Florian Merlau, Marc Nebel, Marius Mühleck, Maximilian Müller

Juni 2023

1 Einführung und Ziele

1.1 Aufgabenstellung [Merlau]

Die vorliegende Architekturdokumentation umfasst die Aufgabenstellung für das Projekt der Webanwendung für die Deutsche Bahn. Das Hauptziel besteht darin, eine Anwendung zu entwickeln, die es Benutzern ermöglicht, Verspätungen von Zügen zu analysieren sowie Informationen zu den Preisen und Tarifen abzurufen.

Die Aufgabenstellung beinhaltet die folgenden Hauptaspekte:

- Verspätungsanalyse: Die Anwendung soll in der Lage sein, Daten über Zugverspätungen zu sammeln, zu verarbeiten und Benutzern in einer übersichtlichen und verständlichen Form darzustellen. Dazu sollen verschiedene Datenquellen wie beispielsweise Live-Daten von der Deutschen Bahn oder historische Daten verwendet werden.
- Preis- und Tarifanalyse: Neben den Verspätungen soll die Anwendung auch Informationen zu den Ticketpreisen und Tarifen für Fernstrecken liefern. Benutzer sollen die Möglichkeit haben, nach günstigen Angeboten zu suchen, Preistrends zu analysieren und die Preise für verschiedene Reisezeiträume zu vergleichen.
- Verbindungssuche: Die Anwendung soll es Benutzern ermöglichen, nach Verbindungen zwischen verschiedenen Bahnhöfen zu suchen. Dabei sollen Informationen wie Abfahrtszeiten, Ankunftszeiten, Umstiege und Reisedauer berücksichtigt werden. Die gefundenen Verbindungen sollen übersichtlich und leicht verständlich präsentiert werden.

Die Aufgabenstellung legt den Rahmen für die Entwicklung der Webanwendung fest und dient als Leitfaden für das Projektteam. Sie stellt sicher, dass alle relevanten Anforderungen erfasst werden und das Endprodukt den Erwartungen der Benutzer und Stakeholder entspricht.

Es ist wichtig, dass die Aufgabenstellung während des Projekts regelmäßig überprüft und aktualisiert wird, um sicherzustellen, dass die Anforderungen angemessen berücksichtigt und umgesetzt werden. Durch eine klare Definition der Aufgabenstellung wird eine solide Grundlage für die erfolgreiche Umsetzung des Projekts geschaffen.

1.2 Qualitätsziele [Merlau]

- Leistungsfähigkeit: Die Anwendung soll eine schnelle und reaktionsschnelle Performance bieten, um Benutzern ein effizientes Arbeiten zu ermöglichen. Ladezeiten sollten minimiert werden, um eine optimale Benutzererfahrung zu gewährleisten.
- Skalierbarkeit: Die Architektur der Anwendung sollte so konzipiert sein, dass sie problemlos mit steigender Benutzerzahl und wachsenden Datenmengen umgehen kann. Skalierbarkeit ist entscheidend, um die Leistung auch bei zunehmender Belastung aufrechtzuerhalten.
- Zuverlässigkeit: Die Anwendung sollte hochverfügbar sein und Ausfallzeiten minimieren. Es sollten Mechanismen zur Fehlererkennung, Fehlerbehebung und Wiederherstellung implementiert werden, um eine hohe Zuverlässigkeit und Robustheit zu gewährleisten.
- Benutzerfreundlichkeit: Die Anwendung sollte eine intuitive Benutzeroberfläche bieten, die Benutzern eine einfache und effektive Interaktion ermöglicht. Eine klare und verständliche Darstellung von Informationen sowie eine leicht erlernbare Bedienung tragen zur Benutzerzufriedenheit bei.
- Funktionalität: Die Anwendung sollte alle erforderlichen Funktionen und Features bereitstellen, um die Anforderungen der Benutzer zu erfüllen. Die Umsetzung der funktionalen Anforderungen sollte sorgfältig geplant und implementiert werden, um eine korrekte und vollständige Funktionalität sicherzustellen.
- Erweiterbarkeit: Die Architektur der Anwendung sollte flexibel und erweiterbar sein, um zukünftige Änderungen und Erweiterungen zu ermöglichen. Es sollte möglich sein, neue Funktionen und Module hinzuzufügen, ohne bestehenden Code grundlegend zu ändern. Dies erhöht die Anpassungsfähigkeit und erleichtert die Wartung und Weiterentwicklung der Anwendung.

1.3 Stakeholder [Merlau & Müller]

Rolle	Kontakt	Erwartungshaltung
CEO	*Roy Oberhauser*	*erwartet eine benutzerfreundliche Webanwendung mit genauen Informationen, zuverlässiger Leistung und regelmäßiger Kommunikation.*
Entwicklerinnen und Entwickler	*Marc Nebel, Florian Merlau, Marius Mühleck, Maximilian Müller*	• *Klar definierte Architektur* • *Dokumentation der wichtigen Entscheidungen* • *Beschreibung der Qualitätsattribute* • *Aktualität und Pflege der Dokumentation* • *Unterstützung bei der Implementierung* • *Verständliche und nachvollziehbare Darstellung* • *Integration mit anderen Dokumentationswerkzeugen*

2 Randbedingungen [Merlau]

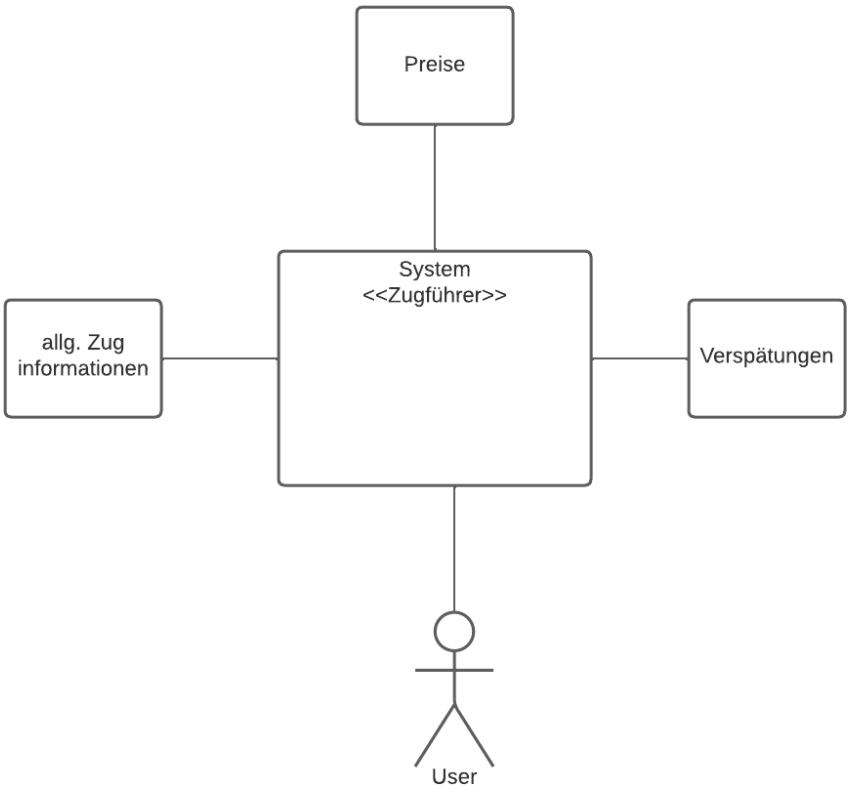
- Webanwendung: Das Projekt sieht die Entwicklung einer Webanwendung vor, wobei eine native App nicht erwartet wird.
- Web UI: Die Webanwendung sollte eine ansprechende Benutzeroberfläche haben und auf modernen Webframeworks wie React, Angular, Vue.js oder Svelte (mit React, Angular oder Vue) basieren.
- Benutzererfahrung (UX): Die Benutzererfahrung ist ein wichtiger Faktor und sollte bei der Entwicklung der Webanwendung berücksichtigt werden. Die Anwendung sollte benutzerfreundlich und intuitiv bedienbar sein.
- Microservice-Architekturstil: Die Architektur der Anwendung sollte den Microservice-Architekturstil verwenden, bei dem jeder Microservice (einschließlich Datenbank) in seinem eigenen Docker-Container läuft. Dadurch wird eine flexible und skalierbare Architektur ermöglicht.
- Pub/Sub-Echtzeitdaten: Die Webanwendung sollte in der Lage sein, Echtzeitdaten aus einer Webquelle zu verwenden. Hierbei kann eine echte oder simulierte webbasierte Quelle für Echtzeitdaten von der Deutschen Bahn Api.
- API-Test und -Dokumentation: Zur Test- und Dokumentationszwecken der APIs sollte Swagger, apiDoc oder ein äquivalentes Werkzeug verwendet

werden, um die APIs zu testen und zu dokumentieren.

- NoSQL-Datenbank: Für die Speicherung von Daten sollte zumindest teilweise eine NoSQL-Datenbank in Betracht gezogen werden. Die Verwendung einer NoSQL-Datenbank wird empfohlen, wobei auch eine Mischung aus NoSQL und SQL (hybrid) möglich ist.
- Backend-Optionen: Für das Backend der Anwendung stehen verschiedene Optionen zur Verfügung, wie z.B. NodeJS/Express oder LoopBack, Python/Falcon oder Flask oder Django REST Framework, Spring Boot, ktor, Go (mit Frameworks wie Micro oder Microservices mit Go).
- Cloud-Bereitstellung und dockerbasierte Microservices: Die Anwendung sollte in der Cloud bereitgestellt werden, wobei die Microservices im Backend in Docker-Containern laufen.

3 Kontextabgrenzung [Merlau]]

3.1 Fachlicher Kontext [Merlau]



Benutzerinteraktion über das Frontend

- Das System ermöglicht den Benutzern, direkt über das Frontend mit der Anwendung zu interagieren.
- Das Frontend bietet eine benutzerfreundliche Oberfläche, über die Benutzer verschiedene Funktionen und Informationen abrufen können.

Verspätungsinformationen

- Die Verspätungsdaten werden über eine API abgerufen, die Zugriff auf Echtzeitdaten zu Zugverspätungen bietet.
- Die erhaltenen Verspätungsdaten werden in der Anwendung gespeichert, um später analysiert und dargestellt zu werden.
- Die Verspätungsinformationen werden in der Fahrplanauskunft angezeigt, um Benutzern eine genaue Vorstellung von möglichen Verspätungen zu geben.

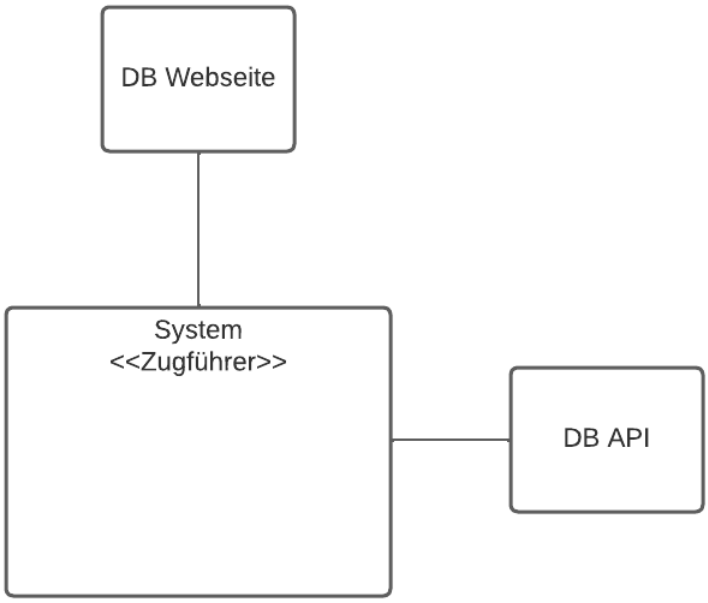
Preisinformationen

- Die Preise für Zugreisen werden durch Web-Scraping von der Webseite der Deutschen Bahn erfasst.
- Die erfassten Preisdaten werden in der Anwendung gespeichert und stehen den Benutzern zur Verfügung.
- Die Preisinformationen ermöglichen es den Benutzern, die Kosten ihrer Zugreisen einzuschätzen und alternative Verbindungen oder Sparangebote zu berücksichtigen.

Allgemeine Zuginformationen

- Die Zugdaten werden über eine API abgerufen, die Zugriff auf Informationen wie Abfahrts- und Ankunftszeiten, Zugtypen, Wagenreihungen usw. bietet.
- Die erhaltenen Zuginformationen werden in der Anwendung gespeichert und in der Fahrplanauskunft angezeigt.
- Die Zuginformationen helfen den Benutzern dabei, die verfügbaren Zugverbindungen zu verstehen und ihre Reisen entsprechend zu planen.

3.2 Technischer Kontext [Merlau]



4 Technischer Kontext

Web-Scraping

- Das System nutzt Web-Scraping-Techniken, um Preisinformationen von der Webseite der Deutschen Bahn zu erfassen.
- Es greift auf die Webseite zu, extrahiert die relevanten Preisdaten und speichert sie in der Anwendung.

Deutsche Bahn API

- Das System verwendet die Deutsche Bahn API, um aktuelle Zuginformationen wie Abfahrts- und Ankunftszeiten, Zugtypen und Verspätungen abzurufen.
- Die API bietet eine standardisierte Schnittstelle zur Abfrage von Zuginformationen, auf die das System zugreift.

Datenverarbeitung und Aufbereitung

- Das System verarbeitet die abgerufenen Informationen, um sie für die Benutzeransicht aufzubereiten.

- Preis- und Verspätungsinformationen werden analysiert, formatiert und in der Anwendung präsentiert.
- Die Zuginformationen werden entsprechend der Benutzeranforderungen strukturiert und dargestellt.

5 Lösungsstrategie [Nebel]

5.1 Einleitung in die Lösungstrategie

Um eine klare Darstellung der architektonischen Ziele zu gewährleisten und den Übergang zu den folgenden Tabellen zu erleichtern, werden in diesem Unterkapitel die angewendeten architektonischen Prinzipien erläutert und in Bezug auf die entsprechenden architektonischen Ziele gesetzt.

- Zuverlässigkeit:
  - Einzelne Container werden so robust wie möglich gebaut, so dass diese auch nach Fehlern weiterhin verfügbar sind. Entsprechende abgefangene Fehler spiegeln sich in einem minimalen Verlust von Daten wieder, der Hauptteil der Daten ist weiterhin verfügbar
  - Potentielle kritische Fehler in Drittanbieter Software (wie z.B. PostgresSQL) werden durch Neustart Richtlinien von Docker abgefangen.
  - Um eine hohe Ausfallsicherheit der Software zu gewährleisten, wurde sie nach dem Prinzip des Defensive Programming entwickelt. Dieser Ansatz ermöglicht es, einen minimalen Datenverlust in Kauf zu nehmen, während gleichzeitig eine robuste Funktionalität gewährleistet wird.
  - Das Prinzip des Loose Coupling erhöht die Zuverlässigkeit des Codes durch die Verringerung der Abhängigkeiten zwischen den Komponenten. Dadurch werden potenzielle Auswirkungen von Änderungen in einer Komponente auf andere Komponenten reduziert, was die Gesamtstabilität der Software verbessert.
  - Microservices-Architektur verbessert die Code Zuverlässigkeit durch die Aufteilung der Anwendung in unabhängige Dienste, was wiederum Fehlerisolierung ermöglicht. Dadurch wird die Gesamtfunktionalität der Anwendung weniger anfällig für Störungen und Ausfälle.
- Funktionalität:
  - Alle Services werden dem Nutzer über ein React Frontend zur Verfügung gestellt. Dieser kann dann über eine intuitiv gestaltete Benutzeroberfläche auf generierte Daten zugreifen
  - Alle Programmkomponenten, also Frontend, Verspätungsanalyse, Preisanalyse und Verbindungsdatenspeicherung befinden sich in einzel-

nen Microservices welche nur über Datenströme voneinander abhängig sind.

  - Die Nutzung der Pandas Python Bibliothek ist von entscheidender Bedeutung für die effiziente Datenverarbeitung in Python. Dies erlaubt dem Benutzer eine zeitnahe Anzeige neuer berechneter Daten und Preise, wodurch die Funktionalität maßgeblich verbessert wird.

- Benutzerfreundlichkeit:
  - Das Design des Frontend lehnt sich an das Design der offiziellen Bahn Website an, um dem durchschnittlichen Deutsche Bahn Kunden einen möglichst intuitiven Einstieg in die Nutzung unseres Produktes zu ermöglichen.
  - Erzeugte Daten werden in einfachen aber umfangreichen dynamischen Grafiken angezeigt um dem Benutzer einen schnellen und einfachen Überblick über die Datenlage seiner Verbindung zu ermöglichen.
- Leistungsfähigkeit:
  - Für eine möglichst schnelle Verspätungsdatenverarbeitung kommt Pandas zum Einsatz.
  - Um schnelle Datenspeicherung und Abrufe zu Gewährleitsten wird PostgreSQL als Datenbank Container für alle Microservices genutzt.
  - Rechenintensive Berechnungen auf die Verbindungs- sowie die Preisdaten werden direkt bei Erhalt der Daten durchgeführt und gespeichert um die Bereitstellungszeit der Daten zu minimieren.
- Skalierbarkeit:
  - Um eine Basis Skalierbarkeit zu erreichen werden einzelne Komponenten in Microservices entwickelt. Diese sind, abgesehen von Abhängigkeit der Datenströme, voneinander unabhängig und können entsprechend beliebig erweitert werden.
  - Datenspeicherung mit PostgreSQL Containern welche einfach durch weitere gespiegelte Container skaliert werden können.
  - Durch die Anwendung des Loose Coupling Prinzips wird der Code flexibler und anpassungsfähiger in Bezug auf die Skalierung, da die Komponenten weniger stark voneinander abhängig sind.
  - Docker stellt eine effiziente und skalierbare Lösung bereit, um Container mit geringem Aufwand zu skalieren.
  - Durch die Nutzung von REST Schnittstellen wird die Skalierbarkeit erleichtert, da sie die Bereitstellung von Datenzugriff in separaten Containern ermöglichen, ohne dass umfangreiche Änderungen am bestehenden Code vorgenommen werden müssen.

- Erweiterbarkeit
  - Python wird als Programmiersprache für Berechnungen verwendet da diese einfach zu verstehen und entsprechend erweiterbar ist. Darüber hinaus existieren viele mächtige Drittanbieter Bibliotheken welche einfach für zusätzliche Funktionen eigebunden werden können.
  - Java kommt als Programiersprache für das Frontend zum Einsatz. Diese ist ebenfalls einfach veerständlich sowie allgemein breit vertreten was Erweiterungen unkompliziert und problemfrei gestaltet. Wie bei Python existieren hier viele vordefinierte Bibliotheken welche den Code einfach modular erweitern können.
  - Das Prinzip des Loose Coupling ermöglicht eine effiziente Erweiterung von Komponenten, indem neue und bestehende Elemente geringere Abhängigkeiten voneinander aufweisen. Dies führt zu einer Reduzierung der Auswirkungen von Änderungen auf die Gesamtfunktionalität der Software. Durch die Verwendung klar definierter Schnittstellen können Komponenten einfach ausgetauscht werden.
  - REST ermöglicht die Nutzung von standardisierten HTTP-Methoden und URIs, um auf Ressourcen zuzugreifen. Dadurch wird der Code klar strukturiert und leicht erweiterbar, da neue Funktionen einfach durch Hinzufügen neuer Ressourcen oder Erweiterung der vorhandenen Ressourcen implementiert werden können, ohne den gesamten Code zu ändern.

5.2 Architektonischen Ziele

Nachfolgend befindet sich die Architektonischen Ziele (abgekürzt mit AZ) in Tabellarischer Form. Diese sind, in der letzten Spalte, mit den Architektonischen Prinzipien (abgekürzt mit AP) dieses Projekts verknüpft.

Prio.	AZ	Abkürzung	Eigenschaften	APs
1	Zuverlässigkeit	AZ:ZUVK	keine Abstürze, Immer verfügbare Daten, Genaue Daten	AP:SQLD, AP:DOCK, AP:DEPR, AP:LOCO, AP:MISE
2	Funktionalität	AZ:FUNK	UI, Datenspeicherung, Verspätungsanalysen, Preisanalysen	AP:WEBF, AP:MISE, AP:SEDA
3	Benutzerfreundlichkeit	AZ:BEFR	Intuitive/einfache UI	AP:WEBF, AP:SEDA
4	Leistungsfähigkeit	AZ:LEIF	schnelle Datenabrufe, schnelles Generieren von neunen Daten	AP:SQLD, AP:SEDA
5	Skalierbarkeit	AZ:SKAL	Erweiterbarkeit der Daten, Skalierung der Nutzungskapazität	AP:SQLD, AP:DOCK, AP:LOCO, AP:MISE, AP:REST
6	Erweiterbarkeit	AZ:ERWB	Modularität, einfach wartbarer Code	AP:PYTH, AP:JAVA, AP:LOCO, AP:REST

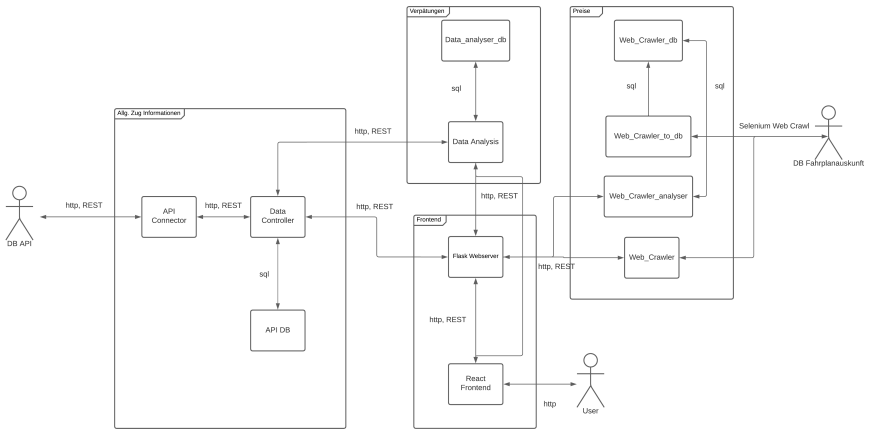
5.3 Architektonische Prinzipien

Die folgende Tabelle zeigt die Architektonischen Prinzipien (abgekürzt mit AP), welche in diesem Projekt Verwendung finden. Die letzte Spalte enthält dabei die Architektonischen Ziele (abgekürzt mit AZ) die von den jeweiligen AP abgedeckt werden.

AP	Abkürzung	Beschreibung	AZs
Webfrontend	AP:WEBF	React Webfrontend für intuitive Nutzung der Anwendung von allen Plattformen via HTTP	AZ:BEFR, AZ:FUNK
SQL DB	AP:SQLD	Teilrelationales speichern von Daten in PostgreSQL Datenbanken	AZ:ZUVK, AZ:LEIF, AZ:SKAL
Docker	AP:DOCK	Aufteilen der einzelnen Software Komponenten auf verschiedene Container, sorgt für Modularität und Ausfallsicherheit, Lässt sich einfach auf andere Plattformen portieren	AZ:ZUVK, AZ:SKAL
Python	AP:PYTH	Einfache und mächtige Hochsprache, vereinfacht Erweiterungen und Wartung des Codes	AZ:ERWB
Java	AP:JAVA	Mächtige Hochsprache für Webentwicklung	AZ:ERWB
Defensive Programming	AP:DEPR	Robuster, Absturzsicherer Code für Ausfallsicherheit auf Kosten minimaler Datenfehler	AZ:ZUVK
Loose Coupling	AP:LOCO	Unabhängige Code Komponenten sorgen für einfache Skalierbarkeit, Erweiterbarkeit, sowie Ausfallsicherheit	AZ:ZUVK, AZ:SKAL, AZ:ERWB
Microservices	AP:MISE	Aufteilung von Hauptkomponenten auf Microservices sorgt für Ausfallsicherheit und Erweiterbarkeit	AZ:FUNK, AZ:SKAL, AZ:ZUVK
Schnelle Datenverarbeitung	AP:SEDA	Schnelle Datenverarbeitung mit Pandas Datenframes	AZ:FUNK, AZ:LEIF, AZ:BEFR
REST Schnittstellen	AP:REST	Einfache und Modulare Kommunikation zwischen Microservices mit Flask Rest Schnittstellen	AZ:SKAL, AZ:ERWB

5.4 Technische Realisierung der Lösungsstrategie

Um die oben genannten Punkte zu gewährleisten und den obig definierten fachlichen Kontext abzudecken wurden die einzelnen technischen Realisierungen in folgende Microservices aufgegliedert. Jeder Kasten repräsentiert dabei einen eigenen Microservice mit entsprechender fachlicher Kategorisierung. Diese laufen dabei in eigenen Docker Containern und sind entsprechend, abgesehen von Daten Abhängigkeiten, voneinander unabhängig.



Allgemeine Zuginformationen: Diese Kategorie umfasst drei verschiedenene Container. Der erste Container ist der **API Connector**, der in regelmäßigen Zeitabständen Zugverbindungsdaten von der offiziellen API der Deutschen Bahn abruft. Diese Daten werden über einen Flask Endpoint an den Container **Data Controller** weitergeleitet. Der Data Controller selbst stellt eine Verbindung zur **API DB** her und dient nicht nur der Speicherung der Daten, sondern auch der Bereitstellung von Informationen für andere Microservices. Durch diese Architektur wird eine umfassende Erfassung und Bereitstellung von Zuginformationen ermöglicht.

Verspätungen: Diese Kategorie enthält zwei Container, die als Microservice fungieren und die Verarbeitung, Speicherung und Bereitstellung von Verspätungsdaten ermöglichen. Der Container **Data Analysis** bezieht Zugverbindungsdaten vom Zuginformations Microservice, führt verschiedene Berechnungen und Analysen durch und speichert die daraus resultierenden Durchschnittswerte letztendlich im Container **Data Analyzer DB** ab. Die so gesammelten Daten werden dann über einen REST Endpoint auf dem **Data Analysis** Container für andere Microservices bereitgestellt. Durch diese Architektur wird eine umfangreiche Erfassung und Analyse von Zugverspätungen ermöglicht.

Preise: Dieser Microservice besteht aus vier separaten Containern, die aktuelle Fernstreckenpreise von der offiziellen DB-Website crawlen, speichern und vergleichend darstellen. Für letzteres werden vom Container **Web Crawler Analyser** dynamisch Grafiken erstellt, die über einen Flask REST Endpoint

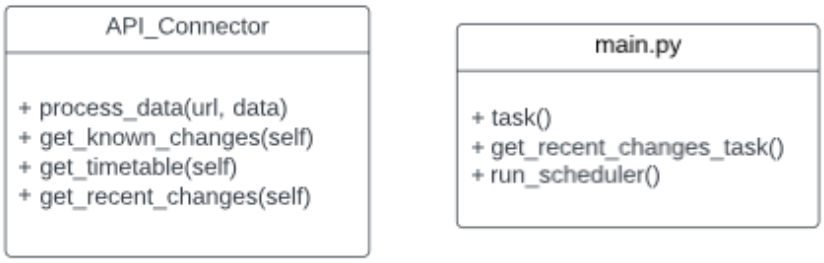
dem Frontend zur Verfügung gestellt werden. Die verwendeten Werte werden zudem im Container **Web Crawler DB** gespeichert. Der Crawlvorgang selbst wird vom Container **Web Crawler to DB** mithilfe von Selenium durchgeführt. Zusätzlich ermöglicht der Container **Web Crawler** das direkte Crawlen von Daten ohne Datenbankzugriff und stellt diese direkt als Grafik über einen Flask REST Endpoint zur Verfügung. Durch diese umfassende Architektur können die Preise dynamisch erfasst, gespeichert und visualisiert werden.

Frontend: Diser Microservice besteht aus zwei Containern: dem **React Frontend** und dem **Flask Webserver**. Das **React Frontend** beinhaltet einen React Webserver, der dem Benutzer eine benutzerfreundliche Oberfläche bietet und den Zugriff auf die anderen Microservices ermöglicht. Jeder Microservice hat seine eigene Subseite im Frontend. Der **Flask Webserver** dient als Proxy-Server für den React-Server, um sichere Verbindungen gemäß den CORS-Richtlinien zu den REST-Endpunkten der anderen Microservices herzustellen. Dadurch wird eine sichere Kommunikation zwischen dem Frontend und den anderen Microservices gewährleistet. Durch diese Architektur kann der Benutzer intuitiv und schnell auf die erzeugten Daten dieser Software zugreifen.

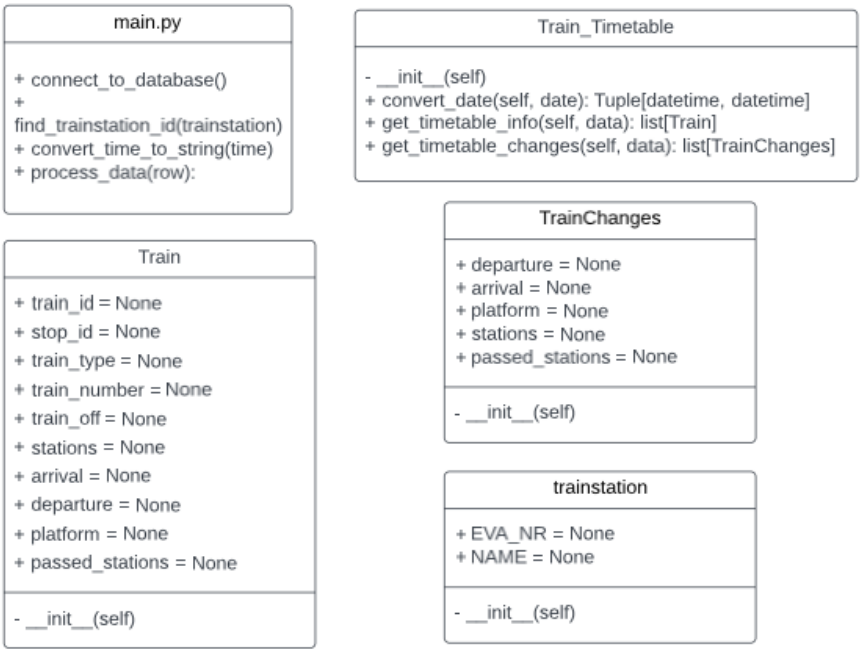
Durch die Strukturierung der Software in Form eines einfachen, modularen und leistungsstarken Microservices wird das angestrebte Ziel erreicht, den Benutzer mit umfassenden Informationen über den Bahnverkehr in Deutschland zu versorgen. Diese Architektur ermöglicht es, die Anforderungen des Systems effektiv zu erfüllen und eine zuverlässige Bereitstellung der benötigten Informationen zu gewährleisten.

5.5 Bausteinsicht [Mühleck]

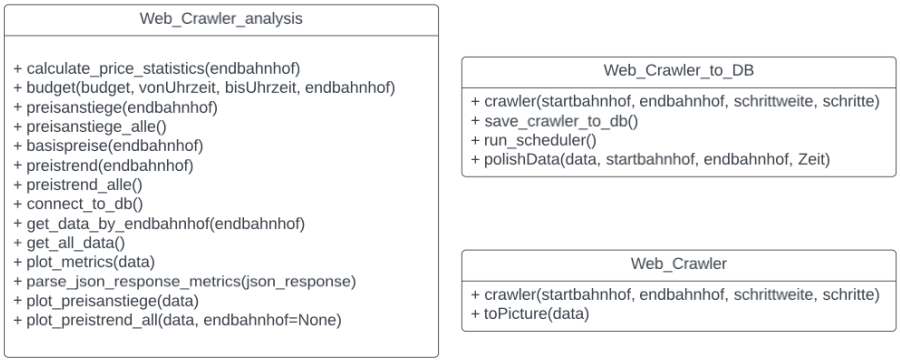
Hier sind die Klassen bzw. Funktionen des API Connector Containers gezeigt.



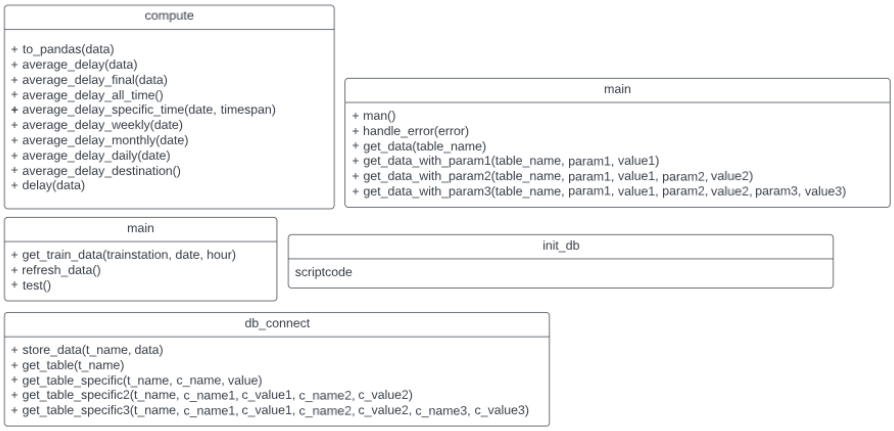
Hier sind die Klassen bzw. die Funktionen im Data Controller Container gezeigt.



Hier sind die Klassen bzw. die Funktionen der Web Crawler Container gezeigt.

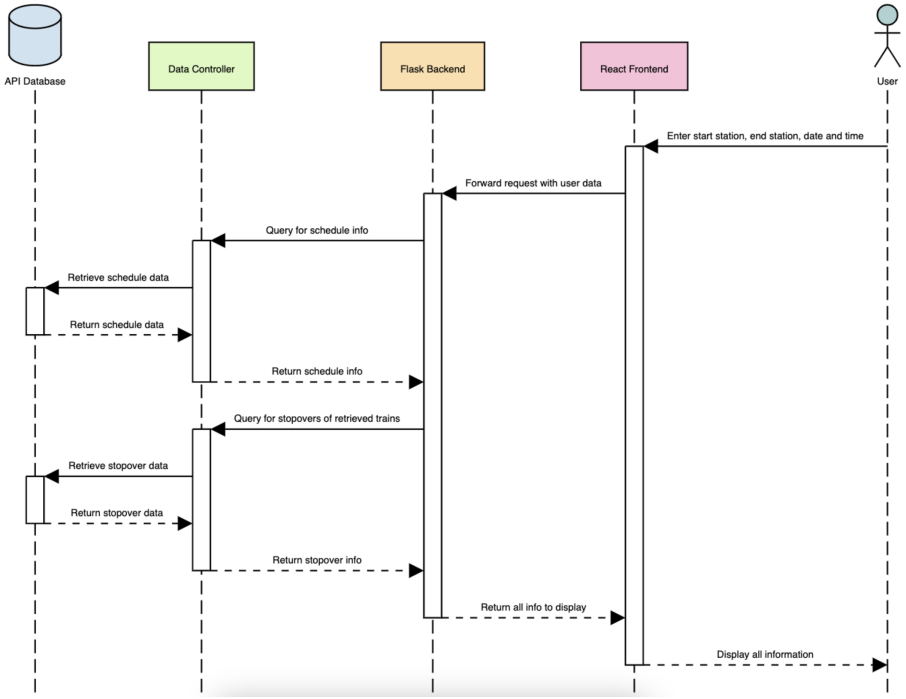


Hier sind die Klassen bzw. die Funktionen des Data Analyzer Container gezeigt.



6 Laufzeitsicht [Mühleck]

6.1 Fahrplanauskunft

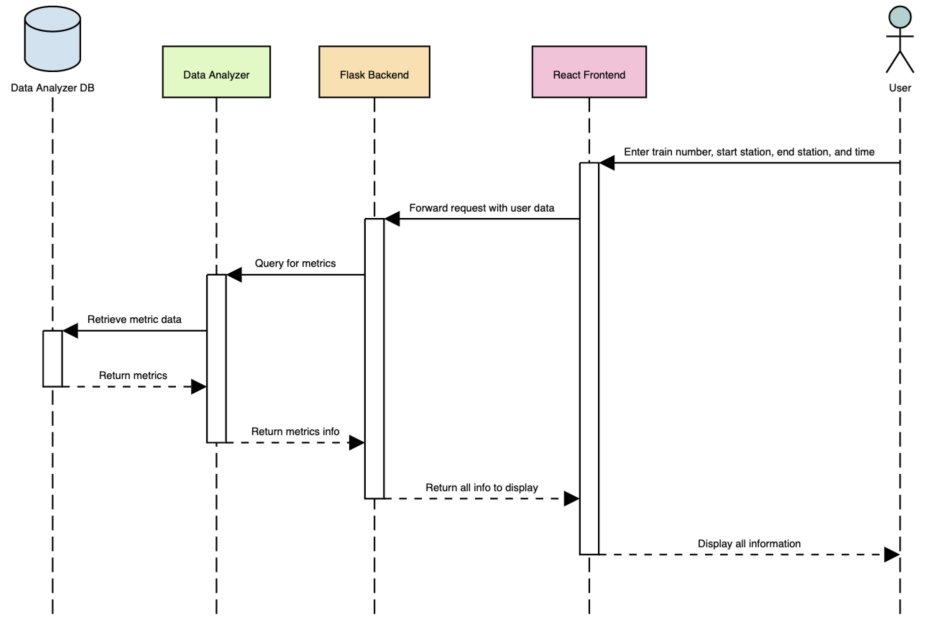


Das Sequenzdiagramm beschreibt den Ablauf der Fahrplanauskunft.

1. Der Benutzer (User) interagiert mit unserem React Frontend. Er gibt die Start- und Endhaltestelle, das Datum und die Uhrzeit ein.

2. Das Frontend leitet diese Daten direkt an unser Flask Backend weiter.
3. Das Flask Backend sendet wiederum eine Anfrage an unseren Data Controller, von dem es Infos über den passenden Fahrplan benötigt.
4. Der Data Controller holt sich die benötigten Informationen aus einer externen API-Datenbank (API Database).
5. Nachdem der Data Controller die Fahrplandaten erhalten hat, sendet er diese an das Flask Backend zurück.
6. Nun fragt das Flask Backend den Data Controller nach Zwischenhalten der abgerufenen Züge. Der Data Controller holt sich diese Daten nun wiederum aus der API-Datenbank.
7. Nachdem der Data Controller die Daten über die Zwischenhalte erhalten hat, sendet er diese Informationen auch an das Flask Backend.
8. Das Flask Backend, nun mit allen Informationen ausgestattet, sendet sämtliche Daten an das React Frontend zurück.
9. Schließlich zeigt das React Frontend alle gesammelten Informationen an, die der Benutzer angefragt hat. Die Anzeige der Informationen ist der letzte Schritt in diesem Prozess.

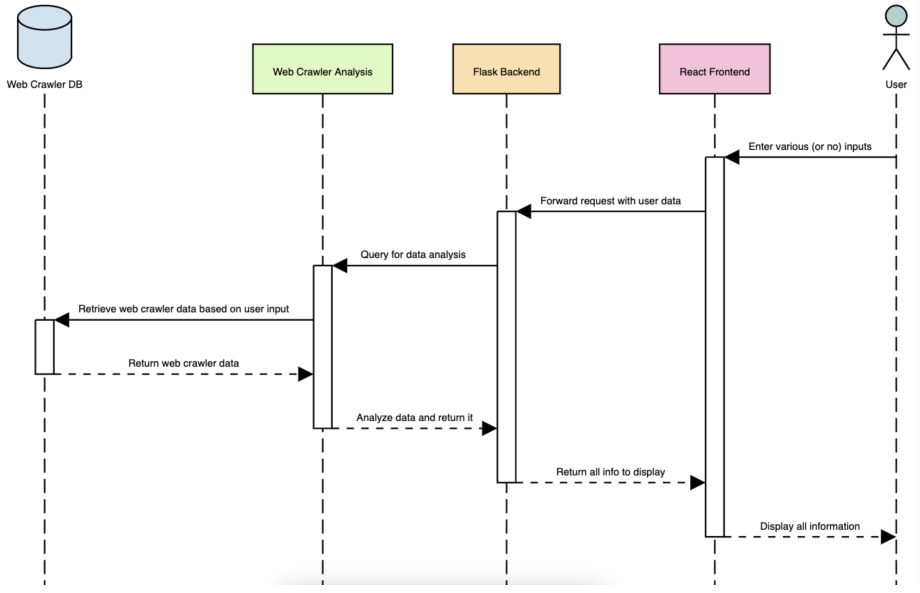
6.2 Verspätungsanalyse



Das Sequenzdiagramm beschreibt den Ablauf der Verspätungsanalyse.

1. Der User startet den Prozess, indem er seine Daten in das Formular im React Frontend eingibt und abschickt.
2. Das React Frontend leitet die vom User eingegebenen Daten an unser Flask Backend weiter.
3. Das Flask Backend sendet die Anfragedaten an den Data Analyzer. Es fragt nach spezifischen Metriken, die anhand der vom Benutzer bereitgestellten Informationen berechnet werden sollen.
4. Der Data Analyzer kommuniziert dann mit seiner Datenbank (Data Analyzer DB), um die entsprechenden Metrikdaten abzurufen.
5. Nachdem die Daten abgerufen wurden, gibt die Data Analyzer DB diese an den Data Analyzer zurück.
6. Der Data Analyzer liefert als nächstes die berechneten Metriken an das Flask Backend zurück.
7. Das Flask Backend, nun im Besitz der relevanten Metriken, sendet diese Informationen an das React Frontend.
8. Das React Frontend präsentiert dem Benutzer dann die angeforderten Metrikindikatoren.

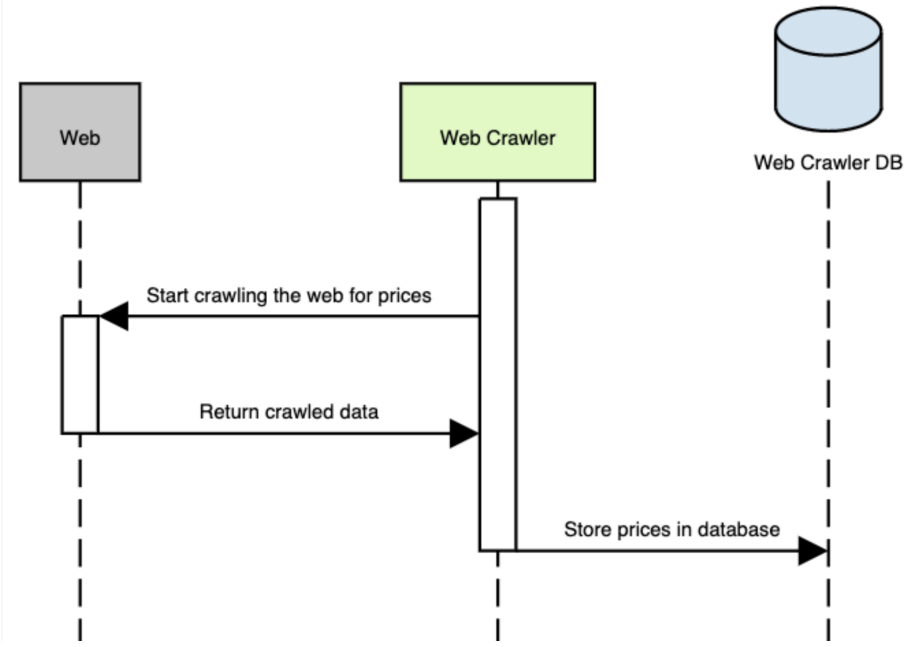
6.3 Preisanalyse



Das Sequenzdiagramm beschreibt den Ablauf der automatisierten im Hintergrund laufenden Preisanalyse.

1. Der User startet den Prozess, indem er seine Daten in das Formular im React Frontend eingibt und abschickt oder im Fall der Allgemeinen Preisanalyse einfach die Seite aufruft.
2. Das React Frontend leitet die vom User eingegebenen Daten an unser Flask Backend weiter.
3. Das Flask Backend sendet die Anfragedaten anschließend an die Web Crawler Analyse. Diese analysiert die gegebenen Daten und gibt die Preisanalysen zurück.
4. Die Web Crawler Analyse holt die entsprechenden Daten aus ihrer Datenbank (Web Crawler DB).
5. Nach Erhalt der Daten gibt die Web Crawler DB diese an die Web Crawler Analyse zurück.
6. Die Web Crawler Analyse führt dann eine Analyse mit den abgerufenen Daten durch und gibt die Ergebnisse an das Flask Backend zurück.
7. Das Flask Backend, welches jetzt über die analysierten Daten verfügt, sendet diese Informationen an das React Frontend.
8. Das React Frontend stellt die analysierten Daten anschließend für den Benutzer dar.

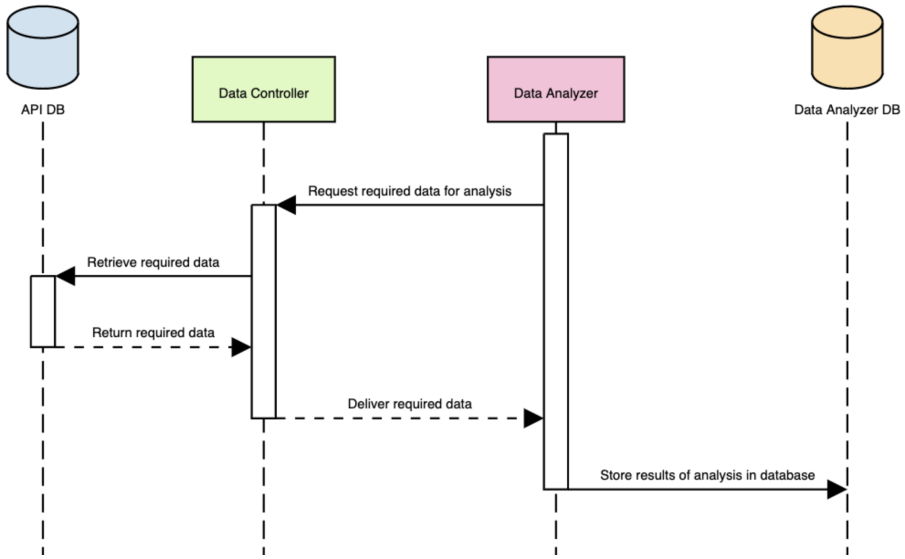
6.4 Preis Crawling



Das Sequenzdiagramm beschreibt den Ablauf des automatisierten im Hintergrund laufenden Preis Crawlings.

1. Der "Web Crawler" startet den Prozess, indem er das Internet nach bestimmten Daten - hier Preisen - durchsucht.
2. Sobald der Web Crawler die benötigten Daten gefunden hat, bekommt er diese vom Internet zurückgeliefert.
3. Die erhaltenen Daten speichert der Web Crawler in der "Web Crawler Datenbank".

6.5 Fahrplananalyse



Das Sequenzdiagramm beschreibt den Ablauf der Fahrplananalyse.

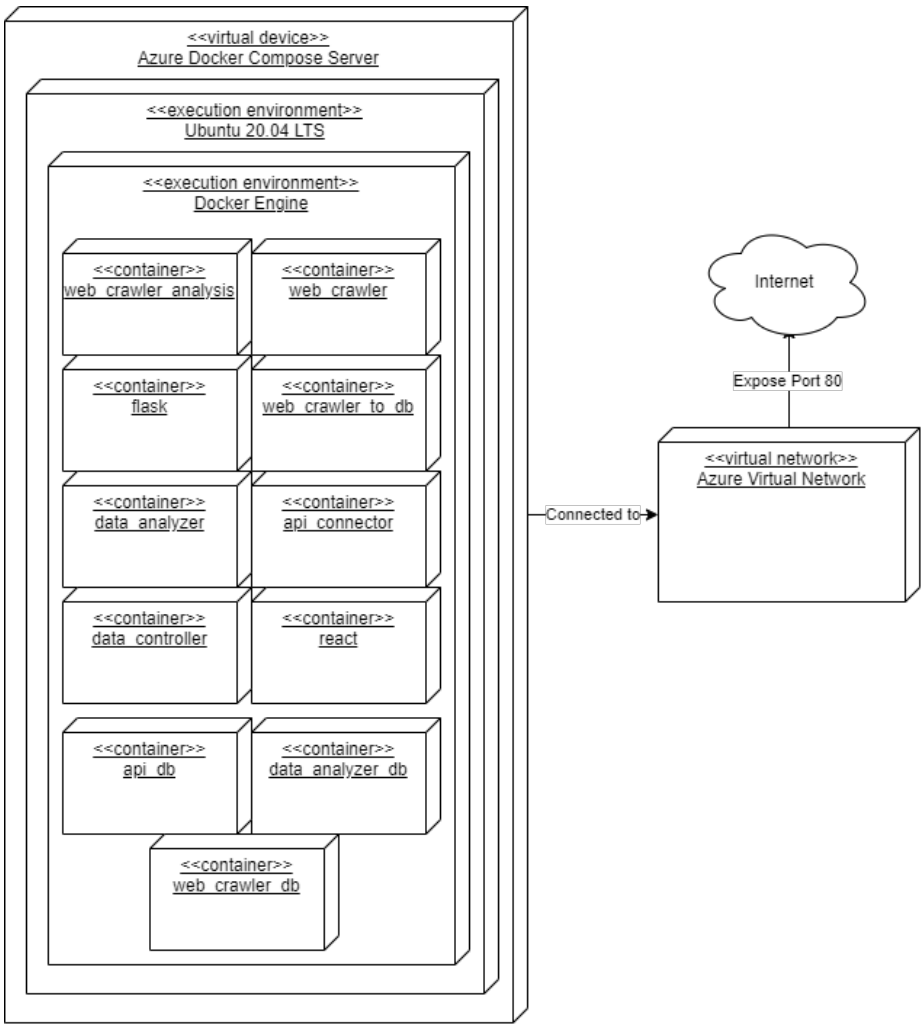
1. Der Data Analyzer startet den Prozess, indem er eine Datenanfrage an den Data Controller sendet. Dieser soll die für die Analyse benötigten Daten liefern.
2. Der Data Controller bearbeitet diese Anfrage, indem er die erforderlichen Daten aus der API-Datenbank (ADB) abruft.
3. Sobald der Datenabruf abgeschlossen ist, gibt die API DB die Informationen an den Data Controller zurück.
4. Der Data Controller leitet daraufhin die abgerufenen Daten an den Data Analyzer weiter.
5. Mit den erhaltenen Daten führt der Data Analyzer nun seine Analyse durch.

6. Nach Abschluss der Analyse speichert der Data Analyzer die ermittelten Ergebnisse in der Data Analyzer Datenbank (DADB).

7 Verteilungssicht [Nebel]

7.1 Infrastruktur Ebene Azure Cloud

Das folgende Verteilungsdiagramm zeigt die Software auf einem Azure Docker Host nach vollständigem Deployment des Docker Compose. Dieses Schaubild entspricht dem derzeitigem Deployment dieser Software wärrend der Certification.



Begründung Aufgrund der einfachen und zuverlässigen Hochverfügbarkeit der Azure Cloud-Plattform wird das Softwareprojekt auf einem Standard Docker Compose-Server von Azure gehostet. Dabei werden die bestehenden Microservices in einem Compose-Verbund ausgeführt. Durch die Nutzung der Azure-Plattform können die Vorteile der Hochverfügbarkeit,

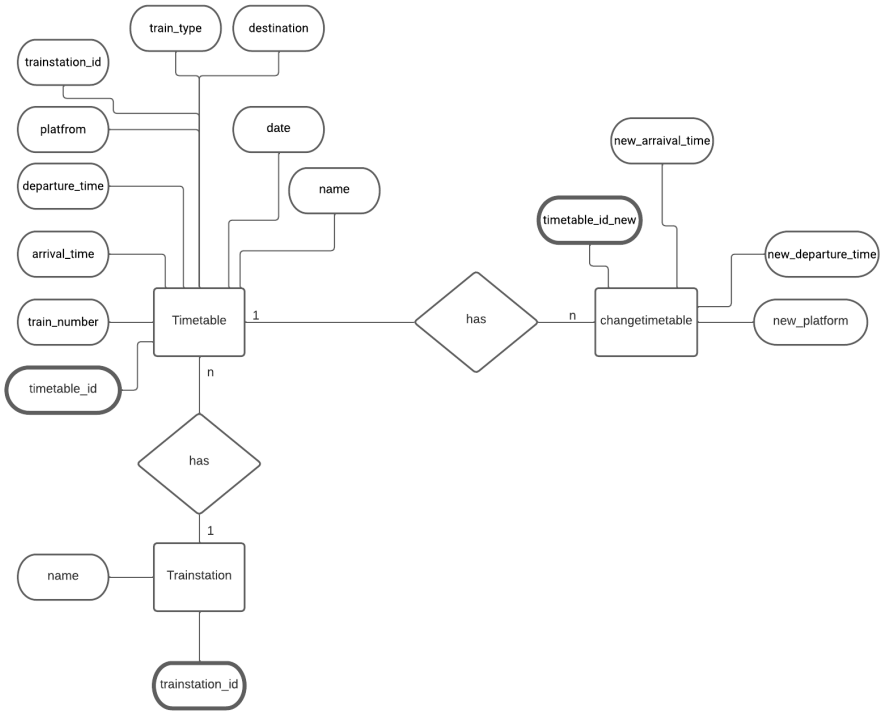
Skalierbarkeit und Zuverlässigkeit von Azure genutzt werden, um eine stabile und leistungsfähige Umgebung für das Projekt zu gewährleisten.

Qualitäts- und/oder Leistungsmerkmale Standard Azure Docker Compose Server mit 8GB RAM und 2vCPU Cores

Zuordnung von Bausteinen zu Infrastruktur Alle Microservices laufen in der Docker Engine des Azure Docker Compose Server. Die Container **Web crawler**, **Web crawler to db** und **Api connector** verbinden sich über das virtuelle Azure Netzerk mit ihren respektiven externen APIs. Der Container **React** wird dabei über das Azure Virtual Network auf **Port 80** über die externe Adresse exposed.

8 Querschnittliche Konzepte [Merlau]

8.1 Datenbank/ER Diagramm [Merlau]]



Datenbankstruktur

Die Datenbankstruktur des DB APIs basiert auf dem Konzept der Querschnittsbeziehungen. Es gibt insgesamt drei Tabellen, die über Schlüssel miteinander verknüpft sind.

TrainStation

- **TrainStation_ID:** Eindeutige ID des Zugbahnhofs (Primärschlüssel).
- **Name:** Name des Bahnhofs.
- Weitere Informationen wie Standort, Kapazität oder Serviceangebote können ebenfalls gespeichert werden.

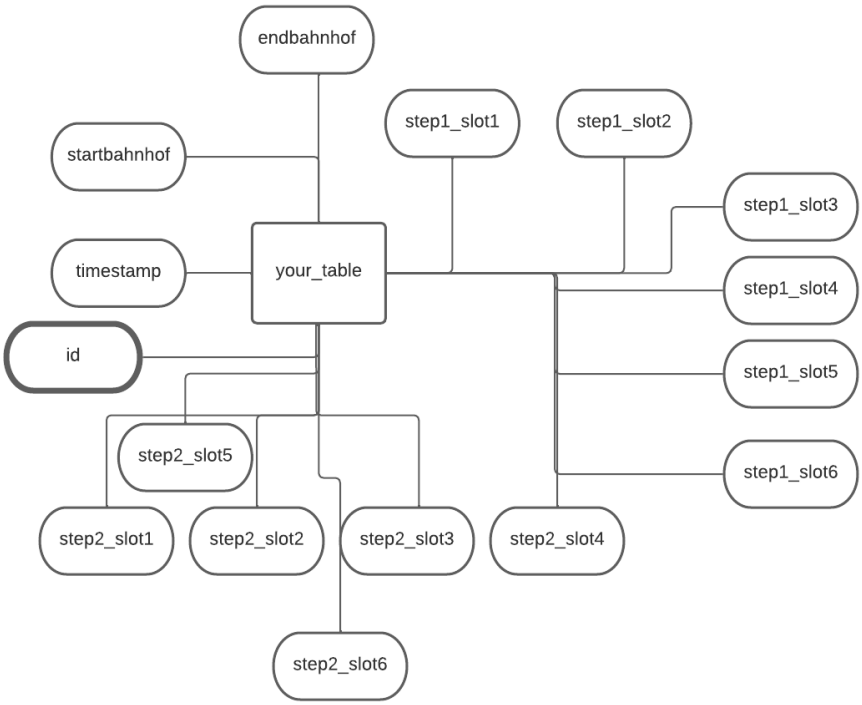
TimeTable

- **ID:** Eindeutige ID des Fahrplans (Primärschlüssel).
- **Date:** Datum des Fahrplans.
- **Time:** Uhrzeit des Fahrplans.
- **ArrivalTime:** Nummer des Zugs.
- **ArrivalTime:** Geplante Ankunftszeit.
- **DepartureTime:** Geplante Abfahrtszeit.
- **Platform:** Geplanter Bahnsteig.
- **TrainType:** Typ des Zugs (z.B. Regionalzug, Intercity).
- **Destination:** Liste der Zielhaltestellen des Zugs.
- **TrainStation_ID::** ID des zugewiesenen Zugbahnhofs (Fremdschlüssel).
- **TrainName:** Name des Zugs.

ChangeTimeTable

- **Timetable_ID:** ID des Fahrplans, auf den sich die Änderungen beziehen (Fremdschlüssel).
- **NewArrivalTimes:** Aktualisierte Ankunftszeiten.
- **NewTrainNumbers:** Aktualisierte Zugnummern.
- **NewPlatforms:** Aktualisierte Bahnsteige.
- **NewDepartureTimes:** Aktualisierte Abfahrtszeiten.

has Es besteht eine 1:n-Beziehung namens "has", die es ermöglicht, die TrainStation-Tabelle zu umgehen und die anderen Tabellen direkt über die IDs miteinander zu verbinden. Die Zugbahnhof_ID in der TimeTable-Tabelle dient als Fremdschlüssel, um eine Beziehung zwischen den Fahrplänen und den Zugbahnhöfen herzustellen.

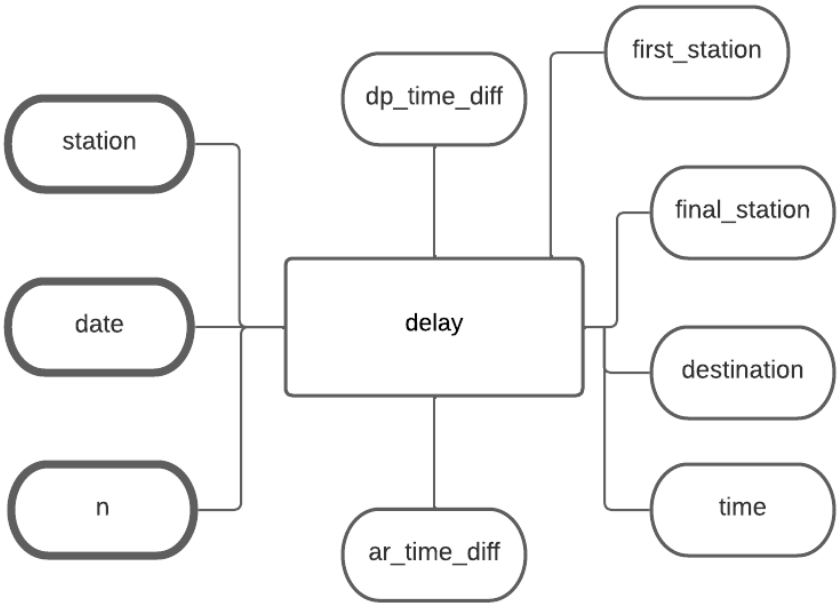


your_table

Diese Tabelle enthält Daten zu verschiedenen Schritten (‘step‘) und Slots (‘slot‘) im Zusammenhang mit einer Reise. Die Tabelle hat folgende Spalten:

- `id`: Eine eindeutige ID für jeden Eintrag in der Tabelle.
- `timestamp`: Ein Zeitstempel, der den Zeitpunkt des Eintrags in der Tabelle angibt.
- `startbahnhof`: Der Startbahnhof der Reise als Text.
- `endbahnhof`: Der Endbahnhof der Reise als Text.
- `stepX_slotY`: Numerische Werte (double precision) für die verschiedenen Slots (1-6) in jedem Schritt (1-5) der Reise. Diese Werte können verschiedene Informationen darstellen, die mit jedem Slot in Verbindung stehen.

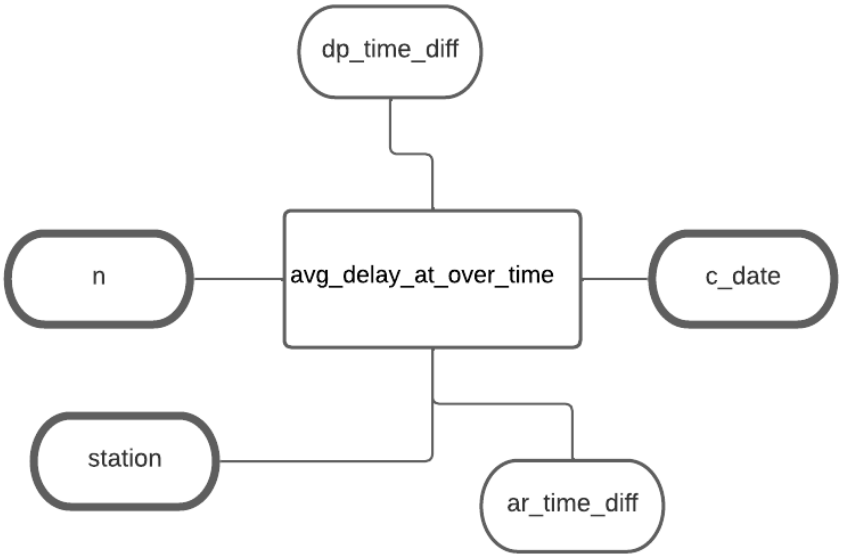
Die Tabelle ermöglicht die Speicherung und Organisation von Daten zu Reiseschritten und zugehörigen Slots. Jeder Eintrag in der Tabelle enthält Informationen zu einem bestimmten Zeitpunkt und den zugehörigen Werten für die Slots.



delay

Diese Tabelle speichert alle Verzögerungen und Basiswerte, die für weitere Berechnungen benötigt werden. Sie ist nicht dafür vorgesehen, außerhalb dieses Services verwendet zu werden. Die Tabelle hat folgende Spalten:

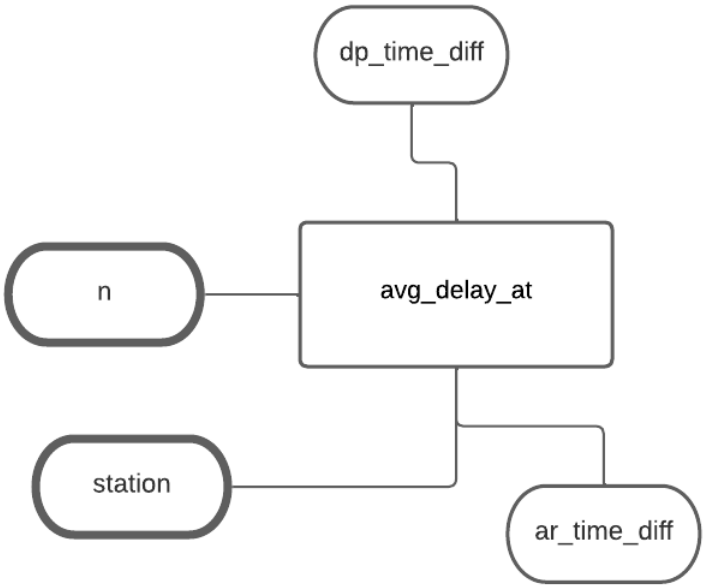
- `date`: Ein Datumswert, der als Primärschlüssel fungiert.
- `n`: Ein Ganzzahlwert, der als Primärschlüssel fungiert.
- `station`: Ein Textwert, der als Primärschlüssel fungiert.
- `ar_time_diff`: Ein Textwert, der die Ankunftszeitdifferenz repräsentiert.
- `dp_time_diff`: Ein Textwert, der die Abfahrtszeitdifferenz repräsentiert.
- `first_station`: Ein boolescher Wert, der angibt, ob es sich um die erste Station handelt.
- `final_station`: Ein boolescher Wert, der angibt, ob es sich um die Endstation handelt.
- `destination`: Ein Textwert, der das Ziel repräsentiert.



avg__delay__at__over__time

Diese Tabelle enthält den Durchschnitt der Verzögerungen zu unterschiedlichen Zeitpunkten. Die Tabelle hat folgende Spalten:

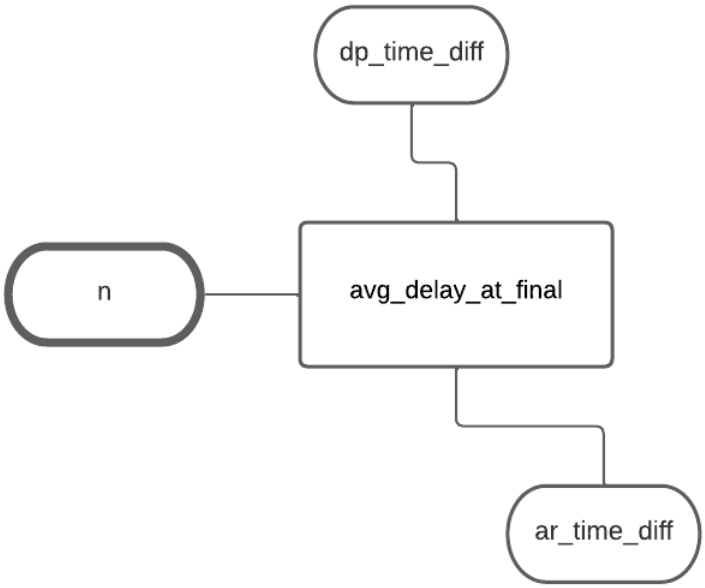
- `n`: Ein Ganzzahlwert, der als Primärschlüssel fungiert.
- `c_date`: Ein Datumswert, der als Primärschlüssel fungiert.
- `station`: Ein Textwert, der als Primärschlüssel fungiert.
- `ar_time_diff`: Ein Textwert, der die Ankunftszeitdifferenz repräsentiert.
- `dp_time_diff`: Ein Textwert, der die Abfahrtszeitdifferenz repräsentiert.



avg__delay__at

Die Tabelle `avg_delay_at` enthält den Durchschnitt aller vorhandenen Verzögerungen. Sie hat folgende Spalten:

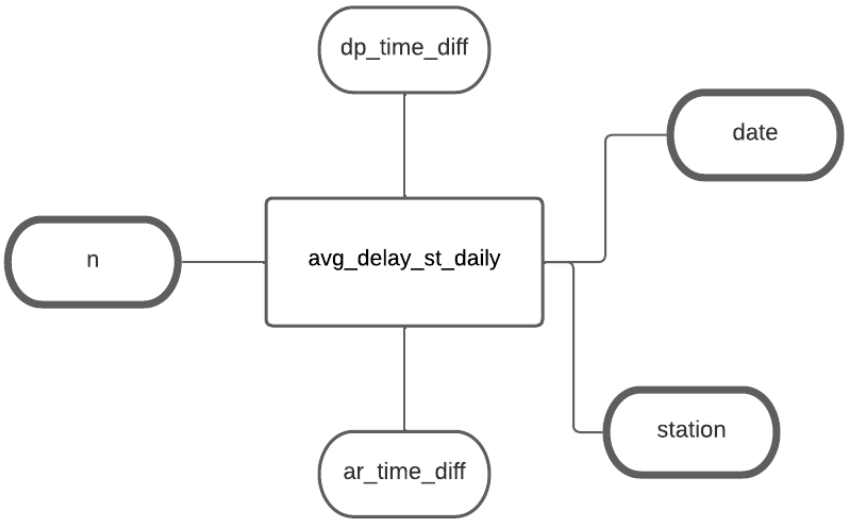
- `n`: Ein Ganzzahlwert, der als Primärschlüssel fungiert.
- `station`: Ein Textwert, der als Primärschlüssel fungiert.
- `ar_time_diff`: Ein Textwert, der die Ankunftszeitdifferenz repräsentiert.
- `dp_time_diff`: Ein Textwert, der die Abfahrtszeitdifferenz repräsentiert.



avg__delay__at__final

Die Tabelle `avg_delay_at_final` enthält den Durchschnitt der Verzögerungen am Anfang und am Ende einer gegebenen Linie. Sie hat folgende Spalten:

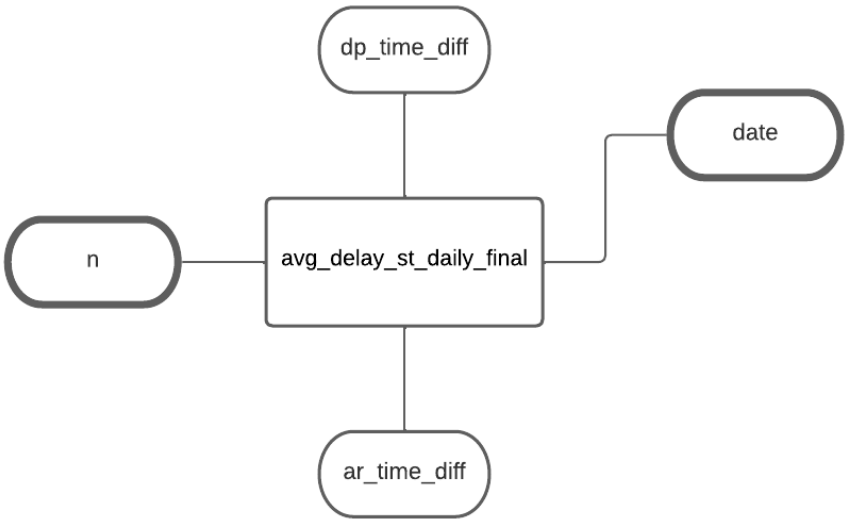
- `n`: Ein Ganzzahlwert, der als Primärschlüssel fungiert.
- `ar_time_diff`: Ein Textwert, der die Ankunftszeitdifferenz repräsentiert.
- `dp_time_diff`: Ein Textwert, der die Abfahrtszeitdifferenz repräsentiert.



avg__delay__st__daily

Die Tabelle `avg_delay_st_daily` enthält die durchschnittliche tägliche Verzögerung pro Linie. Beachte, dass die Linien pro Tag eindeutig sind, daher ist diese Tabelle derzeit identisch mit der Tabelle `delay`. Sie hat folgende Spalten:

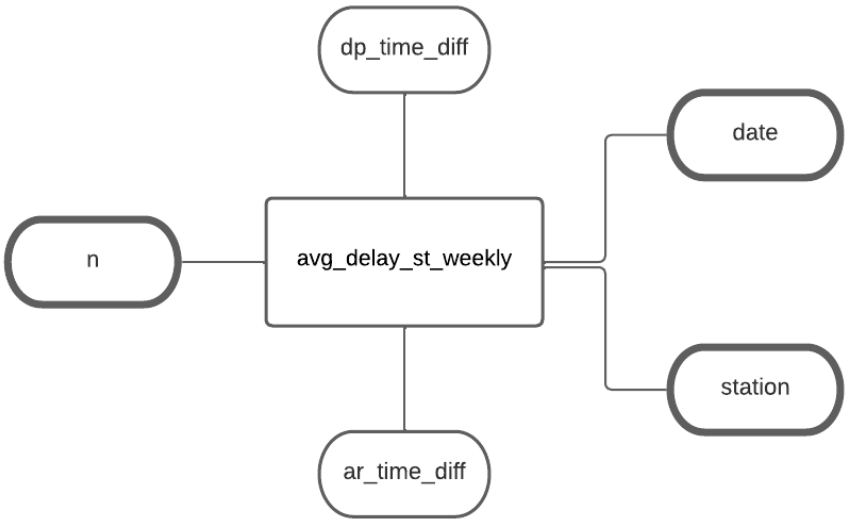
- `n`: Ein Ganzzahlwert, der als Primärschlüssel fungiert.
- `date`: Ein Datumswert, der als Primärschlüssel fungiert.
- `station`: Ein Textwert, der als Primärschlüssel fungiert.
- `ar_time_diff`: Ein Textwert, der die Ankunftszeitdifferenz repräsentiert.
- `dp_time_diff`: Ein Textwert, der die Abfahrtszeitdifferenz repräsentiert.



avg__delay__st__daily__final

Die Tabelle `avg_delay_st_daily_final` enthält die durchschnittliche tägliche Verzögerung pro Linie am Anfang und am Ende der Reise. Sie hat folgende Spalten:

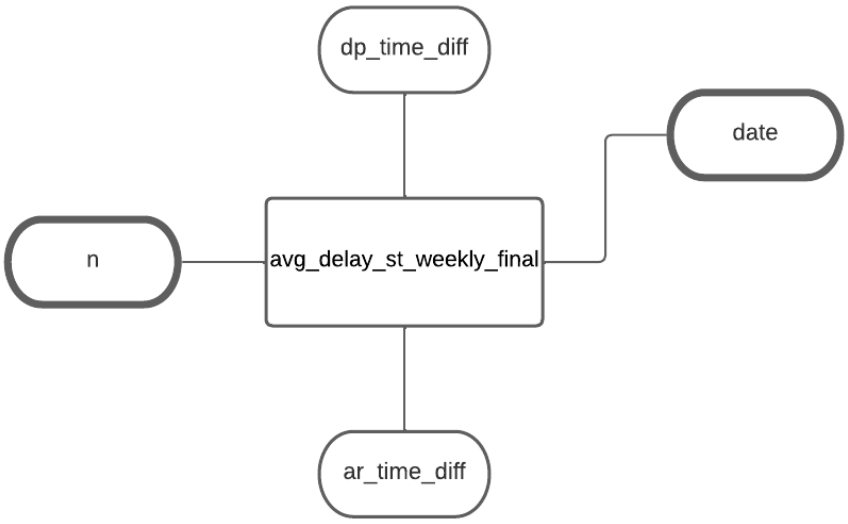
- `n`: Ein Ganzzahlwert, der als Primärschlüssel fungiert.
- `date`: Ein Datumswert, der als Primärschlüssel fungiert.
- `ar_time_diff`: Ein Textwert, der die Ankunftszeitdifferenz repräsentiert.
- `dp_time_diff`: Ein Textwert, der die Abfahrtszeitdifferenz repräsentiert.



avg_delay_st_weekly

Die Tabelle `avg_delay_st_weekly` ist ähnlich wie die oben genannte Tabelle, jedoch auf wöchentlicher Basis. Sie hat folgende Spalten:

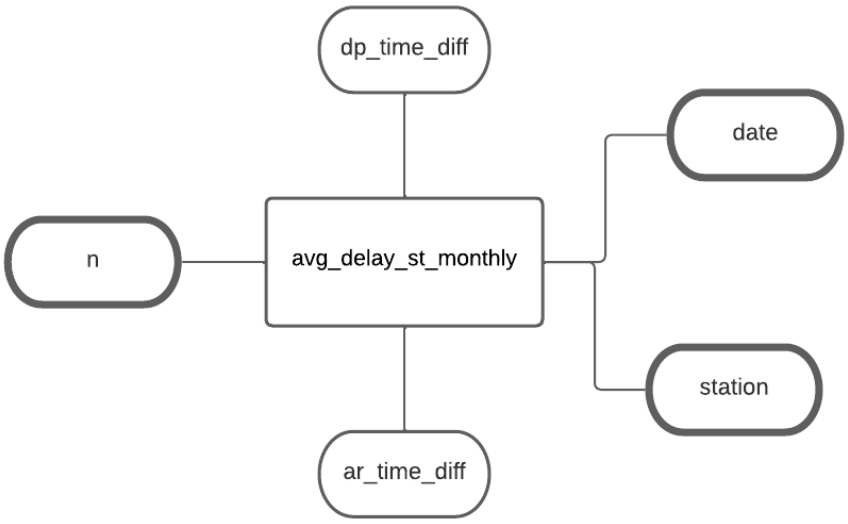
- `n`: Ein Ganzzahlwert, der als Primärschlüssel fungiert.
- `date`: Ein Datumswert, der als Primärschlüssel fungiert.
- `station`: Ein Textwert, der als Primärschlüssel fungiert.
- `ar_time_diff`: Ein Textwert, der die Ankunftszeitdifferenz repräsentiert.
- `dp_time_diff`: Ein Textwert, der die Abfahrtszeitdifferenz repräsentiert.



avg__delay__st__weekly__final

Die Tabelle `avg_delay_st_weekly_final` enthält die durchschnittliche wöchentliche Verzögerung pro Linie am Anfang und am Ende der Reise. Sie hat folgende Spalten:

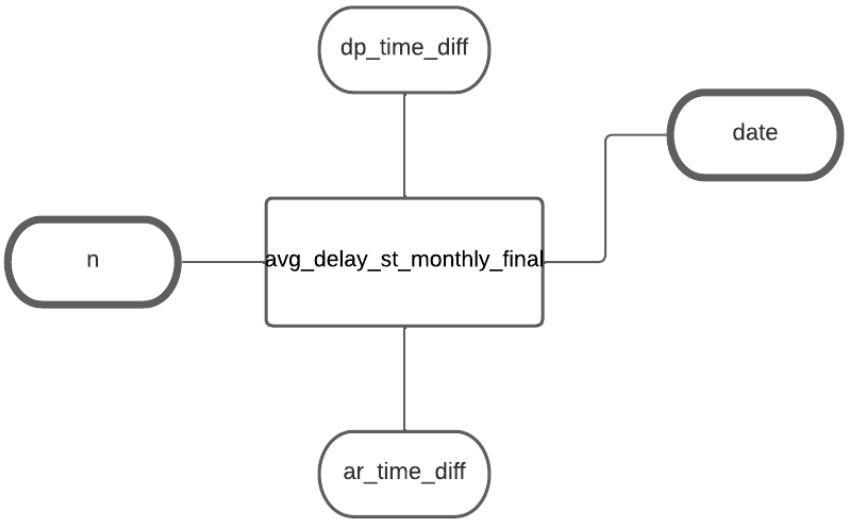
- `n`: Ein Ganzzahlwert, der als Primärschlüssel fungiert.
- `date`: Ein Datumswert, der als Primärschlüssel fungiert.
- `ar_time_diff`: Ein Textwert, der die Ankunftszeitdifferenz repräsentiert.
- `dp_time_diff`: Ein Textwert, der die Abfahrtszeitdifferenz repräsentiert.



avg__delay__st__monthly

Die Tabelle `avg_delay_st_monthly` enthält die durchschnittliche monatliche Verzögerung pro Station für Ankunfts- und Abfahrtszeiten. Sie hat folgende Spalten:

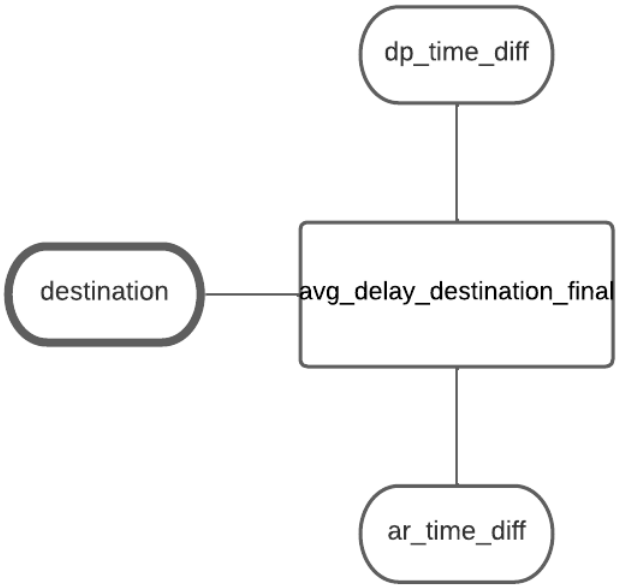
- `n`: Ein Ganzzahlwert, der als Primärschlüssel fungiert.
- `date`: Ein Datumswert, der als Primärschlüssel fungiert.
- `station`: Ein Textwert, der den Namen der Station repräsentiert.
- `ar_time_diff`: Ein Textwert, der die Ankunftszeitdifferenz repräsentiert.
- `dp_time_diff`: Ein Textwert, der die Abfahrtszeitdifferenz repräsentiert.



avg__delay__st__monthly__final

Die Tabelle `avg_delay_st_monthly_final` enthält die durchschnittliche monatliche Verzögerung am Start- und Zielpunkt einer Reise. Sie hat folgende Spalten:

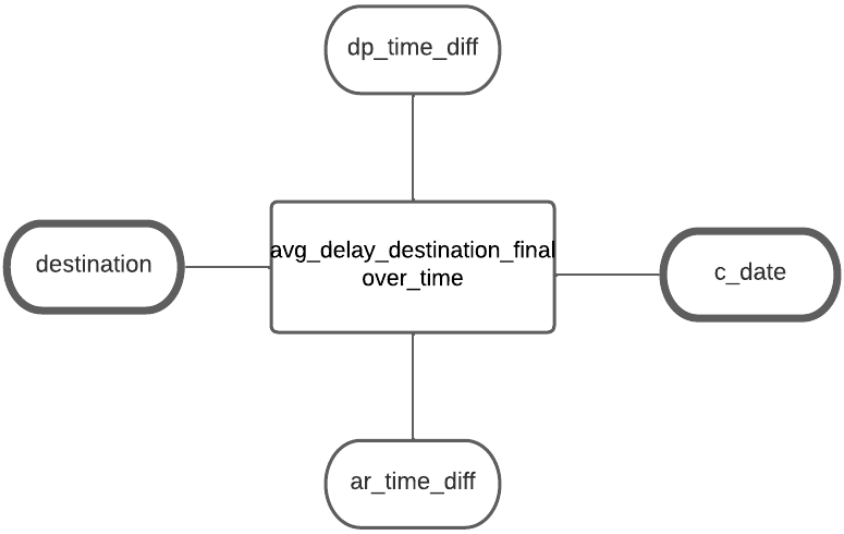
- `n`: Ein Ganzzahlwert, der als Primärschlüssel fungiert.
- `date`: Ein Datumswert, der als Primärschlüssel fungiert.
- `ar_time_diff`: Ein Textwert, der die Ankunftszeitdifferenz repräsentiert.
- `dp_time_diff`: Ein Textwert, der die Abfahrtszeitdifferenz repräsentiert.



avg_delay_destination_final

Die Tabelle `avg_delay_destination_final` enthält die durchschnittliche Verzögerung am Start- und Zielpunkt einer gesamten Strecke (z. B. von AA nach Ulm). Sie hat folgende Spalten:

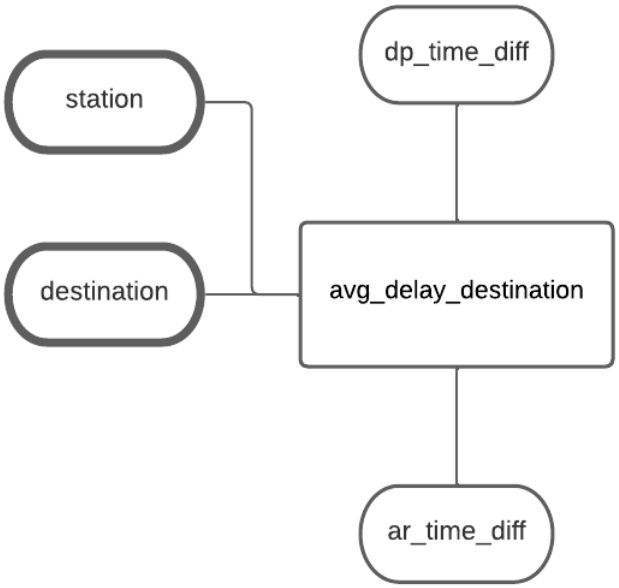
- `destination`: Ein Textwert, der das Ziel repräsentiert und als Primärschlüssel fungiert.
- `ar_time_diff`: Ein Textwert, der die Ankunftszeitdifferenz repräsentiert.
- `dp_time_diff`: Ein Textwert, der die Abfahrtszeitdifferenz repräsentiert.



avg__delay__destination__final__over__time

Die Tabelle `avg_delay_destination_final_over_time` enthält die durchschnittliche Verzögerung am Start- und Zielpunkt einer gesamten Strecke über verschiedene Zeitpunkte hinweg. Sie hat folgende Spalten:

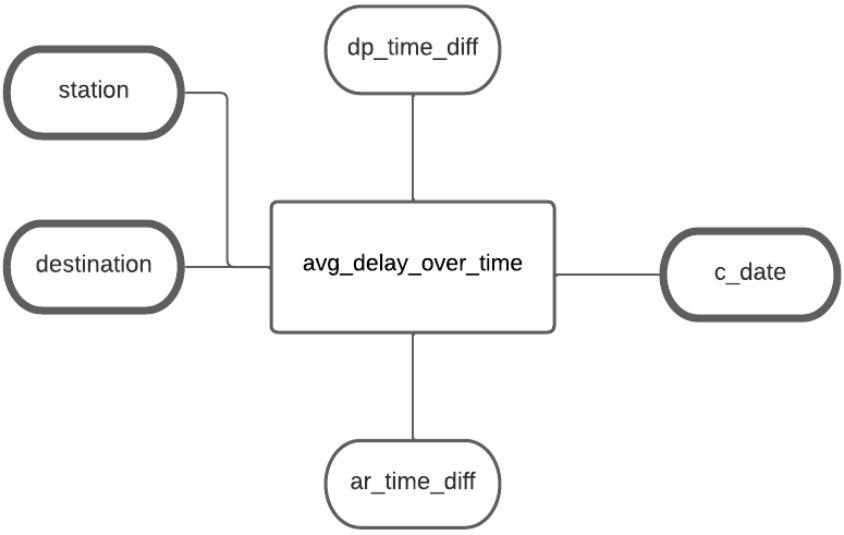
- `destination`: Ein Textwert, der das Ziel repräsentiert und als Primärschlüssel fungiert.
- `c_date`: Ein Datumswert, der als Primärschlüssel fungiert.
- `ar_time_diff`: Ein Textwert, der die Ankunftszeitdifferenz repräsentiert.
- `dp_time_diff`: Ein Textwert, der die Abfahrtszeitdifferenz repräsentiert.



avg_delay_destination

Die Tabelle `avg_delay_destination` enthält die durchschnittliche Verzögerung pro Station für eine gesamte Strecke. Sie ist etwas redundant, aber wichtig für Stationen mit mehreren Zielen. Sie hat folgende Spalten:

- `destination`: Ein Textwert, der das Ziel repräsentiert und als Primärschlüssel fungiert.
- `station`: Ein Textwert, der den Namen der Station repräsentiert und als Primärschlüssel fungiert.
- `ar_time_diff`: Ein Textwert, der die Ankunftszeitdifferenz repräsentiert.
- `dp_time_diff`: Ein Textwert, der die Abfahrtszeitdifferenz repräsentiert.



avg__delay__destination__over__time

Die Tabelle `avg_delay_destination_over_time` enthält die durchschnittliche Verzögerung pro Station für eine gesamte Strecke über verschiedene Zeitpunkte hinweg. Sie hat folgende Spalten:

- `destination`: Ein Textwert, der das Ziel repräsentiert und als Primärschlüssel fungiert.
- `station`: Ein Textwert, der den Namen der Station repräsentiert und als Primärschlüssel fungiert.
- `c_date`: Ein Datumswert, der als Primärschlüssel fungiert.
- `ar_time_diff`: Ein Textwert, der die Ankunftszeitdifferenz repräsentiert.
- `dp_time_diff`: Ein Textwert, der die Abfahrtszeitdifferenz repräsentiert.

8.2 REST-API [Merlau]

Die RESTful-API wurde in folgenden Containern implementiert:

- Data Controller
- Web Crawler
- Web Crawler Analyser
- Data Analyser

8.2.1 Data_Controller

GET /get_all

Hier können alle Daten in der Datenbank auf einmal abgerufen werden.

POST /delay_data

Hier können Daten zu Verspätungen übermittelt werden.

POST /data

Hier werden Daten vom API Connector für den normalen Fahrplan empfangen.

GET /destination/string:train_number

Mit dieser Schnittstelle können alle Haltestellen abgerufen werden, die von einem Zug mit der angegebenen Zugnummer angefahren werden.

GET /timetable/string:trainstation/string:date

Über diese Schnittstelle können die Fahrpläne für einen bestimmten Zugbahnhof an einem bestimmten Datum abgerufen werden.

GET /timetable/string:trainstation/string:date/string:hour

Diese Schnittstelle ermöglicht es, nach Zugbahnhof-ID, Datum und Stunde zu filtern und alle Fahrpläne für diese spezifische Stunde abzurufen.

8.2.2 Der Web_Crawler

POST '/plot'

Über diese Schnittstelle erhält man ein JSON mit einem Base64-codierten Bild.

8.2.3 Web_Crawler_Analyser

GET /metrics/<endbahnhof>

Diese Schnittstelle liefert den Durchschnittspreis, den Höchstpreis und den Mindestpreis für einen Endbahnhof in Schritten.

GET /budget/float:b/int:vonUhrzeit/int:bisUhrzeit/<endbahnhof>

Hier werden für ein bestimmtes Budget (b), eine bestimmte Uhrzeit von und bis, sowie den gewählten Endbahnhof die Preise zurückgegeben.

GET /preisanstiege/<endbahnhof>

Diese Schnittstelle gibt die Preisanstiege für den gewählten Endbahnhof zurück.

GET /preisanstiege_alle

Hier werden alle Preisanstiege zurückgegeben.

GET /preistrend/<endbahnhof>

Diese Schnittstelle liefert den Preistrend für den gewählten Endbahnhof zurück.

GET /basispreise/<endbahnhof>

Hier werden die Basispreise für den gewählten Endbahnhof zurückgegeben.

GET /preistrend_alle

Diese Schnittstelle gibt alle Preistrends zurück.

8.2.4 Data_Analyser

GET /<table_name>/<param1>/<value1>/<param2>/<value2>/<param3>/<value3>

Dieser Endpunkt ermöglicht es, nach bestimmten Parametern in der angegebenen Tabelle zu suchen. Du kannst bis zu drei Parameter-Wert-Paare angeben, um die Suche einzuschränken.

GET /<table_name>/<param1>/<value1>/<param2>/<value2>

Dieser Endpunkt ermöglicht die Suche mit zwei Parametern und ihren entsprechenden Werten in der angegebenen Tabelle.

GET /<table_name>/<param1>/<value1>

Dieser Endpunkt ermöglicht die Suche mit einem Parameter und seinem Wert in der angegebenen Tabelle.

GET /<table_name>/

Dieser Endpunkt gibt alle Einträge in der angegebenen Tabelle zurück, ohne Einschränkung durch spezifische Parameter.

Siehe oben die Tables.

Für weitere Details kann die API-Dokumentation in Swagger in unserem Projekt eingesehen werden im Porjektordner 23s-cdc-teamg7/src.

8.3 Deutsche Bahn API [Merlau]

Im Rahmen des Querschnittskonzepts nutzen wir die Timetable-API der Deutschen Bahn. Mit dieser API können wir die Timetable-Daten sowie Informationen zu bekannten Änderungen (knownChanges) und kürzlich vorgenommenen Änderungen (recentChanges) abrufen. Es besteht eine Limitierung von 60 Abfragen pro Minute für jede dieser Datenarten. Wir aktualisieren die knownChanges und recentChanges alle 30 Sekunden. Den Timetable rufen wir einmal täglich um 0:01 Uhr ab.

Um Daten abzurufen, senden wir eine Anfrage mit einer Trainstation-ID und einem Datum mit Uhrzeit. Daraufhin erhalten wir die entsprechenden Daten zurück.

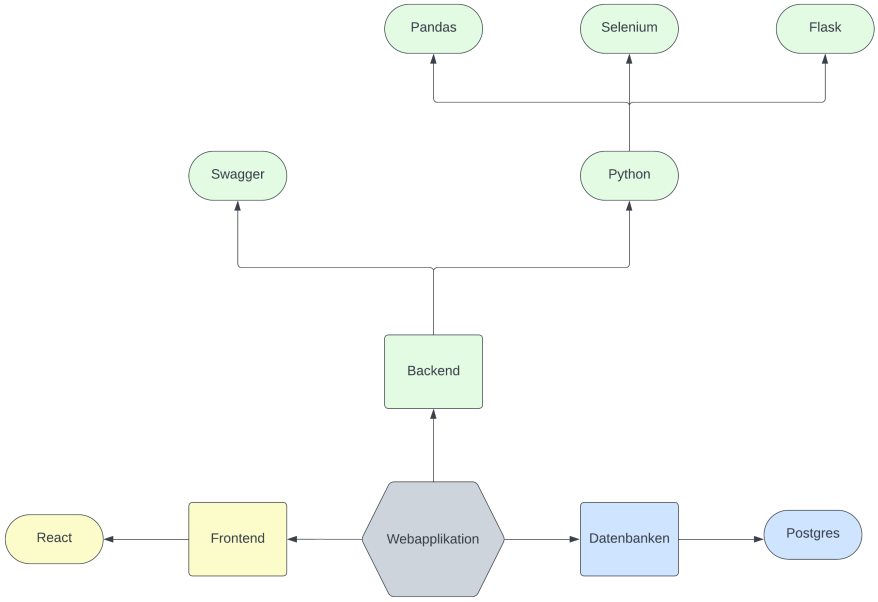
Weitere Informationen finden Sie unter dem folgenden Link: https://developers.deutschebahn.com/db-api-marketplace/apis/product/timetables/api/26494#/Timetables_10213/overview

8.4 User Interface [Mühleck]

Das Frontend wurde quasi als SPA (Single Page Application) designed. Man landet zuerst auf der Homepage und wird dann beim Klick auf den Start Button einmalig auf die zweite Seite weitergeleitet. Auf dieser Seite befinden sich alle Funktionen der Website in verschiedenen Tabs, diese leiten nicht auf andere Seiten weiter. Die einzelnen Funktionen sind visuell klar voneinander abgetrennt, aber jederzeit leicht zu erreichen. Das Frontend wurde repsonsive erstellt, um sowohl auf Desktop- als auch Mobilgeräten zu funktionieren.

9 Architekturentscheidungen [Müller]

Zunächst wird ein kleiner Überblick über die Verwendeten Technologien gegeben, welcher dann im einzelnen spezifiziert wird durch die Beschreibung und Beweggründe der Nutzung der jeweiligen Technologien ergänzt wird.



9.1 Frontend

Table 2: Technologien Frontend

React	• Reaktive Aktualisierungen ermöglichen automatische UI-Aktualisierungen beim Ändern des Zustands. • Komponentenbasierte Architektur ermöglicht die Entwicklung wiederverwendbarer und modularer UI-Komponenten. • Gut dokumentierte API und reichhaltige Ressourcen für Entwickler. • Einwegdatenfluss macht den Code vorhersehbarer und erleichtert das Debugging. • React Bootstrap ermöglicht eine schnellere und visuell ansprechendere Implementation durch die Bereitstellung von Templates.

9.2 Datenbanken

Table 3: Technologien Datenbank

Postgres	• Datenstruktur kann relational dargestellt werden. • Zuverlässigkeit und Robustheit durch ACID-Transaktionsunterstützung. • Regelmäßige Updates, Sicherheitspatches und umfangreiche Dokumentation.

9.3 Backend

Technologien Backend

Python	• Python bietet eine einfache Syntax. • Angebot umfangreicher Bibliotheken für die Datenanalyse, beispielsweise: Pandas, NumPy, Matplotlib. • Breite Community-Unterstützung • Python läuft auf verschiedenen Betriebssystemen, einschließlich Windows, macOS und Linux. Dies ermöglicht eine reibungslose Entwicklung und Ausführung von Datenanalyseprojekten auf verschiedenen Plattformen. • Python kann nahtlos mit anderen Tools und Sprachen integriert werden. Es ermöglicht beispielsweise die Verwendung von SQL-Datenbanken.
Selenium	• Selenium ist gut geeignet für das Scraping von Webseiten, die auf JavaScript basieren und dynamische Inhalte generieren. • Selenium ermöglicht das Simulieren von Benutzerinteraktionen wie Klicken auf Schaltflächen, Ausfüllen von Formularen und Scrollen. • Selenium unterstützt verschiedene Browser wie Chrome, Firefox, Safari und Edge, was Flexibilität und Kompatibilität bietet. • Selenium bietet Funktionen zur Steuerung von Wartezeiten und zum Warten auf bestimmte Bedingungen, um robuste und zuverlässige Scraping-Skripte zu erstellen. • Selenium bietet Möglichkeiten, Captchas zu umgehen, indem manuelle Eingaben simuliert oder auf Captcha-Lösungsdienste zurückgegriffen wird.
Pandas	• Tabellenähnliche Datenstrukturen (DataFrames) • Einfache Datenbereinigung und -transformation • Datenvisualisierung mit Matplotlib und Seaborn • Integration mit anderen Bibliotheken
Swagger	• Einfache und einheitliche Dokumentation von APIs • Automatische Generierung von interaktiven API-Dokumentationen • Erleichtert die Kommunikation zwischen Entwicklern, Frontend- und Backend-Teams • Vereinfacht das Testen und Debuggen von APIs

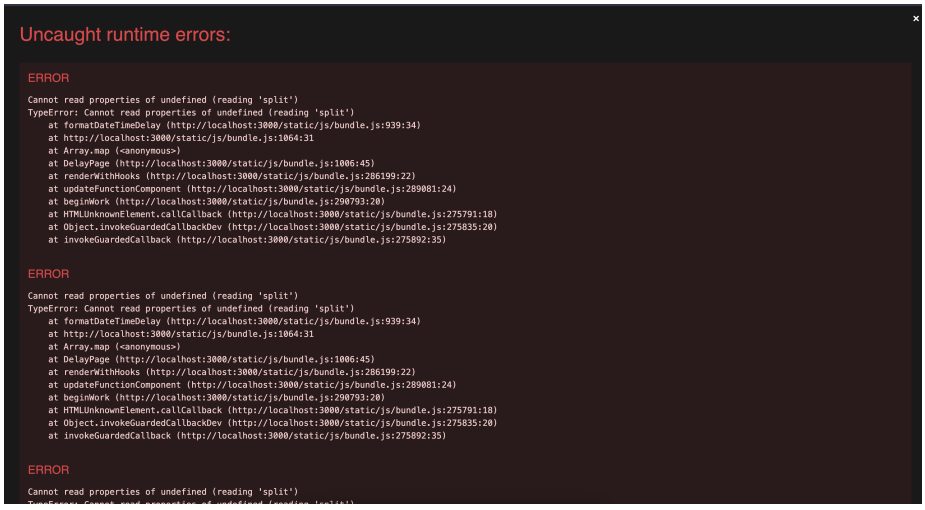
Table 5: Fortsetzung Technologien Backend

Flask	• Leichtgewichtig und einfach zu erlernen. • Flask ist sehr flexibel und erlaubt es Entwicklern, ihre Anwendungen nach ihren eigenen Bedürfnissen anzupassen. Es erzwingt keine bestimmte Projektstruktur und bietet eine Vielzahl von Erweiterungen und Plugins, die für verschiedene Anwendungsfälle verwendet werden können. • Flask basiert auf einem modularen Aufbau, bei dem Funktionen durch verschiedene Erweiterungen hinzugefügt werden können. • Flask bietet eine umfangreiche Unterstützung für das HTTP-Protokoll, was die Entwicklung von RESTful APIs und Webanwendungen erleichtert. • Allgemeine Wahl der Programmiersprache ist auf Python gefallen, daher wird die Integration verschiedener Python Bibliotheken und -Tools erleichtert, z. B. SQLAlchemy für die Datenbankintegration.

10 Risiken und technische Schulden [Müller]

10.1 Verspätungsanalyse

Aus Zeitgründen wurde die Visualisierung der Verspätungsanalyse im Frontend nicht vollständig implementiert. Dies führt zu dem folgenden Error:



Dieser Error geht aus der File: frontend/src/pages/delays.js (Zeile 163) hervor. Dieser Fehler entsteht wenn der Wert des Feldes avgdelaydestinationfinal nicht definiert ist.

10.2 Leistungsanforderungen an den Host

Durch Rechenaufwendige Datensammlung durch das Webcrawling können die Anforderungen an den Cloud Host, die durch die Hochschule zur Verfügung gestellten Instanzen übersteigen und somit nicht einsetzbar sein.

10.2.1 Eventualfallplanung

Es ist möglich für den Anwender eine Beispielausgabe für dynamisch angestoßene Crawl-Vorgänge auszugeben. Des Weiteren kann die Datenbank, welche durch den Crawler befüllt wird lokal befüllt werden. Diese Daten werden dann in die Cloud migriert, sodass alle weiteren Analysen und deren Services ohne Einschränkung genutzt werden können. Ein weiterer Ansatz ist, andere Anbieter in Betracht zu ziehen oder multiple Instanzen bei einem Anbieter zu initialisieren und dann zwischen diesen zu rooten.

10.2.2 Risikominderung

Der Aufwand wird gemindert, durch das definieren einiger Langsteckenbahnhöfe für das Crawling, welche sorgfältig ausgewählt wurden um möglichst viele Strecken abzudecken, ohne eine Vielzahl an Crawler Iterationen anstoßen zu müssen. Des weiteren wird der Zeitraum, nach dem gecrawlt wird auf folgende Reiseantritte beschränkt:

- Buchungsdatum +1 Tag
- Buchungsdatum +1 Woche
- Buchungsdatum +2 Wochen
- Buchungsdatum +3 Wochen
- Buchungsdatum +4 Wochen

Ebenfalls könnte untersucht werden in welchem zeitlichen Abstand der Webcrawler angestoßen werden muss um ein gesundes Mittelmaß zwischen Informationsverlust und Performance zu finden.

10.3 Datenquellen

Sowohl die API der Deutschen Bahn, als auch deren Website zur Ermittlung von Preisen unterliegen fortlaufenden Änderungen, diese Änderungen können bewirken, dass die Funktionalität nicht gewährleistet werden kann.

10.3.1 Eventualfallplanung

- Der Entwickler sollte die aktualisierte Dokumentation der API oder Website überprüfen, um die genauen Änderungen zu verstehen. Es ist wichtig, auf neue Endpunkte, geänderte Datenstrukturen oder aktualisierte Authentifizierungsmethoden zu achten. Durch die gründliche Analyse der Dokumentation erhält der Entwickler einen klaren Überblick über die Anpassungen, die in seinem Programm vorgenommen werden müssen.
- Der Entwickler sollte die Programmlogik an die geänderte API oder Website anpassen. Es ist wichtig sicherzustellen, dass die Anfragen, Parameter und Datenverarbeitung den neuen Spezifikationen entsprechen. Durch die Anpassung der Programmlogik gewährleistet der Entwickler, dass das Programm reibungslos mit der aktualisierten API oder Website interagiert und die gewünschten Ergebnisse erzielt werden können.
- Es ist empfehlenswert, umfangreiche Tests durchzuführen, um sicherzustellen, dass das Programm nach den Anpassungen ordnungsgemäß funktioniert. Dabei sollten verschiedene Szenarien getestet werden, um die reibungslose Integration gewährleisten zu können. Durch gründliche Tests kann der Entwickler potenzielle Fehler oder Probleme identifizieren und entsprechende Maßnahmen ergreifen, um die Stabilität und Funktionalität des Programms zu garantieren.
- Der Entwickler sollte die Fehlerbehandlungsmechanismen überprüfen und sicherstellen, dass das Programm angemessen auf Fehler oder unerwartete Antwortformatänderungen reagiert. Gegebenenfalls sollte er die Fehlerprotokollierung, Wiederholungsversuche und alternative Pfade anpassen. Es ist wichtig sicherzustellen, dass das Programm robust ist und korrekt auf auftretende Fehler reagiert, um eine unterbrechungsfreie Funktionalität zu gewährleisten. Der Entwickler sollte die vorgenommenen Änderungen entsprechend dokumentieren, um zukünftige Wartung und Aktualisierungen zu erleichtern.
- Überprüfung, ob die von Ihnen verwendeten Bibliotheken oder SDKs ebenfalls aktualisiert werden müssen, um mit den Änderungen in der API oder Website kompatibel zu sein.
- Dokumentation der durchgeführten Anpassungen in Ihrem Programm, einschließlich der spezifischen Änderungen in der API oder Website. Dies hilft dabei, die Wartung und zukünftige Aktualisierungen effizienter zu gestalten.

10.3.2 Risikominderung

- Halten Sie sich über offizielle Kanäle der Deutschen Bahn auf dem Laufenden, um Informationen über geplante Änderungen oder Versionierung der API zu erhalten. Dies kann über Entwickler-Blogs, Mailinglisten oder Foren erfolgen.
- Richten Sie Überwachungssysteme ein, die Änderungen in der API oder der Website erkennen können. Dadurch können Sie schnell auf Änderungen reagieren und Anpassungen vornehmen, um einen reibungslosen Betrieb Ihrer Anwendung sicherzustellen.
- Durch die Verwendung von generischen Funktionen können Sie die Abhängigkeit von spezifischen Implementierungsdetails reduzieren und Ihr Programm flexibler gegenüber zukünftigen Änderungen machen.

10.4 Aufwand

Durch die Komplexität in der Form der Daten, welche benötigt werden um sinnvolle Informationen Ableiten zu können, kann der Implementationsaufwand unerwartet ansteigen. Die vielseitigen Funktionen, die die Webapplikation bieten soll, könnten bewirken, dass der Aufwand, welcher in Verbindung mit der Kommunikation der einzelnen Services steht, den geplanten Rahmen übersteigt. Ebenfalls nicht zu vernachlässigen ist der zeitliche Aufwand der Einarbeitung von Entwicklern in neue Technologien, welche zur Erfüllung der Anforderungen essentiell sind.

10.4.1 Eventualfallplanung

Da es eine fixe Deadline in der Fertigstellung des Projektes gibt ist es leider nicht möglich den zeitlichen Rahmen der Bearbeitung zu verändern. Die einzige Möglichkeit besteht darin, dass die Entwickler sich mehr Zeit nehmen um das Projekt zu bearbeiten. Ist dies nicht möglich oder wirkt nicht genug entgegen können Zusatzfunktionen gestrichen werden, welche die Kernkompetenzen der Funktionalität nicht Maßgeblich einschränken (bspw. Comfort Features).

10.4.2 Risikominderung

Es wird ein klarer Plan erstellt welche Features essentiell sind, welche nützlich sind und welche schön wären. Einerseits werden hierdurch Kernfunktionalitäten definiert und somit ermöglicht, dass der Fokus in der Entwicklung nicht abschweift. Andererseits kann, wenn das Problem aufgetreten ist, anhand dieser Rangliste pragmatisch entschieden werden Welche Features von der Entwicklungsausgeschlossen werden sollen.

10.5 Aussagekraft

Die mangelnde Aussagekraft und Qualität der gewonnenen Informationen stellt ein Risiko dar, insbesondere aufgrund der Komplexität der Datenform und der vielseitigen Funktionen, die die Webapplikation bieten soll. Ebenfalls stellt die gegebene Limit der Deutschen Bahn API ein Risiko das in der mangelnden Menge an Daten, die Abgefragt werden können.

10.5.1 Eventualfallplanung

- Implementieren Sie Mechanismen zur Validierung und Bereinigung der Daten, um sicherzustellen, dass nur qualitativ hochwertige und korrekte Informationen verwendet werden. Überprüfen Sie die Daten auf Inkonsistenzen, fehlende Werte oder andere Fehler und beheben Sie diese, bevor Sie sie in die Analyse oder Verarbeitung einbeziehen.
- Stellen Sie sicher, dass die Daten aus verschiedenen Quellen effektiv integriert und verknüpft werden. Dies ermöglicht eine umfassendere und aussagekräftigere Analyse, indem Zusammenhänge zwischen den Daten hergestellt werden.
- Verwenden Sie fortschrittliche Analysetechniken und -modelle, um die gewonnenen Informationen besser zu verstehen und aussagekräftige Erkenntnisse zu gewinnen. Nutzen Sie beispielsweise maschinelles Lernen, um Muster und Zusammenhänge in den Daten zu erkennen und Vorhersagen zu treffen.

10.5.2 Risikominderung

Verwenden Sie agile Entwicklungsmethoden, wie z.B. Scrum, um flexibel auf Änderungen und Herausforderungen reagieren zu können. Durch regelmäßige Iterationen und Feedbackschleifen können Anpassungen vorgenommen und Risiken frühzeitig erkannt werden. Vielversprechend ist auch die Erstellung eines Prototypen und Proof-of-Concept-Implementierungen, um die Machbarkeit und Funktionalität der gewünschten Features zu validieren. Dies hilft dabei, potenzielle Probleme und Schwierigkeiten frühzeitig zu erkennen und anzugehen.

10.6 Rechtliche Hürden

Da Daten von der Deutschen Bahn analysiert und ausgegeben werden kann es zu Urheberrechsverletzungen kommen.

10.6.1 Eventualfallplanung

- Falls Sie feststellen, dass eine Urheberrechtsverletzung stattgefunden hat, sollten Sie den Rechteinhaber kontaktieren. Erklären Sie die Situation und bieten Sie an, die Angelegenheit zu klären. In einigen Fällen kann eine Vereinbarung über die Nutzung der Inhalte getroffen werden, falls dies möglich ist.
- Untersuchen Sie den Vorwurf der Urheberrechtsverletzung sorgfältig. Prüfen Sie, ob die behauptete Verletzung tatsächlich stattgefunden hat und ob Sie die erforderlichen Rechte zur Nutzung des betroffenen Materials haben. In einigen Fällen können missverständliche oder falsche Anschuldigungen erhoben werden.
- Entfernen Sie unverzüglich die betroffenen Inhalte aus Ihrer Webapplikation.
- Bei schwerwiegenden Urheberrechtsverletzungen ist es ratsam, rechtlichen Rat von Fachleuten einzuholen. Ein Anwalt mit Fachkenntnissen im Urheberrecht kann Ihnen dabei helfen, die rechtlichen Konsequenzen zu verstehen und angemessene Maßnahmen zu ergreifen.

10.6.2 Risikominderung

Um zukünftige Urheberrechtsverletzungen zu vermeiden, sollten Sie sicherstellen, dass Sie über die erforderlichen Rechte zur Nutzung von Inhalten verfügen. Wenn Sie auf Inhalte Dritter angewiesen sind, sollten Sie prüfen, ob Lizenzen oder Vereinbarungen erforderlich sind, um die Nutzung rechtmäßig durchzuführen.