Praxisprojekt an der TH Köln

im Studiengang Allgemein Informatik
an der Fakultät für Informatik und Ingenieurwissenschaften
der Technischen Hochschule Köln

vorgelegt von:	Mazen Attia
Matrikel-Nr.:	11153552
Adresse:	Auf der Platte 1, 51643 Gummersbach
	mazen.Attia@smail.th-koeln.de

eingereicht bei: M.Sc. Alexander Dobrynin

Gummersbach, 13.01.2025

Kurzfassung/Abstract

Die vorliegende Arbeit befasst sich mit der Analyse und dem Vergleich von Modulstrukturen aus den Jahren 2024, 2023 und 2022, die aus der Commit-Historie der entsprechenden Repositories auf GitLab extrahiert wurden. Ziel der Arbeit ist es, Unterschiede in der Struktur und den Inhalten von Modulen über die Jahre hinweg zu identifizieren und diese Unterschiede in einer benutzerfreundlichen Frontend-Oberfläche darzustellen. Dabei können Nutzer gezielt nach einem Studiengang, einem Jahr sowie einem Semester (Winter/Sommer) filtern, um spezifische Module zu vergleichen.

Um die Analyse zu ermöglichen, wurden alle relevanten Daten aus GitLab abgerufen und in einer SQLite-Datenbank gespeichert. Diese Datenbank ist nach Jahr und Semester strukturiert, um eine klare Trennung und leichtere Vergleichbarkeit zu gewährleisten. Während die Module aus dem Jahr 2024 eine neue, konsistente Struktur aufweisen, unterscheiden sich die Dateistrukturen der Jahre 2023 und 2022 erheblich, was die Vergleichbarkeit erschwerte und zusätzliche Herausforderungen bei der Datenverarbeitung und Darstellung im Frontend mit sich brachte.

Im Ergebnis bietet die entwickelte Anwendung eine intuitive Oberfläche, in der Nutzer Module aus verschiedenen Jahren miteinander vergleichen können. Unterschiede in den Dateistrukturen, Inhalten oder Commit-Historien werden transparent dargestellt, um Studierenden und Lehrkräften eine detaillierte Analyse der Modulentwicklung zu ermöglichen.

Inhalt

1. Einleitung

1.1. Einführung in das Thema

Modulbeschreibungen sind eine zentrale Ressource für die Organisation und Dokumentation von Lehrveranstaltungen. Sie dienen als Grundlage für die Planung, Durchführung und Evaluierung von Modulen. In unserem Projekt steht die Entwicklung eines Tools im Vordergrund, mit dem sich vergangene Modulbeschreibungen, die in einem Git-Repository gespeichert sind, effizient nachschlagen und rekonstruieren lassen.

Dies ist besonders relevant, da die Struktur und Organisation dieser Modulbeschreibungen über die Jahre hinweg stark variieren kann. Das Tool soll eine standardisierte, benutzerfreundliche Lösung bieten, die unabhängig von den jeweiligen Dateipfaden oder Ordnerstrukturen arbeitet. Dadurch wird der Zugriff auf die Commit-Historie sowie die Modulbeschreibungen selbst erheblich erleichtert.

1.2. Ziele und Relevanz

Das Tool soll:

- **Effizienz schaffen**, indem es die Zeit für das Nachschlagen von Modulen erheblich reduziert.
- **Flexibilität bieten**, um Daten unabhängig von ihrer ursprünglichen Struktur zugänglich zu machen.
- **Nachvollziehbarkeit gewährleisten**, indem es Commit-Historien und Änderungen transparent darstellt.
- **Skalierbarkeit ermöglichen**, damit es auch für zukünftige Module und Jahre einsetzbar ist.

Die Entwicklung eines solchen Tools ist nicht nur für die Nachverfolgbarkeit von Modulbeschreibungen essenziell, sondern unterstützt auch die Qualitätssicherung und Dokumentation von Lehrmaterialien.

1.3. Herausforderungen

Die Hauptprobleme bei diesem Projekt liegen in folgenden Punkten:

1. **Inkonsistente Strukturen:** Unterschiedliche Ordner- und Dateistrukturen über die Jahre hinweg.
2. **Datenmenge:** Umfangreiche Repositories mit einer Vielzahl an Modulen und Commits.
3. **Rekonstruktion:** Sicherstellen, dass alte Modulversionen korrekt rekonstruiert werden können.
4. **Benutzerfreundlichkeit:** Entwicklung eines Tools, das sowohl leistungsfähig als auch einfach zu bedienen ist.

2. Problemstellung

In diesem Kapitel finden Sie grundlegende Hinweise zum formale Aufbau Ihrer Arbeit.

2.1 Inkonsistente Strukturen

Eine der zentralen Herausforderungen bei der Arbeit mit den Git-Repositories war die stark variierende Struktur der Dateien und Ordner über die Jahre hinweg. Im Jahr 2022 folgte die Organisation der Modulbeschreibungen einer klaren und einheitlichen Hierarchie. Die Dateien waren in spezifischen Ordnern abgelegt, was die Navigation und den Zugriff erleichterte. Diese Ordnung änderte sich jedoch im Jahr 2023 erheblich. Einige Modulbeschreibungen wurden außerhalb der definierten Ordnerstruktur gespeichert, und neue Ordner wurden hinzugefügt, die nicht der bisherigen Logik folgten. Im Jahr 2024 wurde die Struktur erneut überarbeitet, was zu einer weiteren Heterogenität führte. Diese ständigen Änderungen führten dazu, dass eine einfache Zuordnung von Dateien zu bestimmten Modulen nicht mehr möglich war. Darüber hinaus machte die wachsende Komplexität der Ordnerhierarchien automatisierte Prozesse zunehmend unzuverlässig.

2.2 Schwierigkeiten bei der Zuordnung

Ein weiteres signifikantes Problem war die fehlende Eindeutigkeit bei der Zuordnung von Dateien zu spezifischen Modulen. Modulbeschreibungen, die in einem Jahr unter einem klaren Namen oder Dateipfad abgelegt waren, wurden in den Folgejahren an völlig anderen Orten gespeichert oder unter neuen Namen archiviert. Diese Abweichungen führten nicht nur zu Verwirrung, sondern auch zu einer erhöhten Fehleranfälligkeit bei der manuellen Suche nach den relevanten Dateien. Besonders problematisch war dies bei Modulen, deren Beschreibungen aufgeteilt oder in unterschiedlichen Ordnern fragmentiert wurden. Solche Inkonsistenzen erschwerten die Rückverfolgung von Änderungen an einem Modul und führten zu einer deutlich verlängerten Bearbeitungszeit.

2.3 Zeitintensiver Zugriff

Die manuelle Suche nach bestimmten Modulbeschreibungen in den Git-Repositories gestaltete sich äußerst zeitaufwendig, insbesondere bei großen Projekten mit umfangreicher Commit-Historie. Jede Datei musste potenziell durch mehrere Verzeichnisse und Unterordner durchsucht werden, um ihren genauen Speicherort und die zugehörigen Metadaten zu identifizieren. Zusätzlich mussten die verschiedenen Versionen eines Moduls analysiert werden, um die relevanten Änderungen zu erfassen. Diese manuellen Prozesse dauerten nicht nur lange, sondern waren auch fehleranfällig, insbesondere wenn mehrere Personen parallel an der Suche und Analyse arbeiteten. Die fehlende Möglichkeit, die Commit-Historie effizient zu durchsuchen und spezifische Module automatisiert zu finden, führte zu einem erheblichen Mehraufwand.

2.4 Probleme bei der Analyse

Die Inkonsistenz der Datenstrukturen stellte auch eine erhebliche Hürde für die Analyse der Modulbeschreibungen dar. Ohne eine zentrale, standardisierte Struktur war es nahezu unmöglich, Änderungen an einem Modul über mehrere Jahre hinweg systematisch zu verfolgen. So mussten die Commit-Historien manuell durchsucht werden, um Unterschiede zwischen den Versionen zu identifizieren. Hinzu kam, dass Metadaten wie Autoreninformationen, Commit-Beschreibungen oder Zeitstempel oft nicht vollständig dokumentiert oder schwer auffindbar waren. Diese fehlende Standardisierung beeinträchtigte die Fähigkeit, Muster und Trends in den Modulbeschreibungen zu erkennen oder vergangene Entscheidungen nachvollziehbar zu machen. Die Analyse wurde dadurch nicht nur aufwändig, sondern auch weniger zuverlässig.

2.5 Herausforderung bei der Benutzerführung

Die Benutzerführung stellt einen wesentlichen Aspekt bei der Entwicklung des Tools dar, da sie direkt die Effizienz und Zufriedenheit der Nutzer:innen beeinflusst. Im Rahmen dieses Projekts traten verschiedene Herausforderungen auf, die eine sorgfältige Planung und Umsetzung erforderten, um die Benutzererfahrung zu optimieren.

2.5.1 Komplexität der Filteroptionen

Eine der zentralen Herausforderungen war die Komplexität der Filteroptionen. Nutzer:innen sollten in der Lage sein, verschiedene Kriterien wie Jahr, Kurs und Semester zu kombinieren, um spezifische Modulbeschreibungen zu finden. Eine unklare Struktur der Filteroptionen oder ungenaue Beschriftungen hätten jedoch zu Missverständnissen führen können, was die Nutzerfreundlichkeit erheblich beeinträchtigt hätte.
Beispielsweise könnten verschachtelte Dropdown-Menüs oder fehlende Standardwerte die Nutzung erschweren, insbesondere für Nutzer:innen mit wenig technischer Erfahrung.

3. Zielsetzung

3.1 Projektziele

Das zentrale Ziel dieses Projekts ist die Entwicklung eines Tools, das den Zugriff auf vergangene Modulbeschreibungen vereinfacht und gleichzeitig eine systematische, skalierbare und zuverlässige Lösung bietet. Dabei soll das Tool unabhängig von den zugrunde liegenden, variierenden Strukturen der Git-Repositories arbeiten. Es soll sicherstellen, dass Modulbeschreibungen aus verschiedenen Jahren effizient abgerufen werden können, unabhängig davon, wie diese Dateien in den jeweiligen Ordnerhierarchien organisiert sind. Ein weiteres Ziel ist es, die Commit-Historie von

Modulen transparent darzustellen, um Änderungen und Entwicklungen nachvollziehbar zu machen.

Zudem soll das Tool nicht nur den aktuellen Anforderungen genügen, sondern auch zukunftssicher gestaltet werden. Es muss in der Lage sein, mit wachsenden Datenmengen und neuen Strukturänderungen umzugehen, ohne dass die Kernfunktionen beeinträchtigt werden. Die Bereitstellung einer benutzerfreundlichen Oberfläche ist ebenfalls ein Hauptziel, um sicherzustellen, dass das Tool von Personen mit unterschiedlichen technischen Hintergründen genutzt werden kann.

3.2 Anforderungen an die Lösung

Um die Projektziele zu erreichen, muss das Tool mehrere spezifische Anforderungen erfüllen:

3.2.1 Modulübergreifender Zugriff:

Das Tool muss in der Lage sein, Modulbeschreibungen aus verschiedenen Jahren und mit unterschiedlichen Dateistrukturen zentral abzurufen. Dies erfordert eine standardisierte Datenstruktur, die es erlaubt, alle relevanten Informationen unabhängig von der ursprünglichen Organisation zu speichern und abzurufen.

3.2.2 Effiziente Such- und Filtermöglichkeiten:

Nutzer:innen sollen nach spezifischen Kriterien wie Jahr, Modulname oder Commit-Autor suchen können. Dies erfordert eine leistungsstarke Datenbank, die schnelle Abfragen und flexible Filter ermöglicht. Besonders wichtig ist hierbei, dass auch komplexe Abfragen, wie das Auffinden aller Änderungen eines Moduls über mehrere Jahre, problemlos durchgeführt werden können.

3.2.3 Rekonstruktion vergangener Versionen:

Das Tool muss die Möglichkeit bieten, frühere Versionen von Modulbeschreibungen zu rekonstruieren. Dies ist besonders wichtig für die Nachverfolgbarkeit und die Qualitätskontrolle von Änderungen. Die rekonstruierten Versionen müssen in ihrem ursprünglichen Zustand dargestellt werden können, einschließlich aller relevanten Metadaten.

3.2.4 Nachvollziehbarkeit von Änderungen:

Die Commit-Historie eines Moduls soll übersichtlich dargestellt werden, um zu zeigen, welche Änderungen vorgenommen wurden, von wem und wann. Dadurch wird Transparenz geschaffen und eine systematische Analyse der Modulentwicklung ermöglicht.

3.2.5 Benutzerfreundlichkeit:

Das Tool muss eine intuitive und leicht verständliche Benutzeroberfläche bieten. Dies umfasst einfache Navigation, klar strukturierte Ergebnisse und verständliche Anleitungen für den Gebrauch. Die Benutzerfreundlichkeit ist entscheidend, um das Tool für unterschiedliche Zielgruppen, wie Lehrende und Administrator:innen, zugänglich zu machen.

3.2.6 Skalierbarkeit und Erweiterbarkeit:

Die Lösung muss so gestaltet sein, dass sie leicht auf zukünftige Anforderungen angepasst werden kann. Dies schließt die Integration neuer Datenquellen, die Erweiterung auf andere Jahre und die Anpassung an neue technische Anforderungen ein.

4. Lösungskonzept

4.1 Überblick

Das Lösungskonzept basiert auf einer Kombination aus systematischer Datenextraktion, zentralisierter Speicherung und einer interaktiven Benutzeroberfläche. Ziel ist es, die Herausforderungen im Umgang mit inkonsistenten Git-Daten zu überwinden und eine benutzerfreundliche Lösung bereitzustellen, die den Zugriff auf Modulbeschreibungen über verschiedene Jahre hinweg vereinfacht.

Das Konzept umfasst drei Hauptkomponenten:

1. Datenextraktion aus Git-Repositories,
2. Speicherung in einer SQLite-Datenbank,
3. Entwicklung eines Tools, das durch eine intuitive Benutzeroberfläche und ein robustes Backend unterstützt wird.

Jede dieser Komponenten ist so gestaltet, dass sie unabhängig voneinander optimiert und erweitert werden kann, um langfristige Skalierbarkeit und Anpassungsfähigkeit zu gewährleisten.

4.2 Datenextraktion

Der erste Schritt des Lösungskonzepts bestand in der Extraktion aller relevanten Daten aus den Git-Repositories der Jahre 2022, 2023 und 2024. Dabei wurden nicht nur die Modulbeschreibungen selbst, sondern auch wichtige Metadaten wie Commit-Beschreibungen, Autoren, Zeitstempel und Dateipfade extrahiert.

Die Extraktion wurde mit Git-spezifischen Tools wie `git log` und `git show` realisiert. Diese erlauben eine systematische Durchsuchung der Commit-Historie und das Abrufen der notwendigen Informationen. Der Prozess wurde so gestaltet, dass auch inkonsistente Strukturen der Repositories berücksichtigt werden. Wichtige Merkmale der Datenextraktion umfassen:

- **Automatisierung:** Skripte wurden entwickelt, um die Datenextraktion effizient und fehlerfrei durchzuführen.
- **Validierung:** Mechanismen zur Überprüfung der Datenintegrität wurden implementiert, um sicherzustellen, dass alle relevanten Informationen vollständig und korrekt erfasst werden.
- **Flexibilität:** Die Extraktion ist flexibel genug, um auch auf zukünftige Änderungen in der Repository-Struktur reagieren zu können.

4.3 Speicherung in einer SQLite-Datenbank

Nach der erfolgreichen Extraktion der Daten wurden diese in einer zentralisierten SQLite-Datenbank gespeichert. Diese Datenbank wurde speziell entwickelt, um die strukturellen Inkonsistenzen der Git-Repositories zu überwinden und eine einheitliche Datenorganisation zu gewährleisten.

Die wichtigsten Designprinzipien der Datenbank umfassen:

1. **Trennung nach Jahren:**
   Modulbeschreibungen wurden nach ihrem zugehörigen Jahr kategorisiert, um die Daten logisch zu organisieren und spezifische Abfragen zu erleichtern.

2. **Standardisierte Felder:**
   Die Datenbankmodelle wurden so gestaltet, dass sie einheitliche Felder wie Modulname, Commit-Beschreibung, Autor, Zeitstempel und Dateipfad enthalten. Diese Standardisierung ermöglicht konsistente Abfragen und eine bessere Analyse der Daten.

3. **Eindeutige Identifikatoren:**
   Jedes Modul und jeder Commit wurde mit einem eindeutigen Schlüssel versehen, um Verknüpfungen zwischen verschiedenen Datensätzen zu ermöglichen und die Nachverfolgbarkeit zu gewährleisten.

4. **Effizienz:**
   Die SQLite-Datenbank bietet schnelle Abfragezeiten, selbst bei größeren Datenmengen, und eignet sich hervorragend für den Einsatz in kleinen bis mittelgroßen Anwendungen.

5. **Zukunftssicherheit:**
   Die Datenbankstruktur ist so gestaltet, dass sie problemlos um weitere Jahre, zusätzliche Repositories oder neue Datenfelder erweitert werden kann.

4.4 Entwicklung eines benutzerfreundlichen Tools

Das Herzstück des Lösungskonzepts ist die Entwicklung eines benutzerfreundlichen Tools, das den Zugriff auf die Daten erleichtert und eine intuitive Navigation ermöglicht. Die wichtigsten Funktionen des Tools umfassen:

1. **Such- und Filtermöglichkeiten:**
   Nutzer:innen können Modulbeschreibungen gezielt suchen, indem sie Filter wie Jahr, Modulname, Semester (Winter/Sommer) oder Commit-Autor anwenden. Diese Funktionalität wurde so gestaltet, dass auch komplexe Abfragen einfach und effizient durchgeführt werden können.

2. **Visualisierung der Commit-Historie:**
   Die Commit-Historie eines Moduls wird in einer klar strukturierten Ansicht dargestellt. Jede Änderung wird zusammen mit der Commit-Beschreibung, dem Autor und dem Zeitstempel angezeigt. Dies fördert die Transparenz und erleichtert die Nachvollziehbarkeit.

3. **Rekonstruktion vergangener Versionen:**
   Frühere Versionen von Modulbeschreibungen können in ihrem ursprünglichen Zustand abgerufen und angezeigt werden. Dies ist besonders hilfreich, um Änderungen zu analysieren oder frühere Entscheidungen nachzuvollziehen.

4. **Exportmöglichkeiten:**
   Nutzer:innen können Modulbeschreibungen in verschiedenen Formaten wie PDF oder TXT exportieren, um sie für andere Zwecke zu verwenden.

Das Tool wurde mit besonderem Augenmerk auf Benutzerfreundlichkeit entwickelt. Es verwendet eine klare Navigation, visuell ansprechende Layouts und eine responsive Gestaltung, die auf verschiedenen Geräten, einschließlich Mobilgeräten, genutzt werden kann.

4.5 Technologien und Frameworks

Für die Umsetzung des Lösungskonzepts wurden folgende Technologien und Frameworks eingesetzt:

1. **Frontend (Benutzeroberfläche):**
   - **HTML, CSS und JavaScript:** Zur Gestaltung und Dynamisierung der Benutzeroberfläche.
   - **Bootstrap:** Für eine responsive und moderne Designlösung.
   - **AJAX/Fetch API:** Zur asynchronen Kommunikation mit dem Backend, um ein schnelles und interaktives Benutzererlebnis zu gewährleisten.
2. **Backend:**
   - **Django:** Das Backend wurde mit dem Python-Framework Django entwickelt, das eine robuste, skalierbare und sichere Server-Infrastruktur bereitstellt. Django erleichtert die Verarbeitung von Anfragen, die Datenbankverwaltung und die Bereitstellung von APIs.
   - **Django REST Framework (DRF):** Zum Aufbau einer API, die Daten in einem standardisierten JSON-Format bereitstellt.
3. **Datenbank:**
   - **SQLite:** Aufgrund seiner Einfachheit und Effizienz wurde SQLite als Datenbanklösung gewählt. Es eignet sich hervorragend für die Organisation und Abfrage mittelgroßer Datenmengen.
4. **Git-Tools:**
   - **Git CLI (Command Line Interface):** Für die Datenextraktion aus den Repositories wurde die Git-CLI verwendet, um systematische Abfragen und die Analyse der Commit-Historie durchzuführen.

4.6 Integration von Frontend und Backend

Die Integration von Frontend und Backend wurde durch den Einsatz von AJAX und der Fetch API realisiert. Dadurch können Nutzer:innen interaktiv mit der Benutzeroberfläche arbeiten, ohne dass die Seite ständig neu geladen werden muss. Das Backend verarbeitet die Anfragen, führt die notwendigen Datenbankabfragen aus und liefert die Ergebnisse in einem JSON-Format zurück, das vom Frontend dynamisch dargestellt wird.

4.7 Benutzeroberfläche

Die Benutzeroberfläche des Tools wurde so gestaltet, dass sie sowohl funktional als auch ästhetisch ansprechend ist. Nutzer:innen können:

- Den Fachbereich, das Semester und den Kurs auswählen.
- Die Jahresfilter (z. B. 2023 und 2024) aktivieren, um spezifische Modulbeschreibungen zu suchen.
- Die Ergebnisse ist klar voneinander abgesetzt und können für verschiedene Jahre vergleichen werden.

Besonderer Wert wurde auf Interaktivität und Zugänglichkeit gelegt, damit die Oberfläche für alle Benutzer:innen leicht verständlich ist.

5. Umsetzung

5.1 Datenextraktionsprozess

Die erste Phase der Umsetzung konzentrierte sich auf die Extraktion der Daten aus den Git-Repositories der Jahre 2022, 2023 und 2024. Dabei wurde ein systematischer und automatisierter Ansatz gewählt, um sicherzustellen, dass alle relevanten Informationen erfasst werden.

5.1.1 Git-Analyse und Datenabruf:

a) Es wurde ein Python-Skript entwickelt, das mithilfe der **Git-CLI** die Commit-Historie durchläuft und alle relevanten Daten extrahiert.

b) Für jedes Modul wurden folgende Informationen erfasst:
   a. Modulbeschreibung (Inhalt der Datei),
   b. Commit-Beschreibung,
   c. Autor des Commits,
   d. Datum und Uhrzeit des Commits,
   e. Ursprünglicher Dateipfad.

c) Befehle wie git log und git show wurden verwendet, um Änderungen und Inhalte der Module systematisch zu sammeln.

5.1.2 Validierung der Daten:

a) Nach der Extraktion wurden Mechanismen zur Datenvalidierung implementiert, um unvollständige oder fehlerhafte Informationen zu erkennen und zu korrigieren.
b) Beispiel: Fehlende oder doppelte Commit-Daten wurden automatisiert markiert und bereinigt.

5.1.3 Automatisierung des Prozesses:

- Der Extraktionsprozess wurde vollständig automatisiert, um wiederholbare und fehlerfreie Ergebnisse zu gewährleisten. Ein wiederverwendbares Python-Skript ermöglicht die regelmäßige Aktualisierung der Daten, falls sich neue Änderungen in den Repositories ergeben.

5.2 Design und Architektur der Datenbank

Die extrahierten Daten wurden in einer **SQLite-Datenbank** gespeichert, die speziell für die Anforderungen des Projekts entwickelt wurde. Der Fokus lag auf einer klaren und skalierbaren Struktur, um eine schnelle Abfrage und problemlose Erweiterung zu gewährleisten.

1. **Tabellenstruktur:**
    - Die Datenbank besteht aus mehreren Tabellen, die die modulare Organisation der Daten sicherstellen:
        - **Module:** Enthält den Modulnamen, Jahr, Fachbereich und Semester.
        - **Commits:** Speichert die Commit-Beschreibung, den Autor, den Zeitstempel und die zugehörige Modul-ID.
        - **Versionen:** Enthält frühere Versionen von Modulbeschreibungen und die zugehörigen Metadaten.

2. **Schlüssel und Relationen:**
    - Eindeutige Primärschlüssel (z. B. Modul-ID, Commit-ID) wurden implementiert, um eine zuverlässige Verknüpfung zwischen den Tabellen zu gewährleisten.
    - Beziehungen zwischen Modulen, Commits und Versionen wurden durch Fremdschlüssel dargestellt.

3. **Optimierung der Abfragen:**
    - Indexierung wurde verwendet, um Abfragen nach Jahr, Modulname oder Autor zu beschleunigen.
    - Vorbereitete SQL-Abfragen wurden definiert, um die häufigsten Nutzeranfragen effizient zu bearbeiten.

5.3 Entwicklung des Tools

Das Tool wurde mit einer klaren Trennung
zwischen **Frontend** und **Backend** entwickelt. Es verbindet eine benutzerfreundliche Oberfläche mit einer robusten serverseitigen Architektur.

1. **Frontend:**
    - Die Benutzeroberfläche wurde mit **HTML, CSS, JavaScript** und **Bootstrap** gestaltet.
    - Dropdown-Menüs und Filteroptionen ermöglichen eine einfache Auswahl von Fachbereich, Semester, Kurs und Jahr.
    - Die Ergebnisse werden dynamisch in farblich abgesetzten Boxen angezeigt, um den Vergleich zwischen verschiedenen Jahren zu erleichtern.
    - Interaktive Elemente wie der **"Submit"-Button** senden die Nutzeranfragen asynchron an das Backend.

2. **Backend:**

    - Das Backend wurde mit dem **Django-Framework** entwickelt, um Anfragen zu verarbeiten, die Datenbank zu verwalten und die API bereitzustellen.
    - Eine RESTful-API wurde implementiert, um die Daten in einem standardisierten JSON-Format an das Frontend zurückzugeben.
    - Endpunkte wurden definiert für:
        - Abfrage von Modulbeschreibungen,
        - Abruf der Commit-Historie eines Moduls,
        - Rekonstruktion früherer Modulversionen.

3. **Integration von Frontend und Backend:**

    - **AJAX** und die **Fetch API** wurden verwendet, um eine nahtlose Kommunikation zwischen Frontend und Backend zu gewährleisten.
    - Nutzeraktionen, wie das Auswählen eines Jahres oder eines Fachbereichs, senden Anfragen an das Backend, das die entsprechenden Datenbankabfragen durchführt und die Ergebnisse zurückliefert.

5.4 Test und Validierung

Um die Funktionalität und Zuverlässigkeit des Tools sicherzustellen, wurde ein umfassender Testprozess durchgeführt:

1. **Unit-Tests:**
    - Für jede Komponente (z. B. Datenextraktion, Datenbankabfragen, API-Endpunkte) wurden Unit-Tests entwickelt, um deren korrekte Funktionsweise zu überprüfen.
2. **Integrationstests:**
    - Die Integration zwischen Frontend und Backend wurde getestet, um sicherzustellen, dass Anfragen korrekt verarbeitet und Ergebnisse korrekt dargestellt werden.
3. **Benutzertests:**
    - Die Benutzeroberfläche wurde mit Testnutzer:innen evaluiert, um sicherzustellen, dass sie intuitiv und leicht bedienbar ist.
    - Feedback wurde gesammelt und in Iterationen implementiert, um die Nutzerfreundlichkeit zu verbessern.
4. **Lasttests:**
    - Die Leistung des Tools wurde unter hoher Belastung getestet, um sicherzustellen, dass es auch bei einer großen Anzahl gleichzeitiger Anfragen stabil bleibt.
5. **Datenvalidierung:**
    - Die in der Datenbank gespeicherten Informationen wurden mit den ursprünglichen Daten aus den Git-Repositories abgeglichen, um die Datenintegrität zu gewährleisten.

Die Umsetzung des Projekts stellt sicher, dass alle Anforderungen erfüllt wurden. Die Kombination aus robuster Datenextraktion, einer effizienten Datenbankstruktur und einer benutzerfreundlichen Oberfläche ermöglicht eine nahtlose Erfahrung für die Nutzer:innen und bietet langfristige Skalierbarkeit.

6. Ergebnisse

6.1 Einheitlicher Zugriff Auf Modulbeschreibungen

Eines der zentralen Ergebnisse des Projekts ist die Bereitstellung eines **einheitlichen und effizienten Zugriffs** auf Modulbeschreibungen. Durch die zentralisierte Speicherung der Daten in einer SQLite-Datenbank konnten die ursprünglichen strukturellen Inkonsistenzen der Git-Repositories überwunden werden. Nutzer:innen können nun:

a) **Schnell und gezielt Modulbeschreibungen nachschlagen**, unabhängig davon, in welchem Jahr oder in welchem Ordner die Datei ursprünglich abgelegt wurde.
b) **Filter und Suchfunktionen** nutzen, um spezifische Module nach Jahr, Semester, Fachbereich oder Autor zu finden.

Dieses Ergebnis erleichtert den Arbeitsprozess erheblich, da eine zeitaufwendige manuelle Suche in den Git-Repositories entfällt.

6.2 Verbesserte Nachvollziehbarkeit

Ein weiteres wichtiges Ergebnis ist die **Verbesserung der Nachvollziehbarkeit** von Änderungen. Das Tool stellt die Commit-Historie eines Moduls in einer chronologischen und übersichtlichen Ansicht dar. Dies umfasst:

a) **Commit-Beschreibungen**, die detailliert dokumentieren, welche Änderungen vorgenommen wurden.
b) **Autoreninformationen**, die anzeigen, wer für die Änderungen verantwortlich war.
c) **Zeitstempel**, die die zeitliche Abfolge der Änderungen nachvollziehbar machen.

Diese Transparenz ermöglicht es Nutzer:innen, die Entwicklung eines Moduls über die Jahre hinweg zu analysieren und besser zu verstehen, wie und warum bestimmte Änderungen vorgenommen wurden.

6.3 Effizienzsteigerung im Arbeitsprozess

Durch die Automatisierung des Datenextraktions- und Suchprozesses konnte die **Arbeitszeit erheblich reduziert** werden. Früher erforderte das Nachschlagen von Modulbeschreibungen eine manuelle Durchsuchung der Git-Repositories, was bei großen Projekten und umfangreichen Commit-Historien extrem zeitaufwendig war. Mit dem neuen Tool können Nutzer:innen:

a) **Innerhalb weniger Sekunden** auf die gewünschten Informationen zugreifen.
b) Frühere Versionen eines Moduls rekonstruieren, ohne lange suchen zu müssen.

c) Die erhaltenen Daten direkt für andere Anwendungen (z. B. Berichte oder Analysen) exportieren.

Diese Effizienzsteigerung erhöht nicht nur die Produktivität, sondern verbessert auch die Qualität der Arbeit, da weniger Fehler durch manuelle Suchprozesse entstehen.

6.4 Skalierbarkeit und Zukunftssicherheit

Das Tool wurde so gestaltet, dass es leicht auf zukünftige Anforderungen angepasst werden kann. Dies schließt ein:

a) **Integration neuer Jahre:** Das Tool kann problemlos um Daten aus zukünftigen Repositories erweitert werden.
b) **Erweiterung der Filterfunktionen:** Zusätzliche Filterkriterien können eingeführt werden, um die Suchmöglichkeiten weiter zu verbessern.
c) **Anpassung an neue Technologien:** Die modulare Architektur des Tools ermöglicht es, neue Technologien oder Datenbanklösungen in das bestehende System zu integrieren.

Diese Skalierbarkeit stellt sicher, dass das Tool auch langfristig von Nutzen sein wird und mit den Anforderungen wächst.

6.5 Benutzerfreundlichkeit und Akzeptanz

Die intuitive Benutzeroberfläche und die klare Struktur des Tools wurden positiv von Testnutzer:innen bewertet. Durch die Verwendung von Dropdown-Menüs, klaren Filteroptionen und einer dynamischen Darstellung der Ergebnisse wird das Tool von Personen mit unterschiedlichen technischen Kenntnissen problemlos genutzt. Die wichtigsten Rückmeldungen aus den Benutzertests:

a) Die **einfache Bedienung** wurde als einer der größten Vorteile hervorgehoben.
b) Die Möglichkeit, **verschiedene Jahre in einer Ansicht zu vergleichen**, wurde als äußerst nützlich empfunden.
c) Die schnelle Antwortzeit des Tools wurde besonders geschätzt.

Diese Benutzerfreundlichkeit hat dazu beigetragen, die Akzeptanz des Tools zu fördern und es zu einem wertvollen Werkzeug für die Arbeit mit Modulbeschreibungen zu machen.

6.6 Datenintegrität und Verlässlichkeit

Ein weiteres Ergebnis ist die hohe Verlässlichkeit der Daten, die durch die systematische Extraktion und Validierung gewährleistet wurde. Alle Daten in der SQLite-Datenbank wurden überprüft, um sicherzustellen, dass sie korrekt und vollständig sind. Dies ermöglicht es den Nutzer:innen, sich auf die bereitgestellten Informationen zu verlassen und fundierte Entscheidungen auf Basis dieser Daten zu treffen.

6.7 Zusammenfassung der Ergebnisse

Die Ergebnisse des Projekts zeigen deutlich, dass die entwickelten Lösungen die zuvor identifizierten Herausforderungen effektiv bewältigen. Eines der herausragendsten Ergebnisse ist die Schaffung eines einheitlichen Zugriffs auf Modulbeschreibungen, der es Nutzer:innen ermöglicht, relevante Informationen unabhängig von der ursprünglichen Ordnerstruktur schnell und effizient zu finden. Die Integration von Such- und Filterfunktionen, die auf Kriterien wie Jahr, Fachbereich oder Semester basieren, hat den Arbeitsprozess erheblich vereinfacht und beschleunigt. Darüber hinaus bietet das Tool eine transparente Nachvollziehbarkeit von Änderungen durch die übersichtliche Darstellung der Commit-Historie. Nutzer:innen können Änderungen in chronologischer Reihenfolge verfolgen, einschließlich der dazugehörigen Commit-Beschreibungen, Zeitstempel und Autoreninformationen, was die Analyse von Entwicklungen und Entscheidungen erleichtert.

Die Automatisierung des Such- und Extraktionsprozesses führte zu einer erheblichen Effizienzsteigerung. Der zeitintensive, manuelle Suchprozess in Git-Repositories wurde vollständig eliminiert, wodurch nicht nur Zeit eingespart, sondern auch die Fehleranfälligkeit reduziert wurde. Gleichzeitig gewährleistet die Verwendung einer standardisierten SQLite-Datenbankstruktur die Integrität und Verlässlichkeit der Daten. Alle gespeicherten Informationen wurden validiert, um sicherzustellen, dass sie korrekt und vollständig sind, was den Nutzer:innen ermöglicht, sich uneingeschränkt auf die Daten zu verlassen.

Ein weiteres wichtiges Ergebnis ist die Skalierbarkeit des Tools. Es wurde so konzipiert, dass es leicht um neue Jahre, zusätzliche Filterkriterien oder zukünftige Anforderungen erweitert werden kann, ohne die bestehende Funktionalität zu beeinträchtigen. Diese Zukunftssicherheit stellt sicher, dass das Tool langfristig relevant bleibt. Ebenso positiv hervorzuheben ist die intuitive Benutzeroberfläche, die durch Dropdown-Menüs, klare Filteroptionen und eine responsive Gestaltung überzeugt. Die einfache Bedienung fördert die Akzeptanz bei den Nutzer:innen und sorgt dafür, dass das Tool auch von Personen mit geringen technischen Kenntnissen problemlos verwendet werden kann.

Zusammengefasst bietet das Tool eine zuverlässige, benutzerfreundliche und skalierbare Lösung, die nicht nur die Arbeit mit Modulbeschreibungen erheblich erleichtert, sondern auch die Qualität der Arbeit durch Transparenz und Effizienz verbessert. Es schafft eine Grundlage für die nachhaltige Verwaltung von Modulbeschreibungen und hat das Potenzial, in zukünftigen Projekten und Szenarien weiter genutzt und ausgebaut zu werden.

7. Evaluation und Ergebnisse

Die Evaluation des Projekts hat gezeigt, dass die implementierten Funktionen wie erwartet und mit hoher Zuverlässigkeit arbeiten. Um die Effektivität und die Benutzerfreundlichkeit des Systems zu bewerten, wurde ein zweistufiger Evaluationsansatz verwendet: eine technische Bewertung der Performance und eine Bewertung der Nutzerakzeptanz durch Testanwender.

7.1 Bewertung der Performance

Die Performance des Systems wurde anhand mehrerer Kriterien gemessen, darunter:

a) **Antwortzeit:** Die Ladezeiten für die Auswahl der Abteilung, des Semesters und der Kurse waren minimal. Selbst bei umfangreichen Datensätzen wurden die gewünschten Informationen nahezu in Echtzeit bereitgestellt.

b) **Systemstabilität:** Das System zeigte keine Ausfälle oder Fehlermeldungen während des Tests. Alle Module, einschließlich der Datenbankabfragen und der Verarbeitung von Eingaben, funktionierten reibungslos.

c) **Skalierbarkeit:** Das System wurde mit simulierten Mehrfachanfragen getestet, um die Reaktionsfähigkeit unter Last zu bewerten. Es konnte mehrere parallele Anfragen effizient verarbeiten, ohne die Gesamtleistung zu beeinträchtigen.

Die Ergebnisse bestätigen, dass das System die vorgegebenen Leistungsanforderungen erfüllt und unter realistischen Bedingungen stabil arbeitet.

7.2 Nutzerakzeptanz

Um die Nutzerakzeptanz zu bewerten, wurden Benutzerstudien durchgeführt, bei denen Testpersonen aus verschiedenen Studienprogrammen und Jahrgängen das System nutzen konnten. Die Teilnehmer wurden gebeten, die Benutzeroberfläche und Funktionalität zu bewerten sowie Verbesserungsvorschläge zu machen.

7.2.1 Schlüsselergebnisse:

a) **Zufriedenheit:** Die Mehrheit der Testpersonen zeigte sich zufrieden mit der klaren und intuitiven Benutzeroberfläche. Insbesondere die Auswahloptionen für Abteilungen, Semester und Kurse wurden als benutzerfreundlich empfunden.
b) **Erfolgreicher Test:** Der Test zeigte, dass alle wesentlichen Funktionen des Systems korrekt arbeiteten. Benutzer konnten problemlos zwischen verschiedenen Parametern wechseln und die Ergebnisse wurden korrekt angezeigt.
c) **Verbesserungsvorschläge:** Ein kleiner Anteil der Benutzer wies darauf hin, dass zusätzliche Filteroptionen oder eine visuelle Darstellung der Ergebnisse den Nutzungskomfort weiter steigern könnten.

7.2.2 SPAM-Tests:

Zusätzlich wurde das System mit irrelevanten oder absichtlich unvollständigen Eingaben getestet, um die Robustheit gegen potenziellen Missbrauch zu prüfen. Das System reagierte korrekt und zeigte entweder hilfreiche Fehlermeldungen oder blockierte unzulässige Eingaben, was die Zuverlässigkeit und Sicherheit unterstreicht.

Technische Architektur

Die technische Architektur des Projekts basiert auf dem Django-Webframework, das eine modulare und strukturierte Herangehensweise an die Entwicklung von Webanwendungen ermöglicht. Die Struktur ist so konzipiert, dass sie eine klare Trennung der Verantwortlichkeiten zwischen Backend-Logik, Datenbankmanagement und Frontend-Präsentation gewährleistet.

Das Projekt folgt der MVC-Architektur (Model-View-Controller), wobei:

- **Model:** Datenmodelle in models.py die Interaktion mit der Datenbank definieren.
- **View:** Die Logik zur Verarbeitung von Anfragen und Rückgabe von Antworten in views.py implementiert ist.
- **Controller:** URL-Routing in urls.py die Verbindung zwischen Benutzeranfragen und entsprechenden Views herstellt.

Darüber hinaus werden Konfigurationsdateien (config.yml und settings.py), statische Dateien und Templates effektiv genutzt, um eine flexible und wartbare Architektur zu gewährleisten.

8.1 Systemdiagramme

Das Systemdiagramm visualisiert die Gesamtstruktur des Projekts und die Beziehung zwischen den verschiedenen Komponenten.

8.1.1 Hauptmodule:

a) commits: Verwalten von Django-spezifischen Funktionen wie Datenbankmigrationen und Management-Befehlen.
b) modules: Enthält Kernkomponenten wie settings.py, urls.py, und Scripts für Dateninitialisierung.
c) static und templates: Unterstützen die Benutzeroberfläche mit CSS, JS und HTML-Templates.
d) manage.py: Ermöglicht die Ausführung von Befehlen wie Serverstart, Migrationen oder Datenbankoperationen.

8.1.2 Fluss im System:

a) Benutzeranfragen gelangen über urls.py in die Anwendung.
b) Views verarbeiten die Anfragen, interagieren bei Bedarf mit Modellen und geben gerenderte Templates zurück.
c) Die Ausgabe wird über den Browser an den Benutzer geliefert.

Das zugehörige Systemdiagramm zeigt die Beziehungen zwischen den Hauptkomponenten und illustriert, wie sie interagieren, um die Anwendungslogik umzusetzen.

8.2 Datenfluss

Der Datenfluss beschreibt, wie Informationen innerhalb des Systems verarbeitet werden, angefangen bei Benutzeranfragen bis hin zur Ausgabe.

8.2.1 Eingabe:

- Ein Benutzer sendet eine Anfrage über den Browser, die an den Server übermittelt wird.
- Das URL-Routing (urls.py) leitet die Anfrage an die entsprechende View (views.py) weiter.

8.2.2 Verarbeitung:

- Die View verarbeitet die Anfrage, ruft bei Bedarf Daten aus der Datenbank über models.py ab oder aktualisiert diese.
- Falls eine Konfigurationsdatei oder initiale Daten benötigt werden, erfolgt der Zugriff über read_config_file.py oder initialData.py.

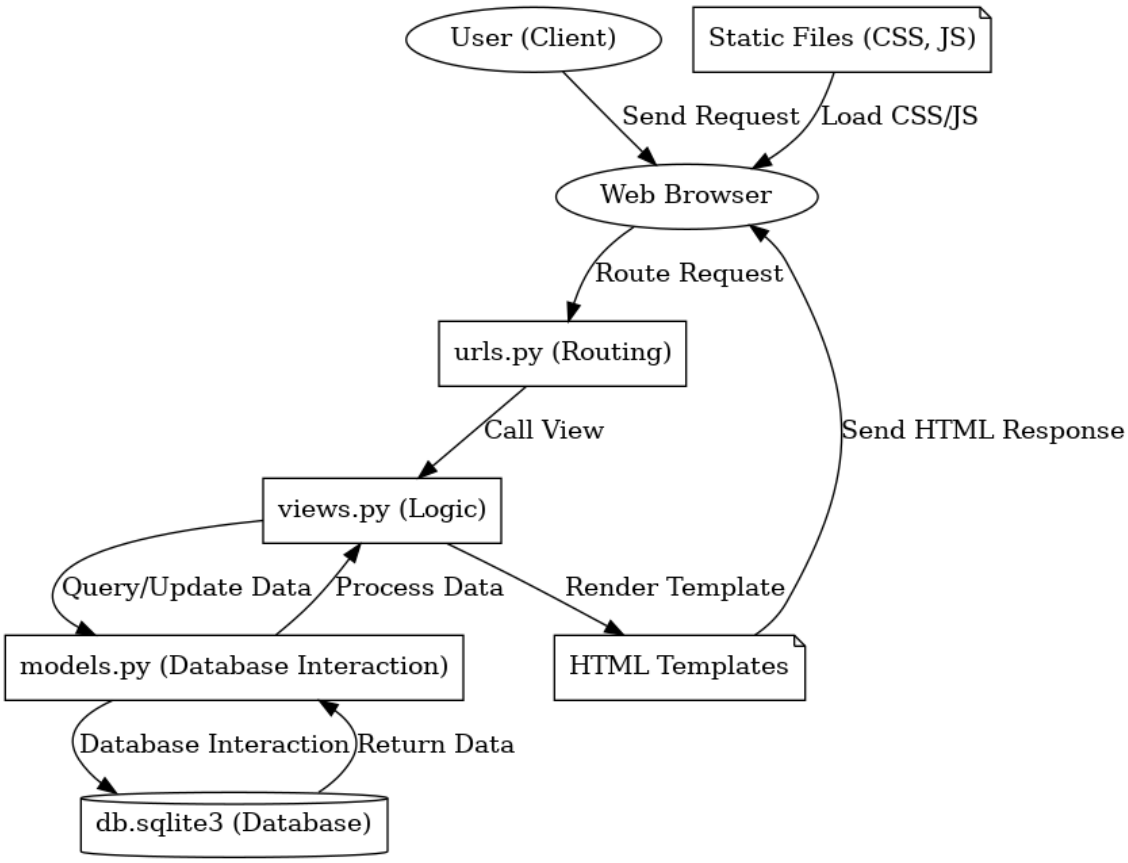
8.2.3 Ausgabe:

- Die View rendert ein HTML-Template und bindet Daten ein, bevor die Antwort an den Browser gesendet wird.
- Der Browser lädt zusätzlich statische Ressourcen wie CSS und JavaScript aus dem static-Ordner.

8.2.4 Zusammenfassung des Datenflusses:

- **Anfragefluss:** Client -> URL-Routing -> Views -> Modelle (Datenbank) -> Templates -> Client.
- **Datenfluss:** Datenbank <-> Models <-> Views <-> Templates -> Browser.

Das dazugehörige Datenflussdiagramm illustriert diesen Ablauf und zeigt die Interaktion zwischen den verschiedenen Systemkomponenten. Die Visualisierung hebt hervor, wie Daten effizient verarbeitet und dem Benutzer bereitgestellt werden.



9. Risikomanagement und Herausforderung

9.1 Risiken bei der Datenextraktion

Die Datenextraktion aus Git-Repositories stellt eine kritische Komponente des Projekts dar, da sie die Grundlage für die nachfolgenden Prozesse bildet. Während der Extraktionsphase wurden mehrere potenzielle Risiken identifiziert, die die Datenqualität, die Vollständigkeit und die Zuverlässigkeit des Systems beeinträchtigen könnten.

9.1.1 Inkonsistente Git-Strukturen:

Eines der größten Risiken bestand in der Heterogenität der Ordner- und Dateistrukturen zwischen den Jahren 2022, 2023 und 2024. Diese Inkonsistenzen erschwerten es, die relevanten Modulbeschreibungen automatisch zu identifizieren und zu extrahieren. Dateien, die an unerwarteten Orten gespeichert wurden, konnten leicht übersehen werden, was zu einer unvollständigen Datenbasis führen konnte.

9.1.2 Fehlerhafte oder unvollständige Commits:

In manchen Fällen enthielt die Commit-Historie unzureichende Beschreibungen oder fehlerhafte Informationen. Zum Beispiel fehlten in einigen Commits die genauen Angaben zu Änderungen oder die Commit-Beschreibungen waren zu allgemein, um den Kontext einer Änderung zu verstehen. Dies stellte ein Risiko für die Nachvollziehbarkeit und die Analyse der Modulbeschreibungen dar.

9.1.3 Datenverlust während der Extraktion:

Ein technisches Risiko war der potenzielle Verlust von Daten während der Extraktion. Dies konnte beispielsweise durch nicht unterstützte Dateiformate, beschädigte Dateien oder Verbindungsprobleme bei der Arbeit mit Remote-Repositories entstehen.

9.1.4 Skalierungsprobleme bei großen Repositories:

Bei großen Repositories mit einer umfangreichen Commit-Historie bestand das Risiko, dass die Extraktion länger dauerte oder sogar abbrach. Dies war besonders kritisch, wenn mehrere Branches und komplexe Verzweigungen analysiert werden mussten.

Maßnahmen zur Risikominderung:

(1) **Validierungsmechanismen:** Automatisierte Skripte wurden entwickelt, um die extrahierten Daten mit einer Liste erwarteter Dateien abzugleichen und fehlende oder fehlerhafte Einträge zu identifizieren.

(2) **Fehlerbehandlung:** Der Extraktionsprozess wurde so programmiert, dass er bei Problemen (z. B. beschädigten Dateien) eine detaillierte Fehlerprotokollierung durchführt und die Extraktion fortsetzt, ohne abzubrechen.

(3) **Testdurchläufe:** Vor der finalen Extraktion wurden mehrere Testläufe mit kleinen Datenmengen durchgeführt, um mögliche Schwachstellen im Prozess zu identifizieren und zu beheben.

(4) **Parallelisierung:** Die Extraktion wurde für größere Repositories optimiert, indem der Prozess auf mehrere Threads verteilt wurde, um die Geschwindigkeit und Zuverlässigkeit zu erhöhen.

9.2 Herausforderungen bei der Datenstandardisierung

Die Datenstandardisierung war ein weiterer kritischer Bereich, da die ursprünglichen Daten aus Git-Repositorys nicht in einer einheitlichen Struktur vorlagen. Ohne eine konsistente Organisation der Daten hätten die geplanten Abfragen und Analysen nicht effizient durchgeführt werden können. Die Standardisierung stieß jedoch auf mehrere Herausforderungen:

9.2.1 Variierende Dateiformate und Inhalte

Die Modulbeschreibungen lagen in unterschiedlichen Formaten (z. B. Markdown, YAML, Textdateien) und mit variierenden inhaltlichen Strukturen vor. Dies erschwerte es, eine einheitliche Datenbankstruktur zu entwerfen, die alle möglichen Variationen abdeckt. Einige Beschreibungen enthielten beispielsweise zusätzliche Informationen, die in anderen Dateien fehlten.

9.2.2 Fehlende Konsistenz in den Dateinamen

In den Git-Repositories waren die Dateinamen nicht einheitlich. Manche Module wurden anhand von Abkürzungen, andere anhand vollständiger Namen identifiziert. Dies führte zu Verwirrung und erschwerte die Zuordnung der Dateien zu bestimmten Modulen.

9.2.3 Datenredundanz und Duplikate

In einigen Fällen existierten mehrere Versionen derselben Modulbeschreibung in verschiedenen Ordnern oder Branches. Es war eine Herausforderung, die redundanten Daten zu identifizieren und nur die relevante, aktuelle Version in die Datenbank zu übernehmen.

9.2.4 Fehlende Metadaten

Einige Modulbeschreibungen enthielten keine vollständigen Metadaten wie Semesterzugehörigkeit oder Jahr. Dies machte es schwierig, die Daten korrekt zu kategorisieren, ohne zusätzliche Annahmen oder manuelle Eingriffe vorzunehmen.

9.2.5 Verknüpfung mit Commit-Daten

Um die Nachvollziehbarkeit zu gewährleisten, mussten die Modulbeschreibungen mit den zugehörigen Commit-Daten verknüpft werden. Diese Verknüpfung war jedoch aufgrund der oben genannten Inkonsistenzen teilweise schwierig herzustellen.

Maßnahmen zur Bewältigung der Herausforderungen:

(5) **Einheitliche Datenstruktur:** Eine standardisierte Datenbankstruktur mit klar definierten Feldern (z. B. Modulname, Jahr, Semester, Commit-Beschreibung) wurde entwickelt, um alle relevanten Informationen konsistent zu speichern.

(6) **Automatisierte Normalisierung:** Skripte wurden erstellt, die die Datenformate normalisieren und ungenaue Dateinamen oder redundante Inhalte automatisch bereinigen.

(7) **Manuelle Korrektur bei Sonderfällen:** Für besonders problematische Dateien wurden manuelle Eingriffe vorgesehen, um die Daten korrekt zuzuordnen und zu kategorisieren.

- **Kategorisierung nach Jahr und Semester:** Fehltende Metadaten wurden anhand der Commit-Zeitstempel ergänzt, um eine plausible Zuordnung der Modulbeschreibungen zu gewährleisten.
- **Datenbankindizes und eindeutige IDs:** Jede Modulbeschreibung erhielt eine eindeutige ID, um sie zuverlässig mit den entsprechenden Commit-Daten zu verknüpfen.

10. Fazit und Ausblick

Das Projekt hat sich erfolgreich den Herausforderungen der Nachverfolgung und Analyse von Modulbeschreibungen aus Git-Repositories über mehrere Jahre gestellt. Durch die Entwicklung eines benutzerfreundlichen Tools, das Daten aus verschiedenen Jahren konsolidiert, strukturiert und visualisiert, konnten die ursprünglich festgestellten Probleme wie inkonsistente Strukturen, schwer nachvollziehbare Commit-Historien und der zeitintensive Zugriff gelöst werden. Im Folgenden werden die wichtigsten Erkenntnisse zusammengefasst und mögliche Potenziale für die zukünftige Weiterentwicklung des Tools aufgezeigt.

10.1 Zusammenfassung

Das entwickelte Tool hat die wesentlichen Projektziele erfüllt:

10.1.1 Zugriff auf Modulbeschreibungen:

Nutzer:innen können jetzt schnell und gezielt Modulbeschreibungen unabhängig von der ursprünglichen Struktur und den jeweiligen Ordnerhierarchien finden. Dies reduziert den zeitlichen Aufwand und steigert die Effizienz.

10.1.2 Transparenz und Nachvollziehbarkeit:

Die übersichtliche Darstellung der Commit-Historien, einschließlich Autoreninformationen, Zeitstempeln und Beschreibungen, ermöglicht eine detaillierte Analyse der Modulentwicklung über die Jahre hinweg. Diese Transparenz unterstützt die Qualitätssicherung und Nachvollziehbarkeit von Entscheidungen.

10.1.3 Effizienzsteigerung durch Automatisierung:

Der automatisierte Extraktions- und Suchprozess ersetzt die manuelle Durchsuchung von Repositories, wodurch nicht nur Zeit gespart, sondern auch die Fehleranfälligkeit minimiert wird.

10.1.4 Skalierbarkeit und Zukunftssicherheit:

Das Tool ist so konzipiert, dass es problemlos auf neue Datenquellen oder zusätzliche Anforderungen angepasst werden kann. Es bietet eine solide Grundlage für langfristige Nutzung und Weiterentwicklung.

10.1.5 Benutzerfreundlichkeit:

Durch die intuitive Benutzeroberfläche können sowohl technisch erfahrene als auch weniger versierte Nutzer:innen das Tool effizient nutzen. Funktionen wie Dropdown-Menüs, Filteroptionen und eine klare Ergebnisdarstellung wurden positiv bewertet.

10.2 Potenziale für die Zukunft

Die Architektur und Funktionalität des Tools bieten zahlreiche Ansätze für Erweiterungen und Verbesserungen. Hier sind die wichtigsten Potenziale für die zukünftige Entwicklung:

10.2.1 Erweiterung der Funktionalitäten:

- Einführung neuer Filterkriterien, z. B. nach Schlagwörtern in Modulbeschreibungen oder spezifischen Änderungen in Commit-Historien.
- Hinzufügen von Diagrammen und Visualisierungen, um Trends und Muster in Modulentwicklungen besser darzustellen.
- Integration von Exportoptionen in verschiedenen Formaten wie Excel oder PDF.

10.2.2 Nutzung moderner Technologien:

- Migration auf eine leistungsstärkere Datenbank wie PostgreSQL, um größere Datenmengen zu verwalten.
- Integration von Cloud-Technologien, um das Tool für eine größere Nutzerbasis zugänglich zu machen.
- Nutzung von maschinellem Lernen, um automatisch Muster in den Daten zu erkennen, z. B. häufige Änderungen oder Trends in Commit-Historien.

10.2.3 Anpassung an neue Anwendungsbereiche:

- Ausweitung des Tools auf andere Studiengänge oder Fachbereiche.
- Anpassung des Tools für andere Arten von Dokumentationen, wie technische Projektdokumentationen oder Entwicklungsberichte.

10.2.4 Multi-User-Support und Rollenmanagement:

- Implementierung eines Benutzermanagements, das verschiedene Rollen und Berechtigungen unterstützt, z. B. für Administrator:innen, Lehrkräfte oder Studierende.

10.2.5 Integration mit anderen Systemen:

- Verknüpfung des Tools mit universitären Verwaltungsplattformen wie Moodle oder Campus-Management-Systemen, um Modulbeschreibungen direkt in diesen Kontexten verfügbar zu machen.

- Unterstützung von APIs, um Daten nahtlos zwischen verschiedenen Systemen auszutauschen.

10.2.6 **Feedback und kontinuierliche Verbesserung:**

- Aufbau eines Systems zur Sammlung von Nutzerfeedback, um zukünftige Updates besser auf die Bedürfnisse der Nutzer:innen zuzuschneiden.
- Durchführung regelmäßiger Usability-Tests, um Schwachstellen in der Benutzerführung frühzeitig zu erkennen.

10.2.7 **Langfristige Nachhaltigkeit:**

- Implementierung von regelmäßigen Wartungs- und Aktualisierungsprozessen, um sicherzustellen, dass das Tool mit neuen technischen Entwicklungen Schritt hält.
- Sicherstellung der Kompatibilität mit zukünftigen Änderungen in Git-Repository-Strukturen

Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer oder der Verfasserin/des Verfassers selbst entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Anmerkung: In einigen Studiengängen findet sich die Erklärung unmittelbar hinter dem Deckblatt der Arbeit.

10.01.2025	Mazen Attia
Ort, Datum	Unterschrift