SGDb: Semantic Video Game Database
Technical Report

Anastasia Chernysheva *a.chernysheva@oth-aw.de*
Jakob Götz *j.goetz2@oth-aw.de*
Ardian Imeraj *a.imeraj@oth-aw.de*
Patrice Korinth *p.korinth@oth-aw.de*
Philipp Stangl *p.stangl1@oth-aw.de*

***Zusammenfassung*—Dieser Technical Report beschreibt die Architektur von SGDb – eine webbasierte Anwendung mit einer Graphen-basierten Suche von Videospielen.**

I. Einleitung

Vor über einem halben Jahrhundert wurde das erste Videospiel von William Higinbotham veröffentlicht [1], bei dem zwei Spieler gegeneinander auf einem Oszilloskop spielen konnten. Das Spiel bestand aus einem einfachen Tennis-Spiel, bei dem ein Punkt erzielt wurde, wenn der Ball auf das andere Ende des Bildschirms gespielt wurde. Im Jahr 2022 waren es 2,95 Milliarden aktive Spieler, die täglich mehr als 10 Milliarden Stunden an Videospielen verbringen [2]. Die Anzahl der Videospieler als auch die Anzahl der Spiele steigt stetig an, weshalb es für einen Spieler immer schwieriger wird, sich in der Vielzahl an Spielen zurechtzufinden. Die Suche nach einem passenden Spiel kann sehr zeitaufwendig sein. Dazu soll eine Semantic Video Game Database (SGDb) mit einer Graphen-basierten Suche für Videospiele entwickelt werden.

In den weiteren Abschnitten des Technical Reports wird zuerst auf die Lösungsstrategie in Abschnitt II und die Datenbeschaffung in Abschnitt III eingegangen. Im nächsten Abschnitt IV wird das System aus Bausteinsicht beschrieben. Daran anschließend wird in Abschnitt V die Verteilungssicht der Anwendung beschrieben. In Abschnitt VI werden die angewandten Entwicklungswerkzeuge vorgestellt. Abschließend wird ein Fazit und Ausblick in Abschnitt VII gegeben.

II. Lösungsstrategie

Eine Gegenüberstellung der wichtigsten Ziele und Lösungsansätze für die Architektur befinden sich in der nachfolgenden Tabelle I. Die Abbildung 1 gibt einen Überblick über die Architektur.

Tabelle I
Qualitätsziele und Lösungsansätze

Qualitätsziel	Lösungsansatz
Intuitive Bedienung	Benutzeroberfläche mit Svelte
Separierung der Zuständigkeiten	„Backend for Frontend“-Muster
Verarbeitung großer Graphen	Graphen mit WebGL zeichnen

III. Datenbeschaffung

Die Daten für den Prototypen werden über den `/games` API-Endpunkt der IGDB-API [3] bezogen. Dieser liefert einen Datensatz zurück, der wesentliche Informationen zu einem Spiel beinhaltet. Zu den Informationen zählen: Spiele-ID, Spieletitel, Spieleplattform, Beschreibung, Veröffentlichungsdatum, Genre, Bewertung, Entwicklername- und Land sowie eine URL des Coverbildes.

Der Datensatz wird im JSON-Format gespeichert und ist auf 500 Spiele, die eine hohe Bewertung (>85) erzielt haben, beschränkt. Außerdem werden einige numerische Werte wie das Veröffentlichungsdatum, das im Unix-Zeitstempel[1] angegeben ist und der Standort des Entwicklerunternehmens als dreistelliger Country-Code[2] definiert. Im Skript `igdb.py` wurde eine Funktion `change_format` implementiert, welche die Unix-Zeit in ein reguläres Zeitformat `DD-MM-YYYY` und den ISO-Country-Code in Ländernamen konvertiert.

Der Datensatz wird anschließend in „Refine“ [6] geladen. Darin werden zunächst die Überschriften geändert, da diese fehlerhaft konvertiert sind und irrelevante Spalten werden entfernt. Außerdem werden die Bewertungen in Ganzzahlen umgewandelt. Da leere Zeilen im späteren Verlauf zu Fehlern führen, werden diese mit „`Nicht Vorhanden`“-Strings gefüllt. Im Anschluss werden die gereinigten Daten zu RDF-Triples strukturiert und in die Datenbank geladen. Ein RDF-Triple besteht aus einem Subjekt, einem Prädikat und einem Objekt.

IV. Bausteinsicht

Diese Sicht zeigt die statische Zerlegung des Systems in Bausteine sowie deren Beziehungen.

A. Datenbank

Zur Beschreibung der Videospiele werden Ontologien von „`schema.org`“ verwendet. Damit das Erscheinungsdatum auch als Datum interpretierbar ist, wird „`xsd:dateTime`“ aus „XML Schema“ [7] benutzt. Um Videospiele vom Typ „`schema:VideoGame`“ zu deklarieren, wird das RDF-Schema „`typeof`“ eingesetzt. Für die Verwaltung und Vernetzung dieser Informationen wird die „Ontotext GraphDB“ [8] verwendet. Die Triples werden in sogenannten *Repositories* gespeichert, die ähnlich wie Tabellen in relationalen Datenbanken organisiert sind. Der Unterschied zu einer relationalen Datenbank ist, dass Ontotext Informationen mithilfe von Klassifizierungen und Interpretationsregeln organisiert. Ein Repository kann als eine einzelne Ontotext-Datenbank angesehen

[1] Dieser ist auch als „The Epoch“ bekannt und zählt die Anzahl der vergangenen Sekunden seit 1.Januar 1970 [4].

[2] von der International Organization for Standardization (ISO) entwickelt und im ISO-3166 publiziert wurde [5].

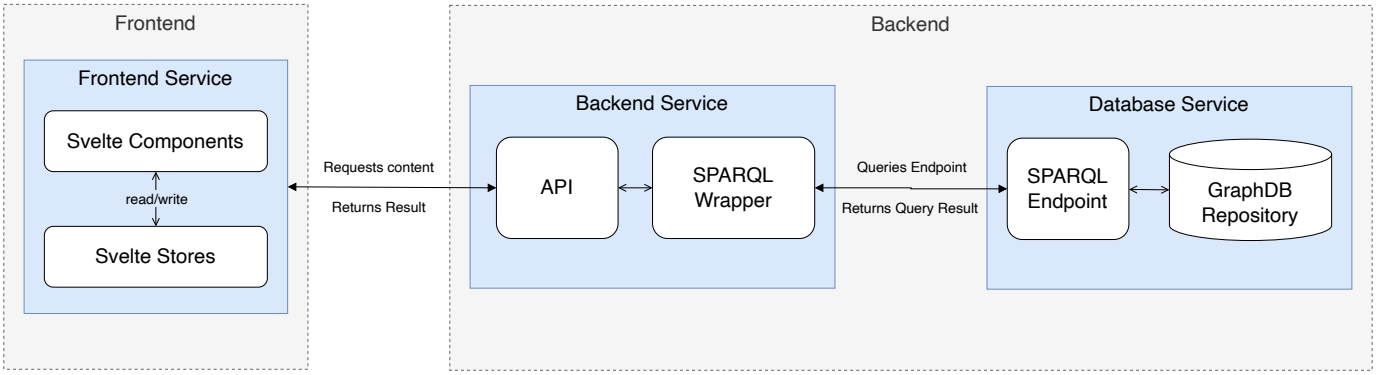

Abbildung 1. Überblick über die Architektur von SGDb. Die Architektur besteht aus drei Teilen: Das Frontend bietet dem Nutzer ein graphisches Dashboard (Abschnitt IV-C), das Backend stellt (Abschnitt IV-B) die API bereit und die Datenbank (Abschnitt IV-A) speichert persistent Daten.

werden, die für sich selbst fungiert. Somit können mehrere Repositories gleichzeitig existieren, diese sind jedoch von den anderen isoliert. Die Unterteilung der Daten in Klassen, Subklassen oder Instanzen sowie die Definition der Beziehung untereinander wird durch Ontologien festgelegt. Um die RDF-Daten im Repository abfragen zu können, stellt die Datenbank einen SPARQL-Endpunkt zur Verfügung.

B. Backend

Das Backend ist mit der Programmiersprache Python und dem Framework „FastAPI“ [9] realisiert. Es dient als Schnittstelle zwischen Datenbank und Frontend. Das Frontend kommuniziert mit dem Backend über eine RESTful-API, die mithilfe des FastAPI Frameworks implementiert ist. Zusätzlich ist es für die Verarbeitung von Ressourcen zuständig. Hierunter fällt die Aufgabe, Daten vom Frontend so aufzubereiten, dass diese für eine Datenbank-Abfrage genutzt werden können. Des Weiteren werden Ergebnisse aus einer Datenbank-Abfrage vom Backend aufbereitet und über die Endpunkt der API dem Frontend zur Verfügung gestellt. Das Rückgabeformat ist bei jedem API-Endpunkt JSON.

Folgende Endpunkte, in Tabelle II, stellt die API im Backend für das Frontend zur Verfügung:

Tabelle II
BACKEND-API FÜR DAS FRONTEND

Endpunkt	**Beschreibung**
`GET /`	Graphen für Startseite
`POST /`	Filter für den Graphen
`GET /search/{search}`	Suche eines Videospiels
`GET /detail/{game}`	Detailseite zu Videospiel

Die `Startpage`-Route gibt eine Menge von Schlüssel-Werte-Paaren zurück. Davon ist jeder Schlüssel ein Jahr, beginnend von 1985 bis 2022, und der Wert eine Liste von Videospielen, die in diesem Jahr erschienen sind. Die `Filter`-Route gibt einen aktualisierten Graphen, basierend auf den gewählten Filtern in der POST-Anfrage, zurück. Das Format und die Struktur des Graphen sind identisch mit demjenigen, der von der Startpage zur Verfügung gestellt wird. Die `Search`-Route verarbeitet eine Suchanfrage und gibt drei Listen zurück. Die erste Liste enthält alle Spieletitel, welche auf die gestellte Suchanfrage zutreffen. Die zweite Liste enthält alle Spieletitel, welche auf die gestellte Suchanfrage zutreffen und im aktuell angezeigten Graphen zu finden sind. Die dritte Liste enthält alle Spieletitel, welche auf die gestellte Suchanfrage zutreffen und nicht im aktuell angezeigten Graphen zu finden sind. Aufgrund der angewandten Einstellungen in der Filter Route kann es passieren, dass gesuchte Spiele nicht im angezeigten Graphen enthalten sind. Die `Detailpage`-Route gibt alle Details zu einem Videospiel, die in Abschnitt IV-C4 näher beschrieben werden, zurück.

Für den Zugriff auf die Datenbank wird ein Graphenobjekt definiert. Über das definierte Graphenobjekt können nun Abfragen direkt in der Python Datei vollzogen werden. Die Antwort der Abfragen wird noch weiter formatiert, sodass die benötigten Informationen aus dem Ergebnis leichter zugänglich sind.

Die Filter sind so gestaltet, dass sich über die Funktion `combine_filter` verschiedene Filteroptionen wie Datum, Genre, Bewertung, Entwickler und Plattform beliebig kombinieren lassen. Ist ein Filterargument gegeben, so wird für das jeweilige Argument eine zugehörige Abfrage erstellt, die Spiele zurückliefert, die dem Filter entsprechen. Anschließend werden alle Abfrage-Ergebnisse verglichen und nur diejenigen zurückgegeben, die auf alle Filtereinstellungen zutreffen.

Die Funktion `recommendations` nutzt die Spieledetails um Vorschläge für ähnliche Spiele zu generien. Jedoch werden nicht Spiele zurückgeliefert, die exakt den Kriterien entsprechen, sondern diejenigen, die am meisten Übereinstimmung zum Spiel aufweisen.

Um die Anzeige der Filteroptionen für Genre, Hersteller und Plattform des Filters im Frontend dynamisch zu gestalten, werden die Optionen jedes Mal beim Aufrufen der Startseite von der Datenbank abgefragt. Diese werden dann in der Backend API gezählt, um zusätzlich zu ermöglichen, dass die Menge der Spiele mit den jeweiligen Optionen einsehbar ist.

C. Frontend

Das Frontend ist für die Benutzerschnittstelle verantwortlich. Hauptbestandteile des Frontends sind die Suchmaske, der Graph und die Detailseite zu einem Videospiel. Die Interaktion mit dem Backend erfolgt über die REST-Schnittstelle.

1) Graph: Für die Darstellung des Graphen und die Interaktion mit diesem wird „Sigma.js“ [10] verwendet. Sigma.js rendert Graphen mit WebGL. Damit lassen sich größere Graphen schneller zeichnen als mit Canvas- oder SVG-basierten Lösungen. Das Graphenmodell wird in einer separaten Bibliothek namens „Graphology“ [11] verwaltet. Dies ist eine Standardbibliothek mit Algorithmen aus der Graphentheorie und allgemeinen Hilfsprogrammen wie z.B. Graphengeneratoren.

Für das Graphen Layout wird der ForceAtlas2 Algorithmus [12] verwendet. Vor dem Start von ForceAtlas 2 Layout muss die Startposition jedes Knotens festgelegt werden. Daher müssen zwei Attribute namens x und y für alle Knoten des Diagramms definiert werden. Dazu wird zuerst ein Graph Objekt erzeugt, das jeden Knoten zufällig positioniert, indem die Koordinaten gleichmäßig nach dem Zufallsprinzip auf dem Intervall $[0, 1)$ ausgewählt werden.

2) Suche: Es gibt zwei Arten von Videospiel-Suche: Die direkte Suche im Graphen und die Suche per Texteingabe. Dazu ist ein Eingabefeld für die textuelle Suche vorhanden, das bei einer Benutzereingabe basierend auf den eingegebenen Suchbegriff vom Backend die besten Suchtreffer abfrägt. Ist die Suche erfolgreich, wird der Knotenpunkte zum gesuchten Videospiel im Graphen hervorgehoben.

3) Filter: Die Filter werden in vier Kategoriern unterteilt: Genre, Spieleplattform, Spielentwickler und Erscheinungsjahr. Die ersten drei Kategorien werden durch Checkboxen dargestellt, um eine Mehrfachauswahl (z.B. mehrere Genres) zu ermöglichen. Die Filterinhalte werden jeweils mit der Gesamtanzahl der Vorkomnisse angezeigt. Das Erscheinungsjahr ist ein Schieberegler mit einem, auf die Spiele abgestimmten, Wertebereich. Die Filterselektion wurde auf fünf mögliche Filteritems begrenzt, da eine uneingeschränkte Auswahl an Filterkombinationen in den allermeisten Fällen zu keinem sinnvollen Ergebnis geführt hat. Für eine visuelle kategorische Differenzierung der Filterselektion verändert sich die Farbe der Filterbox und es wird ein *Tag* der gleichen Farbe generiert. Werden die Filtereinstellungen angewendet, erfolgt ein POST-Request an die `/`-Route.

4) Detailseite: Die Detailseite zeigt die spezifischen Informationen über ein bestimmtes Spiel an. Die Komponente führt asynchron eine HTTP-Anfrage an die REST-API aus, um Details zu einem Spiel abzurufen, wenn sie gerendert wird, und verwendet dann die empfangenen Daten, um die Seite mit Spielinformationen zu füllen. Der Endpunkt lautet „/detail/`game_name`“, wobei „`game_name`“ der Name des angeforderten Spiels ist. Die Daten werden anschließend in verschiedenen Elementen in der Vorlage der Komponente angezeigt. Die Komponente enthält auch einige HTML-Metatags, die Informationen über das Spiel enthalten, die für Social Media-Plattformen wie Twitter relevant sind. Die Metatags enthalten den Titel des Spiels, eine Beschreibung und ein Bild, das im Tweet angezeigt werden soll. Die Daten für diese Metatags werden ebenfalls von der REST-Schnittstelle abgerufen, wenn die Komponente gerendert wird. Schließlich enthält die Komponente ein Hauptelement, in dem die eigentlichen Spielinformationen angezeigt werden. Die Informationen umfassen ein Bild des Spiels, den Titel des Spiels und weitere Details wie die Plattformen, auf denen das Spiel verfügbar ist, und das Veröffentlichungsdatum.

V. Verteilungssicht

Dieser Abschnitt beschreibt den Betrieb von SGDb. Zentraler Bestandteil der Verteilungsstruktur sind Docker-Container [13] (Abbildung 2). Jeder Baustein von SGDb ist für sich isoliert in einem eigenen Docker-Container untergebracht. Damit das gesamte System mit allen verteilten Komponenten in der korrekten Reihenfolge gestartet wird, werden die Docker- Container mit Docker Compose orchestriert. Docker Compose übernimmt die Netzwerkkonfiguration und die Vergabe von Host-Namen an die jeweiligen Docker-Container. Darüber hinaus können Umgebungsvariablen verwaltet werden.

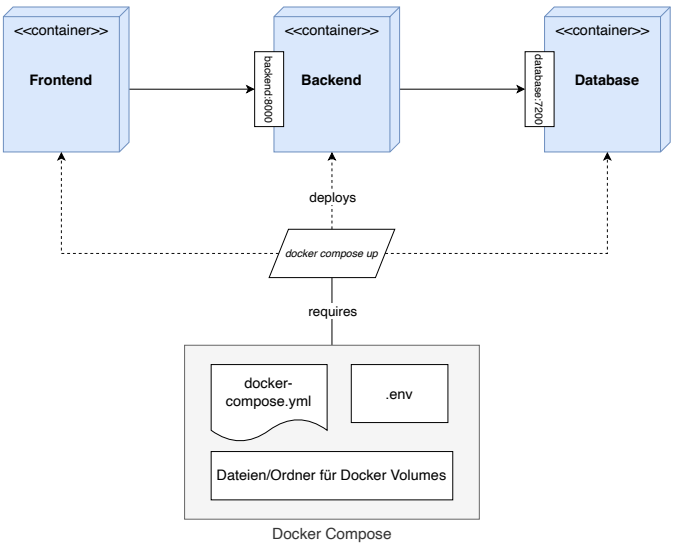

Abbildung 2. Verteilungssicht

VI. Entwicklungswerkzeuge

Dieser Abschnitt geht auf die verwendeten Entwicklungswerkzeuge im Backend (Abschnitt VI-A) und Frontend (Abschnitt VI-B) ein.

A. Backend

Der Hauptbestandteil des Backends wird mit dem Framework FastAPI umgesetzt, das auf den Frameworks „Starlette“ [14] und „Pydantic“ [15] basiert. Starlette bietet Funktionen wie WebSockets und Pydantic ermöglicht die Validierung von Datenmodellen, die von FastAPI genutzt werden. Durch Starlette kann ebenso das Test-Framework „Pytest“ [16] für Unit-Tests direkt genutzt werden. Für die Bereitstellung wird der ASGI (Asynchronous Server Gateway Interface) Server

„Uvicorn“ [17] verwendet. Zur Interaktion mit der Datenbank (RDF Triplestore) wird die Python-Bibliothek „SPARQLWrapper“ [18] genutzt. Diese dient als Python-Wrapper für SPARQL und ermöglicht so die Ausführung von Abfragen in der SPARQL-Syntax. Dazu bietet es die Möglichkeit, das Ergebnis in das gewünschte Format zu formatieren, sodass diese Daten leichter in Python weiterverarbeitet werden können.

B. Frontend

Das Frontend wird unter Zuhilfenahme des Frontend-Frameworks „Svelte“ [19] realisiert. Die Verwaltung der Abhängigkeiten erfolgt mit „npm“ [20]. Als Build-Tool ist „Vite“ [21] dafür zuständig, die Frontend Anwendung aus dem Quellcode zu erstellen. Für die Entwicklung wird ein Vite-Dev-Server (mit Reload-Funktionalität) zum Bereitstellen der Anwendung verwendet. Für die Gestaltung der Benutzeroberfläche wird „Tailwind CSS“ [22] verwendet. Im Gegensatz zu anderen CSS-Framework, die eine Vielzahl von vordefinierten Komponenten bereitstellen, bietet Tailwind CSS eine Vielzahl von niedrigschwelligen Gestaltungsoptionen, die es Entwicklern ermöglichen, genau die benutzerdefinierten Designs zu erstellen, die sie benötigen. Im Frontend wird das Vite-native Test-Framework „Vitest“ [23] in Kombination mit der „Svelte Testing Library“ [24] und „c8“ [25] für die Testabdeckung verwendet.

VII. Fazit und Ausblick

Im Rahmen der Arbeit wurde mit einem begrenzten Datensatz von 500 Spielen eine erste Version der Anwendung entwickelt, welche die Anforderungen des Fachkonzepts erfüllt. Ein Benutzer kann per Graph oder per Texteingabe suchen, die Suchergebnisse filtern und Details zu einem Spiel einsehen. Die Anwendung ist darauf ausgelegt, auch ohne Komplikationen mit einem größeren Datensatz zu arbeiten.

Literatur

[1] W. Higinbotham. „Tennis for Two.“ Accessed: 2021-09-01. (1958).

[2] N. Gilbert. „Number of Gamers Worldwide 2022/2023.“ Zugegriffen am: 05.01.2023. (2022), Adresse: https://financesonline.com/number-of-gamers-worldwide/.

[3] IGDB. „IGDB-API.“ [Online]. (2022), Adresse: https://api-docs.igdb.com.

[4] Datumsformat. „UNIX-Zeitstempel konvertieren.“ [Online]. (2022), Adresse: https://www.datumsformat.de/unix/.

[5] I. Standarts. „ISO 3166 Country Codes.“ [Online]. (2022), Adresse: https://www.iso.org/iso-3166-country-codes.html.

[6] Ontotext. „Ontotext Refine.“ [Online]. (2022), Adresse: https://www.ontotext.com/products/ontotext-refine/.

[7] Datypic. „XSD DateTime.“ [Online]. (2023), Adresse: http://www.datypic.com/sc/xsd/t-xsd_dateTime.html.

[8] Ontotext. „Ontotext GraphDB.“ [Online]. (2022), Adresse: https://www.ontotext.com/products/graphdb/.

[9] S. Ramírez. „FastAPI Documentation.“ [Online]. (2022), Adresse: https://fastapi.tiangolo.com/.

[10] Sigma. „Sigma.js.“ [Online]. (2022), Adresse: https://www.sigmajs.org/.

[11] Graphology. „Graphology.“ [Online]. (2022), Adresse: https://graphology.github.io/.

[12] M. Jacomy, T. Venturini, S. Heymann und M. Bastian, „ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software,“ *PloS one*, Jg. 9, Nr. 6, e98679, 2014.

[13] Docker. „Docker.“ [Online]. (2022), Adresse: https://www.docker.com/.

[14] „Starlette.“ [Online]. (2022), Adresse: https://www.starlette.io/.

[15] „Pydantic Documentation.“ [Online]. (2022), Adresse: https://docs.pydantic.dev/.

[16] K. et al. „Pytest Documentation.“ [Online]. (2022), Adresse: https://pytest.org/.

[17] „Uvicorn Documentation.“ [Online]. (2022), Adresse: https://www.uvicorn.org/.

[18] I. H. E. al. „SPARQLWrapper Documentation.“ [Online]. (2022), Adresse: https://sparqlwrapper.readthedocs.io/en/latest/.

[19] Svelte. „Svelte: Cybernetically Enhanced Web Apps.“ [Online]. (2023), Adresse: https://svelte.dev/.

[20] npm. „npm.“ [Online]. (2023), Adresse: https://www.npmjs.com/.

[21] E. You. „Vite: Next Generation Frontend Tooling.“ [Online]. (2023), Adresse: https://github.com/vitejs/vite.

[22] Tailwind Labs. „Tailwind CSS.“ [Online]. (2023), Adresse: https://tailwindcss.com/.

[23] Vitest. „Vitest: A Vite-native unit test framework.“ [Online]. (2022), Adresse: https://vitest.dev/.

[24] Testing Library. „Svelte Testing Library.“ [Online]. (2022), Adresse: https://github.com/testing-library/svelte-testing-library.

[25] B. Coe. „C8: Output coverage reports using Node.js built in coverage.“ [Online]. (2023), Adresse: https://github.com/bcoe/c8.