Bachelorthesis

Konzeption und Implementierung eines React-Framewvorks zur flexiblen Erweiterung digitaler Datenbestände

Prüfer(in):
Prof. Dr. Christian Soltenborn

Verfasser(in):
Thomas Benjamin Hopf
101485

Heinrichstrasse 23
40764 Langenfeld

Wirtschaftsinformatik
Cyber Security

Eingereicht am:
10.06.2025

Sperrvermerk

Diese Arbeit enthält vertrauliche Informationen über die Firma Unternehmen GmbH. Die Weitergabe des Inhalts dieser Arbeit (auch in Auszügen) ist untersagt. Es dürfen keinerlei Kopien oder Abschriften - auch nicht in digitaler Form - angefertigt werden. Auch darf diese Arbeit nicht veröffentlicht werden und ist ausschließlich den Prüfern, Mitarbeitern der Verwaltung und Mitgliedern des Prüfungsausschusses sowie auf Nachfrage einer Evaluierungskommission zugänglich zu machen. Personen, die Einsicht in diese Arbeit erhalten, verpflichten sich, über die Inhalte dieser Arbeit und all ihren Anhängen keine Informationen, die die Firma Unternehmen GmbH betreffen, gegenüber Dritten preiszugeben. Ausnahmen bedürfen der schriftlichen Genehmigung der Firma Unternehmen GmbH und des Verfassers.

Die Arbeit oder Teile davon dürfen von der FHDW einer Plagiatsprüfung durch einen Plagiatsoftware-Anbieter unterzogen werden. Der Sperrvermerk ist somit im Fall einer Plagiatsprüfung nicht wirksam.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

1 Einleitung

Datenspeicherung und Verarbeitung ist ein zentraler Bestandteil des betrieblichen Ablaufs der Bayer AG. Dies gilt insbesondere für Abteilungen, die für Research and Development (R&D) zuständig sind. Alle Projekte, die im Sinne der Forschung durchgeführt werden, werden mitsamt zusätzlicher Attribute in einer Datenbank (Newport) hinterlegt, wodurch Budgetierung und Projekt-Planung effizient und effektiv möglich ist. Zwar besteht die Möglichkeit, viele Informationen in Newport zu hinterlegen, jedoch lassen sich die vorgegebenen Spalten nicht erweitern. Dadurch ist es nicht möglich, benutzerspezifische Informationen, die über den Horizont von Newport hinausgehen, zu speichern und verwenden. Um dieses Problem zu lösen, soll eine Benutzeroberfläche entwickelt werden, die diese Speicherung ermöglicht. Im Folgenden soll da Projekt vollumfänglich von Konzeption bis Umsetzung beschrieben, und Abläufe und Prinzipien erklärt werden.

2 Grundlagen und Hintergrund

Für die Umsetzung des Projekts ist theoretisches Wissen rund um Datenspeicherung -verarbeitung und -verteilung sowie Webdesign notwendig. Während grundlegendes Informatisches Fachwissen vorausgesetzt wird, werden alle weiteren für das Verständnis dieser Arbeit relevanten Konzepte im Folgenden erläutert.

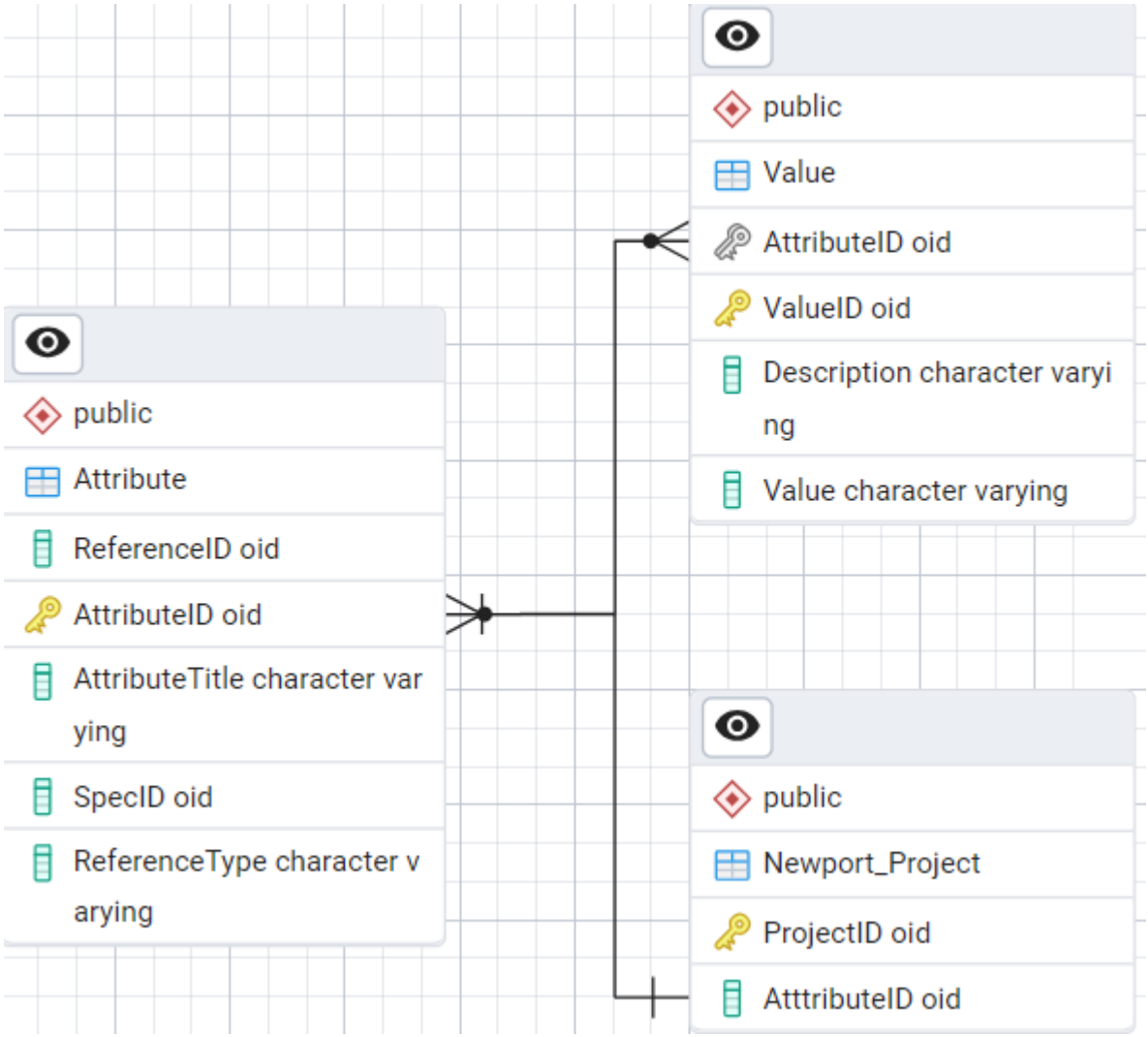
2.1 Fixed-Schema Datenbanken

Die Newport-Datenbank hat aufgrund ihrer großen Datenmenge und Nutzerbasis ein unveränderbares Schema. Das bedeutet, dass die vorgegebenen Spalten nicht bearbeitet, und auch keine neuen Spalten hinzugefügt werden können. Dadurch wird verhindert, dass die Anzahl an Spalten der Datenbank unkontrolliert wächst, was sich negativ auf Recheneffizienz und Übersichtlichkeit auswirken könnte. [1] Ein Nachteil dieses Konzepts ist jedoch die geringere Flexibilität. Ohne die Möglichkeit, neue Spalten hinzuzufügen, sind Nutzer auf die zum Zeitpunkt der Erstellung der Datenbank vorgesehenen Informationen beschränkt. Im Rahmen dieses Projekt soll diese Flexibilität durch Verwendung des Ättribute-Value-Pair=Modells in einer umschließenden Architektur gewährleistet werden, ohne aber die Vorteile einer Fixed-Schema Datenbank zu mindern

2.2 Attribute-Value-Pair Modell

Im Rahmen des Attribute-Value-Pair Modells (In Zukunft AVPM) werden Attribute, durch welche ein Objekt genauer beschrieben wird, in eine eigenständige Tabelle ausgegliedert. Diese Tabelle ist über einen Fremdschlüssel mit dem jeweils tragenden Objekt verbunden. Die Attribute sind in der Tabelle als Schlüssel-Wert-Paare gespeichert. Diese Struktur ermöglicht eine flexible Erweiterung der Attribute, ohne dass die Struktur der Datenbank verändert werden muss:

[1] quelle bitte



2.3 Technologischer Rahmen

Um dieses Modell im Backend umzusetzen, und dann im Front-End nutzbar zu machen, bedarf es verschiedener Technologien. Im Backend werden die Daten in einer Aurora PostgreSQL Datenbank gespeichert[2]. Aurora PostgreSQL (Im Folgenden Aurora) ist ein Dienst, der die Erstellung von relationalen Datenbanken ermöglicht. Auf dieser relationalen Datenbank ermöglicht eine in NodeJS [3] erstellte API den Zugriff auf und die Bearbeitung der gespeicherten Daten. Auf diese API greift dann widerum eine React-App zu, die dann ermöglicht, durch eine Benutzeroberfläche die API zur Kommunikation

[2]Diese Wahl beruht auf Standards im Unternehmen. Dazu mehr in der Evaluierung
[3]Bessere Kompatibilität mit React als alternativen

mit der Datenbank zu benutzen. Die Architektur ist bewusst mehrgliedrich gehalten, um Erweiterungen wie Daten-Redundanz zu ermöglichen [4] Die Wahl dieses technologischen Rahmens beruht auf vorhandenem Wissen und Infrastruktur im Unternehmen, sowie auch Abwägung der Entwicklungsgeschwindigkeit. Grundsätzlich stehen Enwticklern in der Web-Entwicklung jedoch viele Möglichkeiten zur Verfügung

2.4 Alternative Technologien

Für die Datenbank wäre es denkbar gewesen, anstatt einer relationalen auf eine sogenannte NoSQL Datenbanken zurückzugreifen. Eine solche Lösung wird auch im AWS-System angeboten. Der Grund, warum sich gegen diese Option entschieden wurde, liegt in der Natur der Datengrundlage. NoSQL eignet sich vor allem für große Mengen unstrukturierter Daten, während traditionelle Datenbanken besser mit strukturierten Daten umgehen können [5] Im Rahmen der Anwendung muss zwingendermaßen eine strenge Typisierung von Daten gegeben sein, da auf dessen Basis unter anderem Grafiken erstellt werden sollen. Aus diesem Grund ist eine strukturierte Datenbank für den speziellen Zweck des Projekts besser geeignet.

Die Entscheidung, NodeJS für die API zu verwenden, beruht hauptsächlich auf der zeitlichen Komponente der Entwicklung. React basiert auf einem NodeJS-Environment, weshalb es sich anbietet, die gleiche Umgebung für die APi zu verwenden. Grundsätzlich hätte auch jede andere Umgebung zur API-Erstellung ihren Zweck erfüllt, und diese Entscheidung beruht eher auf Bequemlichkeit. React hingegen als Umgebung wurde mit dem Hintergedanken der Modularität gewählt. Da das Projekt nicht nur eine einzene Anwendung, sondern eher einen Komplex kleinerer Tools beherbergen soll, bietet sich ein modulares Framework wie React an, wodurch dieses schon zu Beginn als Anforderung feststand, und als Basis aller weiteren Entscheidungen verwendet wurde. Zudem wird React unternehmensintern immer häufiger für verschiedenste Anwendungen verwendet, weshalb die Infrastruktur für das Deployment bereits aufgebaut ist.

[4] Unter anderem sollen die Daten in ein großes Daten Warehouse der Bayer Cropscience kopiert werden
[5] nosql source

3 Anforderungen und Zieldefinition

Die Anforderungen dieses Projekts gehen auf konkrete Bedürfnisse einer Nutzergruppe zurück, die im Arbeitsalltag intensiv mit dem System „Newport“ arbeitet.

Im Rahmen dieser Bachelor-Arbeit soll das Minimum Viable Product (MVP) des Projekts erarbeitet werden, also einer ersten lauffähigen Version mit den Kernfunktionen, die von der Nutzergruppe als unverzichtbar herausgearbeitet wurden. Entsprechend liegt in der Anforderungskonzeptiond er Fokus auf den sogenannten "Must-HaveAnforderungen, ohne welche die Anwendung ihren Zweck verfehlen würde.

Die durch interne Gespräche erfassten Nutzeranforderungen lassen sich in zwei Kategorien einteilen. An erster Stelle stehen die funktionalen Anforderungen, die sich direkt auf die Interaktion mit der Anwendung beziehen. Ebenfalls wichtig sind jedoch auch nicht-funktionale Anforderungen, die sich auf Qualität, Sicherheit, Performance und Wartbarkeit der Lösung beziehen.

3.1 Funktionale Anforderungen

[6] Im Rahmen des Minimum Viable Products (MVP) wurden folgende funktionale Anforderungen als wesentlich identifiziert. Sie bilden die Grundlage für die Nutzung des Systems durch Fachanwendende ohne tiefgehendes technisches Vorwissen. 1. Anlegen neuer dynamischer Attribute Nutzer*innen sollen die Möglichkeit haben, eigene Attribute anzulegen, die einem bestehenden Projektelement (z. B. einer Formulierung oder einem Projekt) zugewiesen werden können. Hierbei müssen folgende Angaben möglich sein:

Name des Attributs

Beschreibung (optional)

[6] Die beiden folgenden Abschnitte dienen mehr als Platzhalter als alles anderes. Ich habe zwar schon Informationen zu den Anforderungen, werden diese aber in den kommenden Wochen erst konkreter Ausformulieren

Datentyp (z. B. Text, Zahl, Datum)

2. Zuweisung von Attributen zu konkreten Einträgen
Bereits definierte Attribute sollen spezifischen Datenbankeinträgen zugeordnet und mit Werten befüllt werden können. Dies umfasst:

Auswahl eines vorhandenen Attributs

Eingabe eines entsprechenden Werts

Validierung des Werts entsprechend des Datentyps

3. Anzeige und Bearbeitung vorhandener Attribut-Werte
Im Benutzerinterface sollen alle bereits zugewiesenen Attribute mitsamt ihren Werten für ein bestimmtes Projektelement angezeigt und editierbar sein. Die Darstellung soll dynamisch erfolgen und sich an der Anzahl und Art der zugewiesenen Attribute orientieren.

4. Löschung von Attribut-Zuweisungen
Nutzer*innen sollen die Möglichkeit haben, bestehende Attribut-Wert-Paare wieder zu entfernen, ohne dabei das globale Attribut selbst zu löschen. Dadurch bleibt das Attribut für andere Einträge verfügbar.

5. Benutzerführung und Eingabeunterstützung
Die Oberfläche muss durch verständliche Beschriftungen, Platzhaltertexte, Tooltips oder andere visuelle Hinweise den Nutzer bei der Eingabe unterstützen. Validierungsfehler sollen unmittelbar und verständlich angezeigt werden.

3.2 Nicht-Funktionale Anforderungen

Neben den funktionalen Anforderungen, die die direkten Systemfähigkeiten betreffen, wurden verschiedene nicht-funktionale Anforderungen

identifiziert. Diese beschreiben die Qualitätsmerkmale des Systems und sind essenziell für den erfolgreichen Betrieb in einem unternehmenskritischen Umfeld wie dem von Bayer.

1. Performance

Das System muss eine hohe Reaktionsgeschwindigkeit aufweisen, um eine flüssige Nutzererfahrung zu gewährleisten.

Die Ladezeit für die Anzeige eines Projektelements inklusive dynamischer Felder soll unter 500 ms liegen (bei bis zu 20 zusätzlichen Attributen).

Schreiboperationen (z. B. Hinzufügen eines Attribut-Werts) sollen ohne spürbare Verzögerung erfolgen.

2. Skalierbarkeit

Die Lösung soll so ausgelegt sein, dass sie mit wachsender Datenmenge und Nutzerzahl ohne grundlegende Umstrukturierung betrieben werden kann.

Die Datenbankstruktur muss große Mengen an dynamischen Attribut-Werten performant verarbeiten können.

Die Architektur (Frontend, Backend, Datenbank) muss eine horizontale Skalierung ermöglichen (z. B. API-Lastverteilung).

3. Datenintegrität und Validierung

Es muss sichergestellt werden, dass sämtliche gespeicherte Daten vollständig, korrekt und konsistent sind.

Alle Eingaben durch die Nutzer:innen müssen auf Plausibilität und Formatkonformität geprüft werden.

Die Datenbank muss durch Constraints und Foreign Keys inkonsistente Zustände verhindern.

4. Wartbarkeit

Die Codebasis soll modular und verständlich strukturiert sein, sodass zukünftige Weiterentwicklungen oder Fehlerbehebungen effizient durchgeführt werden können.

Der Quellcode muss dokumentiert sein (Kommentare, Readmes, API-Spezifikationen).

Wiederverwendbare Komponenten und Services sollen sauber gekapselt sein.

5. Sicherheit

Das System muss grundlegende Sicherheitsanforderungen erfüllen, um unberechtigte Zugriffe zu verhindern und sensible Daten zu schützen.

Schreibende Aktionen (z. B. das Anlegen neuer Attribute) sollen nur autorisierten Nutzern erlaubt sein.

Die API muss gegen typische Angriffe (z. B. SQL-Injection, CSRF) abgesichert sein.

Benutzerfreundlichkeit (Usability) Die Benutzeroberfläche muss so gestaltet sein, dass auch nicht-technische Anwender:innen effektiv mit dem System arbeiten können.

Komplexe Prozesse wie das Anlegen neuer Felder sollen durch schrittweise geführte Dialoge unterstützt werden.

Fehlerzustände (z. B. ungültige Eingaben) müssen klar und verständlich kommuniziert werden.

3.3 Abgrenzung

Es ist zwingend notwendig, sich im Rahmen dieser Bachelorarbeit auf das MVP der Anwendung zu fokussieren, um eine abschließende Evaluation zu gewährleisten. Dementsprechend gibt es relevante Anforderungen an das fertige Produkt, die im Rahmen dieser Bachelorarbeit nicht umgesetzt werden. All diese Anforderungen sollen im Folgenden der Vollständigkeit halber aufgeführt und begründet werden.

1. Rollen-System

Viele verschiedene Nutzer werden für ihre jeweiligen Vorhaben in dieser Anwendung Attribute anlegen, und diesen Werte zuweisen. Es ist durchaus denkbar, dass zu solchen Attributen auch unter anderem geheime Daten gehören werden. Dementsprechend ist es für das finale Produkt zwingend notwendig, unbefugten Zugriff auf eingetragene Daten zu verhindern. Dies wird über ein Rollen-System innerhalb der Datenbank geschehen, und ist in der Theorie bereits konzeptioniert. Vorerst werden Nutzer jedoch angewiesen, keine solcher Daten zu speichern, bis dieses System fertig implementiert ist, was vorraussichtlich erst nach Ende des Bearbeitungszeitraums dieser Arbeit geschehen wird. Daher wird ein solches System in der Implementierung nicht beachtet.

2. Internationalisierung der Oberfläche

Das fertige Produkt soll, wie auch in vielen anderen Anwendungen der Fall (bspw. SAP)[7], in vielen verschiedenen Sprachen verfügbar sein. Das soll erreicht werden, indem dynamische Language Files eingesetzt werden, die auch eigens von Nutzern bereitgestellt werden können. Eine solche Implementation ist jedoch zeitaufwendig und aufgrund der Sprachkenntnisse der Nutzer nicht von elementarer Wichtigkeit, wird deshalb in dieser Arbeit

[7] quelle bitte

keine Verwendung finden.
3. Keine Integration in bestehende interne Systeme
Das Projekt soll zwar publiziert werden, insbesondere um Nutzer-Rückmeldungen erhalten zu können, jedoch soll es noch nicht in die restliche IT-Landschaft integriert werden, da das über lange Zeit und unter großer Vorsicht geschehen muss, um Kompatibilität zu garantieren.

außer diesen Anforderungen existieren mehrere Erfolgskriterien, die für das Gelingen dieses Projekts elementar wichtig sind. Diese sollen im Folgenden herausgearbeitet werden.

3.4 Erfolgskriterien

In erster Linie gilt das Projekt als gescheitert, sollten die funktionalen und nicht-funktionalen Kriterien nicht erfüllt sein. Demnach ist es zwingend erforderlich, dass Nutzer die Anwendung zweckgemäß nutzen können, sowohl in Anbetracht der Funktionalität, als auch der Performanz. Um diese Kriterien in der Evaluation bewerten zu können, werden im Folgenden (soweit möglich) KPIs (Key Performance Indicators) definiert.

3.4.1 Funktionale Zielerreichung

Hierfür soll in erster Linie ein Abgleich zwischen den durch die Nutzer gestellten Anforderungen und der Anwendung durchgeführt werden. Die Anwendung hat alle definierten "Must-HaveAnforderungen zu erfüllen. Zudem sollen Nutzerinterviews geführt werden[8], um die Nutzerakzep-

[8] rausstreichen wenn kb drauf

tanz zu messen, und somit die Zweckmäßigkeit der Benutzeroberfläche zu evaluieren.

3.4.2 Nicht-Funktionale Anforderungen

Bei durchschnittlicher Nutzung sollte sich die Response-Time der Anwendung in verhältnismäßigen Bereichen bewegen. Diese sind unabhängig vom Endgerät des Nutzers, da es sich um eine Cloud-Applikation handelt, und sollten 2 Sekunden nicht überschreiten [9]

3.4.3 Zukunftstauglichkeit und Wartbarkeit

Zusätzlich zu der imminenten Funktionalität des Programms muss gewährleistet sein, dass die Instandhaltung, sowie eventuelle Erweiterungen der Anwendung in Zukunft möglich sind. Hierfür ist es wichtig, dass die Codebasis modular und verständlich strukturiert ist, was durch eine saubere Trennung von Frontend und Backend, sowie durch die Verwendung von Frameworks, die eine klare Struktur vorgeben erreicht wird. Über diese Arbeit hinaus muss demnach eine Dokumentation gegeben sein, mitdessen Hilfe der Code der Anwendung verständlich ist.

[9] Hier ABgleich mit anderen Systemen für Vergleichbarkeit. Außerdem Gespräche mit Joerg maybe, um das herauszuarbeiten, auch it nuzern?

4 Systemdesign und Herausforderungen

Die Architektur ist im Sinne der Übersicht und Erweiterbarkeit klar in drei Teile aufgeteilt. Zur Datenspeicherung wird eine Aurora-Datenbank verwendet, mit welcher über eine NodeJS API von einem in React erstellten User Interface kommuniziert werden kann. Die Datenbank soll in der Lage sein, Attribut-Wert-Paare flexibel beherbergen zu können, und dabei die bestmögliche Performanz aufweisen.

4.1 Datenbankarchitektur

Um die Erweiterbarkeit der Anwendung zu garantieren, muss die Datenstruktur fest sein. [10] Trotzdem muss die Datenbank aber in der Lage sein, flexibel unterschiedliche Attribute und Werte speichern zu können. Das soll folgende Datenstruktur erreichen:

Die Datentruktur lässt sich in drei wesentliche Schichten aufteilen: An erster Stelle stehen die festen Datenobjekte, welche gleichlautend aus der bestehenden Infrastruktur übernommen werden. Diese Schicht bildet das Bindeglied zwischen der bereits existierenden Datenlandschaft und der von dieser Anwendung angehängten flexiblen Daten. Darauf folgt die dynamische Attributzuordnung, die über Fremdschlüssel direkt mit der ersten Schicht verbunden ist. Sie dient zur Verlinkung verschiedener Attribute mit ihren jeweiligen Eltern-Tabellen, speichert zusätzlich aber auch grundlegende Informationen über die Attribute ab. Die dritte Schicht beherbergt die Werte, und verbindet diese mit ihren zugehörigen Attributen. In dieser Schicht sind Informationen zu der Natur der Werte, aber auch

[10] quelle bitte

zum Inhalt abgespeichert. Im Folgenden sollen diese drei Schichten mir ihren jeweiligen Tabellen genauer beleuchtet werden.

4.1.1 Schicht 1 Domänenspezifische Tabellen

„Newport_Project“ bildet das übergeordnete Forschungsprojekt und ihre zugehörigen Daten ab, wobei F̈ormulationn̈ur ein Forschungsobjekt beschreibt, welches an ein Projekt geknüpft sein kann, aber nicht unbedingt muss:

Tabelle 1: Newport_Project

Spalte	Datentyp	Schlüssel?	Beschreibung
ProjectID	Object Identifier (oid)	Primärschlüssel	Primärer Identifier für den Eintrag
AttributeID	Object Identifier (oid)	Fremdschlüssel	Verweis auf zugewiesene Attribute für das jeweilige Projekt

Quelle: Eigene Darstellung

Tabelle 2: Formulation

Spalte	Datentyp	Schlüssel?	Beschreibung
FormulationID	Object Identifier (oid)	Primärschlüssel	Primärer Identifier für den Eintrag
AttributeID	Object Identifier (oid)	Fremdschlüssel	Verweis auf zugewiesene Attribute für das jeweilige Projekt
ProjectID	Object Identifier (oid)	Fremdschlüssel	Verweis auf ein gegebenenfalls zugewiesenes Newport-Projekt

Quelle: Eigene Darstellung

Tabelle 3: Newport_Segments

Spalte	Datentyp	Schlüssel?	Beschreibung
SegmentID	Object Identifier (oid)	Primärschlüssel	Primärer Identifier für den Eintrag
AttributeID	Object Identifier (oid)	Fremdschlüssel	Verweis auf zugewiesene Attribute für das jeweilige Segment
ProjectID	Object Identifier (oid)	Fremdschlüssel	Verweis auf das zugehörige Newport-Projekt

Quelle: Eigene Darstellung

Tabelle 4: Formulation_Segments

Spalte	Datentyp	Schlüssel?	Beschreibung
SegmentID	Object Identifier (oid)	Primärschlüssel	Primärer Identifier für den Eintrag
Formulation	Object Identifier (oid)	Fremdschlüssel	Verweis auf zugewiesene Attribute für das jeweilige Segment
ProjectID	Object Identifier (oid)	Fremdschlüssel	Verweis auf das zugehörige Newport-Projekt

Quelle: Eigene Darstellung

4.2 Herausforderungen

Im Folgenden sollen die Herausforderungen, die die Anforderungen an das Projekt posieren, und mögliche Lösungsansätze für diese diskutiert werden.

4.2.1 Datenkonsistenz

Aufgrund der hohen Flexibilität der einzutragenen Daten muss die Datenbank modular aufgebaut sein. Um die Datenkonsistenz zu garantieren, muss aber verhindert werden, dass durch die API (Direkte Manipulation ist in der Architektur nicht vorgesehen) inkorrekte Daten in die Datenbank eingespeist werden. Dafür müssen alle API-Requests noch vor

Tabelle 5: Attribute

Spalte	Datentyp	Schlüssel?	Beschreibung
ReferenceType	String (oid)	/	Automatisch gesetzte wörtliche Referenz auf den Owner des Attributs
ReferenceID	Object Identifier (oid)	Fremdschlüssel	Direkter Verweis auf den Owner des Attributs
AttributeID	Object Identifier (oid)	Primärschlüssel	Verweis auf das zugehörige Attribut
AttributeTitle	String	/	Titel des Attributs
SpecID	Object Identifier (oid)	Fremdschlüssel	Verweis auf die zugehörige Spezifikation

Quelle: Eigene Darstellung

Tabelle 6: Value

Spalte	Datentyp	Schlüssel?	Beschreibung
AttributeID	Object Identifier (oid)	Fremdschlüssel	Verweis auf das zugehörige Attribut
ValueID	Object Identifier (oid)	Primärschlüssel	Primärer Identifier für den Eintrag
Description	String	/	Name des Werts
Value	String	/	Wert

Quelle: Eigene Darstellung

Absendung auf Validität geprüft werden. Schlägt die Prüfung an, soll der Nutzer über den Fehler inkenntnis gesetzt werden. Da die Datenbank modular aufgebaut ist, muss die Erweitbarkeit und Performanz sichergestellt werden.

4.2.2 Erweiterbarkeit und Performanz

Um die Erweiterbarkeit zu garantieren, speichert die Datenbank auch Informationen, die im momentanten Scope noch nicht zwingend Erforderlich sind, jedoch bei eventuell auftretenden Erweiterungen notwendig werden können. Ein Beispiel dafür ist die Tabelle „Specification“. Sie

Tabelle 7: Specification

Spalte	Datentyp	Schlüssel?	Beschreibung
SpecID	Object Identifier (oid)	Primärschlüssel	Primärer Identifier für den Eintrag
Unit	String	/	Einheit

Quelle: Eigene Darstellung

speichert genauere Informationen über die Eigenschaften der Werte wie der Einheit ab. Das ist noch nicht erforderlich, da alle vorhersehbaren Daten Numerisch sind, sobald sich das aber ändert kann die Erweiterung ablaufen, ohne die Datenbank-Struktur zu verändern, was wichtig ist, da für eine solche Veränderung das gesamte Backend umgeschrieben werden muss. Es ist effizienter, die möglichen Veränderungen sofort einzuplanen. Dabei darf die Performanz aber nicht vernachlässigt werden. Durch die vielen Sicherheitsmechanismen, hinter welchen die Daten auf AWS gelagert werden wird die Response-Time ohnehin verlängert, der restliche Prozess muss sich daran also anpassen. Der Nutzer soll jederzeit über den Fortschritt ausstehender Anfragen informiert werden, die Anwendung muss, auch während einer laufenden Anfrage, weiter bedienbar sein und alle Prozesse rund um den Datenabruf müssen optimiert sein. Ein Beispiel für eine optimierende Planung in der Architektur ist die Spalte „ReferenceType“. Durch diese Spalte, die automatisch gesetzt wird, müssen die großen Tabellen von Attribut und Attribut-Owner nicht miteinander verbunden werden, um sie zuzuordnen, wodurch (bei größeren Datenmengen in den Tabellen) die Response Time deutlich verbessert wird. Trotz dieser Optimierungen darf aber die Architektur nicht so komplex werden, dass die Wartbarkeit darunter leidet.

4.2.3 Wartbarkeit und Benutzerfreundlichkeit

Die Gesamtarchitektur muss stets, trotz aller Herausforderungen, simpel genug sein, um mit möglichst geringem zusätzlichen Arbeitsaufwand gepflegt werden zu können. Dementsprechend muss die Gesamtheit der Architektur bis ins Detail dokumentiert, und alle Entscheidungen festgehalten werden. Neben den Entwicklern, die für die Wartung zuständig sind, müssen aber auch die Nutzer in der Lage sein, die Anwendung zu verstehen und effektiv anzuwenden. Deshalb muss das User Interface so selbsterlklärend wir möglich sein, und alles weitere in einer User-Dokumentation festgehalten werden, auf welche auch offensichtlich genug verwiesen wird.

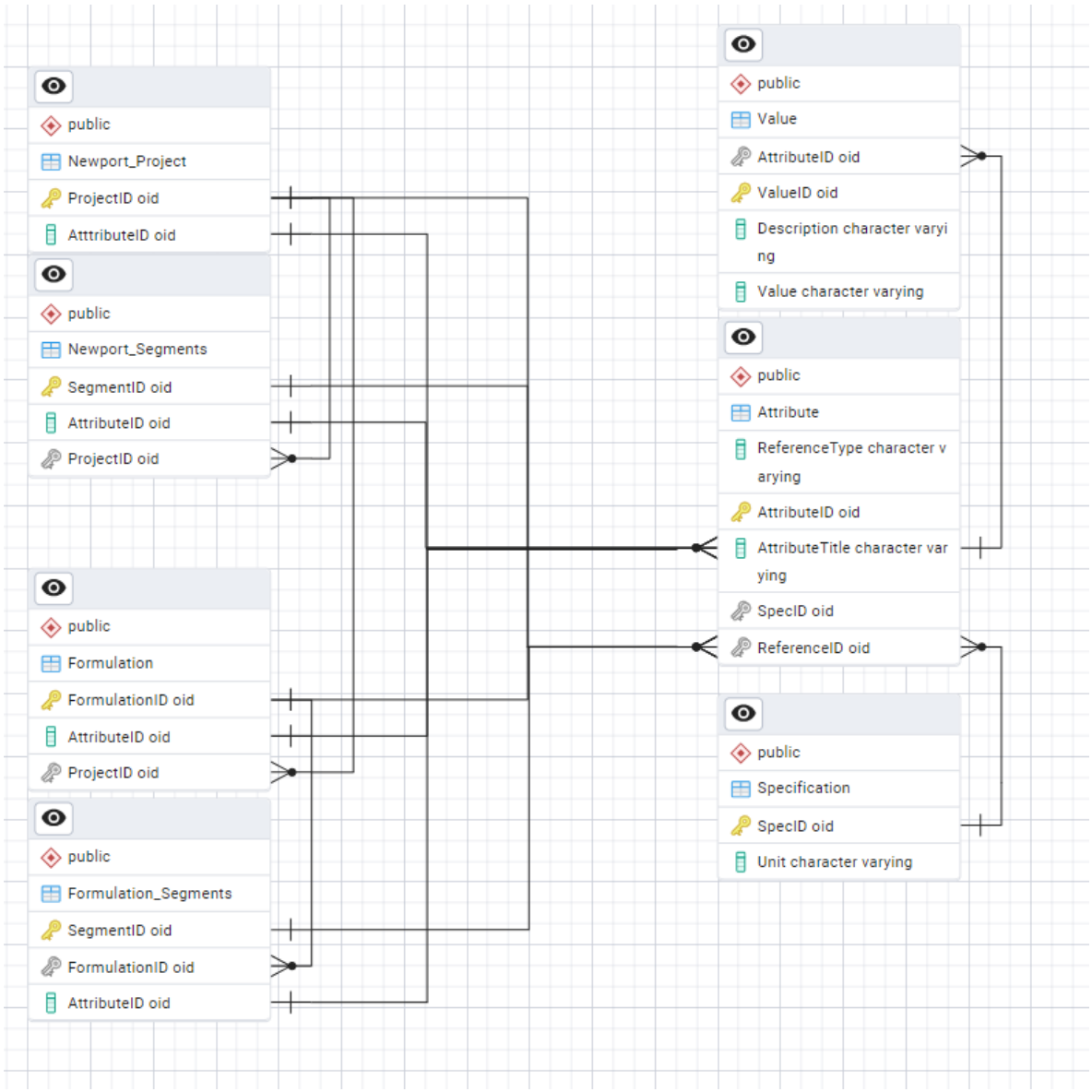


Abbildung 1: Modell der Datenbank

5 Datenmodellierung

Im Sinne der Modularität basiert die Datenmodellierung auf dem Entity-Value-Attribut-Konzept (EAV). Dieses Konzept ermöglicht die Erweiterung des starren relationalen Schema um eine variable Anzahl benutzerdefinierter Attribute, ohne dabei eine Schema-Veränderung zu erfordern. Dadurch sind die dynamischen Attribute von den Domänenobjekten (Projekte und Formulationen) getrennt. Im Folgenden soll der Aufbau und die Funktionsweise des Datenmodells genauer erläutert werden.

5.1 ER-Modell und Tabellenübersicht

Das relationale Schema gliedert sich in 3 Hauptblöcke: Domänentabellen, Attributdefinition und der Werteebene.

5.1.1 Domänentabellen

„Newport_Project“ verbindet die Datenbank über den Primärschlüssel „ProjectID“ mit der Newport-Datenbank aus dem CropScience Warehouse (CSW). Das ermöglicht indirekten Zugriff auf externe Projektdaten. „Newport_Segments“ beinhaltet sogenannte Segmente[11] für die jeweiligen Newport-Projekte, und (sofern gegeben) deren Attribute. „Formulation“ und „Formulation_Segments“ fungieren entsprechend für zu speichernde Formulationen.

[11] Segmente sind in der Enwticklung einer Formulation beispielsweise verschiedene Länder, in denen das Produkt lizensiert werden soll

5.1.2 Attributdefinition

„Attribute“ enthält alle potentiell verwendbaren, dynamischen Felder. Jede Zeile verfügt über eine eindeutige „AttributeID“, einen entsprechenden „AttributeTitle“, und sowohl eine direkte („ReferenceType“), als auch eine indirekte Referenz(„ReferenceID“) auf den Attribut-Owner. Mit der „SpecID“ wird zusätzlich auf eine Tabelle verwiesen, in der genauere Informationen über das Attribut (bspw. Einheit) hinterlegt werden können.

5.1.3 Werteebene

„Value“ speichert die benutzerdefinierten Werte, mit denen das Attribut angelegt wurde. Wesentliche Spalten sind „ValueID“, der Primärschlüssel, „AttributeID“, die Referenz auf das zugehörige Attribut, „Description“ für eine mögliche genauere Beschreibung und „Value“, worin der eigentliche Wert gespeichert wird.

5.2 Beziehungen und Referentialität

Ein Domänenobjekt kann über die AttributeID mit beliebig vielen Attributen verbunden werden. Ein Attribut kann widerum nur einem Domänenobjekt zugewiesen werden. Die referentielle Integrität wird durch die Fremdschlüssel-Paarungen sichergestellt, wonach kein Attribut ohne Verweis auf ein Domänenobjekt erstellt werden kann. Die API ist hingegen dafür zuständig, dass ein Attribut nicht auf mehrere verschiedene Domänenobjekte verweist, indem es benutzerdefiniertes Setzen der AttributeID verhindert.

5.3 Datenkonsistenz und Validierung

Das EAV-Modell sieht vor, dass die Datenintegrität durch die mit der Datenbank verbundene API sichergestellt wird, indem eingegebene Werte stets auf Validität geprüft werden. Folgende Prüfungen sind für die API vorgesehen:

5.3.1 Datentypprüfung

Jedes Attribut hat in der Specification einen hinterlegten Datentyp. Sollte der vom Benutzer angegebene Datentyp nicht in der Menge unterstützter Datentypen vorhanden sein, soll die Erstellung des Attributs verhindert werden und eine entsprechende Fehlermeldung ausgegeben.

5.3.2 Pflichtfelder und Standardwerte

Um eine reibungslose Behandlung fehlerhafter Nutzerangaben sicherzustellen, soll schon das Front-End Pflichtfelder als solche markieren, und das Abschicken der Form verhindern, bis diese ausgefüllt sind. Das Setzen der Schlüssel übernimmt jedoch das Backend, weshalb diese, obwohl sie Pflichtfelder sind, nicht vom Nutzer gesetzt werden müssen.

5.3.3 Transaktionen

Das Anlegen eines neuen Attributs und das direkte Zuweisen eines ersten Werts sollen in einer Transaktion gebündelt sein, um zu verhindern, dass Attribute ohen zugehörige Werte existieren.

5.4 Beispielhafte Speicherung

Angenommen, ein Nutzer legt für „Projekt A“ ein neues Attribut „Versuchsdauer“ (Einheit Integer) an und vergibt den Wert „3“, dann sähen die Abläufe und Datenbankeinträge folgendermaßen aus:

Tabelle 8: Attributdefinition

AttributeID	AttributeTitle	ReferenceType	SpecID
42	Versuchsdauer	Newport_Project	2

Tabelle 9: Specification (Einheitenzuordnung)

SpecID	Unit
2	Integer

Tabelle 10: Speicherung eines Attribut-Wert-Paares

ValueID	AttributeID	Description	Value
101	42	null	3

Durch diese Trennung bleiben das feste Schema in „Newport_Project“ und die dynamischen Erweiterungen klar voneinander getrennt, und trotzdem werden alle zusätzlichen relevanten Informationen abgespeichert.

Mit diesem Datenmodell können beliebig viele neue Attribute hinzugefügt werden, ohne das Grundschema der Domänentabelle anzupassen.

6 Bakend-Implementierung

Im Backend übernimmt eine mit Node.js implementierte API die Geschäftslogik. Da die Daten in AWS hinter einer Firewall liegen, sitzt die API in einer EC2-Instanz, die eine Schnittstelle zwischen der gesicherten AWS-Landschaft und externen Apps bildet. Zustäzlich wird API Gateway, ein AWS-Serice verwendet, um den automatisierten Zugriff auf die API zu ermöglichen. Gespeichert sind die Daten in einer Aurora PostgreSQL Datenbank.

6.1 Datenbank

Zur Implementierung der Datenbank auf der EC2-Instanz muss zuerst eine Verbindung mit derlokalen Datenbankmanagement-System aufgebaut werden. Dafür wird die AWS-CLI verwendet.[12] Mit ihr wird dann[13] ein Tunnel zwischen dem lokalen Port 2222[14] und dem TCP-Port 22 der EC2-Instanz geöffnet: Dieser Tunnel ist zwingend erforderlich, da aufgrund

Listing 1: SSH-Tunnel mit AWS-CLI

```
aws --profile {Profile-ID} ec2-instance-connect open-tunnel --instance-id {Ins
```

von Sicherheitsrichtlinien Bayer's die EC2-Instanz keine öffentliche IP haben darf.
In PgAdmin kann nun eine Verbindung zu der Datenbank hergestellt werden. Dafür werden folgende Informationen benötigt:

1. Host name/Adresse: Die URL der Datenbank auf AWS-Ebene (name.id.server.rds.amazonaws.com)

[12] https://aws.amazon.com/cli/
[13] Nach Authentifizierung mit aws login –{user}
[14] Der Port ist frei wählbar, solange er nicht reserviert ist (bspw. 22 ist nicht wählbar, da belegt durch TCP)

2. Port: 5432 (Standard Postgres-Port)
3. Maintenaince Database: Name der Datenbank auf AWS
4. UserName: Name des Nutzers, mit dem auf die Datenbank zugegriffen werden soll
5. Parameter „SSL Mode“: „prefer“
6. Use SSH Tunneling: „enabled“
7. Tunnel Host: localhost (da der SSH-Tunnel auf dem localhost aufgesetzt ist)
8. Tunnel Port: 2222 (entspricht –local-port in der CLI)
9. Authentication: „Identity File“
10. Identity File: Hier das .pem-File hinterlegen[15]

Mit diesen Einstellungen kann der Server gespeichert werden, und ein Zugriff auf die Datenbank ist möglich. In PgAdmin kann nun wie üblich das geplante Schema umgesetzt werden. Nun müssen die Daten in React abrufbar gemacht werden.

6.2 API

Für die API muss zuerst eine Infrastruktur in AWS errichtet werden, die eine sichere Verbindung zwischen AWS-Externen Clients und den AWS-Internen Daten garantieren kann. Diese Infrastruktur sieht folgendermaßen aus: (Hier bild) An erster Stelle steht die Virtual Private Cloud (VPC). Auf ihr liegt die gesamte Infrastruktur. Inbound und Outbound traffic aller IP-Adressen sind vollständig blockiert, um die Sicherheit der Daten zu gewährleisten. Auf dieser VPC liegt die Datenbank, auf welche Zugriff durch eine EC2-Instanz ermöglicht wird. Die Datenbank ist eine Aurora Postgres

[15]Key-Files können im AWS-Dashboard unter EC2 erstellt werden

Datenbank in Standardausführung[16]. Die EC2-Instanz entspricht ebenfalls der Standardausführung, und ist so eingerichtet, dass Inbound-traffic auf TCP-Ebene ausgehend von AWS-Internen IP-Adressen möglich ist. Zusätzlich ist die EC2-Instanz mit einem Network Load Balancer (NLB) ausgestattet, wodurch andere AWS-Programme direkt auf Elemente innerhalb der VPC zugreifen können. Dieser Zugriff ist durch eine Target Group möglich, über den ein NLB standardmäßig verfügt, und welcher in diesem Fall direkt auf das VPC zeigt. In einem NodeJS-Programm, welches direkt auf der EC2-Instanz liegt, wird eine Proxy errichtet, mit der eine Verbindung nach außen hergestellt wird: In diesem Code-Snippet wird die Proxy auf den Localhost gerichtet, in der Produktivumgebung zeigt die Proxy auf die URL der Umgebung. Die über die Proxy weitergeleiteten Requests werden von API Gateway verarbeitet, auf welchem die oben formulierten Methoden umgesetzt sind. Jede Methode entspricht einer Lambda-Funktion, über welche die Interaktion mit der Datenbank ermöglicht wird: Zuerst wird in Form der Kontextvariable „pool“ der Gesamtkontext (alle Informationen zu) der Datenbank gespeichert. Mit diesen Informationen wird dann eine Query auf der Datenbank ausgeführt, und das Ergebnis zurückgesendet. Die Lambda-Funktionen sind in API Gateway einer eigenen URL zugeordnet, die über die Proxy auch außerhalb von AWS bei gegebener Authentifizierung abrufbar ist. Diese wird durch Cognito abgewickelt.

6.3 Authentifizierung mit Cognito

In Cognito liegt, speziell für die Authentifizierung von Nutzern für dieses Projekt, ein User-Pool. Dieser hat eine eigene Domain, in der hinterlegte

[16]Standardausführung festgelegt durch den Unternehmensinternen Softwarekatalog

Nutzer ihre Credentials angeben können. Werden die Credentials bestätigt, stellt Cognito ein Auth-Token zur Verfügung, mit welchem dann der Zugriff auf die API möglich ist. Im Frontend wird mit einem Login-Button ein Link zu der Cognito-Auth-Seite hergestellt, der den User nach erfolgreichem Login automatisch zurücklenkt.

Listing 2: EC2-Proxy in NodeJS

```
    // Load .env at startup
require('dotenv').config();

const express = require('express');
const morgan = require('morgan');
const { Pool } = require('pg');

// Sanitize and trim environment variables
const dbUser = process.env.DATABASE_USER?.trim();
const dbPass = process.env.DATABASE_PASSWORD?.trim();
const dbHost = process.env.DATABASE_HOST?.trim();
const dbPort = Number(process.env.DATABASE_PORT);
const dbName = process.env.DATABASE_NAME?.trim();

console.log('Starting DB proxy service');
console.log('CWD:', process.cwd());
console.log('Database config:', { host: dbHost, port: dbPort, user: dbUser, databa

// Set up Express
const app = express();

// 1. Morgan for HTTP logging
app.use(morgan(':method :url :status :res[content-length] - :response-time ms'));

// 2. JSON body parsing
app.use(express.json());

// 3. Sanity dump for POST bodies
app.use((req, res, next) => {
if (['POST','PUT','PATCH'].includes(req.method)) console.log('> BODY:', req.body);
next();
});

// 4. Postgres pool using sanitized env
const pool = new Pool({
host: dbHost,
port: dbPort,
user: dbUser,
password: dbPass,
database: dbName,
ssl: { rejectUnauthorized: false }
});

// Async wrapper helper
defaultWrapAsync = fn => (req, res, next) => fn(req, res, next).catch(next);

// 5. Health endpoint
defaultWrapAsync(async (req, res) => {
await pool.query('SELECT 1');
res.sendStatus(200);
});
app.get('/health', defaultWrapAsync(async (req, res) => res.sendStatus(200)));

// 6. Query endpoint
app.post('/query', defaultWrapAsync(async (req, res) => {
const { text, params } = req.body;
const { rows } = await pool.query(text, params);
res.json(rows);
}));

// 7. 404 catch-all
app.use((req, res) => res.sendStatus(404));

// 8. Error handler
```

Listing 3: Lambda-Funktion für die API

```
    import { Pool } from 'pg';

// Create a pool as a singleton so Lambda can reuse TCP connections
const pool = new Pool({
  host:     process.env.DB_HOST,
  database: process.env.DB_NAME,
  user:     process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  port:     process.env.DB_PORT,
  // If you need SSL, uncomment and adjust:
  // ssl: { rejectUnauthorized: false }
});

export const handler = async (event) => {
  const projectId = event.pathParameters?.projectId;
  if (!projectId) {
    return { statusCode: 400, body: 'Missing projectId path parameter' };
  }

  let client;
  try {
    client = await pool.connect();

    const result = await client.query(
      `SELECT "ProjectID", "AttributeID"
         FROM public."Newport_Project"
        WHERE "ProjectID" = $1`,
      [projectId]
    );

    if (result.rows.length === 0) {
      return { statusCode: 404, body: 'Project not found' };
    }

    return {
      statusCode: 200,
      headers: { 'Content-Type': 'application/json',
      'Access-Control-Allow-Origin': 'http://localhost:3000',
      'Access-Control-Allow-Credentials': true,
      // if you need cookies
      // add any other headers your client uses:
      'Access-Control-Allow-Headers': 'Authorization,Content-Type',},
      body: JSON.stringify(result.rows[0])
    };

  } catch (err) {
    console.error('DB error', err);
    return {
      statusCode: 500,
      body: 'Internal server error'
    };

  } finally {
    client?.release();
  }
};
```

7 Frontend-Implementierung

8 Evaluierung

9 Ausblick und Integration

10 Fazit

Anhang

Anhangsverzeichnis

Anhang 1 Gesprächsnotizen

Anhang 1.1 Gespräch mit Werner Müller

Gespräch mit Werner Müller am 01.01.2013 zum Thema XXX:

- Über das gute Wetter gesprochen
- Die Regenwahrscheinlichkeit liegt immer bei ca. 3%
- Das Unternehmen ist total super
- Hier könnte eine wichtige Gesprächsnotiz stehen

Quellenverzeichnis

Ehrenwörtliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorthesis selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Langenfeld, 10.06.2025

Thomas Benjamin Hopf