h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbmd
FACHBEREICH MEDIA

Projektdokumentation

Steampeen - Bewertung von Steamprofilen
anhand multipler Kriterien.

Stand: 28.08.2020

Dozent: Prof. Dr. Stefan Schmunk
Andreas Schieberle
Kurs: Daten- und Informationsmanagement
Studenten:
729668 - Alexander Ceynowa
750659 - Julian Sträter
763407 - Shehroz Sial

Inhaltsverzeichnis

1 Vorbereitungen

Das Kapitel “Vorbereitungen” beschäftigt sich lediglich mit der Theorie und den ursprünglichen Ansätzen; konkrete Umsetzung, Probleme, Lösungen und Änderungen werden später erörtert.

1.1 Überlegungen

Im Rahmen des Projektseminares Daten- und Informationsmanagement wurde den Studenten die Umsetzung eines Projektes aufgetragen. Inhalt und Aufbau dessen war den Studenten überlassen. Nach initialer Überlegung wurde von unserer Gruppe ein Thema beschlossen.

Eine gemeinsame Interessensgrundlage war der Bereich Gaming und das Nutzen der digitalen Plattform “Steam”, über die eine Spielebibliothek aufgebaut und verwaltet werden kann. Auf Steam wird einiges an Metriken über die Spiele erfasst, die ein Benutzer besitzt. Unter anderem die Zeit, die in einem Spiel verbracht wird, Spiele können in der Storefront von Nutzern bewertet werden, es wird (zum Zeitpunkt dieser Dokumentation) eine externe Bewertung von metacritic angezeigt, aber all diese Daten beziehen sich jeweils lediglich auf ein einzelnes Spiel; es gibt keinen Wert, der stellvertretend das komplette Profil eines Nutzers beschreibt. Es gibt zwar das sogenannte Steam-Level, aber dieses sagt nichts über das Profil als solches aus, sondern beschreibt grob gesagt nur die Interaktion mit der Plattform Steam (beispielsweise, wie viele Titel direkt über die Steam-Storefront gekauft wurden). Durch andere (private) Projekte ist Alex bereits mit der Steam-API bekannt und hat einen groben Überblick über die Möglichkeiten, welche Daten aus der Steam-API abgerufen werden können. Hieraus bildete sich der Plan, den (Spiele-)Geschmack eines Nutzers zu bewerten.

1.2 Konkrete Problemstellung

Ein Steam-Profil bietet einiges an Daten über einen Nutzer und seine Interaktion mit Spielen, aber entweder sind die angebotenen Informationen nicht das, was einen Nutzer interessiert, oder die Art, wie die Zahl zustande kommt, ist derart verschlungen, dass sie jede Aussagekraft verliert. Wir möchten also eine Zahl errechnen, die die Bibliothek eines Nutzers und eine, die den Geschmack eines Nutzers beschreibt. Um die Bibliothek zu bewerten, bietet es sich an, die durchschnittliche Bewertung seiner Spiele zu nutzen, aber um den Geschmack zu quantifizieren, halten wir diesen Ansatz nicht für aussagekräftig. Wir möchten hier also noch weitere Faktoren einbinden.

1.3 Der Geschmack als Zahl

Unser ursprünglicher Ansatz hierfür war eine Kombination aus der Bewertung der Spiele, der kumulativen Spielzeit des Nutzers, seinem Steam-Level, der Anzahl seiner Achievements (Errungenschaften innerhalb von Spielen) und VAC/Community-Bans (einfach gesagt seine Interaktion mit anderen Spielern). Die Bans sollten hier lediglich dazu genutzt werden, den Nutzer von unserer Wertung auszuschließen.

Während initialer Tests wurden hier bereits Probleme sichtbar, das Größte barg unser zweites großes Kriterium: die Spielzeit. Würden wir einfach nur die kumulative Spielzeit nutzen, so wären Spieler, die schon länger auf der Plattform Steam aktiv sind, im Vorteil gegenüber neuen Mitgliedern. Hier wurde also der Entschluss gefasst, die durchschnittliche Zeit, die in allen Spielen verbracht wurde, zu nutzen. Dies sollte einen guten Eindruck davon verschaffen, wie aktiv ein Spieler ist, ohne Rücksicht auf die Größe der Bibliothek oder das Alter des Accounts nehmen zu müssen. Auch Achievements waren nicht mehr so aussagekräftig wie einst, da es inzwischen diverse “Spiele” gibt, die darauf ausgelegt sind möglichst viele Achievements zu generieren - als Metrik fielen diese also raus. Bans wurden dann ebenfalls aus der Liste der Kriterien gestrichen, da diese, wie bereits erwähnt, die Interaktion mit der Community beschrieben, aber nicht den Geschmack an Spielen. Unsere finale Liste besteht also nur noch aus der Anzahl der Spiele im Besitz des Nutzers, seiner Spielzeit und der Bewertung der Spiele. Nach weiterer Überlegung nahmen wir das Account-Level eines Nutzers wieder in die Kriterien auf, welches aus in 1.2 genannten Gründen ursprünglich ausgeschlossen wurde. Unsere Argumentation war, dass es eine bewusste Kaufentscheidung für ein Spiel repräsentierte, da Spiele, die in einer Steam-Bibliothek stehen, nicht zwangsläufig auch über Steam erstanden wurden, sondern beispielsweise in einem sogenannten “Bundle” zusammen mit anderen Spielen erstanden wurden.

Die nächste Frage, die sich uns stellte, war, wie wir hieraus nun einen Score errechnen sollten. Tests ergaben, dass eine reine Multiplikation von Zahlen hier keine aussagekräftigen Bewertungen ausgeben würde, also entschieden wir uns dafür, die Bewertung von Spielen als Basiswert für den Score zu nutzen und eine Art Modifikator in Form eines Multipliers einzuführen.

1.4 Datenquellen

Um möglichst aussagekräftige Scores zu generieren, ist die Grundlage auf der diese aufbauen natürlich sehr wichtig. Wir entschieden uns dafür, Daten nicht nur von Steam, sondern auch aus anderen Quellen anzuzapfen. Steam ist und bleibt zwar unsere wichtigste Quelle, ist aber mehr unser Ausgangspunkt und weniger ein entscheidender Faktor für einen Score. Als aktive Quellen suchten wir uns fünf Anbieter aus: Steam, Metacritic, IGDB, GOG und OpenCritic. Die Argumentation hier war, dass zwei Anbieter (Steam und GOG) eine Storefront sind, die Spiele vertreibt und Spieler ein gekauftes Produkt bewerten, Metacritic als Aggregator bietet für die meisten Spiele eine Bewertung von Benutzern und professionellen Reviewern an und Opencric sowie IGDB sind Plattformen, deren Inhalt durch die Nutzer generiert wird (ähnlich Wikipedia), und somit eine Klientel anziehen, der es für ein Spiel wichtig genug erscheint, eine Wertung abzugeben, ohne dafür belohnt zu werden.

2 Technik

2.1 Grundlagen

Das Projekt läuft über einen LAMP-Stack, also Linux, Apache, MySQL und PHP. MySql wird in unserem Fall mittels MariaDB umgesetzt, PHP läuft aktuell auf Version 7.4. Die meisten Skripte werden On-Demand bei Seitenaufruf ausgeführt, um Daten möglichst aktuell zu zeigen, Grunddaten wie die Liste aller Spiele und der Wertung von Spielen werden zyklisch alle vierzehn Tage von einem Cronjob aktualisiert. Zur Kommunikation mit der API von IGDB benutzen wir eine Library von @enisz (https://github.com/enisz/igdb), um die SteamIDs zu konvertieren, nutzen wir eine Library von Mukunda Johnson (https://github.com/mukunda-/steamidparser).

2.2 Datenspeicher

Unsere Daten werden, wie bereits erwähnt, in MariaDB gespeichert. Anfangs gab es die Überlegung Daten eventuell in einer MongoDB zu speichern, aber fehlende Erfahrungen mit NoSQL-Systemen brachte uns schließlich dazu dies zu verwerfen. Die Performance der Datenbank (bei unserer Architektur) lässt hier zwar streckenweise zu wünschen übrig, lässt uns bei dieser Datenmenge aber leichter damit arbeiten. Abbildung 1 zeigt das Layout der Datenbank.

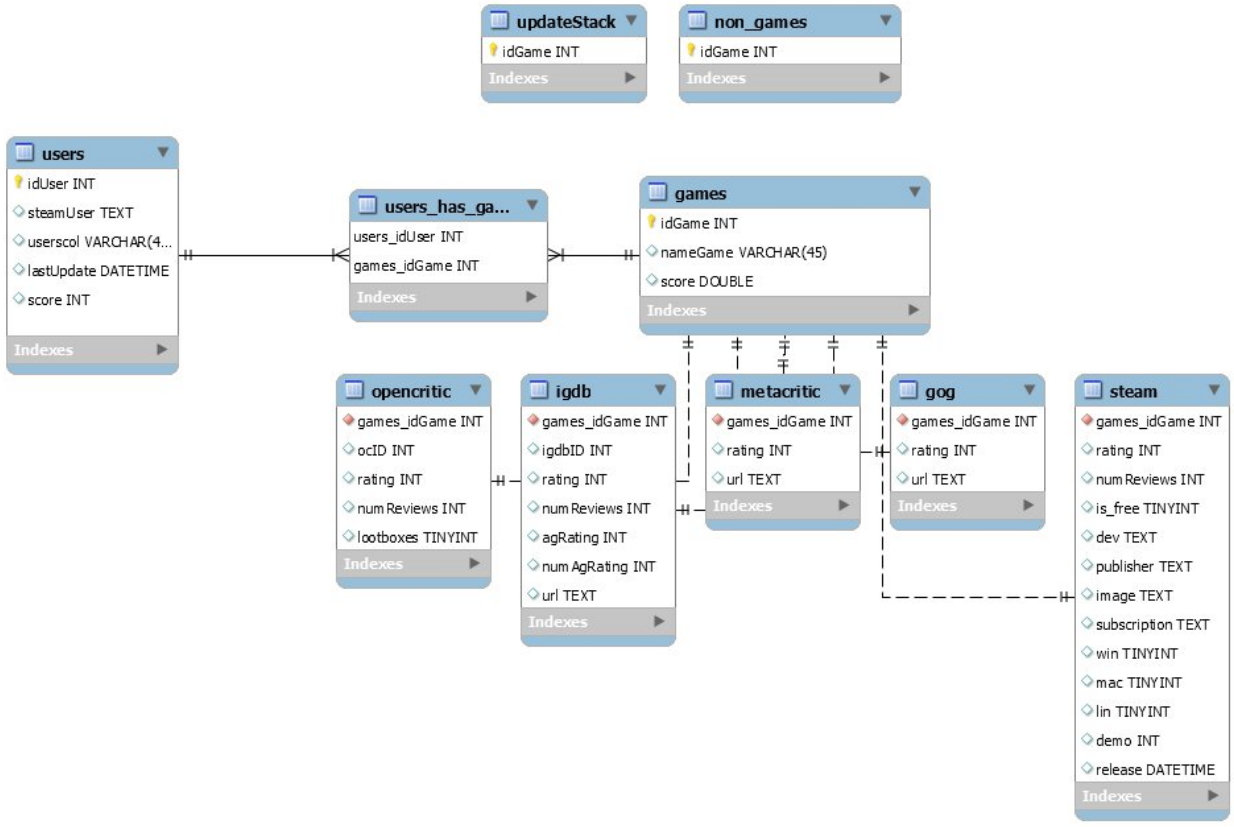


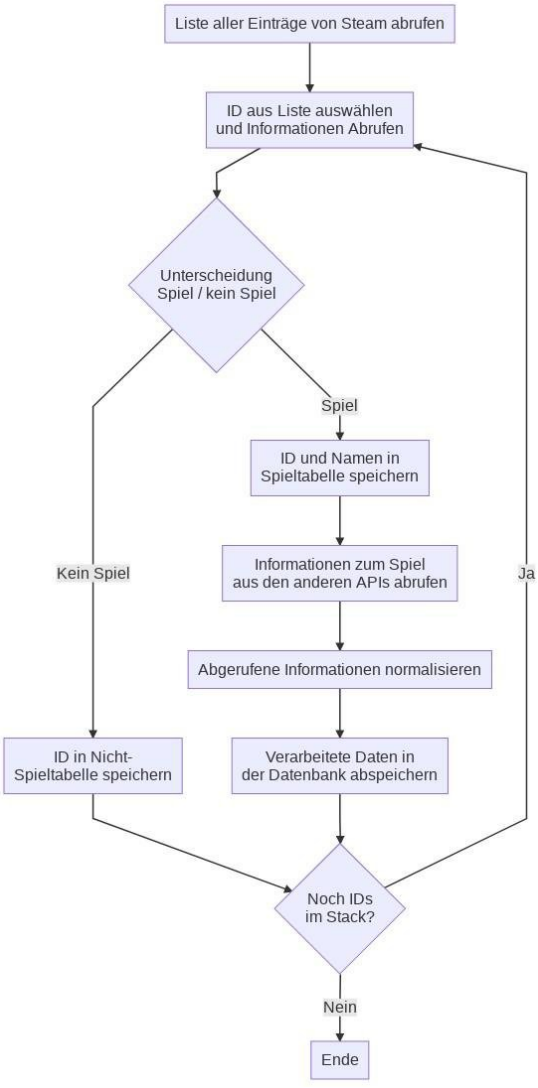
Abbildung 1 - Datenbank Layout

2.3 Interaktion mit APIs

Die APIs werden über PHP mittels file _get_contents() abgerufen und antworten mit Daten im JSON-Format. Die Konstruktion der Anfrage hängt stark von der API ab, GOG benötigt keine Authentifizierung, Steam benötigt einen API-Key in der URL und IGDB benötigt einen API-Key im Körper der Anfrage. Die speziellen Nuancen sind im Quellcode dokumentiert.

3 Datenaufnahme

3.1 Aufnahme “neuer” Titel



Während dem initialen Ingest (erstmaliges Abrufen von Daten von Steam) wurde der gesamte Bestand von Steam als ein json-Array abgerufen und die darin enthaltenen IDs in die Tabelle Updatestack eingetragen. Die Liste wird dann wieder aus der Datenbank abgerufen und in einem Loop bearbeitet. Bei diesem Schritt war Performance kein Primärziel, hier ging es eher um Zuverlässigkeit, da die hieraus entstehenden Datensätze den Grundstamm späterer Updates bilden, speziell wenn ein neuer Ingest von Steam-Daten angestoßen wird. Während dem Loop werden Grundinformationen über das Spiel von Steam abgerufen, in denen auch aufgeführt ist ob es sich um ein Spiel, einen Trailer oder beispielsweise eine Demo handelt. Hier wird lediglich zwischen Spiel und nicht-Spiel unterschieden - nicht-Spiele werden einfach nur durch ihre ID in der non_games-Tabelle gespeichert, "Spiele" werden noch (soweit möglich) um Daten aus anderen APIs erweitert. Diese Daten werden normalisiert und in den entsprechenden Tabellen gespeichert.

Normalisierung der Daten bezieht sich hier auf GOG und Steam. IGDB, OpenCritic und Metacritic bewerten auf einem 100-Punkte-Maximum, GOG bewertet auf einem 50-Punkte-Maximum (null bis fünf Sterne) und Steam bewertet auf einem Ja/Nein-System für Empfehlungen. Um uns im 'Front-Backend-Bereich' Arbeit und (längere) Ladezeit zu sparen, macht es Sinn, die Werte alle auf eine null bis hundert Basis umzurechnen.

Während des erstmaligen Updates wurden alle IDs von Steam in Spiele und nicht-Spiele getrennt. Zweck dieses Vorgehens ist es, bei einem Update des Bestandes (nicht der Bewertungen!) bereits bekannte IDs ignorieren zu können. Die Vergabe von IDs bei Steam erfolgt nicht sequenziell, sondern kann jede mögliche Position in einer Reihe von Zahlen einnehmen. Es werden also alle IDs von Steam abgefragt, in den updateStack eingetragen, dort werden alle uns bekannten IDs entfernt und was übrig bleibt, müssen neue IDs sein.

3.2 Updaten von bekannten Titeln

Der Verlauf zum Updaten von Daten in der Datenbank ist recht einfach. Um die Updategeschwindigkeit zu erhöhen, wurde ein "Threadingsystem" integriert.

Jede Service-Tabelle (also Daten von Steam[+Metacritic], GOG, IGDB und OpenCritic) haben eine 'Update'-Spalte. Diese Spalte kann Werte von null bis Acht haben. Jeder Wert größer als null repräsentiert einen Thread. Diese Threads werden pro Service von einer Datei (genauer gesagt den Dateien in handler/collector/updater/) gehandelt, wobei der zu bearbeitende Thread über einen $_GET-Parameter festgelegt wird. Diese Dateien sind für die Ausführung via PHP-cli ausgelegt und funktionieren NICHT über den Browser! Um die ThreadID für die Items festzulegen, wird pro Tabelle eine Liste der IDs (games_idGame) der bereits erfassten Titel angelegt und auf die Anzahl von n Threads verteilt. Sind die Update-ID-Listen erstellt, werden diese in eine MySQL-Query gesteckt, die die ThreadIDs in der Datenbank abspeichern. Hierfür wird die MySQL-Funktion 'WHERE games_idGame IN (?)' verwendet, wobei das ? das Array (beziehungsweise eine Komma-separierte Aufzählung) der games_idGame-IDs beinhaltet.

In den Update-Scripts sind einige Sanity-Checks bereits integriert, es wird beispielsweise getestet, ob die threadIDs innerhalb eines akzeptablen Limits liegen. Das ist weniger eine technische Limitation, sondern mehr eine bewusste Entscheidung. Das Maximum liegt bei 8 Threads, momentan arbeiten wir auf Basis von 4 Threads. Hintergedanke hier sind die APIs - auch wenn wir für IGDB und Steam API-Keys haben, können wir nicht ausschließen, dass für 'inoffizielle' APIs nicht doch eine Art Rate-Limit (im Kontext Anfragen pro Zeitraum / DOS) vorhanden ist und wir eventuell geblockt/geblacklistet werden könnten. Hier wäre vielleicht noch Kommunikation mit den API-Anbietern eine Option.

Sind alle Daten abgerufen und aktualisiert, so wird auch die durchschnittliche Bewertung von Titeln neu berechnet und gespeichert.

Daten Validierung bezieht sich hier auf einen simplen Test, ob:

- A. das Spiel noch verfügbar ist, wenn nicht wird der Datensatz entfernt.
- B. wir die korrekten Daten aus den APIs abgerufen haben.
- C. wir den korrekten Datensatz im Ingest übernommen haben.

Anmerkung zu C. - Während dem ursprünglichen Ingest kann es durchaus vorgekommen sein, dass die falschen Datensätze aus Critic-APIs übernommen wurden (Beispiel: Das gewünschte Spiel wäre Cities: Skylines, die Bewertung, die wir als Primärergebnis von GOG bekommen haben, wäre Cities: Skylines DLC, somit ist die Bewertung zwar teilweise relevant, aber nicht der gewünschte Datensatz). Diese Datensätze werden soweit möglich während dem Update gefiltert und aktualisiert oder entfernt.

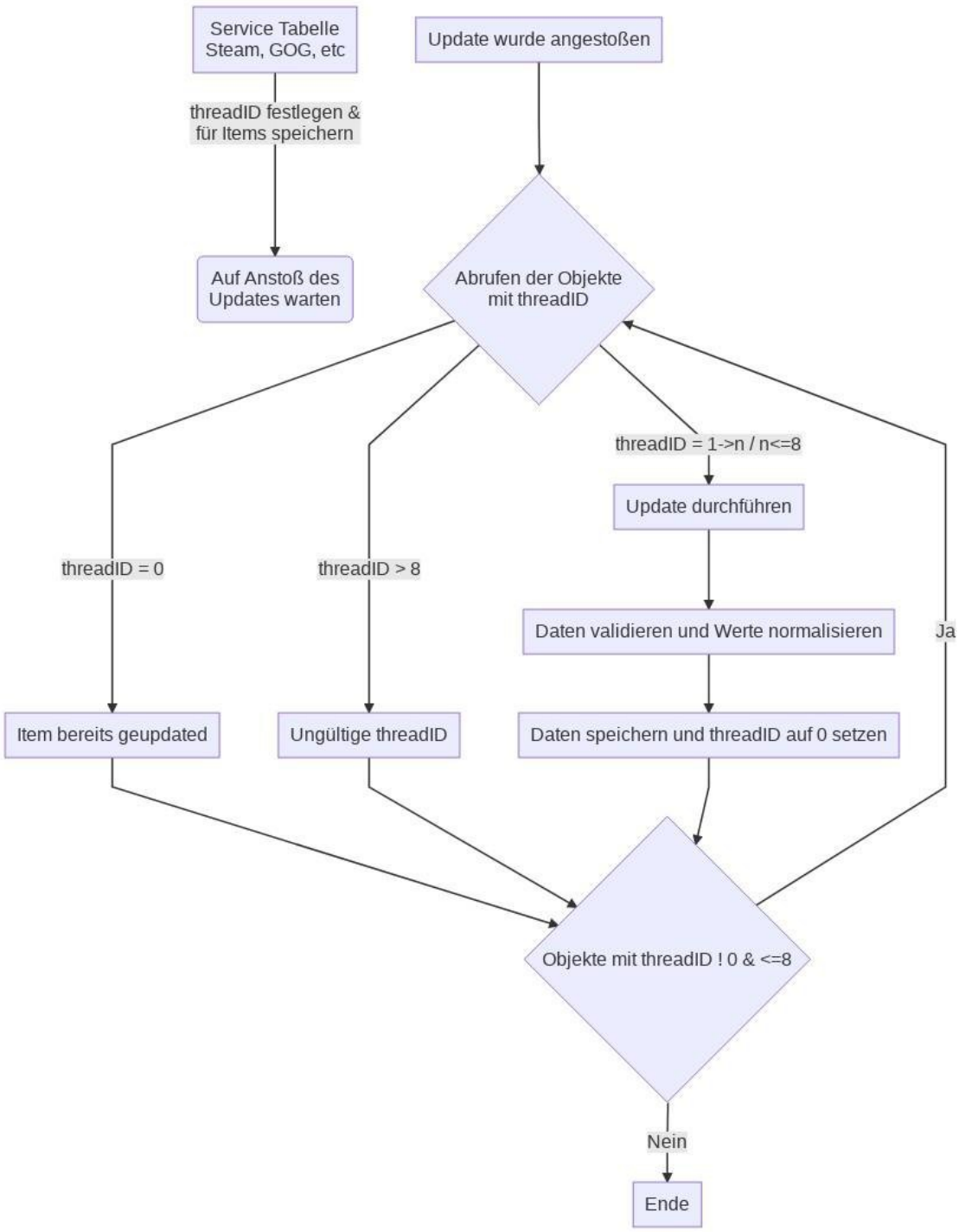


Abbildung 3 - Flowchart für das Update eines Datensatzes

3.3 Sanity Checking

In Kapiteln 3.1 und 3.2 wurde bereits das Konzept “Sanity Checking” angesprochen, darauf soll hier jetzt noch weiter eingegangen werden. Wie bereits erwähnt, ist es durchaus möglich, für ein Spiel die falschen Zusatzdaten abzufragen. Unser momentaner Ansatz ist der Vergleich vom Titel der Anfrage und dem Titel der Antwort. Durch die Struktur der APIs und mögliche lokale Unterschiede funktioniert hier kein boolescher Ja/Nein Vergleich, da bereits ein fehlendes Leerzeichen zu einem falsch-negativ Ergebnis führen könnte. Der Einfachheit halber haben wir uns für ein Likeness/Hemmschwellen-System entschieden, bei dem beide Titel verglichen

werden. Der Vergleich resultiert in einer Zahl auf der Prozentskala und muss eine von uns festgelegte Hemmschwelle übersteigen, um als “der korrekte” Datensatz gehandelt zu werden. Diese Hemmschwelle beträgt aktuell 60 % und hat sich als ausreichend zuverlässig erwiesen. Eine Alternative hier wäre beispielsweise eine gewichtete Levenshtein-Distanz, über die der Updateprozess um einiges intelligenter gestaltet werden könnte. Unser aktuelles Vorgehen berücksichtigt nur die erste Antwort der APIs, könnte mittels Levenshtein aber alle Antworten mit der Anfrage und einander abgleichen und so das beste Ergebnis ermitteln.

3.4 Erfassen von Nutzern

Möchte ein Spieler unsere Plattform nutzen, so benötigt er lediglich einen Identifikator von Steam. Dies kann eine SteamID oder der Link zum Steamprofil sein. Dieser Identifikator wird von einer externen Library (SteamID Parser / Mukunda Johnson) geparsed um die eingegebene ID oder den Link zu einem bestimmten Typ von ID zu machen. Steam identifiziert Nutzer mit einer sogenannten SteamID, diese können mehrere Formate haben. Eine reguläre SteamID wäre beispielsweise STEAM_0:1:15255225, eine SteamID64 wäre 76561197990776179. Die Kommunikation mit den APIs von Steam benötigt spezifisch SteamID64 formatierte IDs, also müssen IDs in anderen Formen konvertiert werden, wurde ein Link zu einem Steamprofil angegeben, muss dieser zur ID aufgelöst werden. Mit der ID des Nutzers können dann bei Steam die entsprechenden Nutzerdaten abgerufen werden. Erst wird geprüft, ob das Profil des Nutzers öffentlich sichtbar ist, danach wird getestet, ob der Nutzer bereits in der Datenbank vorhanden ist. Ist der Nutzer bereits vorhanden, so wird geprüft, wann der Datensatz das letzte Mal aktualisiert wurde. Bei einem Alter von 24 Stunden werden die gespeicherten Daten aktualisiert, sind sie noch aktuell, so wird der Nutzer direkt auf die Profilseite geleitet. Ist der Nutzer noch nicht vorhanden, so werden seine Daten aufbereitet und gespeichert. Es kann jedoch auch sein, dass ein Nutzer seine Bibliothek/sein Profil öffentlich stellt, die Spielzeit jedoch auf privat gestellt und somit nicht sichtbar ist. Das heißt, es ist uns nur möglich, den Inhalt der Bibliothek zu bewerten, jedoch nicht den Geschmack eines Nutzers. In diesem Fall wird der Score automatisch auf null gesetzt.

3.5 Berechnung des Nutzerscores

Der wichtigste Teil des Projekts ist die bereits erwähnte Bewertung der Spielebibliothek eines Nutzers. Um diesen Score zu berechnen, benutzen wir folgende Formel:

$$\frac{1}{n_b}\sum_{b=1}^{n_b} a_b \times log(\frac{(\frac{1}{n_b}\sum_{z=1}^{n_b} a_z \times l_p + 1) \times (1 - m_x) - 1}{1 - m_x})$$

Wobei $n_b$die Anzahl der Spiele, $a_b$die Bewertungen der Spiele, $a_z$die in Spielen verbrachte Zeit, l_p das Steam-Level des Nutzers und $m_x$ein Modifikator des Wertes 0,3 für die logarithmische Funktion repräsentieren.

Auf der linken Seite des “log” berechnen wir die durchschnittliche Bewertung der Spiele eines Nutzers, der Logarithmus ist ein Multiplikator mit sinkender Rendite basierend auf der durchschnittlichen Spielzeit mal dem Steam-Levels des Nutzers. Der Einsatz des Levels in der Formel sollte ein vom Spieler erbrachtes Investment in seine Bibliothek repräsentieren - sollte

der Nutzer beispielsweise lediglich darauf abzielen, eine möglichst große Bibliothek aufzubauen, so würde dieser seinen Bestand weniger über Steam und mehr über dritte Anbieter mit niedrigeren Kosten aufbauen. Die durchschnittliche (im Vergleich zur kumulierten) Spielzeit sollte es den Nutzern zu einem gewissen Maß ermöglichen, unabhängig der Dauer der Mitgliedschaft bei Steam von uns eine möglichst aussagekräftige Bewertung zu erhalten. Jemand, der länger auf Steam vertreten ist, hat vermutlich mehr Spiele als jemand, der noch nicht so lange ein Mitglied ist. Auf diese Art versuchen wir, die Differenzen ein wenig zu mildern.

4 Design //ENTFERNT

5 Lessons Learned

Im Verlauf des Projekts sind uns vermehrt die Effekte von Feature-Creep begegnet. Alex' Stil ist eher organischer Natur. Einfach gesagt beginnt er mit einer Idee und setzt diese nach und nach (unstrukturiert) um. Fällt ihm "unterwegs" etwas ein, das an dieser Stelle gut passen würde, so setzt er es in der Regel auch um. Ein Beispiel wäre die Pagination; ein "gecreeptes" Feature wäre gewesen, den Nutzer die Anzahl der angezeigten Einträge auswählen zu lassen. In diesem Fall wäre das ein minimaler Mehraufwand gewesen, aber nach einem Dutzend solcher "Kleinigkeiten" ginge dann doch ein ziemlicher Zeitaufwand damit einher. Die Selbstkontrolle und Kategorisierung solcher Features/Funktionen in unterschiedliche Dringlichkeiten "notwendig, nett, spielerei, usw." und diese wiederum in verschiedene Meilensteine zu ordnen, hat hier viel geholfen. Ebenfalls ein Problem war das Sprintverhalten. Zwei oder drei Tage wird aktiv, viel und lange gearbeitet, danach ist man quasi ausgelaugt und braucht ein paar Tage, um wieder fit zu sein. Hier wäre es vermutlich effizienter gewesen, die Arbeitskräfte und Zeit besser einzuteilen.

Kollektiv wurden etwa 94,7 Stunden in Programmierung, Layout und Grafikdesign investiert, Planung und Besprechungen wurden hier nicht erfasst; Timetracking lief über Funktionen von Gitlab (/spend) und die Zeiterfassung von PHPStorm.

Zu den Tools lässt sich sagen, dass wir mit unserer Auswahl sehr zufrieden waren. Zur Versionierung benutzten wir Gitlab, dort freundeten wir uns auch gleich mit den Funktionen für Tickets und Meilensteine an. Die Nutzung von Git an sich gestaltet sich weiterhin als schwierig, aber wenn die Bedienung ein wenig besser verstanden wird, ließe sich hier auch ein automatisiertes Deployment neuer Stable-Versionen unseres Projektes automatisieren.

Das Programmieren mittels PHPStorm gestaltete sich als sehr einfach, da im Team bereits Erfahrungen mit dem Programm vorhanden waren. Features wie Timetracking und automatisch generierte Commit-Nachrichten waren hier ein großer Helfer.

Kommunikation lösten wir initial via Slack, das ließ aber ein paar Wünsche offen - kollektiver Videochat war einer dieser Wünsche. Ein weiteres Problem war, dass man zur Kommunikation mit den Mitgliedern darauf angewiesen ist, dass jeder auch aktiv "slackt" - das ist nicht immer bei jedem der Fall. Kontakt wurde also nicht exklusiv (wie geplant) über Slack gelöst, sondern war ein ziemliches Wirrwarr aus Slack, Jitsi, Discord, Issues in Gitlab und vereinzelt Whatsapp oder E-Mail.

Zur technischen Seite - das größte Problem an unserem Projekt ist aktuell das Layout der Datenbank. Sie ist zwar darauf ausgelegt, möglichst erweiterbar zu sein (speziell im Hinblick auf zusätzliche Critic-APIs), skaliert, aber nicht gut von der Leistung her. Die komplexen Joins, die nötig sind, um die für ein einzelnes Spiel benötigten Daten zu Sammeln, mögen hier noch recht performant sein, aber die “Top Games” Listen benötigen, ohne gecached zu sein, einen beträchtlichen Zeitraum, um zu laden. Eine temporäre Lösung wäre es, die Datenbanken von der aktuellen (mechanischen) Festplatte auf eine SSD umzusiedeln, aber das behandelt eher die Symptome und weniger die Ursache. Die wirkliche Lösung wäre es, entweder die Struktur der Datenbank zu ändern oder auf ein anderes Datenbanksystem umzusteigen.

6 Anhang

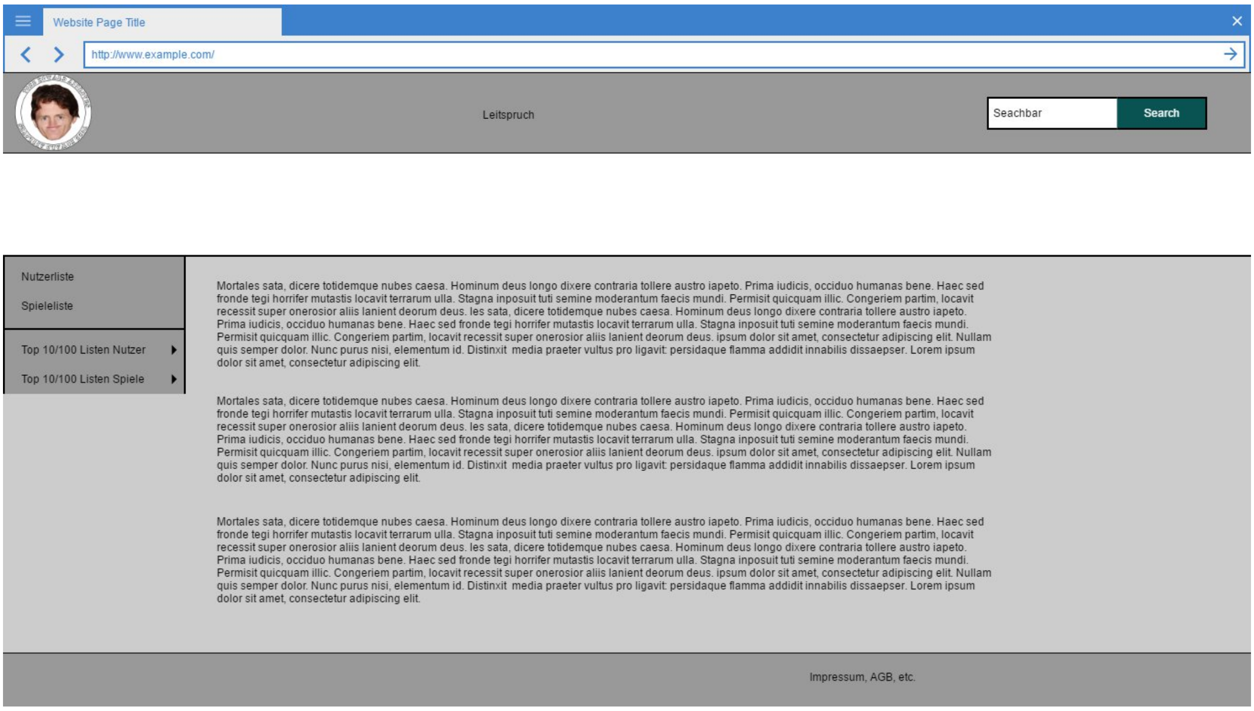


Abbildung A.1 - PureText Pages Wireframe

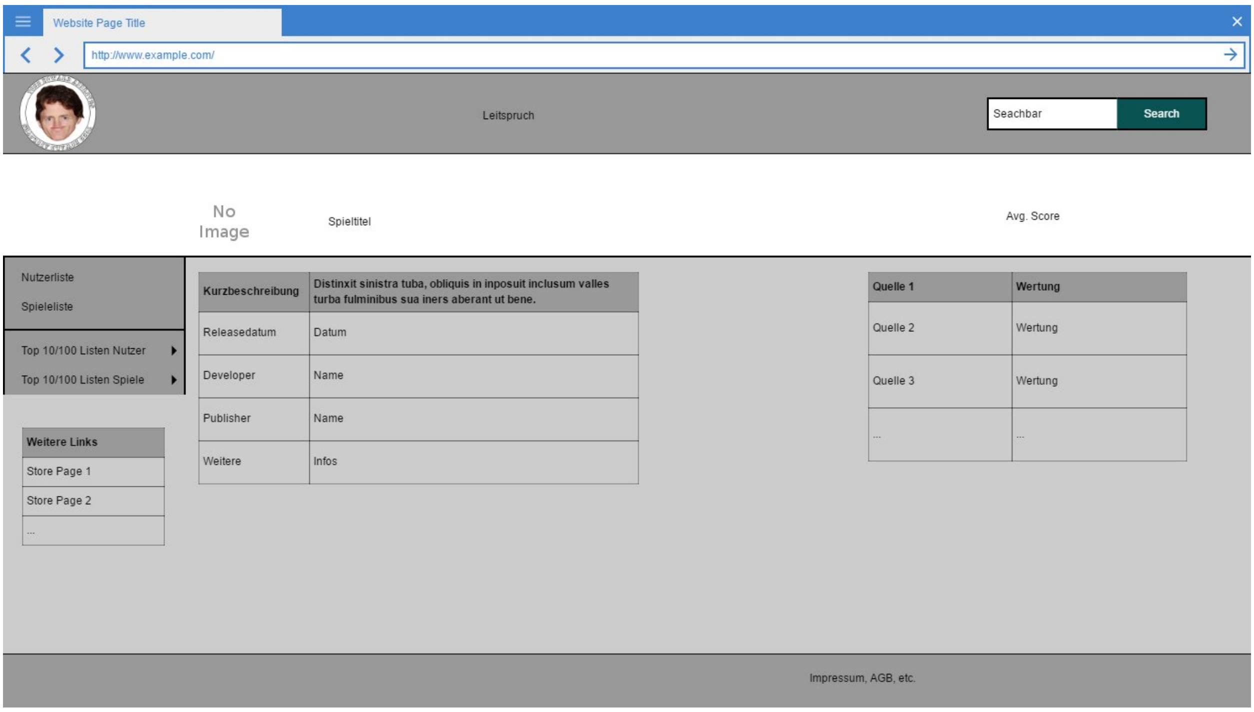


Abbildung A.2 - Spielansicht Wireframe

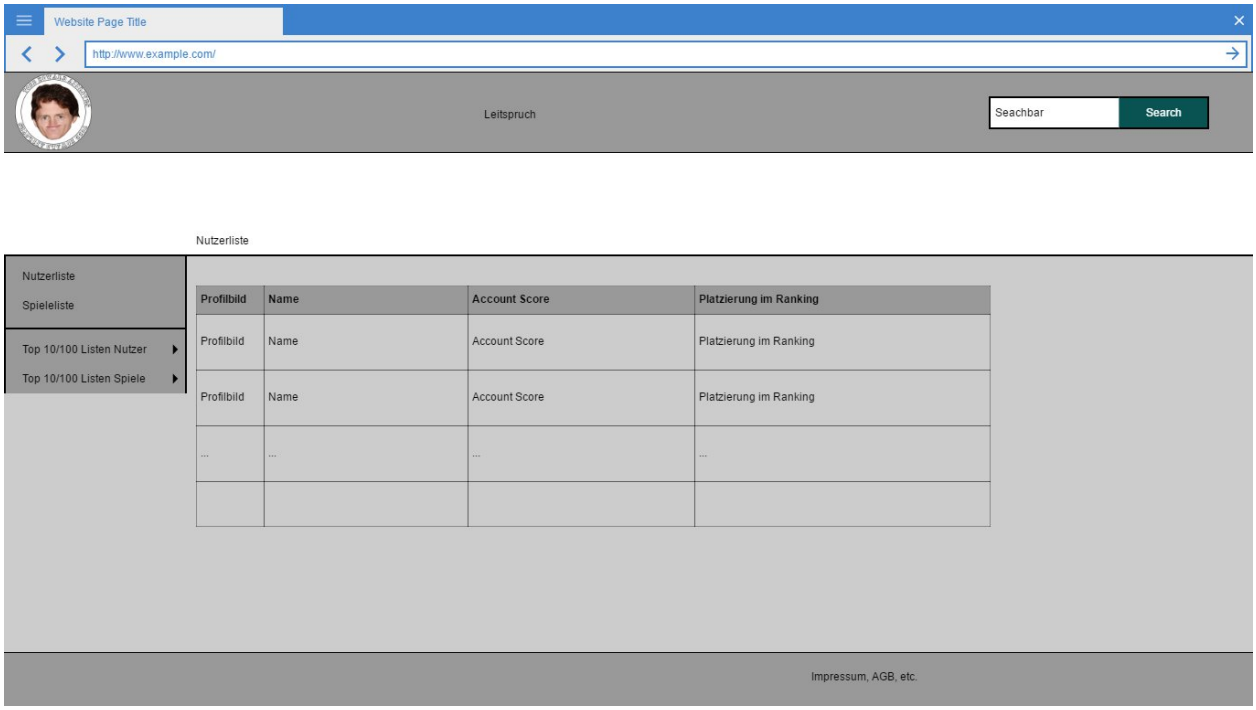


Abbildung A.3 - Nutzerliste Wireframe

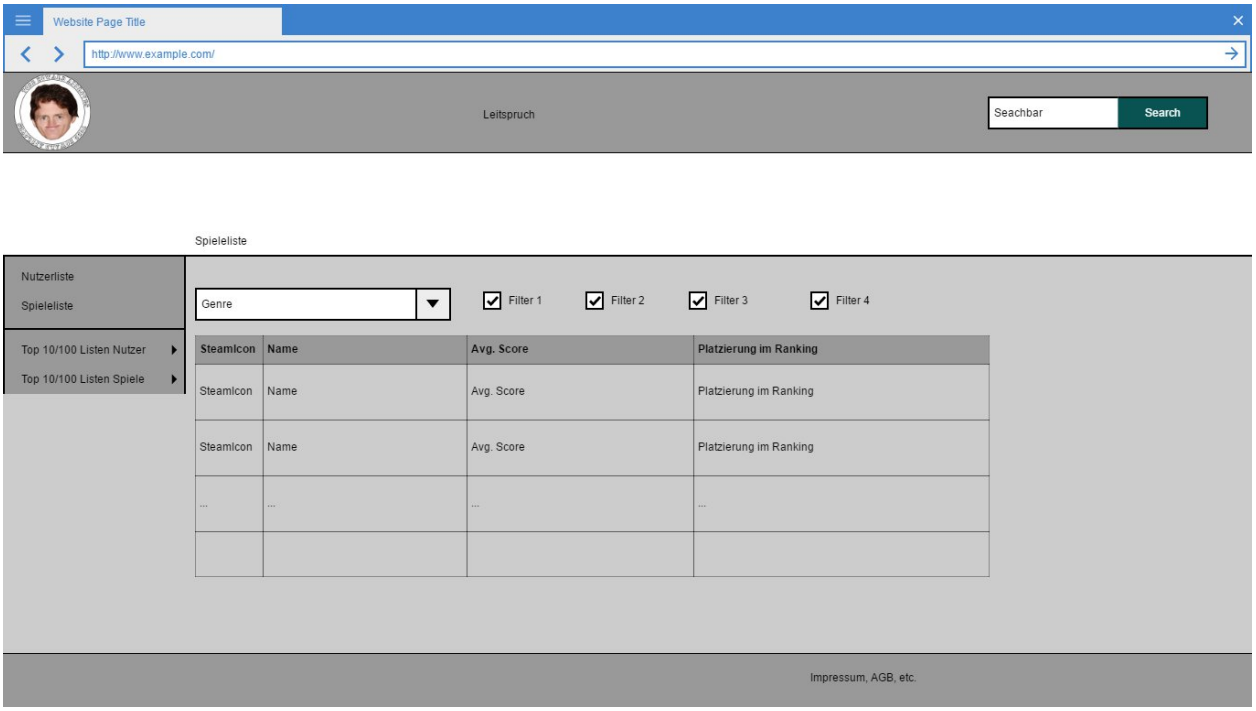


Abbildung A.4 - Spieleliste Wireframe

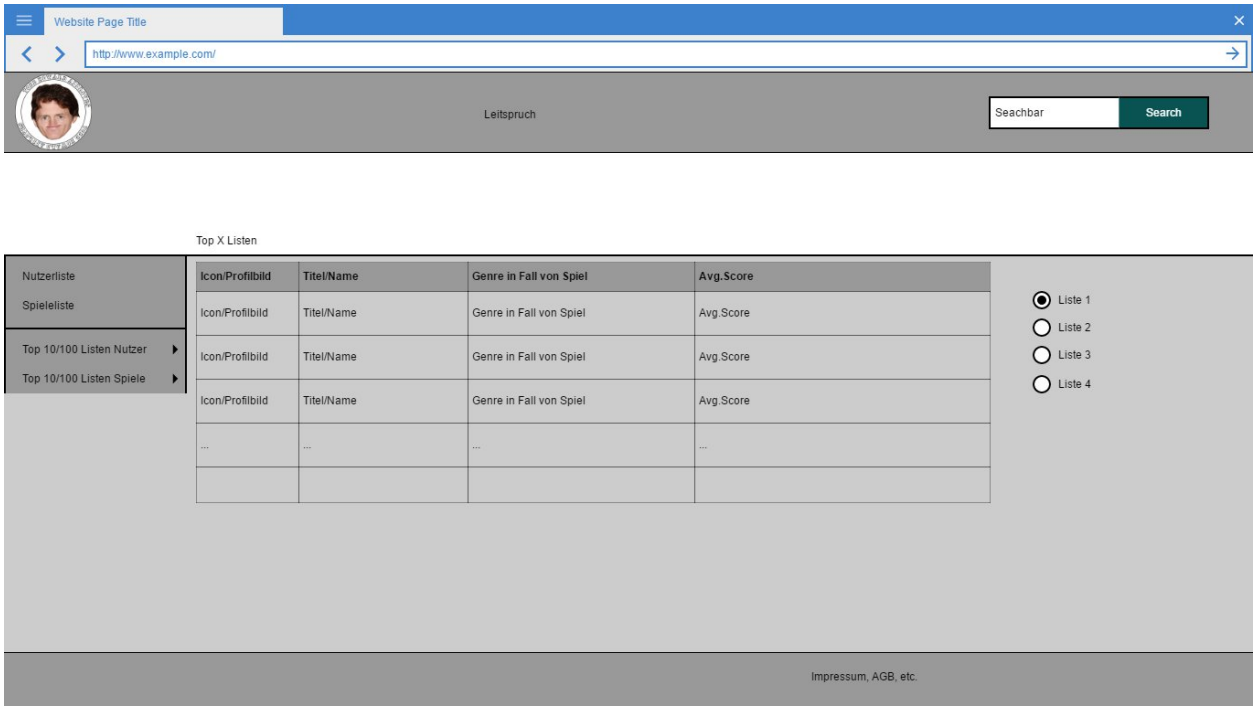


Abbildung A.5 - TopX Listen Wireframe

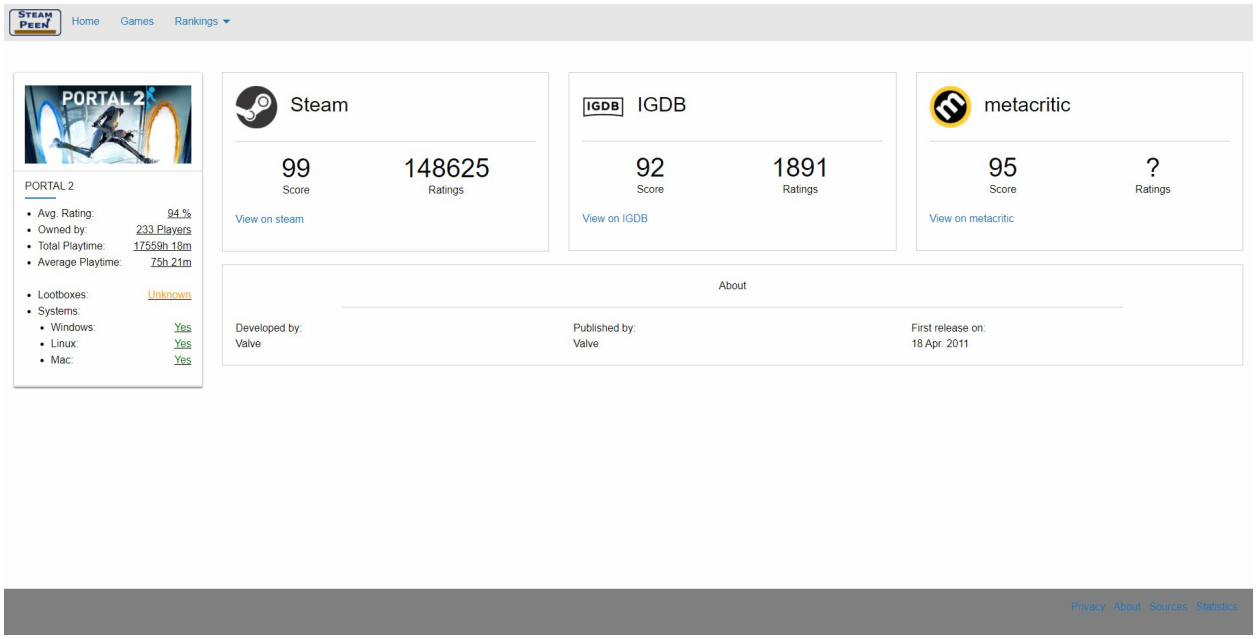


Abbildung A.6 - Spielansicht Fertiges Produkt, Stand 27.08.2020

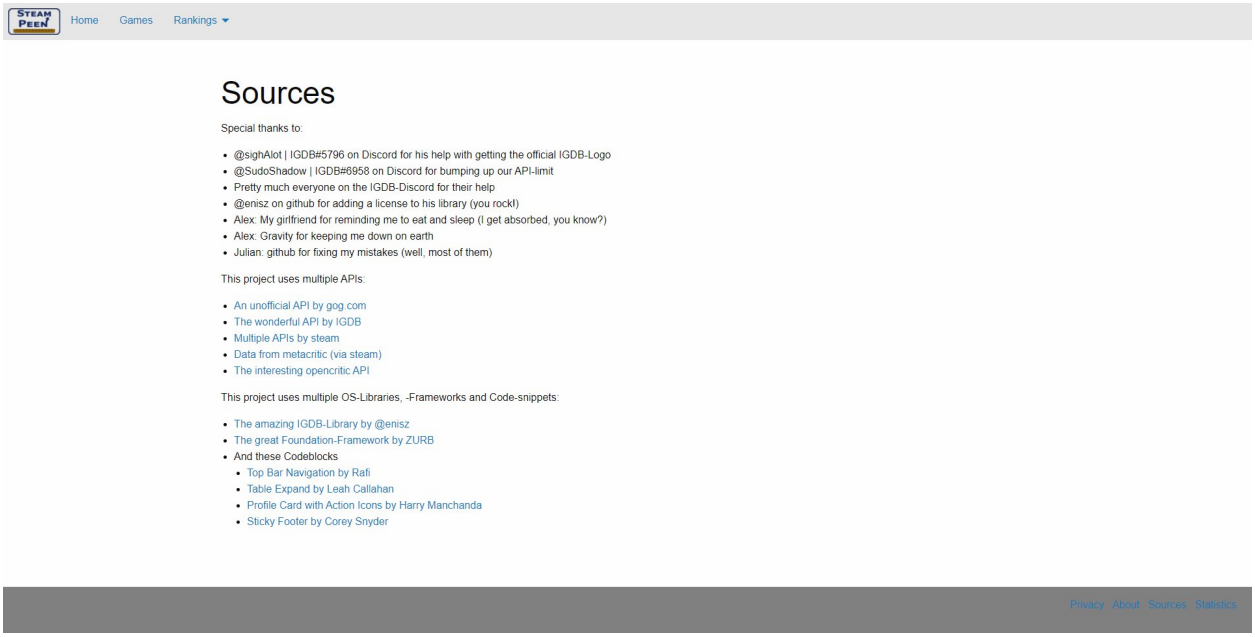


Abbildung A.7 - PureText Pages Fertiges Produkt, Stand 27.08.2020

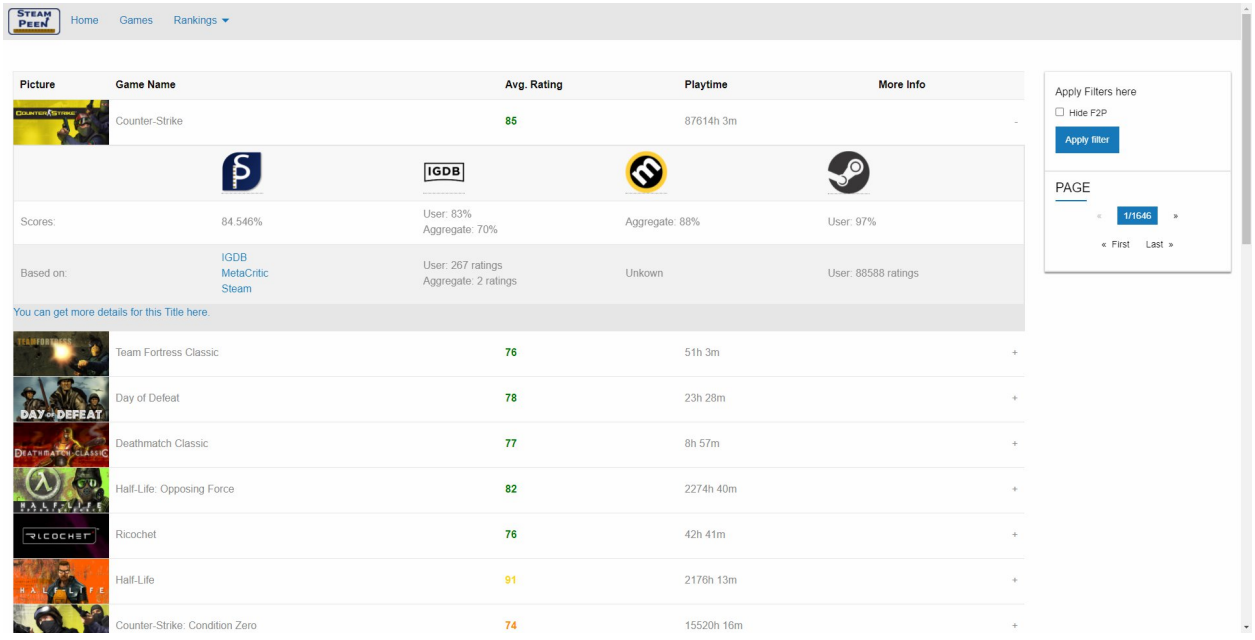


Abbildung A.8 - Spieleliste Fertiges Produkt, Stand 27.08.2020