RYZE DIGITAL

Projektarbeit Sommer 2024
Fachinformatiker für Anwendungsentwicklung
Dokumentation zur betrieblichen Projektarbeit

GitLab-Suchmaschine

Entwicklung einer Suchmaschine für Programmcode in unserer Git-Versionierung

Projektzeitraum:	25.03.2024 – 12.04.2024
Prüfungsbewerber:	Christian Koop
Prüflingsnummer:	11476
Ausbilder:	Bennet Klaus
Projektbetreuer:	Stefan Eyerich
Ausbildungsbetrieb:	RYZE Digital GmbH Berliner Allee 65 64295 Darmstadt

1 Inhaltsverzeichnis

2 Abkürzungs- und Fachwortverzeichnis

Fachbegriff	**Erklärung**
Git	Ein verteiltes Versionskontrollsystem zur Verfolgung von Änderungen in Quellcode
GitLab	Eine zentrale Plattform zur Verwaltung von in Git versionierten Projekten
REST-API	Eine Schnittstelle, die es ermöglicht, über das HTTP-Protokoll auf Daten zuzugreifen (Representational State Transfer)
Breaking Change	Eine Änderung in einer Software, die dazu führt, dass bestehende Funktionalitäten nicht mehr wie erwartet funktionieren (z. B. Änderungen in der API, die eine Anpassung von API-Nutzern erforderlich machen)
OAuth 2.0	Ein Protokoll, das es ermöglicht, sich bei einer Anwendung über einen Drittanbieter zu authentifizieren (z. B. ein "Login mit Google"-Button)
PostgreSQL	Ein relationales Datenbankmanagementsystem mit dem Schwerpunkt auf Erweiterbarkeit und Konformität mit SQL-Standards
MariaDB	Ein relationales Datenbankmanagementsystem, welches seine Wurzeln in MySQL hat. Kommt bei den meisten Webanwendungen der RYZE Digital GmbH zum Einsatz
TypeScript	Eine Programmiersprache, die auf JavaScript aufbaut und statische Typisierung ermöglicht
Node.js	Eine JavaScript-Laufzeitumgebung, die es ermöglicht, JavaScript außerhalb des Browsers auszuführen – Node.js verwendet die *V8 Engine*, welche auch von Google Chrome genutzt wird
(Docker-) Container	Container sind ein von der *Open Container Initiative* definierter Standard, um Anwendungen in isolierten Umgebungen auszuführen. Man kann sich vorstellen, die Anwendung läuft in einer Art virtuellen Maschine, jedoch ohne virtualisierte Hardware oder Kernel.
Docker	Docker ist eine Container-Plattform, die Container-Images bauen und ausführen kann. Sie zählt zu den bekanntesten – Eine relative neue populäre Alternative ist *PodMan*
Abstract Syntax Tree (AST)	Hierbei handelt es sich um eine Datenstruktur, die zum Beispiel Programmcode repräsentieren kann. Sie lässt sich wie ein Baum mit einem Startpunkt und beliebig vielen Ästen vorstellen.

3 Einleitung

In der nachfolgenden Projektdokumentation wird das Projekt mit dessen Zielen und Motivationen beschrieben und anschließend eine Dokumentation zur Nutzung und Weiterentwicklung bereitgestellt.

3.1 Projektbeschreibung

Das Thema meines Abschlussprojektes ist die Konzeption und Entwicklung einer Suchmaschine für unsere GitLab-Projekte.
Diese soll es unseren Anwendungsentwicklern ermöglichen, ohne weitere Aufwände alle Projekte zu durchsuchen und so zum Beispiel die Nutzung bestimmter Softwarekomponenten einschätzen zu können.

3.2 Projektmotivation

Die Projektidee ist bei größeren Umbaumaßnahmen in unserer Software entstanden. Seit über einem Jahr bin ich in dem Team tätig, welches die Kernsoftware betreut, auf dem die meisten unserer Kundenprojekte aufbauen.
Breaking Changes, die Upgrade-Aufwände in den Projektteams verursachen, versuchen wir wenn möglich zu vermeiden, um keine unnötig hohen Zeitaufwände und damit verbundenen Kosten zu verursachen.
Hierbei hilft es enorm, die einzelnen Kundenprojekte zu kennen, mit all ihren Besonderheiten in der Implementierung.

3.3 Ausgangssituation

Zum aktuellen Zeitpunkt ist es sehr hilfreich, die verschiedenen Kundenprojekte betreut zu haben und an den verschiedenen Projektkomponenten gearbeitet zu haben.
Aber auch das reicht nicht, um vollumfänglich Entscheidungen zu treffen, die in den Kundenprojekten potenziell enorme Aufwände erzeugen, die sich deutlich besser zentral lösen lassen, statt in jedem Kundenprojekt einzeln.

Unsere GitLab-Instanz bietet keine Suche für Dateiinhalte, da diese Suchfunktion mit einer teuren Lizenz verknüpft ist.
Daher durchsuchen wir manuell einige wenige Projekte oder über Befehle in der Kommandozeile mittels Stichworten.
Das kostet viel Zeit für einen unvollständigen Überblick.

3.4 Projektziel

Ziel dieses Projektes ist daher die Bereitstellung einer Web-Anwendung im Unternehmensnetzwerk mit Suchfeld und Zugriffskontrolle.
Es soll uns Entwicklern möglich sein, den Dateiinhalt aller relevanten Projekte durchsuchen zu können und so zum Beispiel die Nutzung bestimmter Klassen in Projekten zu finden.
Außerdem soll eine Zugriffskontrolle sicherstellen, dass einem Benutzer keine Inhalte von Projekten angezeigt werden, auf die dieser keinen Lesezugriff besitzt.

Die Suchmaschine soll dem Benutzer Zeit sparen und sollte dementsprechend die Projekte aufbereitet in einer Datenbank speichern und automatisch aktuell halten.
So soll das wiederholte Einlesen und Durchsuchen einzelner Dateien vermieden werden.

3.5 Beteiligten Personen

Name	**Position**	**Rolle im Projekt**
Christian Koop	Auszubildender	Auftragnehmer / Umsetzung
Matthias Frescha	Managing Director (.TECH)	Auftraggeber
Stefan Eyerich	Teamlead Dev	Projektbetreuer
Rene Drewes	Systemadministrator	Bereitstellung des Zielsystems
Christian Klemmer	Anwendungsentwickler / DevOps	Abnahme GitLab Interaktionen

4 Betriebliches Umfeld

4.1 Umfeld des Auftraggebers

Auftraggeber ist die RYZE Digital GmbH, da es sich hierbei um eine Anwendung für den internen Gebrauch handelt.
Die RYZE Digital GmbH ist ein Agenturverbund aus vier Unternehmen, die unterschiedliche Stärken bündeln und Lösungen in verschiedenen Themen und Formaten entwickeln.
Der Standort Darmstadt, an dem ich tätig bin, auch genannt .TECH, ist auf die Entwicklung von verschiedenen Web-Anwendungen spezialisiert, entwickelt jedoch auch komplexe Anwendungen zur Datenverarbeitung und -aufbereitung für unsere Kunden.
Unsere Softwareprojekte werden in einem internen GitLab mit Hilfe von Git versioniert.

4.2 Umfeld des Auftragnehmers

Auftragnehmer ist Christian Koop, Auszubildender als Fachinformatiker für Anwendungsentwicklung am Standort Darmstadt, der Abteilung .TECH der RYZE Digital GmbH.
Ich bin aktuell in dem Team tätig, welches unsere Kernsoftware, auf dem die meisten unserer Kundenprojekte aufbauen, weiterentwickelt und wartet.
Für unser Team sind Kenntnisse bezüglich Projektimplementierungen notwendig, da häufig bestehende Komponenten angepasst oder sogar ersetzt werden müssen. Dieses detaillierte Projektwissen ermöglicht uns Entscheidungen zu treffen, welche in den Projektteams Aufwände reduzieren können.

5 Projektumfeld

Das Abschlussprojekt wird losgelöst von anderen Projekten im Unternehmen entwickelt und unabhängig von der internen Infrastruktur.
Um diese unabhängige Entwicklung zu ermöglichen, muss ein entsprechendes Projektumfeld geschaffen werden, welches die nötige Infrastruktur bereitstellt.

Während der Konzeption und Entwicklung ist zu beachten, dass große Datenmengen verarbeitet werden und dieser Datenbestand nahezu täglich zunimmt.
Es bestehen zum aktuellen Zeitpunkt bereits über 120 Benutzerkonten mit unterschiedlichen Projekt-Berechtigungen, die effizient verarbeitet werden müssen.
Die Last auf der GitLab-Instanz muss gering gehalten werden, um den normalen Betrieb nicht zu unterbrechen und unnötig hohe Auslastung der Instanz zu vermeiden.

6 Projektplanung

6.1 Projektphasen

Phase	**Aufgabe**	**Eingeplante Zeit**
1	**Analyse**	**7 Stunden**
	Ist-Analyse	1 Stunde
	Soll-Analyse	1 Stunde
	Kosten-Nutzen-Analyse	1 Stunde
	Definition der Suchanfragen-Syntax	2 Stunden
	Formulierung des Lastenheftes inklusive Ressourcenplanung	2 Stunden
2	**Konzeption und Entwurf**	**9 Stunden**
	Indexierung und dazugehörige Datenbank-Struktur	4 Stunden
	Zugriffskontrolle und dazugehörige Datenbank-Struktur	2 Stunden
	Mögliche Interaktion mit Git und GitLab APIs prüfen	1 Stunde
	Web-Oberfläche und dazugehörige Endpunkte	2 Stunden
3	**Implementierung**	**52 Stunden**
	Indexer: Quellcode und Commits eines GitLab-Projektes indexieren	21 Stunden
	Indexer: (Datei-)änderungen in GitLab-Projekten effizient erkennen	3 Stunden

Phase	Aufgabe	Eingeplante Zeit
	Suchanfragen parsen und ausführen	17 Stunden
	Zugriffskontrolle umsetzen	4 Stunden
	In regelmäßigen Abständen Projekte auf Änderungen prüfen und indexieren	1 Stunde
	Web-Oberfläche	6 Stunden
4	**Dokumentation**	**12 Stunden**
	Kosten-Nutzen-Nachbetrachtung	1 Stunde
	Erstellung der Entwicklerdokumentation	2 Stunden
	Erstellung des Projektdokumentation	9 Stunden
–	**Summe**	**80 Stunden**

Das Projekt wurde in vier Phasen gegliedert, um einen strukturierten Arbeitsablauf sicherzustellen und den Projektfortschritt genau verfolgen zu können. Jede Projektphase wurde in mehrere Arbeitsabschnitte, beziehungsweise Meilensteine unterteilt, um für eine bessere Nachvollziehbarkeit zu sorgen.

Die erste Phase diente der allgemeinen Analyse. Hierbei wurde unter anderem eine Ist-Analyse und Soll-Analyse durchgeführt, um die Ausgangssituation des zu lösenden Problems zu ermitteln und das gewünschte Ziel zu formulieren.

Während der Konzeptions- und Entwurfsphase wurden erste Entwürfe für die Datenbankstruktur entworfen und die vorhandenen Schnittstellen beim GitLab angesehen, um einen ersten groben Ablauf der Indexierung ausmachen zu können.
Im Rahmen der Indexierung und Web-Oberfläche wurden sich außerdem erste Gedanken zur Syntax für Suchanfragen gemacht und festgehalten.

In der Implementierungsphase wurden die einzelnen Programmteile implementiert und teilweise mit automatisierten Tests abgedeckt, um während der Entwicklung und auch in Zukunft eine gewisse Programmstabilität gewährleisten zu können.
In der letzten Phase, der Dokumentationsphase, wurden die Dokumentationen für Benutzer und Entwickler verfasst. Diese sollen die Grundlage für eine bessere Nachvollziehbarkeit und Wartbarkeit bilden, so dass sich zukünftige Benutzer und Entwickler in der Anwendung zurechtfinden.

Die Gesamtzeit, die für das Projekt aufgewendet wurde, betrug 80 Stunden.

6.2 Ressourcenplanung

Während der Projektarbeit wurden die nachfolgenden Ressourcen verwendet. Dabei wurde zwischen Personal-, Hard- und Softwareressourcen unterschieden:

- Personalressourcen
    - **Auszubildender Fachinformatiker für Anwendungsentwicklung** – Umsetzung des Projekts
    - **Systemadministrator** – Bereitstellung einer Virtuellen Maschine als Zielsystem
    - **Anwendungsentwickler / DevOps** – Abnahme der Interaktionen mit GitLab
- Hardwareressourcen
    - **Notebook**
- Softwareressourcen
    - **JetBrains WebStorm** – Entwicklungsumgebung
    - **Visual Studio Code** – Entwicklungsumgebung
    - **Linux** – Betriebssystem
    - **Git** – Versionskontrollsystem
    - **Docker** – Container-Platform
    - **Node.js** und installierte **Projekt-Bibliotheken** (siehe Entwicklerdoku)
    - **PostgreSQL** – Datenbankmanagementsystem

Im Hinblick auf anfallende Kosten wurde darauf geachtet, dass die Nutzung der Software kostenfrei ist oder die Lizenzen dem Unternehmen bereits zur Verfügung stehen. Dadurch konnten die Projektkosten auf ein Minimum gehalten werden.

6.3 Entwicklungsprozesse

Vor Beginn der Projektumsetzung musste sich für einen geeigneten Entwicklungsprozess entschieden werden, der die Vorgehensweise bei der Entwicklung festlegt. Nach Abwägung der Vor- und Nachteile verschiedener Entwicklungsprozesse entschied ich mich für das Kanban-System.

Beim Kanban werden Aufgabenpakete definiert, die unterschiedliche Phasen durchlaufen. Im Rahmen dieser Projektarbeit sind die Phasen *TODO*, *In Arbeit*, *Test*, *Fertig* durchlaufen worden.
Die Arbeitspakete habe ich aus meiner vorangestellten Zeitplanung definiert und vereinzelte weitere Arbeitspakete angelegt, wenn bestimmte Baustellen aufgefallen sind.

Es ist relativ nah an der agilen Scrum-Variante, die wir typischerweise in der Entwicklung einsetzen. Der größte Unterschied liegt darin, dass es kein Backlog und keine Sprints beim Kanban gibt.

Ich habe mich für das Kanban entschieden, da ich der einzige Entwickler am Projekt bin und ein recht enger Zeitrahmen vorliegt, dass die Unterteilung in Sprints keinen großen Mehrwert geboten hätte, dafür den Prozess aber komplexer gestaltet hätte.

7. Analysephase

7.1 Ist-Analyse

Die durchgeführte Ist-Analyse hat gezeigt, dass es aktuell keinen guten Weg gibt, einen Überblick über die Nutzung einer bestimmten Komponente oder Klasse zu erhalten.
Ein “Ausschnitt” über die mögliche Verwendung in aktuelleren Kundenprojekten ist mit entsprechendem Zeitaufwand möglich, durch das Durchsuchen von eingerichteten Projekten auf dem internen Entwicklungsserver.

Das Durchsuchen ist also zeitaufwändig und liefert ein unvollständiges Bild, welches sich nur mit noch mehr Zeitaufwand verfeinern lässt, jedoch nie vervollständigen.

7.2 Soll-Analyse

Die Soll-Analyse hat klargestellt, dass man über alle Kundenprojekte suchen möchte und am liebsten auch komplexere Anfragen mit logischen Operationen, um zum Beispiel ähnliche Schreibweisen für die gleichen Komponenten zu suchen.
Die Suchanfragen müssen schnell gestellt und schnell beantwortet werden, um einen ausreichenden Mehrwert zu liefern.
Eine langsame Suche, ob manuell oder automatisiert, möchte niemand öfter bedienen als notwendig.

Das Projekt soll als Container auf das Zielsystem ausgerollt werden, um so unabhängig vom System agieren zu können und flexibel Beispielsweise auf eine neuere Node.js oder PostgreSQL Version updaten zu können.
Solche Updates können dann von einem Entwickler durchgeführt werden, ohne den komplexen Prozess eines IT-Tickets zu durchlaufen.

7.3 Wirtschaftlichkeitsanalyse

7.3.1 Projektkosten

Im Rahmen der Projektumsetzung werden Kosten in Höhe von 2772,50 EUR erwartet. Diese erwarteten Kosten setzen sich aus den Personal- und Ressourcenkosten zusammen, wobei die internen Stunden- und Ressourcenkosten nicht öffentlich gemacht werden dürfen.

Aus diesem Grund wurden die folgenden Personalkosten von 20,00 EUR für Auszubildende und 50,00 EUR für Mitarbeiter pro Stunde angesetzt.
Die angesetzten Kosten für Ressourcen betragen pro Person 12,50 EUR pro Stunde.

Vorgang	Mitarbeiter	Zeit	Personal	Ressourcen	Gesamt
Projekt-umsetzung	1x Auszubildender	80 Stunden	1600 EUR	1000 EUR	2600 EUR
Bereitstellung Zielsystem	1x Systemadministrator	2 Stunden	100 EUR	25 EUR	125 EUR
Abnahme GitLab Interaktionen	1x Auszubildender 1x DevOps	½ Stunden	35 EUR	12,50 EUR	47,50 EUR
Gesamt					**2772,50 EUR**

7.3.2 Amortisationsdauer

Die Amortisationsdauer ist ein wichtiger Faktor bei der Bewertung der Rentabilität eines Projekts. Sie gibt an, wie lange es dauert, bis die Einsparungen oder Vorteile, die ein Projekt bietet, die Kosten des Projektaufwands ausgleichen.

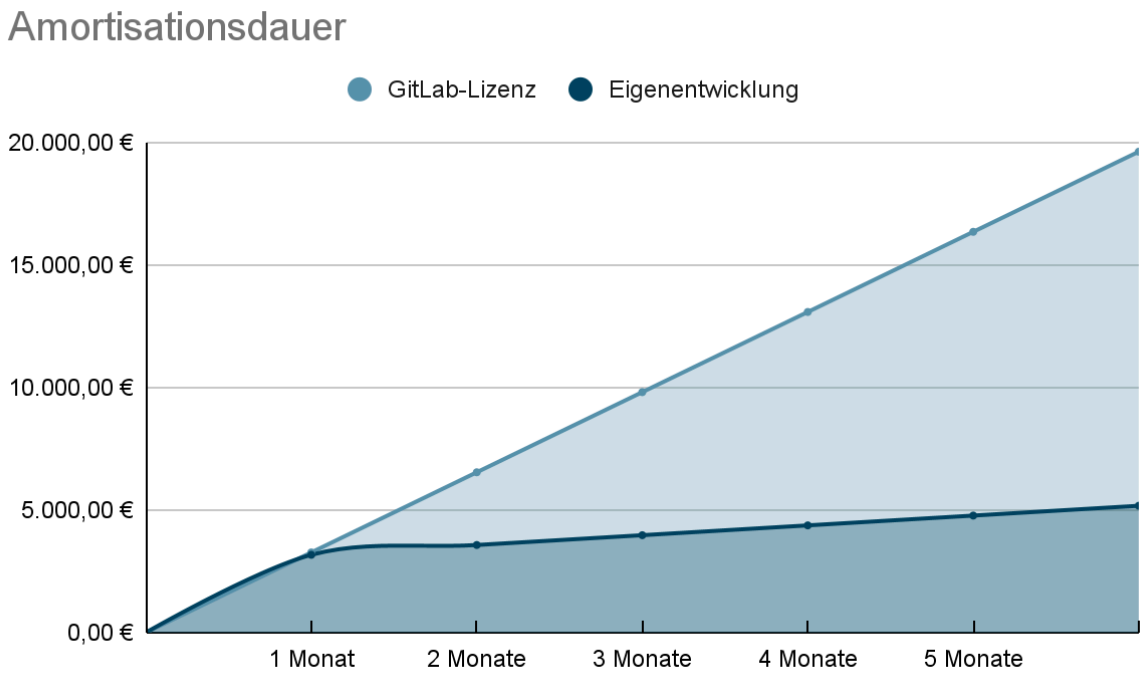
Mit der Projektarbeit wird darauf abgezielt, die Produktivität in bestimmten Entwicklungsphasen bzw. /-Vorgängen zu erhöhen und gleichzeitig die hohen Lizenzkosten zu vermeiden, die mit der offiziellen GitLab-Lösung einhergehen würden.
Die Erhöhung der Produktivität kann hierbei beiden Lösungen angerechnet werden, daher wird sie nachfolgend vernachlässigt.

Bei Vernachlässigung einer erhöhten Produktivität ergeben sich somit die Entwicklungskosten gegenüber den GitLab-Lizenzkosten, zur Betrachtung der Amortisationsdauer.
Auch die Kosten verursacht durch Hosting können vernachlässigt werden, da sowohl unsere GitLab-Instanz als auch die selbstentwickelte GitLab-Suchmaschine auf eigenen Systemen betrieben wird – Die potentiellen Kostenunterschiede können vernachlässigt werden, da keine zusätzliches System oder Hardware angeschafft werden müssten.

Rechnet man zu den einmaligen Entwicklungskosten des Projektes monatlich noch 8 Mitarbeiterstunden für gelegentliche Wartung, entstehen zusätzlich **400 EUR pro Monat** für die Wartung.

Die günstigste GitLab-Lizenz, welche die gewünschte Suchfunktion bereits enthält, kostet **29 USD pro Nutzer pro Monat**.
Die genaue Benutzerzahl in unserem internen System darf nicht genannt werden, daher rechnen wir mit 120 kostenpflichtigen Benutzern, was uns im Monat **3480 USD** an Lizenzkosten verursachen würde. Beim aktuellen Wechselkurs von 0.9407 entspricht das etwa **3273,50 EUR**.

Natürlich kommt die GitLab-Lizenz mit weiteren Funktionen als nur der Suche, jedoch stand auch mit dem vollen Funktionsumfang ein Lizenzkauf in der jüngsten Vergangenheit bereits außer Frage.
Somit ergeben sich Lizenzkosten für **GitLab von etwa 3273,50 EUR pro Monat** und für die selbstentwickelte Lösung einmalige Entwicklungskosten von ca. **2772,50 EUR** und zusätzlich **400 EUR monatlich** für die Wartung.
Daraus ergibt sich eine **Amortisationsdauer von etwas unter einem Monat**.
Dies bedeutet, dass die Projektaufwandskosten in nur wenigen Wochen ausgeglichen sind und somit auch mit fortlaufender Wartung der Anwendung rentabel ist und zur Produktivitätssteigerung beiträgt.



Dem Graphen lässt sich gut ablesen, dass zu Beginn die Entwicklungskosten die Lizenzkosten übersteigen, jedoch nach dem ersten Monat bereits ein klarer Trend der Kostenentwicklung zu erkennen ist.

7.4 Lastenheft

Der folgende Auszug aus dem Lastenheft beschreibt alle mit dem Auftraggeber vereinbarten Punkte

7.4.1 Funktionale Anforderungen

- Die Anwendung lädt automatisch Projekte herunter und indexiert diese
- Die Anwendung ist in der Lage Änderungen in Projekten automatisch zu erkennen
- Die Leseberechtigungen der Benutzer im GitLab werden berücksichtigt
- Mit der GitLab-Instanz soll ressourcensparend interagiert werden

7.4.2 Nicht-Funktionale Anforderungen

- Hohe Stabilität und Performance der Anwendung
- Einfache Wartbarkeit der Anwendung
- Benutzerfreundlich und Intuitiv bedienbar
- Wichtige Werte sind einfach konfigurierbar

8. Entwurfsphase

8.1 Syntax für Suchanfragen

Die Konzeption der Suchsyntax verlief schnell und unkompliziert, da bereits feststeht, was gesucht werden kann. Der Benutzer soll in der Lage sein, sowohl einzelne Wörter zu suchen, als auch ganze Sätze bzw. Codeausschnitte.
Um dies ermöglichen zu können, braucht es besondere Zeichen, wie Anführungszeichen für Texte mit Leerzeichen, und eine Möglichkeit, diese besonderen Zeichen ohne ihre besondere Bedeutung ebenfalls benutzen zu können (Escaping).
Außerdem wäre es praktisch, wenn einzelne Such-Bedingungen sich logisch mit einem “Oder” verknüpfen lassen, umso umfangreichere Suchen mit einer Abfrage zu ermöglichen.

Aus diesen Anforderungen ist folgende Syntax/Grammatik, dargestellt in der *Erweiterte Backus-Naur-Form* (EBNF), entstanden:

```
nonWhitespaceChar ::= (any printable character except ' ')
escapedChar ::= '\' (any printable character)
char ::= nonWhitespaceChar | escapedChar

regex ::= '/' char+ '/'
qualifier ::= char+ ':' (char+ | '"' char+ '"' | regex)
word ::= char+ | '"' char+ '"' | qualifier | regex

term ::= word | '(' andOperation+ ')'
orOperation ::= term ('||' term)*
andOperation ::= orOperation ('&&' orOperation)*

query ::= andOperation+
```

Dieser Grammatik lassen sich vereinfacht die Bestandteile Text, RegEx, Qualifier und logische Operatoren wie “Und”, “Oder” und Klammern herauslesen.
Qualifier sollen hier eine Möglichkeit schaffen, auf Basis von Dateieigenschaften (Metadaten), wie zum Beispiel Dateipfad oder /-endung, zu filtern.
Eine detaillierte Erklärung der einzelnen Such “Bausteine” ist dem Benutzerhandbuch im Anhang zu entnehmen.

8.2 Implementierungsphase

8.2.1 Suchanfrage parsen

Das Parsen der Suchanfrage stellte sich als schwieriger heraus, als gedacht, da in der Planung die Komplexität der Syntax unterschätzt wurde.

Zuerst durchläuft die Suchanfrage den Tokenizer, welcher die einzelnen Zeichen durchgeht und gruppiert und jeweils mit der Information versieht, um was für einen Token es sich handelt. Ein Token könnte hierbei zum Beispiel “Öffnende Klammer” oder “Word” sein (siehe 8.1 Syntax für Suchanfragen).

Anschließend werden die extrahierten Token durch den Parser verarbeitet, welcher einen abstrakten Syntaxbaum (AST) generiert. Es handelt sich hier um einen *rekursiven Abstiegsparser*, der rekursiv die für den Token entsprechenden Parse-Methoden aufruft und so die verschachtelte Struktur des AST korrekt generieren kann. Hier hat eine einfache For-Schleife nicht gereicht, da sich Beispielsweise Klammern verschachteln lassen, was die Struktur komplexer macht.
Hier geschieht die tatsächliche Interpretation der Suchanfrage und die Prüfung auf syntaktische Korrektheit – Also, ob zum Beispiel alle geöffneten Klammern korrekt geschlossen wurden.

Ist der Syntaxbaum korrekt generiert, ist das Weiterverarbeiten im Vergleich sehr einfach, da der Baum nur rekursiv durchlaufen werden muss, um ihn zum Beispiel in ein SQL-Query umzuwandeln, welches der Datenbank zur Ausführung gegeben werden kann.

8.2.2 GitLab-Projekte indexieren

Bei der Implementierung der Indexierung muss besonders der Ressourcenverbrauch der Anwendung beachtet werden. Es ist mit größeren Projekten zu rechnen, die viele Dateien oder auch große Dateien enthalten.
Während der Indexierung muss der Arbeitsspeicherverbrauch bewusst kontrolliert werden und auch mit vielen Dateien in einem Projekt muss zuverlässig umgegangen werden. Zusätzlich kommt hinzu, dass während der Indexierung auch noch Suchanfragen gestellt werden können, die keine unvollständigen Dateien angezeigt bekommen sollten.

Um die Probleme zu adressieren, habe ich mich dazu entschieden ein Dateigrößen-Limit einzuführen, welches bei 25 MiB liegt, was für Textdateien sehr großzügig ist.
Vom GitLab lasse ich mir die aktuellen Dateien als Zip-Archiv geben, und aus diesem beziehe ich die einzelnen Dateien und kann bewusst Dateien überspringen, wenn die Dateigröße oberhalb des Limits liegt.

Die Datenbankoperation selber habe ich in eine Datenbank-Transaktion gepackt, welche darauf achtet, keine strikten “Locks” zu erschaffen. Andernfalls würden eintreffende Suchanfragen auf die Fertigstellung der Indexierung warten müssen, sobald die Anwendung beginnt, eine Datei zu aktualisieren. Da eine Indexierung bei großen Projekt durchaus über eine Minute dauern kann, war das wichtig zu berücksichtigen.

Die Datenbank-Transaktion stellt außerdem die Datenintegrität sicher, sollte es zu einem Fehler oder Programmabsturz während der Indexierung kommen. In so einem Fall würde Postgres die Änderungen in der Datenbank-Transaktion rückgängig machen, statt eine unfertige Indexierung gespeichert zu haben.

9 Fazit

9.1 Soll-/Ist-Vergleich

Die nachfolgende Tabelle zeigt eine Aufstellung der ursprünglichen Problemstellung und dem Projektziel und ob diese wie geplant erreicht wurden und das bisherige Problem lösen oder wo noch Handlungsbedarf besteht.

Kriterium	**IST**	**SOLL**	**Abweichung**
Suchanfragen lassen sich schnell stellen	Einzelne Wörter/Texte lassen sich schnell stellen	Auch komplexe Anfragen lassen sich mit logischen Operationen und RegEx-Pattern formulieren	Erfüllt
Suchanfragen werden schnell beantwortet	Jede Datei muss für jede Anfrage neu gelesen und durchsucht werden	Textdateien werden Indexiert in der Datenbank hinterlegt und ermöglichen so ein schnelles Durchsuchen	Weitestgehend Erfüllt Komplexe Anfragen können durchaus bis zu 12 Sekunden dauern
Alle relevanten Projekte lassen sich für ein Gesamtbild durchsuchen	Nur aktuell, zur Entwicklung eingerichtete Projekte, lassen sich durchsuchen	Alle relevanten GitLab-Projekte werden indexiert und zur Suche bereitgestellt	Erfüllt
Leseberechtigungen werden Berücksichtigt	Es greifen die Berechtigungen auf dem Entwicklungsserver	Der Benutzer muss sich mittels GitLab einloggen und die Leserechte des Accounts werden berücksichtigt	Erfüllt
Suchanfragen können logische Verbindungen aufstellen	Nur mittels umständlichen RegEx	Syntax der Suchanfrage ermöglicht logische Verbindungen und mehr	Erfüllt

Ein möglicher Handlungsbedarf wird hier bei der Geschwindigkeit der Bereitstellung von Suchanfragen sichtbar. Sobald die Anwendung unter Realbedingungen lief, zeigte sich, dass die Wartezeit auf Suchergebnisse höher war als erwartet.
Die Wartezeit für einfache Suchanfragen ohne unterschiedlichen logische Operatoren in der Suchanfrage ist dennoch gering geblieben, weshalb ich auf "Weitestgehend Erfüllt” komme.

Einen weiteren Blick sollten wir auf die Zeitplanung aus *6.1 Projektphasen* legen und diese mit der tatsächlich benötigten Zeit vergleichen.

In der nachfolgenden Tabelle wurden Differenzen zwischen der geplanten und tatsächlich benötigten Zeit, die größer als ± *1 Stunden* sind, farblich hervorgehoben. Grün steht für eingesparte Zeit und Rot für einen unerwartet hohen Zeitaufwand.
Kleinere Abweichungen von der Planung sind zu erwarten und aus diesem Grund nicht hervorgehoben.

Phase	**Aufgabe**	**Eingeplante Zeit**	**Zeit tatsächlich benötigt**	**Differenz**
1	**Analyse**	**7 Stunden**	**6,00 Stunden**	**- 1,00**
	Ist-Analyse	1 Stunde	0,75 Stunden	- 0,25
	Soll-Analyse	1 Stunde	0,75 Stunden	- 0,25
	Kosten-Nutzen-Analyse	1 Stunde	0,75 Stunden	- 0,25
	Definition der Suchanfragen-Syntax	2 Stunden	1,75 Stunden	- 0,25
	Formulierung des Lastenheftes inklusive Ressourcenplanung	2 Stunden	2,00 Stunden	± 0,00
2	**Konzeption und Entwurf**	**9 Stunden**	**8,25 Stunden**	**- 0,75**
	Indexierung und dazugehörige Datenbank-Struktur	4 Stunden	3,75 Stunden	- 0,25
	Zugriffskontrolle und dazugehörige Datenbank-Struktur	2 Stunden	2,50 Stunden	+ 0,50
	Mögliche Interaktion mit Git und GitLab APIs prüfen	1 Stunde	1,25 Stunden	+ 0,25
	Web-Oberfläche und dazugehörige Endpunkte	2 Stunden	0,75 Stunden	- 1,25
3	**Implementierung**	**52 Stunden**	**54,25 Stunden**	**+ 2,25**
	Indexer: Quellcode und Commits eines GitLab-Projektes indexieren	21 Stunden	10,00 Stunden	- 11,00
	Indexer: (Datei-)änderungen in GitLab-Projekten effizient erkennen	3 Stunden	2,25 Stunden	- 0,75
	Suchanfragen parsen und ausführen	17 Stunden	24,25 Stunden	+ 7,25
	Zugriffskontrolle umsetzen	4 Stunden	9,75 Stunden	+ 5,75
	In regelmäßigen Abständen Projekte auf Änderungen prüfen und indexieren	1 Stunde	0,75 Stunden	- 0,25
	Web-Oberfläche	6 Stunden	7,25 Stunden	+ 1,25

Phase	Aufgabe	Eingeplante Zeit	Zeit tatsächlich benötigt	Differenz
4	**Dokumentation**	**12 Stunden**	**11,50 Stunden**	**- 0,50**
	Kosten-Nutzen-Nachbetrachtung	1 Stunde	0,75 Stunden	- 0,25
	Erstellung der Entwicklerdokumentation	2 Stunden	1,50 Stunden	- 0,50
	Erstellung des Projektdokumentation	9 Stunden	9,25 Stunden	+ 0,25
–	**Summe**	**80**	**80 Stunden**	**± 0,00**

Es wird sichtbar, dass das Projekt insgesamt in der geplanten Zeit fertiggestellt werden konnte, es jedoch große Abweichungen während der Implementierungsphase gab.
Die Implementierung der Indexierung ließ sich über elf Stunden schneller realisieren, als erwartet.

Jedoch benötigte die Implementierung von “Suchanfragen parsen und ausführen” über sieben zusätzliche Stunden, die nicht eingeplant waren.
Dies ist auf die komplexe Suchanfragen-Syntax zurückzuführen, die unterschätzt wurde. Zusätzlich kommt hinzu, dass dies mein erster Kontakt mit komplexeren Parsen war, die sich nicht mit einer simplen Schleife gut abbilden lassen.
Es waren daher zusätzliche Recherchen und Prototypen erforderlich.

Auch die Implementierung der Zugriffskontrolle erwies sich als umfangreicher als zu Beginn angenommen und erforderte fast sechs zusätzliche Stunden.
Hier lag die Problematik darin, dass es keine gute Schnittstelle seitens GitLab gibt, um für einen Benutzer alle Projekte, auf die er Zugriff hat, zu listen.
Dies ist ausschließlich mit dem API-Zugriffstoken möglich, den wir beim Login mittels OAuth 2.0 erhalten, da wir mit diesem Token Anfragen im Namen des Benutzers stellen können. Hiermit ist es dann möglich, alle Projekte zu listen, die der Benutzer selbst sehen kann.
Dies bedeutet jedoch, dass die Zugriffsberechtigungen erst nach einem erfolgreichen Login abgefragt werden können und den Login-Vorgang verzögert.

9.2 Ausblick in die Zukunft

In Zukunft können weitere Verbesserungen und Erweiterungen am Projekt vorgenommen werden. Die Benutzeroberfläche ist aktuell sehr einfach und könnte ein frisches Design verkraften.
Auch wäre es denkbar die Suche, nicht nur Dateien, sondern auch zum Beispiel Commits indexieren zu lassen.

Die Wartung der Anwendung sollte ebenfalls kein Problem darstellen, da wenige Bibliotheken installiert wurden und darauf geachtet wurde, dass diese auch in Zukunft eine hohe Chance auf Updates haben.
Die Testabdeckung sollte bei Gelegenheit jedoch erhöht werden, um mögliche zukünftige Entwicklungen zu unterstützen und die Stabilität zu gewährleisten.

10 Anhang

10.1 Zusätzliche Abbildungen

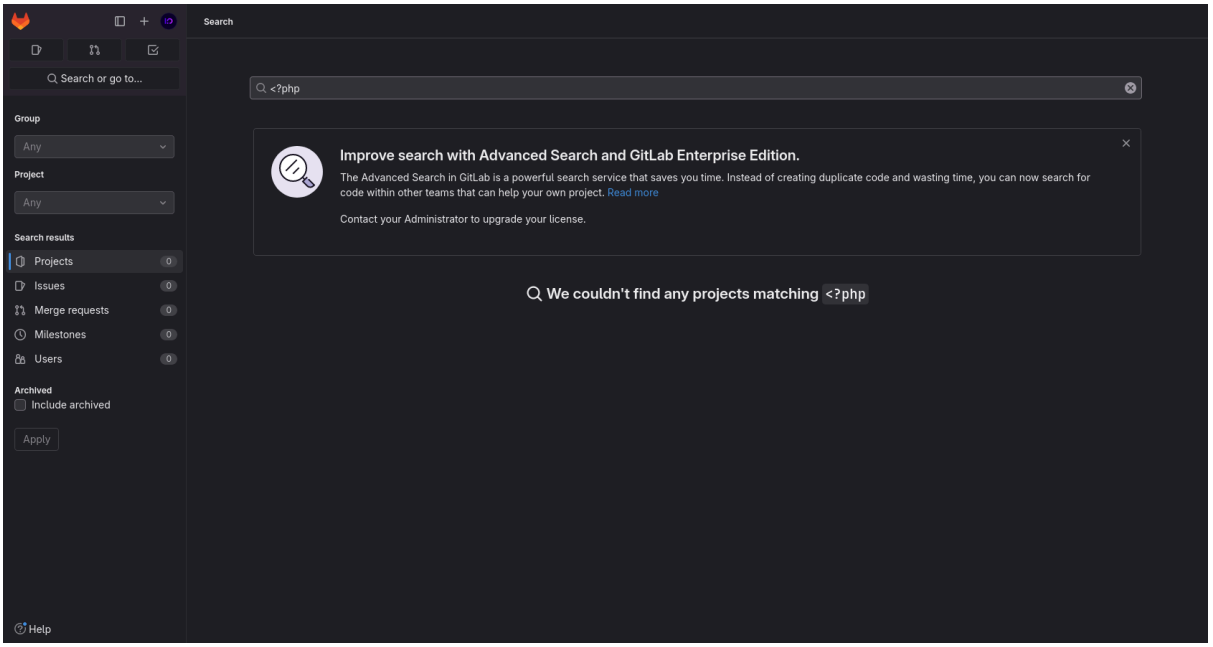


Abbildung 1: Die Suchfunktion im GitLab, die keine Dateien durchsuchen kann

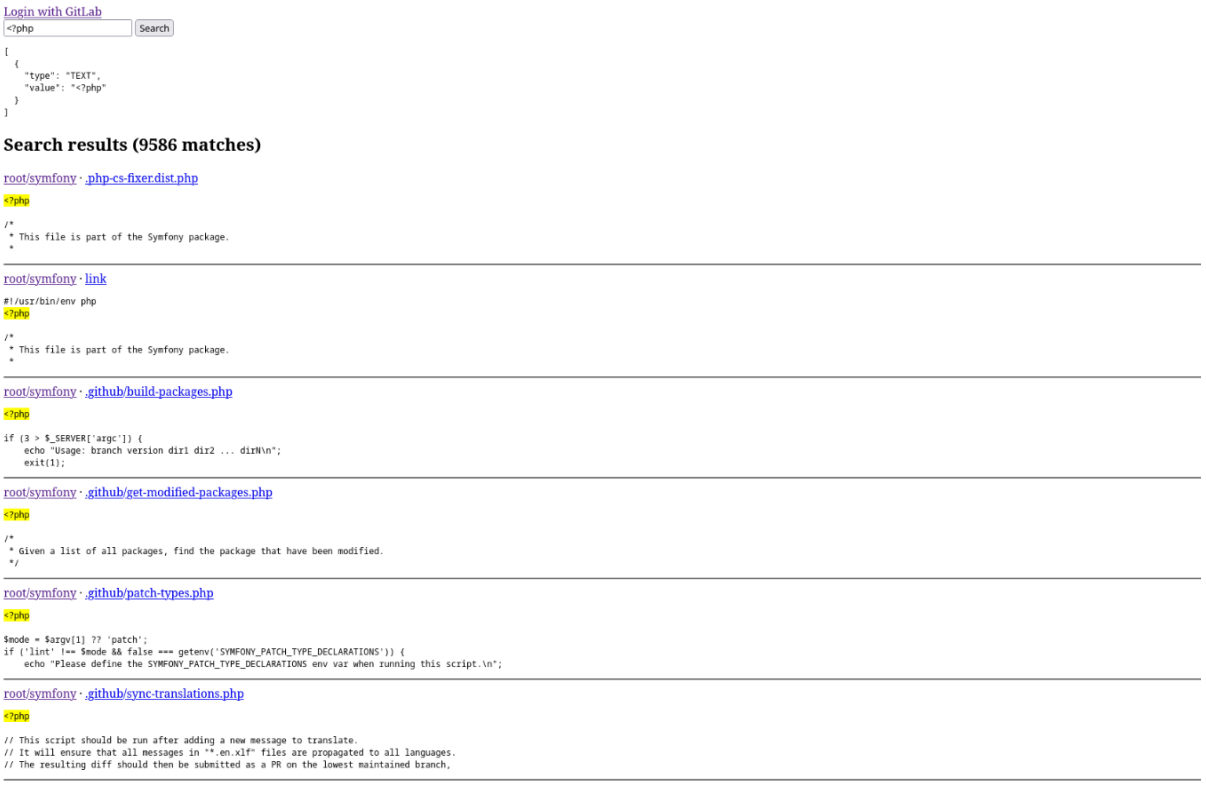


Abbildung 2: Eine, zu *Abbildung 1* equivalente, Suchanfrage in der entwickelten GitLab-Suchmaschine

10.2 Kundendokumentation

Die GitLab-Suchmaschine ist eine Web-Anwendung, die es ermöglicht, den Quellcode von in GitLab versionierten Projekten zu durchsuchen. Hierfür kann eine umfangreiche Suchsyntax verwendet werden, die es ermöglicht, gezielt nach Dateien zu suchen, die bestimmte Kriterien erfüllen.

In der nachfolgenden Dokumentation wird erklärt, wie die Suchmaschine verwendet wird, wie Suchanfragen formuliert werden können und welche Funktionen die Anwendung bietet.

10.2.2 Benutzerhandbuch

Das Benutzerhandbuch richtet sich an den Endnutzer der GitLab-Suchmaschine und erklärt, wie die Anwendung verwendet wird.
Sie richtet sich an Anwendungsentwickler und setzt Grundkenntnisse von GitLab und Kenntnis über die Web-Adresse der GitLab-Suchmaschine voraus.

Anmeldung und Profilverwaltung

Um die GitLab-Suchmaschine zu verwenden, müssen Sie sich mit Ihrem GitLab-Account anmelden. Dies ist notwendig, um die Benutzerberechtigungen abzufragen und sie bei der Ausgabe der Suchergebnisse zu berücksichtigen.

Klicken Sie hierzu auf den "Login mit GitLab" Link oben auf der Seite (siehe Anhang Abbildung 2) und folgen Sie den Anweisungen. Sie werden hierfür zu unserer GitLab-Instanz weitergeleitet und müssen die Anmeldung bestätigen.
Die GitLab-Suchmaschine speichert keine Zugangsdaten und erhält nur Zugriff auf Daten, die bei der Anmeldung genehmigt wurden.

Nutzung der Suchfunktion

In diesem Unterabschnitt wird die Nutzung der Suchfunktion im Detail erklärt.
Die GitLab-Suchmaschine verwendet eine eigene spezielle Syntax, um Suchanfragen zu formulieren. Diese Syntax besteht aus einfachen "Bausteinen", die kombiniert werden können, um komplexe Suchanfragen zu formulieren.

Grundsätzlich ignoriert die Suchmaschine die Groß- und Kleinschreibung.

Sonderzeichen in Suchanfragen

Bevor wir zu den einzelnen Bausteinen kommen, erkläre ich zuerst Sonderzeichen, die in Suchanfragen fehlinterpretiert werden könnten.

Backslash

Ein Backslash (`\`) wird verwendet, um das darauf folgende Zeichen zu escapen.
Escapen bedeutet in diesem Fall, dass das folgende Zeichen als normaler Text interpretiert wird und Teil des aktuellen Bausteins ist.
Das bedeutet, wenn nach einem Backslash gesucht werden soll, muss es grundsätzlich mit einem weiteren Backslash escaped werden (`\\`).

Beispiele

- `\\` wird als `\` interpretiert
- `\"` wird als `"` interpretiert
- `\ ` wird als ` ` (Leerzeichen) interpretiert

Anführungszeichen

Anführungszeichen (`"`) werden verwendet, um Texte aus mehreren Wörtern als einen Baustein zu interpretieren.
So kann nach genau einem bestimmten Satz oder Code-Snippet gesucht werden, statt nach Dateien mit den einzelnen vorkommenden Wörtern.

Beispiele

- `Katzen sind toll` sucht nach Dateien, die die Wörter “Katzen”, “sind” und “toll” enthalten
- `"Katzen sind toll"` sucht nach Dateien, die die Zeichenfolge “Katzen sind toll” enthalten
- `"\"foo\" bar"` sucht nach Dateien, die die Zeichenfolge ““foo” bar” enthalten

Leerzeichen

Leerzeichen werden grundsätzlich als Trennzeichen zwischen Bausteinen interpretiert.
Sie müssen also mittels Backslash escaped werden oder von Anführungszeichen umschlossen sein, wenn sie als Text erkannt werden sollen.

Textbausteine

Textbausteine bestehen aus einem oder mehreren Zeichen und können mithilfe von Anführungszeichen oder Backslashes auch aus mehreren Wörtern bestehen.
Sie stellen die einfachste und intuitivste Art dar, nach Dateien zu suchen.

Beispiele

- `Katze` sucht nach Dateien, die das Wort “Katze” enthalten
- `"Katzen sind toll"` sucht nach Dateien, die die Zeichenfolge “Katzen sind toll” enthalten

Logische Operatoren

- `&&` wird als logisches “Und” interpretiert und verknüpft zwei Bausteine
- `||` wird als logisches “Oder” interpretiert und verknüpft zwei Bausteine
- Klammern (`( )`) können verwendet werden, um die logischen Bedingungen noch feiner zu steuern

Beispiele

- Katze && Hund sucht nach Dateien, die sowohl “Katze” als auch “Hund” enthalten
- `Katze || Hund` sucht nach Dateien, die entweder “Katze” oder/und “Hund” enthalten
- `Katze && (Hund || Maus)` sucht nach Dateien, die “Katze” und entweder “Hund” oder “Maus” enthalten

Reguläre Ausdrücke (Regex-Pattern)

Einigen Entwicklern wird bekannt sein, wie mächtig ein Regex-Pattern sein kann, um nach bestimmten Zeichenfolgen und Mustern zu suchen.
Es sollte bedacht werden, dass die Verwendung von Regex-Pattern die Suche stark verlangsamen kann.

Die GitLab-Suchmaschine unterstützt die Verwendung von solchen Regex-Pattern, indem sie von Slashes (`/`) umschlossen werden.
Bei dem Pattern muss es sich um eine gültige “*POSIX regular expression*” handeln.

Grundsätzlich sollten vereinfachte Pattern aus bekannten Programmiersprachen funktionieren. Bei Zweifel oder Problemen empfehle ich folgende Dokumentation:
PostgreSQL: Documentation: 16: 9.7. Pattern Matching[1]

Beispiele

- `/Katzen-Nummer [0-9]+/` sucht nach Dateien, die die Zeichenfolge “Katzen-Nummer “ und mindestens eine Ziffer enthalten
- `/[0-9a-f]{8}\b-[0-9a-f]{4}\b-[0-9a-f]{4}\b-[0-9a-f]{4}\b-[0-9a-f]{12}/` sucht nach Dateien, die eine UUID enthalten

Qualifier

Ein Qualifier ist ein Baustein, der ein spezielles Attribut einer Datei oder eines Projektes beschreibt.
Mit diesen Qualifiers ist es möglich, die Suche auf Basis von Metadaten einzuschränken.

Der allgemeine Aufbau eines Qualifiers ist “`Attribut:Wert`”, wobei Wert sowohl ein Textbaustein oder ein Regex-Pattern sein kann.

Unterstützte Attribute

- `namespace` bezieht sich auf den Namespace, den das Projekt im GitLab hat
  - **Beispiel**: Einschränken der Suche auf Projekte im `ryze-digital`-Namespace: `namespace:ryze-digital/`
- `path` bezieht sich auf den vollständigen Dateipfad
  - **Beispiel**: Einschränken der Suche auf Dateien im src-Ordner: `path:src/`
- `filename` bezieht sich auf den Dateinamen
  - **Beispiel**: Einschränken der Suche auf Dateien, die “hallo” im Namen haben: `filename:hallo`
- `extension` bezieht sich auf die Dateiendung
  - **Beispiel**: Einschränken der Suche auf PHP-Dateien: extension:php oder `extension:.php`
  - **Beispiel**: Einschränken der Suche auf Twig-Dateien: `extension:twig.html` oder `extension:.twig.html`

Fehlerbehebung und Support

Fehlermeldungen, die in der Webanwendung zu sehen sind, werden automatisch aufgezeichnet und an die Entwickler weitergeleitet.

Sollten konkrete Fragen oder ein Problem auftreten, kann Kontakt per E-Mail an c.koop@ryze-digital.de hergestellt werden oder über ein IT-Ticket.

10.2.3 Entwicklerdokumentation

Die Entwicklerdokumentation richtet sich an Systemadministratoren und Entwickler, die die GitLab-Suchmaschine deployen oder warten wollen.
Die Informationen aus dem Benutzerhandbuch werden vorausgesetzt und lediglich mit weiteren technischen Details ergänzt.

Eingesetzte Technologien (Tech-Stack)

Node.js wurde ausgewählt, da wir im Unternehmen mehrere Entwickler mit JavaScript-Kenntnissen haben und so die Wartung und Weiterentwicklung der Anwendung sichergestellt ist.
TypeScript ist ein Superset von JavaScript, das statische Typisierung ermöglicht und so die Entwicklung und Wartung stabiler und einfacher macht.
PostgreSQL ist bei RYZE Digital bisher noch nicht weit verbreitet, wurde jedoch dennoch ausgewählt, da es im Gegensatz zu MariaDB mehr Funktionen bietet und konform mit SQL-Standards ist. Mitunter bietet es einige Funktionen die für die effiziente Volltextsuche sehr hilfreich sind und in MariaDB nicht oder nur eingeschränkt verfügbar sind

Wichtige Bibliotheken

Ich habe mich gegen den Einsatz eines großen Frameworks entschieden, da ich bisher keines ausprobiert habe und allgemein davon überzeugt war. Um nun auch kein unbekanntes Framework auszuprobieren, habe ich mich für eine Handvoll bekannter Bibliotheken entschieden, die die Grundlage der Anwendung bilden.

Fastify

Fastify ist ein schneller und effizienter Webserver für Node.js, welcher umfangreiche Schnittstellen bereitstellt.
Es wurde ausgewählt, da es eine sehr gute Performance bietet und eine wachsende Community hat, die die Weiterentwicklung sicherstellt und sowohl Dokumentationen als auch (Community-)Plugins vorhanden sind.

Prisma ORM

Prisma ist ein modernes ORM (Object-Relational Mapping) für Node.js und TypeScript, welches es ermöglicht, Datenbankabfragen in TypeScript zu schreiben.
Es erleichtert den Umgang mit Datenbanken, da kein SQL geschrieben und gewartet werden muss. Außerdem bietet es Typsicherheit, die es ermöglicht, Fehler bereits während der Entwicklung zu erkennen.

TSyringe

Bei TSyringe handelt es sich im Grunde um einen Dependency Injection Container für TypeScript.
Die Bibliothek wurde ausgewählt, um mittels Inversion of Control (IoC) die Abhängigkeiten zwischen den einzelnen Komponenten zu reduzieren und auf lange Sicht die Testbarkeit und Wartbarkeit zu verbessern.

Außerdem wird der Programmcode stark vereinfacht, dadurch dass die verschiedenen Abhängigkeiten nicht manuell in den einzelnen Komponenten instanziiert werden müssen.

TSyringe wird von Microsoft entwickelt und hat eine wachsende Community. Im Vergleich zu ähnlichen Bibliotheken für TypeScript, ist TSyringe einfach gehalten und kommt mit weniger Features und ist dadurch auch einfacher zu verwenden.

Sentry SDK

Sentry ist ein Dienst, der es ermöglicht, Fehlermeldungen und Performancedaten zentral zu sammeln und zur Auswertung aufzubereiten.
So können beispielsweise automatisch E-Mails verschickt werden, wenn ein Fehler auftritt.

Bei RYZE Digital ist Sentry bisher noch nicht aktiv im Einsatz, es sind jedoch bereits mehrere Projekte geplant, bei denen ein Einsatz von Sentry geplant ist.
Die GitLab-Suchmaschine geht hier als ein mögliches Beispielprojekt voran und zeigt, was für einen Mehrwert Sentry bieten kann.

Jest und Stryker

Jest ist ein Test-Framework für JavaScript und TypeScript, welches es ermöglicht, Unit- und Integrationstests zu schreiben und auszuführen.
Stryker ist ein Mutationstest-Framework, welches es ermöglicht, die Qualität von Tests zu überprüfen, indem es den Code verändert und schaut, ob die Tests fehlschlagen.

Bei den beiden Bibliotheken handelt es sich um weit verbreitete und bekannte Bibliotheken, mit guten Dokumentationen und einer großen Community.

Entwicklungsumgebung einrichten

Aktuelle Informationen zur Einrichtung der Entwicklungsumgebung gibt es in der `README.md`-Datei des Projektes.

Grundsätzlich muss auf dem Entwicklungssystem Node.js v20 und Docker installiert sein.
Alle weiteren Abhängigkeiten wie eine GitLab-Instanz oder PostgreSQL werden in `dev/` mittels Docker bereitgestellt und können bequem über `./dev/up.sh` gestartet werden.

Die Projektabhängigkeiten lassen sich normal mittels `npm install` installieren und die Anwendung kann mittels `npm run dev` gestartet werden.
Dank `npm run dev` starten die Anwendung automatisch neu, wenn Änderungen am Quellcode vorgenommen werden.

Vor dem ersten Start muss außerdem die `.env.dev` nach `.env` kopiert werden, welche passende Einstellungen für das in `dev/` enthaltene Setup enthält.

Deployment

Das Deployment wurde mittels Docker und Docker-Compose umgesetzt, um eine einfache und konsistente Bereitstellung der Anwendung zu ermöglichen.

Mit Hilfe von Docker können einzelne Anwendungen, wie die GitLab-Suchmaschine oder PostgreSQL-Datenbank in einzelnen isolierten Containern betrieben werden.

Dadurch sind gewisse Details des Zielsystems, wie zum Beispiel die genaue Version von Node.js oder PostgreSQL, nicht mehr relevant,
da die Anwendung in einem eigenen Container läuft und die Abhängigkeiten selbst mitbringt.
Im Projekt-Repository liegt die `Dockerfile`-Datei, um ein Container-Image für die GitLab-Suchmaschine zu bauen. Außerdem kann der Befehl `npm run docker:build` genutzt werden, um das Image zu generieren.

Das Docker-Compose-File liegt in den Händen der Systemadministratoren, da es sich hierbei um eine spezielle Datei mit sensiblen Informationen, wie API-Zugangsdaten handelt.

Konfiguration

Die Konfiguration der GitLab-Suchmaschine erfolgt über Umgebungsvariablen.

Für die Entwicklung ist hier die `.env.dev` im Projekt vorgesehen, welche nach `.env` kopiert und angepasst werden kann. Die Datei wird automatisch beim Programmstart eingelesen.

Für das Produktion-Deployment sollten die Umgebungsvariablen stattdessen mittels Docker-Compose bereitgestellt werden.

Grundlegende Projektstruktur

- Das Datenbank-Schema von Prisma findet sich in prisma/schema.prisma – Alle Anpassungen an der Datenbankstruktur müssen hier vorgenommen werden
- Die Templates für die Web-Oberfläche befinden sich in `resources/views/`
- Der Projektcode ist in `src/` zu finden und verschiedene Module sind in entsprechend benannten Unterordnern zu finden
    - Zum Beispiel findet man die Interpretation der Suchanfragen in `src/search_query/parser/` oder die einzelnen Web-Endpunkte in src/webserver/routes/

Die wichtigsten Module

SearchQuery (Tokenizer, Parser, Highlighter, ...)

Dieses Modul ist mit Abstand das komplexeste und gleichzeitig wichtigste Modul der GitLab-Suchmaschine.
Es ist dafür verantwortlich, Suchanfragen verständlich zu machen und auch Suchergebnisse anhand dieser Anfragen aufzubereiten.

Der Tokenizer und Parser verarbeiten die Nutzereingabe und erstellen daraus einen AST (Abstract Syntax Tree), der die Suchanfrage repräsentiert.
Dieser AST ist sehr einfach weiterzuverarbeiten und wird genutzt, um die Suchanfragen in SQL-Queries umzuwandeln.

Die Testabdeckung dieses Moduls ist und sollte auch in Zukunft 100 % betragen.
Das Parsen von solchen Anfragen ist ein komplexes Thema und Fehler, die die gesamte Anwendung beeinträchtigen, lassen sich hier schnell versehentlich einbauen.

TaskQueue

Die TaskQueue ist sowohl für das Abarbeiten von geplanten Aufgaben, wie dem Indexieren von Projekten, als auch das Planen von solchen Aufgaben.

Bisher existieren nur Tasks, um bestehende und neue Projekte im Hintergrund zu indexieren und auf Änderungen zu überprüfen. Diese Indexierungs-Tasks werden aktuell alle 30 Minuten ausgeführt.

Dateien (FileType-/TextFileDetector)

Dieses Modul dient dazu, den Dateityp einer Datei zu ermitteln. Die GitLab-Suchmaschine möchte ausschließlich Textdateien indexieren und hat kein Interesse an Binärdateien oder Ähnlichem.

Durch dieses Modul sind wir unabhängig von falschen oder fehlenden Dateiendungen. Außerdem ist so keine Liste an bekannten Dateiendungen notwendig, die stetig gepflegt werden müssen. Ermittelt wird ein Mime-Type anhand des Dateiinhalts, oder wenn nicht möglich auf Grundlage der Dateiendung.
Es wurde zusätzlich eine Liste an bekannten MIME-Types hinterlegt, um auch im Sinne des MIME-Types Nicht-Text-Dateitypen wie `application/json` als Textdatei zu erkennen.

Login

Das Login-Modul dient dazu, die einzelnen Schritte der OAuth 2.0 Authentifizierung ausführen zu können, die für den Login mittels GitLab notwendig sind.

OAuth 2.0 ist ein Protokoll, welches uns ermöglicht, uns bei einer Anwendung über einen Drittanbieter zu authentifizieren. So kann die GitLab-Suchmaschine einzelne Nutzer identifizieren und deren Berechtigungen abfragen.