0011. URL: https://www.sciencedirect.com/science/article/pii/S0010028596900110.

[Gus21] Dmitri A Gusev. „Using Modern Chess Software for Opening Preparation“. en. In: (2021). URL: https://eric.ed.gov/?id=ED617407.

[Her85] Hermann Ebbinghaus. *Über das Gedächtnis. Untersuchungen zur experimentellen Psychologie*. Leipzig: Duncker & Humber, 1885.

[KJS24] Friedrich W. Kron, Eiko Jürgens und Jutta Standop. *Grundwissen Didaktik*. ger. 7., vollständig überarbeitete und erweiterte Auflage. UTB 8073. Stuttgart: Ernst Reinhardt Verlag, 2024. ISBN: 978-3-8252-8802-0 978-3-8385-8802-5 978-3-8463-8802-0. DOI: 10.36198/9783838588025.

[KM75] Donald E. Knuth und Ronald W. Moore. „An analysis of alpha-beta pruning“. In: *Artificial Intelligence* 6.4 (1975), S. 293–326. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(75)90019-3. URL: https://www.sciencedirect.com/science/article/pii/0004370275900193.

[Lei73] Sebastian Leitner. *So lernt man lernen: angewandte Lernpsychologie - ein Weg zum Erfolg*. ger. 6. Aufl. Freiburg im Breisgau: Herder, 1973. ISBN: 978-3-451-16265-7.

[lic25] lichess.org. *chess-openings*. Feb. 2025. URL: https://github.com/lichess-org/chess-openings (besucht am 28. 03. 2025).

[Mad20] Neil Madden. *API security in action*. en. OCLC: on1233306337. Shelter Island: Manning Publications, 2020. ISBN: 978-1-61729-602-4.

[Mas12] Mark Massé. *REST API design rulebook: designing consistent RESTful Web Service Interfaces*. en. Beijing Köln: O'Reilly, 2012. ISBN: 978-1-4493-1050-9.

[ÖJ25] Ömür Yanıkoğlu und Jeff Lowery. *eco.json*. Feb. 2025. URL: https://github.com/hayatbiralem/eco.json/commits/master/ (besucht am 28. 03. 2025).

[Pro] Prof Barry Hymer. „The science of learning“. In: (). URL: https://www.chessable.com/science/ (besucht am 03. 02. 2025).

[See19] Mark Seemann. *Dependency Injection Principles, Practices, and Patterns*. eng. New York: Manning Publications Co. LLC, 2019. ISBN: 978-1-61729-473-0 978-1-63835-710-0.

[Sil+17a] David Silver u. a. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. _eprint: 1712.01815. 2017. URL: https://arxiv.org/abs/1712.01815.

[Sil+17b] David Silver u. a. „Mastering the game of Go without human knowledge“. In: *Nature* 550.7676 (Okt. 2017), S. 354–359. ISSN: 1476-4687. DOI: 10.1038/nature24270. URL: https://doi.org/10.1038/nature24270.

[SK23] Software Development Engineering Advisor at Fiserv, USA und Rajesh Kotha. „Architecting Secure REST APIs with Authentication and Authorization Approaches for Web Services“. en. In: *Journal of Mathematical & Computer Applications* (Juni 2023), S. 1–5. ISSN: 27546705, 27546705. DOI: 10.47363/JMCA/2023(2)E132. URL: https://www.onlinescientificresearch.com/articles/architecting-secure-rest-apis-with-authentication-

and-authorization-approaches-for-web-services.pdf (besucht am 18. 04. 2025).

[SM16] Burr Settles und Brendan Meeder. „A Trainable Spaced Repetition Model for Language Learning“. en. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, 2016, S. 1848–1858. DOI: 10.18653/v1/P16-1174. URL: http://aclweb.org/anthology/P16-1174 (besucht am 02. 05. 2025).

[SMS17] Florian Schimanke, Robert Mertens und Ute Schmid. „Spaced Repetition in Mobile Learning Games - A Cure to Bulimic Learning?“ In: Okt. 2017.

[Ste25] Stefan Meyer-Kahlen. „Universal Chess Interface (UCI)“. In: (2025). URL: https://www.shredderchess.com/chess-features/uci-universal-chess-interface.html (besucht am 02. 01. 2025).

[TCE25] TCEC Chessdom. „TCEC-Chess“. In: (2025). URL: https://tcec-chess.com/ (besucht am 27. 01. 2025).

[Vje19] Vjekoslav Nemec. „History Of Chess Computer Engines“. In: (Jan. 2019). URL: https://chessentials.com/history-of-chess-computer-engines/ (besucht am 23. 11. 2024).

[Wik24a] Wikipedia Foundation Inc. „Encyclopaedia of Chess Openings“. In: (Dez. 2024). URL: https://en.wikipedia.org/wiki/Encyclopaedia_of_Chess_Openings (besucht am 29. 03. 2025).

[Wik24b] Wikipedia Foundation Inc. „Komodo (Schach)“. In: (Sep. 2024). URL: https://de.wikipedia.org/wiki/Komodo_(Schach) (besucht am 31. 01. 2025).

[Wik24c] Wikipedia Foundation Inc. „Universal Chess Interface“. In: (Nov. 2024). URL: https://en.wikipedia.org/wiki/Universal_Chess_Interface (besucht am 17. 01. 2025).

[Wik25a] Wikipedia Foundation Inc. „Chess opening book (computers)“. In: (März 2025). URL: https://en.wikipedia.org/wiki/Chess_opening_book_(computers) (besucht am 29. 03. 2025).

[Wik25b] Wikipedia Foundation Inc. „Stockfish (chess)“. In: (Jan. 2025). URL: https://en.wikipedia.org/wiki/Stockfish_(chess) (besucht am 31. 01. 2025).