

# SGDb: Semantic Video Game Database

## Technical Report

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

Vor über einem halben Jahrhundert wurde das erste Videospiel von William Higinbotham veröffentlicht **tennis\_for\_two**, 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 **gamers**. 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, da die Suche nach einem passenden Spiel über verschiedene Plattformen erfolgen muss. Dazu soll eine Semantic Video Game Database (SGDb) entwickelt werden, die eine Suche nach Videospielen ermöglicht. Die Anwendung soll eine Graphen-basierte Suche von Videospielen ermöglichen, die auf einer semantischen Datenbank basiert. In den weiteren Abschnitten des Technical Reports wird zuerst auf die Lösungsstrategie in Abschnitt II eingegangen. Im nächsten Abschnitt IV wird das Gesamtsystem aus Bausteinsicht beschrieben. Anschließend wird in Abschnitt V die Verteilungssicht der Anwendung beschrieben. In Abschnitt VI werden die angewandten Werkzeuge zur Entwicklung der Anwendung vorgestellt. Abschließend wird ein Fazit und Ausblick in Abschnitt VII gegeben.

### II. LÖSUNGSSTRATEGIE

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

Tabelle I  
LÖSUNGSSTRATEGIE

Qualitätsziel	Lösungsansatz
Verarbeitung großer Graphen	Graphen mit WebGL rendern
Separierung der Zuständigkeiten	Backend for Frontend Pattern

### III. DATENBESCHAFFUNG

Für die Datenbeschaffung wird eine Client-Anfrage an die Endpunkte der IGDB-API **igdb-api** gesendet. Als Antwort

wird im Anschluss der Bearbeitung ein Datensatz zurückgegeben, der wesentliche Informationen zu einem Spiel beinhaltet. Zu den Informationen zählen: Spiele-ID, Spieletitel, Beschreibung, Veröffentlichungsdatum, Genre, Spieleplattform, Bewertung, Entwicklername- und Land sowie eine URL des Coverbildes.

Der Datensatz wird im JSON-Format gespeichert und auf 500 Spiele, die eine hohe Bewertung erzielt haben (>85), reduziert. Begründet wird diese Bedingung durch das Bestreben, einen Datensatz zu erzeugen, der über viele Informationen zu den einzelnen Spielen verfügt. Der Datensatz weist an mancher Stelle numerische Werte auf, die jedoch nicht informativ wären für einen Durchschnittsnutzer. So ist das Veröffentlichungsdatum im Unix-Zeitstempel angegeben. Dieser ist auch als „The Epoch“ bekannt und zählt die Anzahl der vergangenen Sekunden seit 1. Januar 1970 **unix**. Der Standort des Entwicklerunternehmens wird als dreistelliger Country-Code definiert, der von der International Organization for Standardization (ISO) entwickelt und im ISO-3166 publiziert wurde **iso**. Es wurde eine Funktion implementiert, die die Unix-Zeit in ein reguläres Zeitformat DD-MM-YYYY und den ISO-Country-Code in Ländernamen konvertiert.

Der Datensatz wird anschließend in OpenRefine **refine** geladen.

### IV. BAUSTEINSICHT

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

#### A. Datenbank

In OpenRefine werden zunächst die Überschriften geändert, da diese fehlerhaft konvertiert sind und irrelevante Spalten werden entfernt. Die Bewertungen werden in Integer umgewandelt, für eine simple Darstellung im Frontend. 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. Diese bestehen aus einem Subjekt, einem Prädikat und einem Objekt. Es werden Ontologien von „**schema.org**“ für die Definition der Spieleinformationen verwendet, da diese Ontologie die geeigneten Beschreibungen für ein Spiel beinhaltet. Damit das Erscheinungsdatum auch als Datum interpretierbar ist, wird „**xsd:dateTime**“ der XMLSchema benutzt. Um die Spiele zum Beginn vom Typ „**schema:VideoGame**“ zu deklarieren, wird das RDF-Schema „**typeof**“ eingesetzt. Für die Verwaltung und Vernetzung dieser Informationen

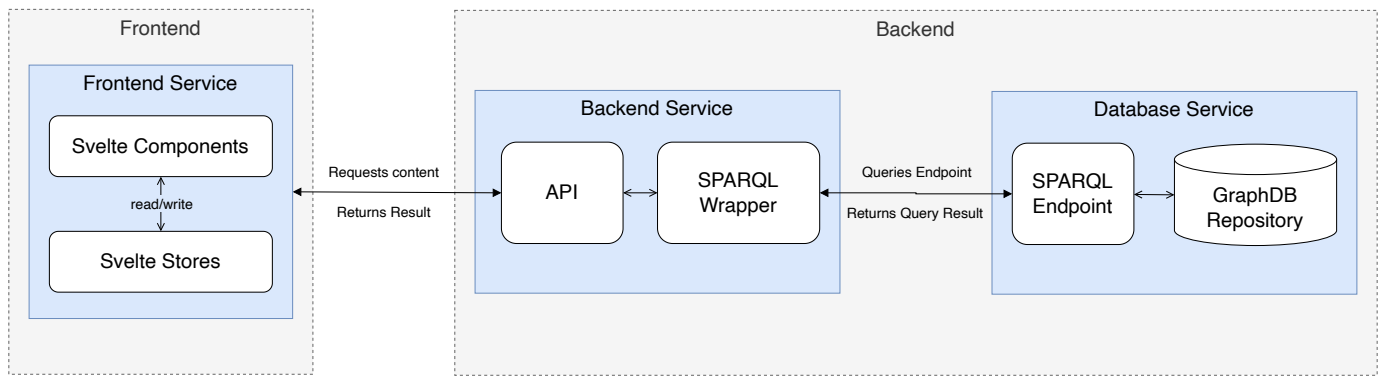


Abbildung 1. Überblick über die Architektur von SGDB. Die Architektur besteht aus drei Teilen: Das Frontend bietet dem Nutzer ein graphisches Dashboard (Abschnitt IV-C), das Backend stellt (Abschnitt IV-B) die API bereit und die Datenbank (Abschnitt IV-A) speichert persistent Daten.

wird die Ontotext GraphDB **graphdb** 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 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 mit RDF-Triples zu arbeiten, verwendet Ontotext GraphDB das RDF4J-Framework sowie die SAIL-API. Um mit den RDF-Daten interagieren zu können, unterstützt Ontotext die graphbasierte Abfragesprache SPARQL. Das webbasierte Verwaltungstool von Ontotext ist leicht bedienbar und ermöglicht den Import und Export der Daten in verschiedenen Formaten, das Clustern/Bearbeiten/Anzeigen des RDF-Graphen, Erstellung neues Repositories, Ausführen der SPARQL-Abfragen u.v.m.

## B. Backend

Das Backend ist mit der Programmiersprache Python und dem Framework „FastAPI“ **fastapi** realisiert. Es dient als Schnittstelle zwischen Datenbank und Frontend. 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 einen Endpunkt dem Frontend zur Verfügung gestellt. Das Format der Rückgaben ist bei jedem Endpunkt JSON. Das Frontend kommuniziert mit dem Backend über eine RESTful-API, die mithilfe des Frameworks implementiert ist.

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

Die Startpage Route gibt als key die Jahre beginnend von 1985 bis 2022 zurück. Jeder key enthält als value die Titel, welche in diesem Jahr veröffentlicht wurden. Die Filter Route gibt, nachdem sie Daten im JSON Format mit den entsprechen-

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}	Detaillseite zu Videospiel

den Filtern erhalten hat, einen aktualisierten Graphen zurück, der die angegebenen Filter berücksichtigt. 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. Denn durch die Einstellungen der Filter Route kann es passieren, dass gesuchte Spiele nicht im angezeigten Graphen enthalten sind. Die Detailpage Route gibt alle in der Datenbank gespeicherten Details zu einem Spiel zurück.

Zur Interaktion mit der Datenbank (RDF Triplestore) wird die Python-Bibliothek „SPARQLWrapper“ **sparqlwrapper** genutzt. Diese dient als Python-Wrapper für SPARQL und ermöglicht so die Ausführung von Queries 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. SPARQL selbst ist eine graphenbasierte Query-Sprache, die mit Daten im RDF-Format arbeitet.

Zuerst wird ein Graphenobjekt definiert, über welches der Zugriff auf die Datenbank realisiert wird. Der Rückgabebetyp ist auf das JSON-Format festgelegt. Über das definierte Graphenobjekt können nun Abfragen direkt in der Python Datei vollzogen werden. Die Antwort der Abfragen ist eine geschachtelte JSON-Ausgabe, sodass diese noch weiter formatiert wird.

Die Funktion der Filter ist so gestaltet, dass eine Hauptklas-

se verschiedene Filteroptionen wie Datum, Genre, Bewertung, Entwickler und Plattform verwaltet. Diese lassen sich beliebig kombinieren. Ist ein Filterargument gegeben, so wird für das jeweilige Argument eine zugehörige Query erstellt, die die Spiele zurückliefert, die dem Filter entsprechen. Anschließend werden alle Query-Ergebnisse verglichen und nur diejenigen zurückgegeben, die auf alle Filtereinstellungen zutreffen.

Des Weiteren behandelt die Filter-Hauptklasse die Funktion, für ein gegebenes Spiel, passende Spiele-Empfehlungen zurückzugeben. Dabei werden ebenfalls die verschiedenen Filterqueries verwendet, die die zugehörigen Argumente des Spiels bekommen. 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) *Graphen Layout*: Für die Darstellung des Graphen und die Interaktion mit diesem wird „Sigma.js“ **sigma** 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“ **graphology** 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 **forceatlas2** 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*:

3) *Filter*: Die Filter werden in vier Kategorien unterteilt: Genre, Spieleplattform, Spielentwickler und Erscheinungsjahr. Die ersten drei Kategorien werden im Frontend als Checkboxinputs ausgegeben, um eine Mehrfachauswahl zu ermöglichen. Sowohl die Filterinhalte als auch die Gesamtanzahl dieser Inhalte werden aus der Datenbank bezogen. Das Erscheinungsjahr ist ein Sliderinput mit einem, auf die Spiele abgestimmten, Wertebereich. Die Filterselektion wurde auf 5 mögliche Filteritems begrenzt, da eine uneingeschränkte Auswahl an Kombinationen in den allermeisten Fällen zu einem 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. Nach der Auswahl der Filter, werden diese mit POST-Request an das Backend übermittelt.

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 **docker** (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.

## 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-**starlette**“ und „Pydantic-**pydantic**“ 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“ **pytest** für Unit-Tests direkt genutzt werden. Als ASGI Server wird Uvicorn genutzt, mit welchen die Anwendung gestartet wird. Als ASGI-Server (Asynchronous Server Gateway Interface) wird Uvicorn-**uvicorn** verwendet, um die Anwendung zu starten. Uvicorn ist ein schneller und leistungsfähiger ASGI-Server, der perfekt für die Verwendung mit FastAPI geeignet ist.

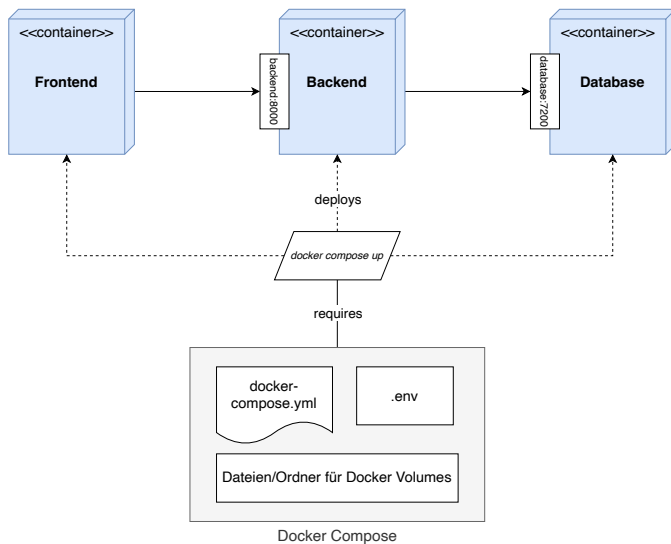


Abbildung 2. Verteilungssicht

### B. Frontend

Das Frontend wird unter Zuhilfenahme des Frontend-Frameworks „Svelte“ **svelte** realisiert. Die Verwaltung der Abhängigkeiten erfolgt mit „npm“ **npm** für auf „Node.js“ basierende Bausteine. Im Frontend ist Vite dafür zuständig, die Anwendung aus dem Quellcode zu erstellen. Dabei gibt es zwei Varianten: Für Entwicklungszwecke wird ein Vite-Dev-Server (mit Reload-Funktionalität) zum Bereitstellen der Anwendung verwendet. Für den Produktiveinsatz werden nur die benötigten Zieldateien unter Verwendung des `Static adapter` erstellt, die dann mit einer beliebigen Server-Software ausgeliefert werden können. Für die Gestaltung der Benutzeroberfläche wird „Tailwind CSS“ **tailwindcss** verwendet. Im Gegensatz zu anderen CSS-Framework, die eine Vielzahl von vordefinierten Komponenten bereitstellen, bietet Tailwind CSS eine Vielzahl von niedrighschwelligen Gestaltungsoptionen, die es Entwicklern ermöglichen, genau die benutzerdefinierten Designs zu erstellen, die sie benötigen. Im Frontend wird das Vite-native Test-Framework „Vitest“ **vitest** in Kombination mit der „Svelte Testing Library“ **stl** und „c8“ **c8** 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.