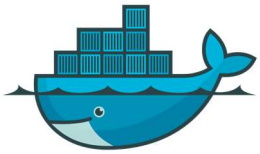
LETTERIX



Modul 347 Projektdokumentation einer Containerized App

Mikka Kummer, Masha Madani

Inhaltsverzeichnis

Projekteinleitung

Die Bewerbung für Lehrstellen ist für viele Sekundarschüler eine große Herausforderung. Sie stehen vor der Aufgabe, ein überzeugendes Bewerbungsschreiben zu verfassen, welches ihre Fähigkeiten und Interessen hervorhebt. Oft fehlt es den Schülern jedoch an Erfahrung und Wissen über die Bewerbungsgestaltung, was zu Unsicherheit und Frustration führen kann. Eine schlecht formulierte Bewerbung kann ihre Chancen auf eine Lehrstelle erheblich beeinträchtigen.

Um Sekundarschülern bei der Bewerbung für Lehrstellen zu helfen, möchten wir eine Applikation entwickeln, welche mit ChatGPT die Schüler dabei unterstützt, qualitativ hochwertige Bewerbungsschreiben zu generieren. Unser Lösungsansatz besteht daraus ein benutzerfreundliches Frontend zu erstellen, welches den Schülern eine einfache und intuitive Benutzeroberfläche bietet, wobei die grösste Komplexität im Hintergrund passiert.

Wir werden verschiedene erlernte Thematiken hier umsetzen, unteranderem die Verwendung von Microservices, um unsere Applikation möglichst flexibel zu gestalten und damit weitere Entwicklung einfach gehalten werden kann, werden wir eine Pipeline konfigurieren, welche die neuesten Änderungen automatisch auf unseren Server pusht.

Schlussendlich ist geplant, dass an der Applikation Letterix weiterlaufend entwickelt wird, wobei die Verwendung von Docker optimal ist. Docker wird es uns ermöglichen eine konsistente Betriebs-, Entwicklungs- und Testumgebung zur Verfügung zu haben, welche unabhängig vom Betriebssystem des Gebrauchers funktionieren wird. Und zu guter Letzt ist uns wichtig, dass wir eine sichere Speicherung von unseren Docker-Images vornehmen, weshalb wir ein Private Registry einrichten möchten.

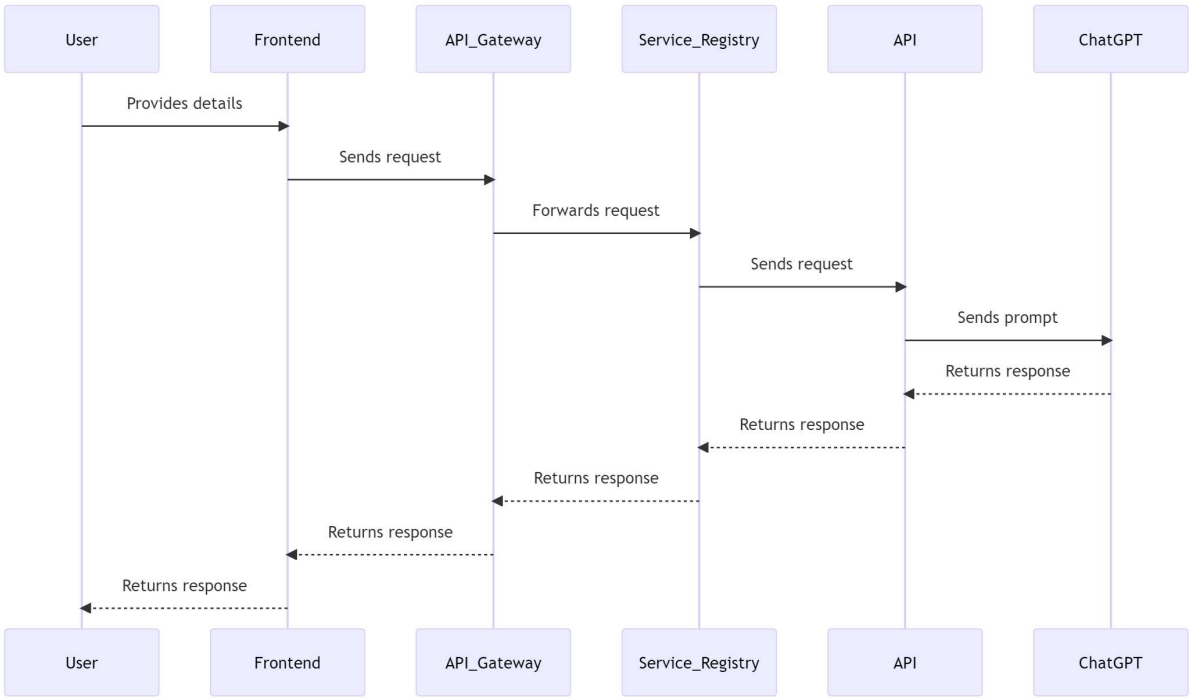


Abbildung 1: Sequenzdiagramm Letterix

Projektauftrag: Containerized App

Projektziel

Das Ziel dieses Projekts ist es, eine containerized App zu entwickeln, die aus mindestens drei unterschiedlichen Containern besteht. Diese Container sollen miteinander interagieren und eine vollständige Applikation aufbauen.

Projektbeschreibung

Das Projekt kann entweder als Gruppen- oder Einzelarbeit durchgeführt werden. Es wird jedoch empfohlen, eine Gruppenarbeit mit 2-3 Lernenden pro Gruppe zu bilden.

Jeder Lernende muss für mindestens einen Container die Hauptverantwortung tragen. Dadurch wird die Aufteilung der Verantwortlichkeiten und die Zusammenarbeit innerhalb des Teams gefördert.

Funktionalitäten der Applikation

Die Applikation muss in der Lage sein, Daten zu erfassen, die durch den Benutzer eingegeben werden.

Die erfassten Daten müssen über das Ende der Container hinaus persistiert werden, um sicherzustellen, dass sie auch nach dem Neubau der Container wieder verfügbar sind.

Beim Neuaufbau der Container müssen die zuvor erfassten Daten wiederhergestellt werden, um eine nahtlose Kontinuität der Benutzererfahrung zu gewährleisten.

Sicherheit der Applikation

Bekannte Sicherheitslücken sollen dokumentiert und so weit wie möglich aufgehoben sein.

Die Abgabe des Projekts ist geplant auf den 11.08.2023.

Planung

Uns ist bewusst, dass die Planung hätte detailliert ausfallen können, wobei wir trotzdem die Deadline erreichen konnten und untereinander selbständig, aber im stetigen Kontakt über unsere Fortschritte.

Anfangs hatten wir eine grobe Planung der Arbeitsschritte, wobei wir durch den Prozess selbständig die spezifischen Schritte ausgemacht haben. In der Entwicklung selber ist es greifbarer, Arbeitsschritte zu definieren und zu erfüllen, vorallem da es uns noch an der nötigen Erfahrung auf verschiedenen Ebenen fehlt, um von Anfang an eine detaillierte Planung zu gestalten, mit den Zeitangaben, welche schwer einzuschätzen sind bei neuen Themen und dem kombinieren von einzelnen erlernten Themen miteinander.

Was schlussendlich zum Erfolg geführt hat, ist dass wir gut alleine unsere Aufgaben mit ihren Verschachtelungen und einzelnen Schritten erarbeiten können und dies auch kompakt einander erklären konnten. Somit konnten wir beide auf Grund des Arbeitspakets von einander lernen und auch weiter entwickeln, falls nötig.

Arbeitspakete & Aufteilung

Arbeitspaket: Frontend-Entwicklung

- React Projektaufsetzung mit dem Konzept «Atomic Design»
- Entwurf der Benutzeroberfläche (UI)
- Umsetzung des Konzepts «Dumb & Smart Components»
- Integration mit dem Backend für den Datenversand; Axios

Aufteilung: Masha – fertig geworden am: 10.07

Arbeitspaket: Backend-Entwicklung (API Gateway Spring Boot Backend)

- Aufsetzen eines Spring Boot-Projekts mit Spring Boot Cloud dependencies für das API-Gateway
- Konfigurieren der Routen und den dazugehörigen Services

Aufteilung: Masha – fertig geworden am: 08.07

Arbeitspaket: Backend-Entwicklung (Service Registry Spring Boot Backend)

- Aufsetzen eines Spring Boot-Projekts mit Spring Boot Cloud und Eureka für das Service Registry
- Konfigurieren des Eureka Clients

Aufteilung: Mikka – fertig geworden am: 08.07

Arbeitspaket: Backend-Entwicklung (Configuration Server)

- Aufsetzen eines Spring Boot-Projekts mit Spring Boot Config Server um die Konfigurationsdateien der Services zu Zentralisieren.

Aufteilung: Masha – fertig geworden am: 08.07

Arbeitspaket: Backend-Entwicklung (Coverletter Service)

- Einrichten eines weiteren Spring Boot-Projekts für den cover-letter REST-Service und die dazugehörigen Endpoints

- Integration der OpenAI-GPT-API für das Generieren des Bewerbungsschreibens

- Implementierung der Datenverarbeitung und -weitergabe an die Chat GPT-API

- Implementierung der Persistierung von generierten Bewerbungsschreiben und den dazugehörigen Daten in einer Postgres Datenbank

Aufteilung: Mikka – fertig geworden am: 07.07

Arbeitspaket: Dockerisierung der Anwendung

- Erstellung eines Dockerfiles für das Frontend

- Erstellung eines Dockerfiles für das Service Registry

- Erstellung eines Dockerfiles für den Cover-letter Service

- Erstellung eines Dockerfiles für den Config Server

- Erstellung eines Dockerfiles für das API-Gateway

- Erstellung einer Docker Compose-Konfiguration für das Zusammenspiel der Container

- Private Docker Registry einrichten

Aufteilung: Mikka & Masha – fertig geworden am: 09.07

Arbeitspaket: Env Files konfigurieren

- ENV Files und Variablen erstellen

Aufteilung: Mikka – fertig geworden am: 10.07

Arbeitspaket: Dokumentation

- Projektauftrag und Beschreibung

- Beschreibung der Architektur und des Funktionsumfangs der Applikation

Aufteilung: Masha – fertig geworden am: 11.07

Arbeitspaket: Testing

- Test-Cases schreiben und durchführen

Aufteilung: Masha – fertig geworden am: 10.07

Arbeitspaket: Deployment auf dem Server

- Bereitstellung der Containeranwendung auf dem Server
- Nginx Konfiguration auf dem Server
- DNS Eintragung in Cloudflare

Aufteilung: Mikka – fertig geworden am: 10.07

Architekturbeschreibung

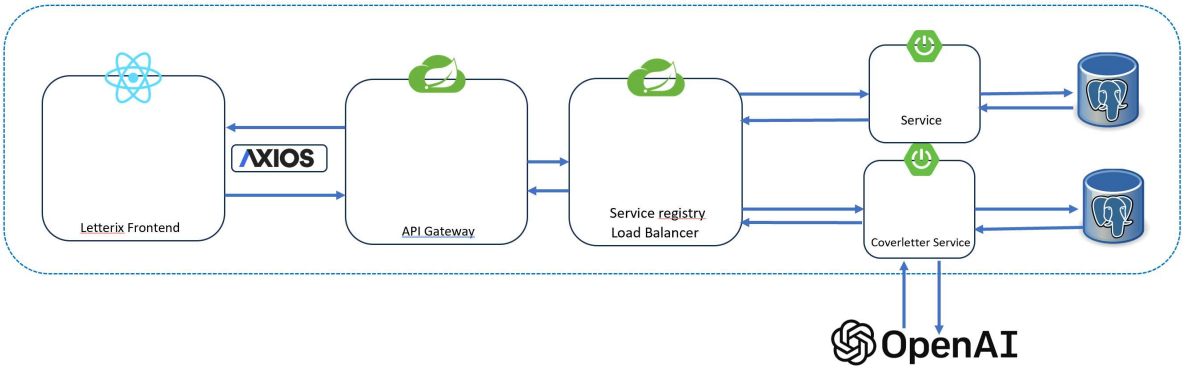


Abbildung 2: Architekturaufzeichnung

Frontend

Unser Frontend für dieses Modul ist einfach und intuitiv gestaltet, hauptsächlich aufgrund von Zeiteinschränkungen, aber auch, um es Bewerbern zu ermöglichen, schnell und einfach ein Bewerbungsschreiben zu erhalten. Dabei haben wir uns an wichtige UI- und UX-Regeln gehalten. Die Benutzeroberfläche ist konsistent und effizient gestaltet, bietet klare Rückmeldungen und gewährleistet eine hohe Zugänglichkeit für Sekundarschüler.

Die wenigen Daten, welche im Frontend eingegeben werden, werden gebraucht, dass unser Backend ein Prompt erstellen kann, welcher schlussendlich an die ChatGPT-API geleitet wird.

Design: Wireframe

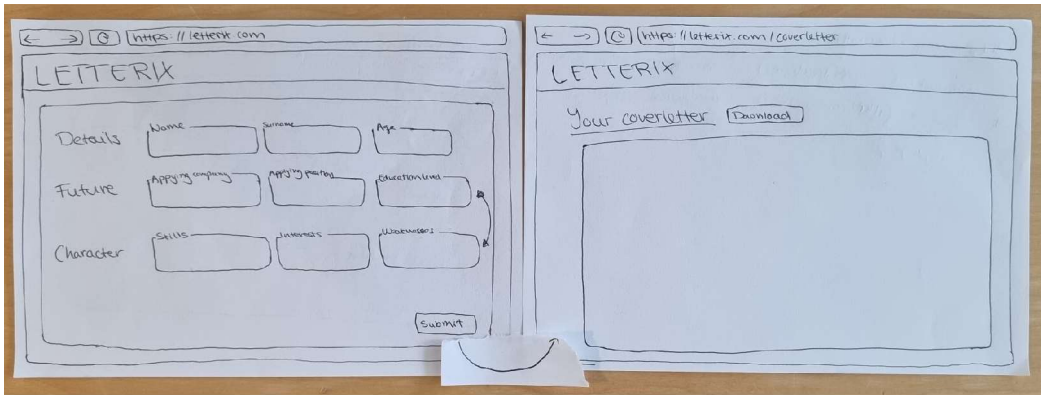


Abbildung 3: Letterix Wireframe

Struktur

Wir haben uns bei der Entwicklung unseres Frontends an die Atomic Design Prinzipien gehalten. Atomic Design ist ein Konzept, welches uns ermöglicht, unser Frontend in kleinere, wiederverwendbare Komponenten zu zerlegen, die in einer hierarchischen Struktur organisiert sind.

Ausserdem haben wir uns an das Prinzip der Smart & Dumb Komponenten gehalten. Smart Komponenten sind verantwortlich für die Logik und die Datenmanipulation, während Dumb Komponenten sich auf die Darstellung und das Rendern der Daten konzentrieren. Diese Trennung ermöglicht eine klare Aufteilung der Verantwortlichkeiten und fördert die Wiederverwendbarkeit und Testbarkeit der Komponenten.

Technische Konzepte

Hier werden wichtige Bibliotheken beschrieben, welche wir in der Frontend Entwicklung gebraucht haben.

Axios
Axios ist eine JavaScript-Bibliothek, die in der Webentwicklung weit verbreitet ist und uns dabei unterstützt, HTTP-Anfragen an externe Ressourcen durchzuführen. Mit Axios können wir Daten von APIs abrufen, sie aktualisieren oder löschen und sogar neue Daten erstellen.

API Gateway

Das API-Gateway spielt eine entscheidende Rolle als zentrale Schaltstelle für unsere API.

Ähnlich einem pulsierenden Verkehrsknotenpunkt ist es dafür verantwortlich, den reibungslosen Fluss von Anfragen zu gewährleisten. Wenn wir es mit einem Hauptbahnhof vergleichen, können wir das Gateway als das Eingangstor zu unserem digitalen Netzwerk betrachten. Jede Person, die eine Verbindung zu unserer API herstellen möchte, muss zuerst durch dieses Gateway passieren. Stellen Sie sich vor, Sie stehen auf einem belebten Bahnhof. Menschen strömen von allen Seiten herbei, um ihre Reise anzutreten. Einige suchen nach Zügen, andere nach Bussen oder U-Bahnen. Der Hauptbahnhof ist der zentrale Ort, an dem alle Reisenden zusammenkommen, bevor sie ihre individuellen Wege fortsetzen.

In ähnlicher Weise nimmt das API-Gateway alle eingehenden Anfragen entgegen, unabhängig davon, ob es sich um eine Anfrage von einem Webbrowser, einer mobilen App oder einem anderen System handelt. Es sammelt sie an einem Ort und leitet sie dann weiter, um die Anfragen an die richtigen Endpunkte oder Dienste in unserer API zu verteilen. Auf diese Weise ermöglicht das Gateway eine nahtlose Kommunikation zwischen den Anwendungen, die unsere API nutzen möchten, und den dahinterliegenden Diensten, die die angeforderten Funktionen bereitstellen.

Das API-Gateway übernimmt auch weitere wichtige Aufgaben wie die Authentifizierung und Autorisierung der Anfragen. Ähnlich den Sicherheitskontrollen an einem Bahnhof sorgt das Gateway dafür, dass nur berechtigte Anfragen durchgelassen werden und unerwünschter Datenverkehr abgewiesen wird.

Es bietet eine Schicht der Kontrolle und Sicherheit, um unsere API und die damit verbundenen Dienste vor bösartigen Angriffen oder unzulässigen Zugriffen zu schützen. Darüber hinaus ermöglicht das Gateway auch die Transformation und Aggregation von Daten. Es kann Anfragen umformen oder anpassen, um die Anforderungen unterschiedlicher Endpunkte oder Anwendungen zu erfüllen. Ähnlich wie ein Bahnhof verschiedene Verkehrsmittel bedient, kann das Gateway unterschiedliche Datenformate oder Protokolle unterstützen und so eine nahtlose Kommunikation zwischen den verschiedenen Komponenten ermöglichen. Insgesamt ist das API-Gateway das Herzstück unserer API-Infrastruktur. Es fungiert als Verkehrskontrolle, Sicherheitswächter und Datenvermittler, um den reibungslosen Fluss von Anfragen und Informationen zu gewährleisten. Indem es die Aufgaben eines Hauptbahnhofs übernimmt, schafft das Gateway eine solide Grundlage für die Skalierbarkeit, Sicherheit und Effizienz unserer API, während es gleichzeitig die Interaktionen zwischen unseren Anwendungen und den damit verbundenen Diensten erleichtert.

Technische Konzepte

Hier werden wichtige technische Konzepte beschrieben, welche wir in unserem Backend verwendet haben.

SpringBoot Cloud

Wir verwenden Spring Cloud, ein Framework, das die Entwicklung von Cloud-nativen Anwendungen erleichtert. Mit Spring Cloud können wir Microservices erstellen, die unabhängig voneinander entwickelt und skaliert werden können. Es bietet Funktionen wie Service-Discovery, Konfigurationsmanagement, Load Balancing und mehr. Spring Cloud integriert sich nahtlos in das Spring-Framework und ermöglicht es uns, Cloud-native Anwendungen effizient zu entwickeln und zu betreiben.

Service Registry

Die Service Registry spielt eine essenzielle Rolle als zentraler Sammelpunkt für unsere Services. Auch hier können wir den Dienst wieder mit einem lebendigen Ort vergleichen, dieses Mal betrachten unser Service Registry als Marktplatz.

Ähnlich wie Menschen auf einem Markt von Stand zu Stand gehen, nutzt das API-Gateway die Service Registry, um einen bestimmten Service anzusprechen.

Stellen Sie sich vor, Sie schlendern über einen geschäftigen Markt, auf dem verschiedene Verkaufsstände ihre Produkte anbieten. Sie können zu einem Stand gehen, um frisches Obst zu kaufen, zu einem anderen Stand für Gewürze und zu einem weiteren für handgefertigte Kunstwerke. Die Service Registry ist wie ein Wegweiser, der Ihnen hilft, den richtigen Stand für Ihre Bedürfnisse zu finden.

In ähnlicher Weise greift das API-Gateway auf die Service Registry zu, um den passenden Service für eine bestimmte Anfrage zu finden. Es fungiert als Vermittler und stellt sicher, dass die Anfragen an den richtigen Service weitergeleitet werden. Die Registry enthält Informationen über die verfügbaren Services, wie ihre Standorte, Versionen und möglicherweise auch ihre Kapazität. Das Gateway nutzt

diese Informationen, um eine reibungslose Kommunikation zwischen den Anwendungen und den entsprechenden Services sicherzustellen.

Ein weiterer wichtiger Aspekt der Service Registry ist ihre Rolle als Load Balancer. Stellen Sie sich vor, ein besonders beliebter Stand auf dem Markt bietet eine Vielzahl von köstlichen Speisen an. Um sicherzustellen, dass sich die Kunden gleichmäßig verteilen und niemand zu lange warten muss, könnte der Stand einen Mitarbeiter haben, der die Besucherströme lenkt. Ähnlich agiert die Service Registry als Load Balancer, wenn wir mehrere Instanzen desselben Services hätten (Dies ist nicht der Fall, da wir Kubernetes nicht Implementieren konnten).

Wenn wir mehrere Instanzen eines Services betreiben, kann die Service Registry den Datenverkehr intelligent auf diese Instanzen verteilen. Sie sorgt dafür, dass keine einzelne Instanz überlastet wird und dass die Anfragen gleichmäßig auf die verfügbaren Instanzen aufgeteilt werden. Dies erhöht die Skalierbarkeit und die Verfügbarkeit unserer Services, da die Last auf mehrere Instanzen verteilt wird und somit ein Ausfall oder eine Überlastung eines einzelnen Dienstes vermieden wird.

Insgesamt ist die Service Registry ein wesentlicher Bestandteil unserer Micro-Service-orientierten Architektur. Sie fungiert als Wegweiser und Koordinator für unsere Services und gewährleistet, dass das API-Gateway den richtigen Service anspricht. Darüber hinaus agiert sie als Load Balancer, um die Last auf mehrere Instanzen zu verteilen und eine optimale Auslastung zu gewährleisten. Durch ihre Funktionen trägt die Service Registry zur Skalierbarkeit, Verfügbarkeit und Effizienz unserer Services bei und fördert eine nahtlose Kommunikation zwischen den verschiedenen Komponenten unserer Architektur.

Technische Konzepte

Hier werden wichtige technische Konzepte beschrieben, welche wir in unserem Service Registry verwendet haben.

SpringBoot Cloud

Wir verwenden Spring Cloud, ein Framework, das die Entwicklung von Cloud-nativen Anwendungen erleichtert. Mit Spring Cloud können wir Microservices erstellen, die unabhängig voneinander entwickelt und skaliert werden können. Es bietet Funktionen wie Service-Discovery, Konfigurationsmanagement, Load Balancing und mehr. Spring Cloud integriert sich nahtlos in das Spring-Framework und ermöglicht es uns, Cloud-native Anwendungen effizient zu entwickeln und zu betreiben.

Eureka Server

Eureka ist ein Service Discovery-Tool, das sehr häufig in Microservices-Architekturen verwendet wird und wurde von Netflix entwickelt und bereitgestellt. Es bietet eine zentrale Registrierungsstelle für Microservices, um sich selbst zu registrieren und ihre Metadaten bereitzustellen. Durch die Integration von Eureka in eine Architektur können Microservices leichter gefunden und aufgerufen werden. Eureka ermöglicht eine dynamische Skalierung und Lastverteilung, da es automatisch erkennt, wenn neue Services hinzugefügt oder entfernt werden. Es ist ein wichtiges Werkzeug für die Bereitstellung hochverfügbarer und skalierbarer Microservices-Umgebungen.

Config Server

Der Config Server mag zwar kein direkter Bestandteil der API-Kette sein, spielt jedoch eine äußerst wichtige Rolle, da er als zentraler Dreh- und Angelpunkt für alle Konfigurationsdateien der Services und des Gateways fungiert. Stellen Sie sich den Config Server als eine Art Bibliothek vor, in der alle Informationen und Anleitungen zur Konfiguration unserer Services und des Gateways gesammelt sind.

Ähnlich wie ein Bibliothekar, der das umfangreiche Wissen eines gesamten Bücherbestands verwaltet, sammelt und organisiert der Config Server alle Konfigurationsdateien. Er stellt sicher, dass die Konfigurationen an einem einzigen Ort zentralisiert und verwaltet werden, um eine einheitliche und konsistente Konfiguration über alle Services und das Gateway hinweg zu gewährleisten.

Wenn ein Service gestartet wird, greift er auf den Config Server zu, um seine spezifischen Konfigurationsinformationen abzurufen. Der Config Server fungiert hier als eine Art Wissensquelle, von der der Service bei Bedarf Informationen abrufen kann. Ähnlich wie ein Entdecker, der auf eine Landkarte oder ein Handbuch zugreift, um den richtigen Weg zu finden, sucht der Service beim Start nach der passenden Konfiguration im Config Server, um sich selbst entsprechend einzurichten.

Der Config Server ermöglicht eine zentrale Verwaltung und Aktualisierung von Konfigurationsdaten. Wenn sich eine Konfiguration ändert, kann sie einfach auf dem Config Server aktualisiert werden, und die betroffenen Services können die aktualisierten Informationen bei ihrem nächsten Start abrufen. Dies erleichtert die Wartung und Aktualisierung der Konfiguration, da Änderungen an einer zentralen Stelle vorgenommen werden können, anstatt jedes einzelne Service manuell zu aktualisieren.

Darüber hinaus trägt der Config Server auch zur Skalierbarkeit und Flexibilität unserer Architektur bei. Wenn wir neue Services hinzufügen oder bestehende Services aktualisieren möchten, können wir dies durch das Aktualisieren der Konfigurationsdateien auf dem Config Server erreichen. Dadurch können wir schnell und einfach auf Veränderungen reagieren, ohne die Services selbst neu bereitstellen oder konfigurieren zu müssen.

Insgesamt spielt der Config Server eine entscheidende Rolle in der Konfigurationsverwaltung unserer Architektur. Er zentralisiert und verwaltet alle Konfigurationsdateien und ermöglicht es den Services und dem Gateway, ihre spezifischen Konfigurationen beim Start abzurufen. Als zentraler Punkt für die Konfigurationserfassung und -verteilung erleichtert der Config Server die Wartung, Aktualisierung und Skalierung unserer Services und trägt zur Konsistenz und Flexibilität unserer Architektur bei.

Technische Konzepte

SpringBoot Cloud config server

Wir verwenden Spring Cloud, ein Framework, das die Entwicklung von Cloud-nativen Anwendungen erleichtert. Mit Spring Cloud können wir Microservices erstellen, die unabhängig voneinander entwickelt und skaliert werden können. Es bietet Funktionen wie Service-Discovery, Konfigurationsmanagement, Load Balancing und mehr. Spring Cloud integriert sich nahtlos in das Spring-Framework und ermöglicht es uns, Cloud-native Anwendungen effizient zu entwickeln und zu betreiben.

Coverletter Service

Der Coverletter Service nimmt eine herausragende Position in unserer aktuellen Architektur ein, da er unser erster und einziger Microservice ist. Dieser Service spielt eine zentrale Rolle bei der Verarbeitung der Benutzerdaten und fungiert als Bindeglied zwischen unserem Frontend und der OpenAI API. Stellen Sie sich den Coverletter Service als eine Art Vermittler vor, der die Benutzerdaten sorgfältig aufbereitet und dann an die leistungsstarke OpenAI API weiterleitet.

Wenn ein Benutzer eine Aktion ausführt oder Informationen im Frontend eingibt, übernimmt der Coverletter Service die Verantwortung, diese Daten zu erfassen und vorzubereiten. Ähnlich einem Schriftsteller, der einen Brief verfasst, nimmt der Service die Informationen des Benutzers und formt daraus einen sogenannten Prompt. Dieser Prompt wird dann an die OpenAI API gesendet, die daraufhin eine intelligente und kontextbezogene Antwort generiert.

Die Antwort der OpenAI API wird auf zweierlei Arten weiterverarbeitet. Zum einen wird die Antwort zurück an den Benutzer im Frontend geschickt, um eine nahtlose Kommunikation und Interaktion zu ermöglichen. Dies erfolgt in Form eines Formatierten Bewerbungsschreibens.

Zum anderen wird die erhaltene Antwort auch in unserer Postgres-Datenbank persistiert, die diesem Service zugeordnet ist. Ähnlich einem Archivar, der Dokumente in einer Datenbank ablegt, speichert der Coverletter Service die Antwortdaten für zukünftige Referenzzwecke. Dies ermöglicht eine spätere Analyse, Verfolgung oder weitere Verarbeitung der generierten Informationen.

Der Coverletter Service stellt somit eine Verbindung zwischen verschiedenen Komponenten unserer Architektur her. Er fungiert als Vermittler zwischen dem Frontend, den Benutzerdaten, der OpenAI API und unserer Postgres-Datenbank. Durch seine Funktionen ermöglicht der Service eine effiziente Verarbeitung, Weitergabe und Speicherung von Informationen und gewährleistet eine nahtlose Benutzererfahrung sowie eine zuverlässige Aufzeichnung der Daten.

Insgesamt spielt der Coverletter Service eine Schlüsselrolle in unserer Architektur, da er die Verarbeitung und Weitergabe von Benutzerdaten an die OpenAI API und die persistente Speicherung der Antwortdaten in unserer Datenbank koordiniert. Als erster Schritt in unserer Microservice-Strategie legt er den Grundstein für zukünftige Erweiterungen und ermöglicht eine skalierbare, intelligente und effiziente Verarbeitung von Benutzerinteraktionen.

Technische Konzepte

Hier werden wichtige technische Konzepte beschrieben, welche wir in unserem ChatGPT Backend verwendet haben.

OpenAI API

Die ChatGPT-API ermöglicht die Kommunikation mit dem ChatGPT-Modell über HTTP-Anfragen. Durch das Senden von Textanfragen an die API erhalten wir maßgeschneiderte Textantworten. Die API nutzt das ChatGPT-Modell, um den eingegebenen Text zu verarbeiten und die passende Antwort zu

generieren. Dadurch können wir natürliche Dialoge und hochwertige Textantworten mit der ChatGPT-API erzielen.

Services (generell)

Zu diesem Zeitpunkt haben wir zwar nur einen einzigen Microservice, aber unsere Zukunftspläne sehen vor, unsere Architektur zu erweitern und weitere Microservices hinzuzufügen. Indem wir neue Services einführen, streben wir eine umfassendere und leistungsfähigere Plattform an, die verschiedene Aspekte unserer Anwendung abdeckt und erweiterte Funktionalitäten bietet.

Ein Beispiel für einen geplanten Microservice ist der User Service, der es uns ermöglichen wird, Benutzerprofile zu verwalten. Er ermöglicht es uns, Benutzerdaten zu speichern, Profile zu erstellen, Anmeldeinformationen zu verwalten und verschiedene Funktionen im Zusammenhang mit der Benutzerverwaltung anzubieten.

Ein weiterer geplanter Microservice ist die Lebenslauf-Generierung. Stellen Sie sich vor, wir haben einen virtuellen Assistenten, der Benutzern dabei hilft, beeindruckende Lebensläufe zu erstellen. Dieser Service würde verschiedene Funktionen bereitstellen, um Benutzerdaten zu analysieren, strukturierte Lebensläufe zu generieren und Vorlagen anzubieten. Damit könnten Benutzer mühelos professionelle Lebensläufe erstellen und ihre Bewerbungsprozesse unterstützen.

Indem wir neue Microservices einführen, streben wir eine modulare und erweiterbare Architektur an, die es uns ermöglicht, einzelne Komponenten unabhängig voneinander zu entwickeln, zu testen und bereitzustellen. Durch die Aufteilung der Funktionalität in verschiedene Services können wir auch die Skalierbarkeit und Wartbarkeit unserer Anwendung verbessern.

Es ist wichtig zu betonen, dass unsere Pläne zur Erweiterung der Architektur nur einige Beispiele sind und dass die Möglichkeiten für zukünftige Microservices vielfältig sind. Wir werden kontinuierlich unsere Anforderungen und die Bedürfnisse unserer Benutzer evaluieren, um die passenden Services zu identifizieren und schrittweise in unsere Architektur zu integrieren.

Insgesamt sehen wir die Einführung weiterer Microservices als einen wichtigen Schritt, um unsere Anwendung zu erweitern und unsere Funktionalität zu verbessern. Indem wir verschiedene Services hinzufügen, wie den User Service und die Lebenslauf-Generierung, werden wir in der Lage sein, eine umfassendere Plattform zu schaffen und ein breiteres Spektrum an Funktionen anzubieten, um den Bedürfnissen unserer Benutzer gerecht zu werden.

Postregs DB

Für jeden unserer API-Services haben wir eine PostgreSQL-Datenbank eingerichtet, um die Daten des jeweiligen Services zu persistieren. Wir haben uns für PostgreSQL entschieden, da wir in der Noser bereits Erfahrung damit gesammelt haben und sie unseren Anforderungen entspricht.

Die Daten dieser Datenbank werden auch auf dem Server persistiert, im falle das die Datenbank neu gestartet werden muss. Dies erreichten wir mit einem Docker Volumen, dass wir im Docker-compose angegeben haben.

PostgreSQL bietet eine zuverlässige Datenbanklösung mit erweiterten Funktionen und einer hohen Skalierbarkeit. Mit PostgreSQL können wir komplexe Datenmodelle entwerfen und effiziente Abfragen durchführen, um eine optimale Leistung zu gewährleisten.

Durch die Verwendung von PostgreSQL können wir auch von ihren Erweiterungen und Plugins profitieren, die für spezifische Anwendungsfälle entwickelt wurden. Wir können die Datenbank flexibel anpassen und erweitern, um den individuellen Anforderungen unserer API-Services gerecht zu werden.

Darüber hinaus bietet PostgreSQL eine umfangreiche Palette von Sicherheitsfunktionen, wie zum Beispiel Zugriffskontrollen, Verschlüsselungsoptionen und Überwachungsmechanismen. Dies ermöglicht uns, die Integrität unserer Daten zu sichern und potenzielle Sicherheitslücken zu minimieren.

Unsere Erfahrung mit PostgreSQL in der Noser hat uns gezeigt, dass es eine effektive Datenbanklösung ist, die unseren Anforderungen gerecht wird.

Docker

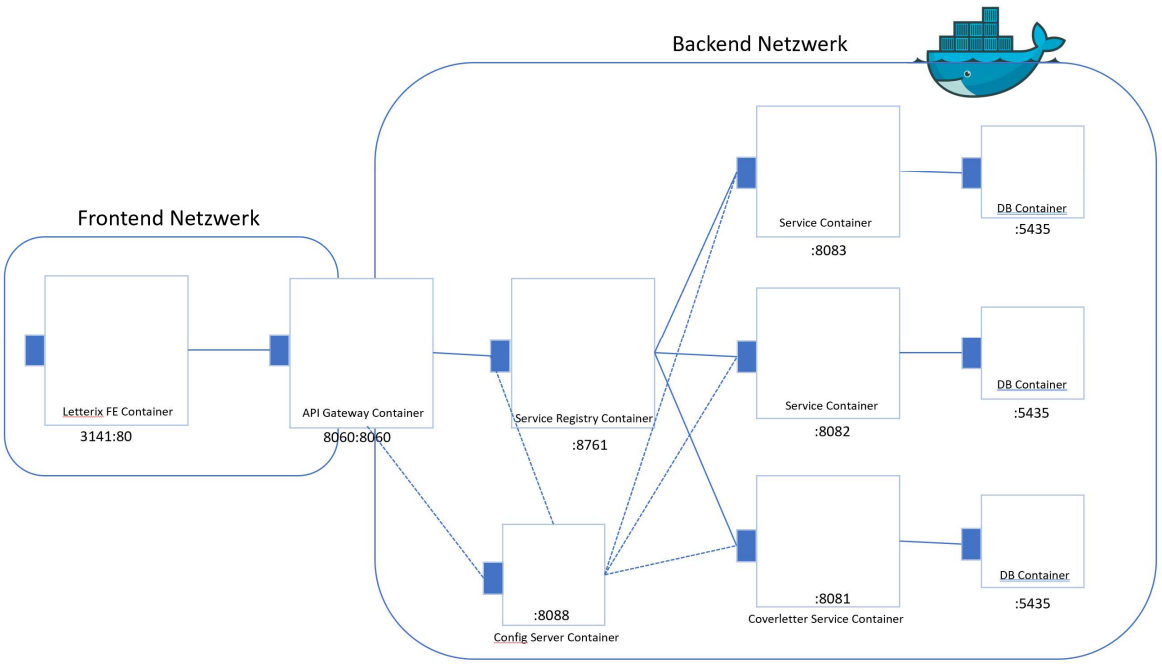


Abbildung 4: Docker Container- und Netzwerkabbildung

Wie oben visualisiert, werden in unserem docker-compose zwei Netzwerke erstellt; eines für das Frontend und eines für die Backend Services. Die Kommunikationsschnittschelle bildet unser API Gateway.

Security

Wir haben auf verschiedenen Ebenen Sicherheitsvorkehrungen getroffen, um die Integrität und Vertraulichkeit unserer Anwendung zu gewährleisten.

Auf Backend-Ebene werden die Eingaben validiert, wobei bei einem Fehler der Request an ChatGPT nicht ausgeführt wird und das Frontend den Fehler behandelt. Zudem nutzen wir CORS und haben CSRF-Regeln implementiert.

Da wir Docker verwenden, haben wir uns dafür entschieden, zwei separate Netzwerke einzurichten. Dadurch können das Frontend und die Backend-Services sich gegenseitig nicht sehen, und abgesehen vom API-Gateway sind die Services nicht öffentlich zugänglich. Die Kommunikation zwischen Frontend und API-Gateway erfolgt ausschließlich über das HTTPS-Protokoll. Darüber hinaus sind unsere Images in einem passwortgeschützten privaten Registry gespeichert.

Indem wir sensible Daten nur auf dem Server in einer «.env» Datei speichern, bleiben sensible Informationen vertraulich und sind vor unbefugtem Zugriff geschützt.

Deployment & CI/CD

Server

Unsere Applikation wird nicht wie ursprünglich angeboten auf dem TBZ LernMaaS betrieben, sondern auf unserem selbst gehosteten Webserver. Diese Entscheidung haben wir aus verschiedenen Gründen getroffen. Zum einen war der Server bereits vorhanden und einsatzbereit, wodurch wir Zeit und Ressourcen sparen konnten. Zudem sind wir mit dem System vertraut, was die Verwaltung erleichtert.

Ein weiterer wichtiger Aspekt ist die zukünftige Weiterführung des Projekts. Da wir vorhaben, das Projekt privat weiterzuentwickeln, war es sinnvoll, von Anfang an die Konfiguration auf unserem eigenen Server vorzunehmen. Dies vermeidet redundante Arbeit und ermöglicht uns eine nahtlose Fortsetzung des Projekts, ohne auf externe Infrastruktur angewiesen zu sein.

Insgesamt bietet uns die Entscheidung, unsere Applikation auf unserem selbst gehosteten Webserver zu betreiben, eine höhere Flexibilität, Effizienz und Kontrolle über die Infrastruktur. Wir können auf vorhandene Ressourcen aufbauen, unsere Vertrautheit mit dem System nutzen und das Projekt kontinuierlich und unabhängig weiterentwickeln.

CI/CD

Für die Implementierung von CI/CD haben wir uns für die Verwendung von GitHub Actions entschieden, was eine neue Erfahrung für uns war, obwohl wir bereits in einem üK erste Einblicke in Actions erhalten hatten. In unseren bisherigen Projekten haben wir immer Bitbucket Pipelines verwendet.

Um unsere Services effizient zu integrieren, hat jeder Service, einschließlich des Frontends, einen eigenen GitHub Workflow. Dieser Workflow wird nur ausgelöst, wenn Änderungen im entsprechenden Ordner vorgenommen wurden. Auf diese Weise können wir sicherstellen, dass nur diejenigen Workflows ausgeführt werden, die direkt von den Änderungen betroffen sind, und dadurch Ressourcen sparen.

Um Ressourcen zu sparen, werden die Docker-Images beim Erstellen eines Pull Requests in unseren Hauptzweig (Main Branch) nur gebaut, aber nicht gepusht. Dies geschieht erst beim Einchecken (Commit) in unseren Hauptzweig. Dadurch wird sichergestellt, dass die Docker-Images nur dann in das Repository hochgeladen werden, wenn Änderungen endgültig akzeptiert wurden und in den Hauptzweig übernommen wurden.

Theoretisch, werden auch die dazugehörigen Services im Docker compose neu gestartet via SSH Zugang in der Pipeline, allerdings, funktioniert dies zum jetzigen Zeitpunkt noch nicht ganz, da der SSH dients ein Problem hat den SSH Key zu erkennen.

Diese Vorgehensweise ermöglicht es uns, den CI/CD-Prozess effizient zu gestalten und Ressourcen zu schonen, indem nur notwendige Aktionen ausgeführt werden. Durch die Nutzung von GitHub Actions können wir eine kontinuierliche Integration und Bereitstellung unserer Services erreichen, während wir gleichzeitig die Flexibilität und Kontrolle über den Entwicklungsprozess behalten.

Testing

Aktion	Vorgehen	Erwartetes Ergebnis	Eigentliches Ergebnis	Resultat
Erforderte Bewerbungsdaten eingeben	Alle Daten korrekt und vollständig eingegeben	Ein Bewerbungsschreiben wird angezeigt	Ein Bewerbungsschreiben wird angezeigt	Passed
Unvollständige Daten eingeben	Felder leer gelassen	Error wird geworfen und angezeigt	Error wird geworfen und angezeigt	Passed
ChatGpt ist überladen	Versuch ein Bewerbungsschreiben zu generieren	Error wird geworfen	Unbekannt – ist während unserer Entwicklung nocht nicht passiert	Unbekannt

Zukünftige Features

Letterix ist eine Idee, die uns vor einigen Monaten gekommen ist. Ursprünglich planten wir die Implementierung eines Chatbots, der mit Sekundarschülern interagiert und ihnen bei der Erstellung von Bewerbungsschreiben unterstützt. Nun haben wir uns entschieden, zunächst eine Website für dieses Projekt zu entwickeln, die zunächst die grundlegenden Funktionen bereitstellt.

In Zukunft möchten wir den Benutzern die Möglichkeit geben, einen Account zu erstellen, um auf ihre früheren Bewerbungsschreiben zugreifen zu können. Außerdem planen wir die Implementierung eines Zahlungstools.

Um unsere ursprüngliche Idee dennoch umzusetzen, haben wir beschlossen, während der Sommerferien den Chatbot zu entwickeln. Dies wird es uns ermöglichen, die interaktive Komponente und die personalisierte Unterstützung bei der Erstellung von Bewerbungsschreiben einzuführen.

Unser Ziel ist es, Letterix zu einer umfassenden Plattform für Bewerbungsschreiben zu entwickeln, die Schülern dabei hilft, professionelle und ansprechende Bewerbungsschreiben zu verfassen. Wir sind gespannt auf die Weiterentwicklung dieses Projekts und freuen uns darauf, unsere Ideen in die Realität umzusetzen.

Fazit

Von Anfang an hatten wir ambitionierte Ansprüche, was jedoch dazu führte, dass wir uns etwas übernommen haben. In der ersten Woche haben wir viel Zeit damit verbracht, eine Applikation zu entwickeln, die wir am Ende des Projektzeitraums größtenteils verworfen haben. Anfangs hatten wir eine Art Monolith-Architektur mit zwei Backends, die über HTTP-Anfragen miteinander kommunizierten. Da wir uns doch für eine richtige Microservice-Architektur entschieden haben, haben wir diesen Ansatz über Bord geworfen.

Das Problem bestand darin, dass wir nicht genau verstanden haben, wie eine solche Architektur mit einem API-Gateway, einer Service Registry usw. aufgebaut und implementiert werden sollte. Gegen Ende der letzten Woche hat uns unser Berufsbildner in einem 20-minütigen Crashkurs erklärt, wie eine solche Architektur aufgebaut werden sollte und welche Tools wir dafür nutzen könnten. Daraufhin haben wir einen 4-5-tägigen Sprint gestartet, um dies umzusetzen. Dies führte dazu, dass nicht alles perfekt ist, aber die grundlegenden Anforderungen unserer Applikation funktionieren.

Von Anfang an hätten wir genauer und insgesamt besser planen können, um etwas weniger Stress zu haben. Hierbei hätten wir ein Tool nutzen können, das uns bei der Planung unterstützt. Die Lernerfahrung der letzten 5 Tage war enorm und hat uns sehr viel Spaß gemacht.