E-Business Architekturen

Prüfungsleistung (Gruppenaufgabe)

Ergebnisprotokolle der Komplexübungen 1, 2, 3b und 4d im Rahmen der Veranstaltung E-Business Architekturen

vorgelegt am
07.05.2023

an der
Hochschule für Wirtschaft und Recht Berlin
Fachbereich Duales Studium

von:	Robert Neubert
	Danny Neupauer
	Hannes Roever
Fachrichtung:	Wirtschaftsinformatik
Studienjahrgang:	WI20C
Studienhalbjahr:	Wintersemester 2022/23
Dozent:	Prof. Dr. Andreas Schmietendorf

Inhaltsverzeichnis

1 Einleitung **1**

2 Aufgabe 1: E-Business Grundlagen **1**
2.1 Anwendung des Begriffs E-Business . . . 1
2.2 Beziehung zu domänenspezifischen Lösungen . . . 1
2.3 Ziele und Erwartungen an E-Business Lösungen . . . 1
2.4 Eigenschaften von E-Business Softwarearchitekturen . . . 1
2.5 E-Business im konkreten Unternehmenskontext . . . 2

3 Aufgabe 2: Serviceverzeichnisse **3**
3.1 Analyse eines Verzeichnisdienstes . . . 3
3.1.1 Vorgehensweise bei der Auswahl eines Verzeichnisdienstes . . . 3
3.1.2 Allgemeines über den Verzeichnisdienst . . . 3
3.1.3 Vergleich von API- und datenorientierten Schnittstellen . . . 5
3.2 Analyse von Web-APIs . . . 6
3.2.1 Erstellung des Bewertungsmodells . . . 6
3.2.2 Einsatz des Bewertungsmodells . . . 7
3.3 API - Spezifikationen . . . 9
3.3.1 Spezifikationsanalyse . . . 9
3.3.2 Analysetools . . . 9
3.3.3 Einschränkungen und Alternativen . . . 10

4 Übung 3b: Entwicklung eigener Service-Angebote **11**
4.1 Möglichkeiten für Implementierung und Deployment . . . 11
4.1.1 Analyse der Möglichkeiten . . . 11
4.1.2 Analytischer Vergleich der Möglichkeiten . . . 11
4.1.2.1 Implementierung . . . 12
4.1.2.2 Deployment . . . 16
4.2 Entwicklung . . . 16
4.2.1 Rahmenbedingungen . . . 16
4.2.1.1 Anforderungen . . . 17
4.2.1.2 Verwendete Sprache(n) . . . 17
4.2.1.3 Komponenten . . . 18
4.2.1.4 Eingesetzte Frameworks und Libraries . . . 19
4.2.1.5 Konfiguration Entwicklungsumgebung . . . 20
4.2.1.5.1 Datenbank . . . 20
4.2.1.5.2 REST API . . . 21
4.2.1.5.3 Client WebApp . . . 21
4.2.1.6 Deployment . . . 22
4.2.2 Umsetzung . . . 22
4.2.2.1 Design . . . 22
4.2.2.2 Implementierung . . . 22
4.2.2.2.1 Client . . . 23
4.2.2.2.2 REST API . . . 23
4.2.2.2.3 Tests . . . 23
4.2.2.3 Deployment . . . 24
4.2.3 Anbindung Datenbank . . . 26

5 Übung 4d: Sicherheit von Web APIs **28**
5.1 Sicherheitsrisiken in Verbindung mit dem HTTP Protokoll . . . 28
5.1.1 HTTPS und TLS . . . 28
5.1.2 Authentifizierungmöglichkeiten HTTP(S) . . . 29
5.1.3 Cookies . . . 29
5.1.3.1 Begriff . . . 29
5.1.3.2 Sicherheitsrisiken . . . 30
5.2 Möglichkeiten zur Risikominderung . . . 30
5.2.1 OWASP . . . 30
5.2.1.1 Broken Object Level Authorization . . . 31
5.2.1.2 Excessive Data Exposure . . . 32
5.2.1.3 Broken Function Level Authorization . . . 33
5.2.1.4 Injection (hier SQL) . . . 33
5.2.1.5 Improper Assets Management . . . 34
5.2.2 OAuth 2 und OIDC . . . 35
5.3 Praktische Anwendung von OAuth2 . . . 35
5.3.1 Testwerkzeuge . . . 35
5.3.2 Implementierung . . . 35

1 Einleitung

hallo[1] hallo2[2]

2 Aufgabe 1: E-Business Grundlagen

2.1 Anwendung des Begriffs E-Business

Was verbinden Sie mit dem Begriff des E-Business? Versuchen Sie die folgenden Aspekte zu berücksichtigen, nennen Sie ggf. weitere.

- Organisatorische Aspekte
- Prozessbezogene Aspekte (z.B. Geschäftsprozess)
- Technologische Aspekte (z.B. Entwicklung & Betrieb)
- Gesellschaftliche Implikationen (z.B. Soziologische Aspekte)

2.2 Beziehung zu domänenspezifischen Lösungen

Welche Beziehungen sehen Sie zu den folgenden Lösungen?

- Systeme für das e-Learning (z.B. Moodle oder Open HPI)
- Systeme für das e-Government (z.B. ELSTER oder Fahrzeugzulassung)
- Systeme für das e–Banking (z.B. Instant Payment)
- Systeme für das e-Commerce (z.B. Web Shops)

2.3 Ziele und Erwartungen an E-Business Lösungen

Welche Ziele und Erwartungen verknüpfen Unternehmen und ihre Kunden mit e-Business-Lösungen?

- Berücksichtigen sie ggf. unterschiedliche Sichten
- Nennen Sie ihnen bekannte Lösungen (z.B. aus den Praktika)
- Identifizieren Sie mögliche Vor- und Nachteile

2.4 Eigenschaften von E-Business Softwarearchitekturen

Über welche Eigenschaften sollten Softwarearchitekturen für e-Business-Lösungen verfügen?

- Fragen des Kommunikationssystems
- Verwendete Rechnerinfrastruktur
- Eigenschaften entwickelter Softwaresysteme

[1] Vgl. Athey (2018), S.5
[2] Vgl. DevInsider (2022), online

2.5 E-Business im konkreten Unternehmenskontext

Wie könnte eine Strategie zur Einführung einer e-Business-Architektur in einem Unternehmen ihrer Wahl aussehen?

- Notwendige Voraussetzungen & Rahmenbedingungen
- Auswirkungen auf das Informationsmanagement (CIO)
- Auswirkungen auf die Entwicklung von Software (Lösungsanbieter)
- Auswirkungen auf den Betrieb von Software (Rechenzentren)
- Mehrwertpotentiale für die Kunden und Lieferanten

Worin sehen Sie weitere Aspekte eines digitalen Unternehmens, die mit dem Begriff des e-Business nicht erfasst werden?

3 Aufgabe 2: Serviceverzeichnisse

3.1 Analyse eines Verzeichnisdienstes

Analysieren Sie die Möglichkeiten eines in Abstimmung mit dem Dozenten zu wählenden Verzeichnisdienstes für Web-APIs.

- Recherche und Auswahl eines Verzeichnisdienstes:
  - Anzahl und Art der registrierten Web-APIs (ggf. auch Open Data)
  - Allgemeiner Funktionsumfang des Verzeichnisdienstes
  - Hinterlegte Klassifikationen – d.h. Organisation der Serviceablage
  - Vorgehensweise zum ggf. Suchen von Serviceangeboten
  - Vorgehensweise zum ggf. Registrieren eigener Serviceangebote
  - Bereitgestellte Entwicklerunterstützung, wie z.B. Beispielcode
- Voraussetzungen zur Nutzung (Registrierung, Kosten, ...)?
- Vergleich von API- und datenorient. Schnittstellen (z.B. Open Data)?

In dieser Unteraufgabe wird sich mit der Nutzung und Analyse von Service-Verzeichnissen beschäftigt. Dazu wird sich zunächst für einen zu betrachtenden Verzeichnisdienst entschieden, welcher im weiteren Verlauf der Aufgabe auf seine Eigenschaften überprüft wird.

3.1.1 Vorgehensweise bei der Auswahl eines Verzeichnisdienstes

Um einen vollumfänglichen Überblick über den Verzeichnisdienst bieten zu können, wurde bei der Auswahl des Verzeichnisdienstes darauf geachtet, einen Verzeichnisdienst ohne Zugangsbeschränkungen mit öffentlich zugänglichen APIs zu wählen. Aufgrund dieser Vorgaben wir uns für das API-Verzeichnis des Bundes entschieden.
Die APIs, die unter der Webadresse https://bund.dev/apis zusammengefasst sind, dienen dem Zweck, den Zugang zu verschiedenen Datensätzen und Verwaltungsverfahren der Bundesverwaltung zu erleichtern. Sie ermöglichen es Entwicklern und anderen interessierten Nutzern, auf eine standardisierte und dokumentierte Art und Weise auf diese Informationen zuzugreifen und sie in eigenen Anwendungen zu nutzen. Dabei können die APIs verschiedene Funktionalitäten bereitstellen, wie beispielsweise die Abfrage von Daten, die Bearbeitung von Anträgen oder die Einreichung von Dokumenten. Durch die Bereitstellung dieser APIs im Rahmen der Open Government Umsetzungsstrategie des Bundes wird eine transparentere und effizientere Zusammenarbeit zwischen Verwaltung und Bürgern angestrebt.

3.1.2 Allgemeines über den Verzeichnisdienst

In ihrer Gesamtheit sind auf der Website insgesamt 47 diverse Web-APIs identifizierbar. Diese APIs können hauptsächlich verschiedenen Bundesbehörden zugeordnet werden. Jedoch lassen sich vereinzelt auch APIs von Landesbehörden sowie von Anstalten des öffentlichen Rechts ausmachen. Die Qualität der bereitgestellten Dokumentationen variiert und erstreckt sich von minimalen Informationen, die lediglich auf der GitHub-Seite darauf hinweisen, dass eine bestimmte API (Rechtsinformationsportal) deaktiviert wurde, bis hin zu umfangreichen und gut strukturierten Dokumentationen, für die eigens eine Webpräsenz entwickelt wurde (FIT-Connect).

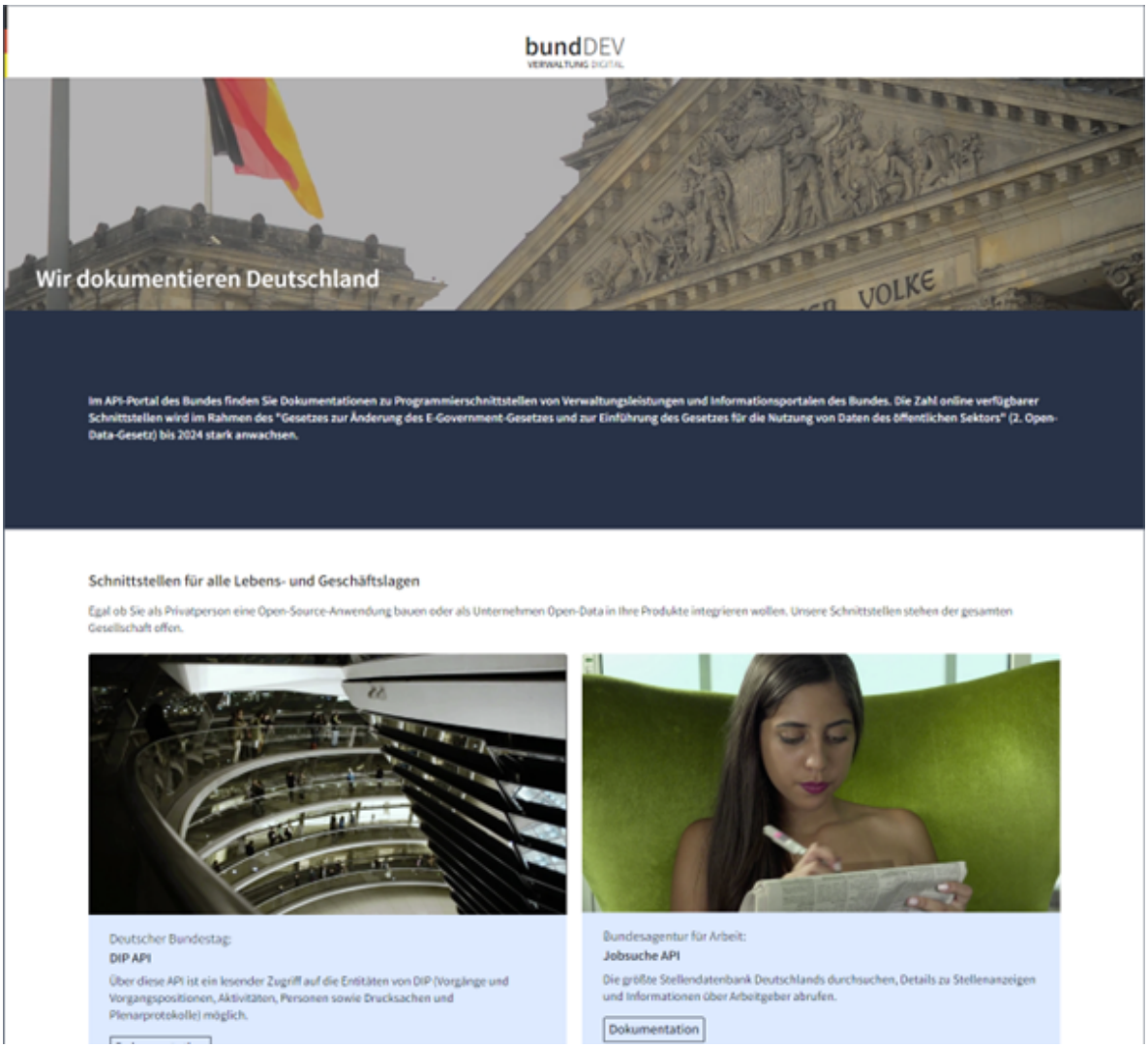


Abbildung 1: Ansicht der Startseite des Verzeichnisdienstes

Das Hochladen selbst entwickelter API´s ist auf der BundDev Platform nicht vorgesehen. Es besteht aber die Möglichkeit über die GitHub Verbindung eigene Forks zu kreieren und diese zu verändern und anzupassen. Da viele Datengrundlagen die von den API´s genutzt werden öffentlich zugänglich sind besteht auch die Möglichkeit mit Orignaldaten zu arbeiten. Nachdem der Code angepasst oder verbessert wurde lässt sich dieser über einen Pull Request in den ursprünglichen Code integrieren. Das Melden von Problemen und Bugs ist in einigen Fällen nur über GitHub möglich. In wenigen Dokumentationen ist eine Kontaktemail hinterlegt. Eine standardisierte Vorlage für das Reporting von Schwierigkeiten besteht nicht.

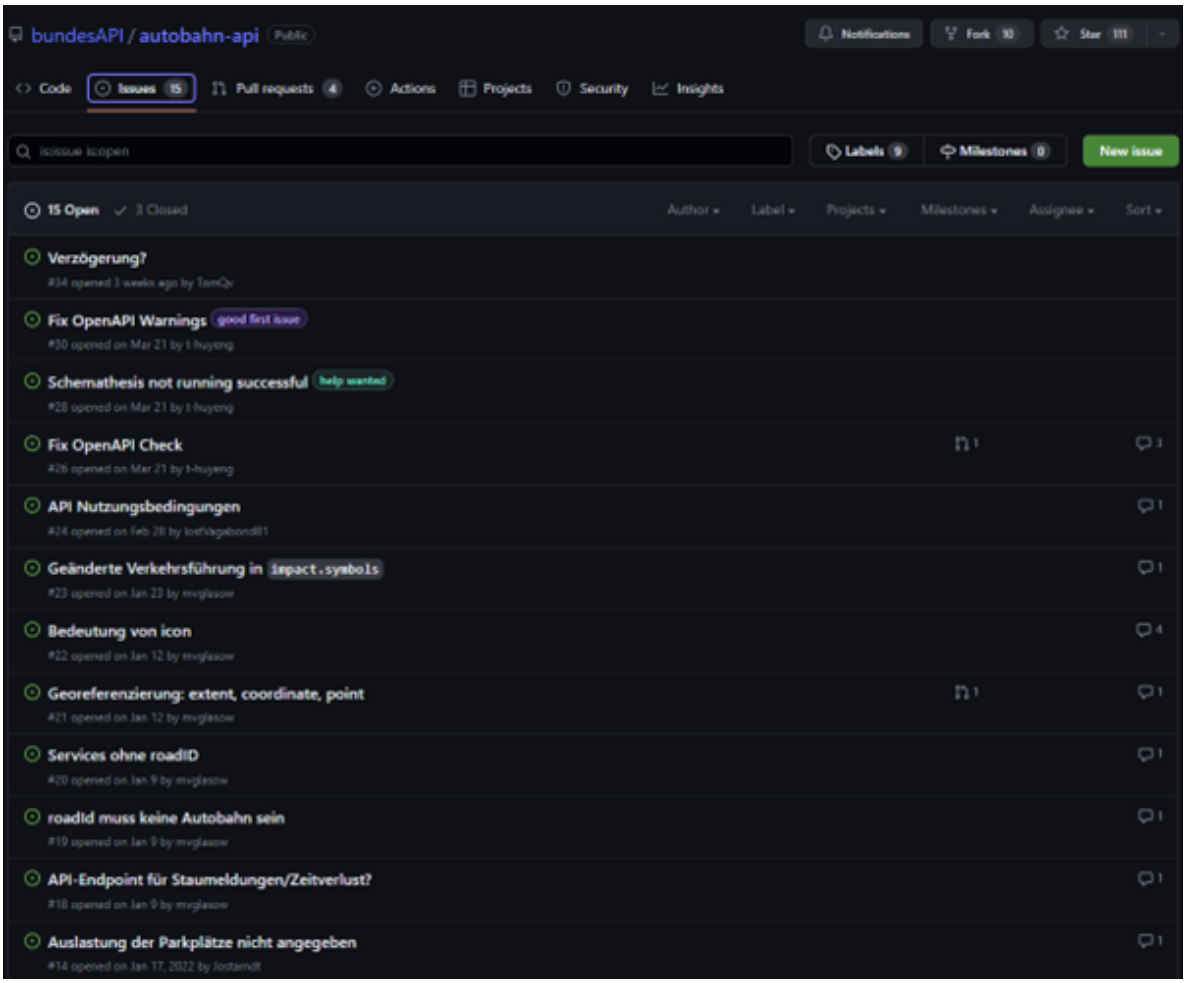


Abbildung 2: Gemeldete Probleme

Wie in Abbildung 2 ersichtlich ist, können bei der Erstellung von Problemen verschiedene Tags zugeordnet werden, um eine Klassifizierung für den Entwickler zu erleichtern. Der Verzeichnisdienst ist ohne vorherige Anmeldung oder Registrierung zugänglich. Ebenso kann der API-Code ohne GIT-Registrierung oder Anmeldung heruntergeladen werden. Das Melden von Problemen über die GIT-Funktion erfordert jedoch eine vorherige Anmeldung und Registrierung. Analog dazu verhält es sich bei verfügbaren Pull Requests und ähnlichen Funktionalitäten.

3.1.3 Vergleich von API- und datenorientierten Schnittstellen

REST: protokollorientierte (https://systempilot.net/edi-rest-api-schnittstellen-systemintegration/) und https://de.wikipedia.org/wiki/Programmierschnittstelle

Schnittstellen-Typ	Funktionsorientierte PS	Dateiorientierte PS	Objektorientierte PS	Protokollorientierte PS
Beschreibung	Stellt Funktionen zur Verfügung, die von anderen Anwendungen aufgerufen werden können.	Bietet die Möglichkeit, auf Daten in Dateien zuzugreifen und diese zu verarbeiten.	Basiert auf Objekten, die Funktionen und Eigenschaften enthalten und von anderen Anwendungen verwendet werden können.	Bietet eine strukturierte Art und Weise, um Daten zwischen Systemen auszutauschen.
Beispiele	DLL, Programm-APIs, Bibliotheken	CSV, XML, JSON	COM, CORBA, SOAP	HTTP, TCP/IP, FTP
Verwendung	Häufig in einfachen Anwendungen verwendet.	Nützlich für Anwendungen, die mit großen Datenmengen arbeiten.	Komplexere Anwendungen, die eine umfangreichere Struktur benötigen.	Weit verbreitet in verteilten Systemen und bei der Kommunikation zwischen verschiedenen Anwendungen.
Vorteile	Schnell und einfach zu implementieren.	Gut geeignet für die Arbeit mit großen Datenmengen.	Bietet eine flexible Art der Datenverarbeitung und -speicherung.	Bietet eine standardisierte Art der Kommunikation zwischen Systemen.
Nachteile	Kann bei komplexen Anwendungen unübersichtlich werden.	Begrenzte Funktionalität im Vergleich zu anderen Schnittstellentypen.	Komplex in der Implementierung und erfordert mehr Aufwand.	Kann weniger effizient als andere Schnittstellentypen sein.

3.2 Analyse von Web-APIs

3.2.1 Erstellung des Bewertungsmodells

Erstellen Sie ein Bewertungsmodell für angebotene Web APIs

- Welche Informationen halten Sie für einen Einsatz notwendig?
  - Spezifikation/Technologie (SOAP, REST, JSON, MIME, ...)
  - Servicebeschreibung (technisch & fachlich)
  - Funktionstüchtigkeit (Qualitätsvereinbarungen)
  - Kontaktinformationen
  - Beispiele zur programmiertechnischen Einbindung
- Informationen zu einem Ansatz für ein Bewertungsmodell – siehe Anlage

Wir haben uns dazu entschieden alle API´s aus dem BundDev Verzeichnis zu bewerten. So ist es möglich eine Übersicht der vom Bund bereitgestellten Schnittstellen zu gewinnen. Zu Beginn wurde das Bewertungsmodell in fünf Kategorien aufgeteilt. Diese lauten: Übersicht, Offenheit, Qualität, Dokumentation und Verfügbarkeit.

1. In diesem Teil wird nur eine Übersicht über die einzelnen API´s dargestellt. Unter welchen URL lässt sich die Dokumentation finden? Wie viele get, post, put, delete und andere werden werden in der API verwendet? Hierbei wird keine Bewertung vorgenommen, sondern es wird nur aufgezählt.
2. Offenheit:
   - Quellcode: Die Offenlegung des Quellcodes einer API fördert Transparenz und Anpassungsfähigkeit, indem sie Entwicklern Einblicke in die Funktionsweise der API gewährt und die Möglichkeit bietet, die API an individuelle Bedürfnisse anzupassen sowie Fehler zu beheben.
   - Insofern ein Token zur Nutzung notwendig ist, sollte dies einfach, kostenlos und umgehend zur Verfügung gestellt werden (nach Registrierung)
   - Zugänglichkeit: Dokumentation, Beispiele, Kontakt, verwendete Sprache (englisch, keine Fachbegriffe und Abkürzungen) sind nachvollziehbar und gut auffindbar
   - Kontakt: Verantwortlicher und ggfs. Entwickler können direkt erreicht werden
3. Qualität:
   - Granularität: zu gering (wenige Ressourcen mit wenigen Routen geben sehr große Objekte zurück) bis zu hoch (für "normale" Usecases müssen für ein clientrelevantes Objekt mehrere Abfragen gestellt werden)
   - TLS: ausschließlich oder Weiterleitung bei HTTP Aufruf (gut) über HTTP möglich (mittel) bis nur unter HTTP erreichbar (schlecht)
   - Statuscode: Werden alle relevanten Statuscodes in den Dokumentationen genannt und ausreichen beschrieben?
   - Ressourcen fachlich korrekt ausgewählt?
   - Routen in angemessener Tiefe?
4. Dokumentation:
   - Routen: beschrieben, Datentypen, Beispiele
   - Ressourcen: benannt, Request und Response Modelle verfügbar (required? usw.)
   - Parameter: benannt, Datentypen, Beispiele
5. Verfügbarkeit und Performance

3.2.2 Einsatz des Bewertungsmodells

Analysieren Sie stichpunktartig 20 registrierte Web APIs

- Verwenden Sie ihr entwickeltes Bewertungsmodell
- Ausführung der Services mittels Musterlösung (keine Programmierung)

Die zu analysierenden Service-APIs sollten möglichst aus unterschiedlichen Serviceverzeichnissen stammen.

Bewertungsteil	**Beschreibung**	**Punkte**
Token/Registrierung	Nicht notwendig ODER notwendig, aber leicht einzurichten. Wenn ja, mit Token sind alle Endpunkte verfügbar	2
	Notwendig, aber auch dann nicht alle Endpunkte verfügbar ODER notwendig, aber mit Hürden einzurichten	1
	Für alle Endpunkte nötig UND mit Hürden einzurichten ODER Token ungültig	0
Quellcode	Liegt vor und ist verlinkt	1
	Liegt nicht vor oder muss erst gesucht werden	0
Request Limit	Wenn nicht angegeben, stellen wir verteilt über 30 Minuten 1000 Anfragen. Wenn das klappt, gibt es zwei Punkte.	
	< 100	0
	< 1000	1
	< 10k	2
	> 10k	3
Kontakt	Ja (Link/E-Mail)	1
	nein	0

Tabelle 1: Bewertungsschema Offenheit

Bewertungsteil	**Beschreibung**	**Punkte**
Granularität	Ausgeglichen, sowohl Listen als auch Einschränkungen auf einzelne Objekte	1
	Sehr viele Use Cases mit einzelnen Endpunkten oder eine Anfrage erfordert verschiedene Ressourcen	0
	Zu geringe Granularität, sehr wenige Endpunkte mit zu umfangreichen Datenmodellen	0
Transportverschlüsselung	HTTPS und gegebenenfalls Weiterleitung von HTTP auf HTTPS	2
	HTTP und HTTPS, aber keine Weiterleitung von HTTP	1
	Nur HTTP	0
Routen	Jede Ressource hat einen Identifier und ist mit Nomen benannt. HTTP-Methoden sind korrekt eingesetzt. Maximale Tiefe beträgt 3.	2
	Mindestens 2 Bedingungen aus 2 sind erfüllt	1
	Eine oder keine Bedingung aus 2 erfüllt	0
Statuscodes	200, 201, 204, 400, 401, 403, 404, 409, 500, 503	2
	Mind. 200, 201, 400, 404, 500	1
	Keine	0
MIME Types	JSON und/ oder XML/ plain text	2
	Nur XML und/ oder plain text	1
	Keine/ plain	0
Versionierung	Verschiedene Majorversionen = 2	
	Nur latest = 1	
	Nicht angegeben = 0	

Tabelle 2: Bewertungsschema Qualität

Bewertungsteil	Beschreibung	Punkte
Routen	alle Routen mit Beispielen (Request/Response), wenn nicht sprechend mit Erläuterung, alle notwendigen und optionalen Parameter sind angegeben	2
	wie zwei, aber nur teilweise erfüllt	1
	Keine Doku	0
Ressourcen	alle Resourcen dokumentiert mit Beispielen, Datentyp und ggfs. default Wert	2
	wie zwei, aber nur teilweise erfüllt	1
	Keine Doku	0
Parameter	alle Parameter mit Datentyp, erlaubtem Werteraum, required und - wenn nicht sprechend - mit Beispiel, ggfs Differenzierung der Rückgabeobjekte ist dokumentiert	2
	wie zwei, aber nur teilweise erfüllt	1
	Keine Doku	0
Statuscodes	Responsecodes mit Zahl, Rückgabestring und Datentyp, Mimetype oder Datenmodell dokumentiert	2
	wie zwei, aber nur teilweise erfüllt	1
	Keine Doku	0
Swagger	Swagger yaml oder json file vorhanden, sowie Swagger UI zum ausprobieren, alle Beispiele und ggfs Auth./Tokens in Swagger UI funktionieren	2
	Swagger yaml file oder json vorhanden, aber keine Swagger UI oder andere Möglichkeit zum ausprobieren	1
	Keine Doku/ Möglichkeit zum Testen	0

Tabelle 3: Bewertungsschema Dokumentation

3.3 API - Spezifikationen

3.3.1 Spezifikationsanalyse

Analysieren Sie Struktur und Elemente einer WSDL, OpenAPI-(Swagger) oder auch GraphQL(Schema)-Spezifikation.

- Verwenden Sie zur Analyse 5 ausgewählte Services
- Stichpunktartige Beschreibung der Struktur/Unterelemente
- Metrische Erfassung der Struktur bzw. eingesetzten Elemente
- Statistische Auswertung Informationen (z.B. Tabellen, Diagramme)

3.3.2 Analysetools

Nutzen Sie ggf. verfügbare Hilfsmittel und gehen Sie auf die entsprechende Funktionsweise der Tools ein

- Beispiele: soapUI, SOAPSonar, Postman https://www.postman.com
- Grafische WSDL-Editoren (z.B. XMLSpy ab Version 8)

3.3.3 Einschränkungen und Alternativen

- Welche Informationen fehlen bei der gewählten Spezifikationen?
- Recherchieren Sie nach alternativen Beschreibungsformen?

4 Übung 3b: Entwicklung eigener Service-Angebote

4.1 Möglichkeiten für Implementierung und Deployment

4.1.1 Analyse der Möglichkeiten

Analysieren Sie mit Hilfe des Internets mögliche Alternativen zur Implementierung und Deployment von Web APIs (speziell WSDL/XML, REST/OpenAPI und GraphQL), wie z.B.:

- IDE NetBeans und GlassFish Server
- IDE Eclipse und Tomcat & Axis-Erweiterung
- Postman API Builder
- Cloud-basierte Entwicklung/Deployment

Nach weitreichender Internetrecherche wurden einige Werkzeuge zur Implementierung und Deployment von Apis zusammengetragen. Zur besseren Einordnung dieser sind Eigenschaften wie Scope und Typ sowie falls vorhanden Laufzeitanforderungen aufgeführt.

Plattform / Service	**Scope**	**Typ**	**Laufzeit**
AWS Lamda mit AWS API Gateway	Implementierung, Deployment	SaaS	-
ApiGee	Implementierung, Deployment	SaaS	-
Postman Api Builder	Implementierung	SaaS	-
Firebase	Implementierung, Deployment	SaaS	-
Swagger Hub	Deployment	SaaS	-
Cloudflare Workers	Deployment	FaaS	-
Supabase	Implementierung, Deployment	PaaS	-
AWS Amplify	Deployment	PaaS	-
AWS ECS	Deployment	IaaS	-
Swagger Codegen	Implementierung	Executable	Java
Apicurio Studio	Implementierung	Executable	Java
ASP.NET	Implementierung	Framework	C#
express.js	Implementierung	Framework	JS
Flask	Implementierung	Framework	Python
Spring Boot	Implementierung	Framework	Java
Ruby on Rails	Implementierung	Framework	Ruby

Tabelle 4: Übersicht über verschiedene Plattformen und Services für die Implementierung, Deployment und Nutzung von APIs.

4.1.2 Analytischer Vergleich der Möglichkeiten

Vergleichen Sie die gefunden Alternativen anhand eines eigenen Bewertungsmodells, mit Hilfe von Kriterien wie z.B.:

- Voraussetzungen zur Verwendung (HW- und SW-Ressourcen)
- Integration von Entwicklung- und Ausführungsplattform
- DevOps orientierte Vorgehensweise (Automationsaspekte)

- Verbreitung, Entwicklersupport, Kosten, Lizenzen

Im Folgenden werden ausgewählte der in 3b.1.1 aufgeführten Implementierungs- und Deploymentmöglichkeiten mittels eines Bewertungsschemas verglichen. Die ausgewählten tools stehen exemplarisch für jeweils eine Implementierungs- oder Deploymentart.

4.1.2.1 Implementierung

Das der Bewertung verschiedener Implementierungsarten zugehörige Schema beinhaltet die Kriterien Komplexität der Implementierung, Komplexität der OpenApi-Spezifikationserstellung, Güte der Dokumentation, Popularität, Kosten und Geschwindigkeit. Zur Bewertung der Implementierungskomplexität werden die Implementierungen der gleichen Api verglichen. Dazu wird die künstliche Intelligenz ChatGPT genutzt. Diese bekommt pro Implementierungsart die gleiche Aufforderung, welche wie folgt aussieht:

> Implement a rest api with one endpoint named /items. This endpoint should return all columns of the mysql database table item and should be able to response the http-codes 200, 404 and 500. Do so using the shortest possible way in Implementierungsart.

Die Ergebnisse wurden dann auf die Herkunft der nötigen libraries sowie die Anzahl der Funktionaufrufe untersucht. Zur Bewertung der Komplexität der OpenApi Spezifikationserstellung wurde recherchiert, ob eine Spezifikationserstellung überhaupt möglich und wenn möglich ohne externe Hilfsmittel/Libraries möglich ist. Außerdem wurde berücksichtigt, ob die Erstellung automatisch oder manuell erfolgt. Zur effektiven Entwicklung von Software sind umfangreiche, verständliche und vor allem aktuelle Dokumentationen von großer Bedeutung. Aufgrund dessen ist auch die Dokumentationsgüte teil des Bewertungsschemas. Hier fließen die Übersichtlichkeit, der Umfang, die Aktualität und die Verständlichkeit ein. Dabei gilt zu beachten, dass diese Bewertungen nicht objektiv messbar sind und daher subjektiv bewertet wurden. Außerdem wurde die Popularität mittels Google Trends bestimmt. Die Geschwindigkeit wurde anhand von Benchmarks gerankt. Ein weiteres sehr wichtiges Kriterium zu Auswahl der Entwicklungswerkzeuge ist die Preisstruktur dieser, weshalb diese ebenfalls aufgeführt ist.

Listing 1: Implementierung in Java

```java
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/items")
public class ItemController {

    public static void main(String[] args) {
        SpringApplication.run(Main.class, args);
    }

    @Autowired
    private JdbcTemplate jdbcTemplate;

    @GetMapping
    public ResponseEntity<List<Item>> getAllItems() {
        try {
            List<Item> items = jdbcTemplate.query(
                    "SELECT * FROM item",
```

```
                    new RowMapper<Item>() {
                        public Item mapRow(ResultSet rs, int rowNum) throws
                            SQLException {
                            Item item = new Item();
                            item.setId(rs.getLong("id"));
                            item.setName(rs.getString("name"));
                            item.setDescription(rs.getString("description"));
                            item.setPrice(rs.getDouble("price"));
                            return item;
                        }
                    });
            if (items.isEmpty()) {
                return new ResponseEntity<>(HttpStatus.NOT_FOUND);
            }
            return new ResponseEntity<>(items, HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

    public static class Item {
        private long id;
        private String name;
        private String description;
        private double price;

        public long getId() {
            return id;
        }

        public void setId(long id) {
            this.id = id;
        }

        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }

        public String getDescription() {
            return description;
        }

        public void setDescription(String description) {
            this.description = description;
        }

        public double getPrice() {
            return price;
        }

        public void setPrice(double price) {
            this.price = price;
        }
    }
}
```

Listing 2: Implementierung in python

```
from flask import Flask, jsonify
from flask_mysqldb import MySQL

app = Flask(__name__)
app.config['MYSQL_HOST'] = 'localhost'
app.config['MYSQL_USER'] = 'username'
app.config['MYSQL_PASSWORD'] = 'password'
```

```
app.config['MYSQL_DB'] = 'database'
mysql = MySQL(app)

@app.route('/items')
def get_items():
    cur = mysql.connection.cursor()
    cur.execute("SELECT * FROM item")
    data = cur.fetchall()
    if data:
        return jsonify(data), 200
    else:
        return jsonify({"message": "No items found"}), 404

@app.errorhandler(500)
def internal_error(error):
    return jsonify({"message": "Internal server error"}), 500

if __name__ == '__main__':
    app.run(debug=True)
```

Listing 3: Implementierung in C#

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;
using Dapper;
using MySql.Data.MySqlClient;
using System;
using System.Linq;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddSingleton<MySqlConnection>(sp =>
    new MySqlConnection(builder.Configuration.GetConnectionString("
        DefaultConnection")));

var app = builder.Build();

app.MapGet("/items", async (HttpContext httpContext, MySqlConnection connection)
    =>
{
    try
    {
        var items = (await connection.QueryAsync<Item>("SELECT * FROM Items")).
            ToList();
        if (items.Count == 0)
        {
            return Results.NotFound();
        }
        return Results.Ok(items);
    }
    catch (Exception ex)
    {
        Console.Error.WriteLine(ex);
        return Results.StatusCode(StatusCodes.Status500InternalServerError);
    }
});

app.Run();

public record Item(int Id, string Name, string Description, decimal Price,
    DateTime CreatedAt);
```

Kriterium	Unterkategorie	Bewertungen	Quellen
Komplexität der Implementierung	notwendige Hilfsmittel/Libraries	1: keine externen Hilfsmittel/Libraries nötig 0: externe Hilfsmittel/Libraries nötig	
	Anzahl der notwendigen Schritte/Funktionsaufrufe	0: mehr als der Durchschnitt der anderen 1: durchschnittlich 2: weniger als der Durchschnitt der anderen	
OpenAPI-Spezifikation	Integration	2: out of the box 1: Drittanbieter library 0: nicht möglich	Spring: [a] Flask: [b] .NET: [c] Postman: [d]
Implementierung		2: automatisch 1: manuell	
Dokumentation	Übersichtlichkeit (logische Untergliederung)	1: übersichtlich 0: unübersichtlich	Spring: [e] Flask: [f] .NET: [g] Postman: [h]
Umfang		1: vollumfänglich 0: unvollständig	
Aktualität		1: aktuell 0: (teils) veraltet	
Verständnis (ggf. mit Beispielen)		1: gut verständlich 0: nicht gut verständlich	
Popularität	-	Rangfolge entsprechend Google Trends (0-3)	[i]
Geschwindigkeit	-	Rangfolge entsprechend TechEmpower Benchmark (0-2)	[j]
Skalierbarkeit	Replizierbarkeit	1: möglich 0: nicht möglich	Glassfish[k]
	Loadbalancing (Clustering)	1: möglich 0: nicht möglich	
	Effizienz	1: geringer Ressourcenverbrauch 0: hoher Ressourcenverbrauch	
Continuous Deployment	-	2: ohne Downtime möglich 1: mit Downtime möglich 0: nicht möglich	

[a] https://spring.io/
[b] https://flask.palletsprojects.com/en/2.1.x/
[c] https://dotnet.microsoft.com/
[d] https://www.postman.com/
[e] https://spring.io/
[f] https://flask.palletsprojects.com/en/2.1.x/
[g] https://dotnet.microsoft.com/
[h] https://www.postman.com/
[i] https://trends.google.com/trends/explore?q=swagger%20ui,postman,insomnia&geo=US
[j] https://www.techempower.com/benchmarks/
[k] https://docs.oracle.com/cd/E19182-01/821-0915/jbi_cluster-create_t/index.html

Tabelle 5: Bewertungskriterien für API-Testtools

4.1.2.2 Deployment

Ähnlich dem Vergleich der Implementierungsmöglichkeiten wurden auch für den Deploymentmöglichkeiten-Vergleich repräsentative Vertreter für die drei verbreitetsten Arten Application Server, Cloud Server und Container gewählt. Diese wurden bezüglich der Skalierbarkeit und der Continuous-Deployment-Fähigkeit verglichen. Dabei wurde zur Bewertung der Skalierbarkeit die Replizierbarkeit und das Möglichkeit von Loadbalancing sowie die Effizienz herangezugen um so die Fähigkeit zum vertikalem Skalieren abzubilden. Die Continuous Deployment Fähigkeit wird damit bestimmt, ob dies grundsätzlich möglich ist und wenn ja, mit oder ohne Server Downtime.

	Kriterium	Aws Api Gateway	Azure Api Management	Linode	AWS EC2	On Premise
Flexibilität Entwicklung	mehrere Sprachen	+	+	+	+	+
	vorgegebene Libraries/Frameworks	+	+	+	+	+
	Api-Dokumentaion/-Spezifikation	+	+	+	+	+
Flexibilität Deployment	Integration in CD Pipelines	/	+	+	+	+
	Restriktionen	?	?	+	+	+
	verschiedene Umgebungen (Test/Production)	+	+	+	+	+
	parallele Versionen möglich	+	+	+	+	+
Skalierbarkeit	Containerisierung möglich	?	+	/	/	/
Kosten	initial	+	+	+	+	-
	laufend	-	-	/	/	/
Abhängigkeiten	vorgeschriebene Libraries/Frameworks	+	+	+	+	+
	Abhängigkeit von Anbieter selbst	-	-	-	-	+
	Laufzeitumgebung	/	/	-	/	+

4.2 Entwicklung

4.2.1 Rahmenbedingungen

Wählen Sie für die weiteren Aufgaben dieser Übung eine konkrete Entwicklungsumgebung aus, begründen Sie Ihre Entscheidung

- Benötigte Softwareversionen und Werkzeuge
- Installation und Konfiguration der Entwicklungsumgebung
- Cloud-basierte Implementierung und Betrieb

Für eine nachvollziehbare Argumentation, warum der eingesetzte Toolstack verwendet wurde und welche Laufzeitumgebung und Art des Deployments als angebracht eingeschätzt wurde, sollen zunächst kurz die Anforderungen an die Anwendung dargestellt werden. Diese sind zwar "simuliert", jedoch (in sehr oberflächlicher Form) an möglichen realen Anforderungen angelehnt. Da die nicht-funktionalen Anforderungen hier eher die Argumentationsgrundlage bilden, stehen diese im Fokus - funktionale

Anforderungen sollten nur in Ausnahmefällen eine Determinante für Techstack und Deployment sein. Überlegungen, welche eine prototypische Umsetzung im gegebenen Rahmen sprengen würden, werden bewusst außer acht gelassen. Dazu gehören: ggfs. initial höhere Entwicklungskosten, verfügbare (Entwicklungs)ressourcen und Skillset der Beteiligten, architektonische Überlegungen und der Einsatz bestimmter Design Patterns, sowie das Thema Tests.
Desweiteren halten wir eine Begründung, warum nun welche Entwicklungsumgebung eingesetzt wurde, nicht für sinnvoll. Welche IDE ein Entwickler verwendet, ob als Git nun Github, Gitlab oder Bitbucket verwendet wird und mit welchem Tool REST Endpunkte getestet werden ist entweder von den Vorlieben und Gewohnheiten des Einzelnen abhängig, oder durch Vorgaben des Arbeitgebers bestimmt (oder beides). Insofern beschränken wir uns bei diesen Punkten auf die Benennung der "Werkzeuge", ohne das Warum weiter zu vertiefen. Stattdessen wollen wir die aus unserer Sicht viel wichtigere Frage beantworten, warum für den genannten Usecase eine bestimmte Sprache, Bibliotheken und Deploymentszenarien gewählt wurden.

4.2.1.1 Anforderungen

Funktionale Anforderungen:

- Anzeige von Basisinformationen zu Coderepositories (Autor, Sprache, Forks, Commits), welche über eine REST API abgerufen werden
- Löschen vorhandener, Hinzufügen neuer und Ändern vorhandener Repositories (Client)
- Persistierung der Änderungen in einer Datenbank
- Bereitstellung als Webapp

Nicht-funktionale Anforderungen:

- unterdurchschnittlich geringe TCO durch:
  - hohe Performanz und geringen Footprint bei der Hardwarenutzung
  - geringe Wartungskosten
  - einfache Verwaltung der Abhängigkeiten
  - einfaches Deployment
- gute Skalierbarkeit
- hohes Level an Sicherheit
- volle Flexibilität hinsichtlich der Laufzeitumgebung
- DB Typ möglichst offen

4.2.1.2 Verwendete Sprache(n)

Client und REST API sollen in Rust geschrieben werden, auch die verwendete Datenbank (Surreal DB) ist in Rust geschrieben. Rust ist eine multi-paradigmatische, noch recht junge (2015) Programmiersprache, die auf konzeptioneller Ebene einige Besonderheiten aufweist. Im Folgenden werden einige dieser Besonderheiten erläutert:

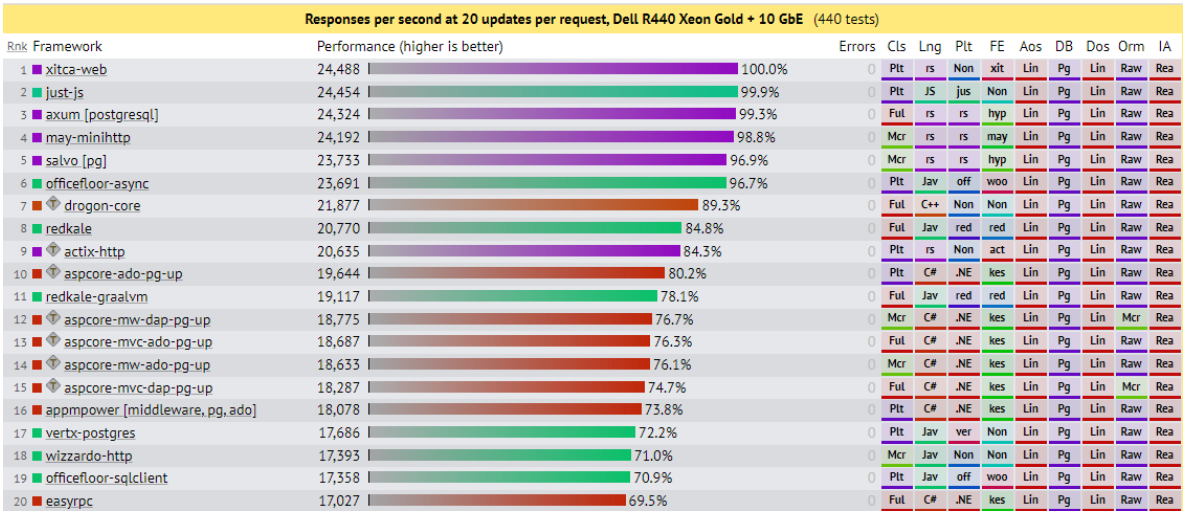
- Memory-Safety und Thread-Safety: Rust erreicht dies durch eine strenge Typisierung und durch Speicherzugriffsregeln, die sicherstellen, dass Speicher nur dann gelesen oder geschrieben werden kann, wenn es korrekt und sicher ist. Dies wird durch die Borrowing- und Ownership-Konzepte erreicht, die den Zugriff auf den Speicher in Rust stark reglementieren. Mit diesen Regeln ist es möglich, Memory-Safety-Garantien zu erzwingen, ohne dass ein Garbage-Collector erforderlich ist, aber auch ohne den in C und C++ verwendeten Ansatz der manuellen Speicherkontrolle.

- Laufzeitstabilität: Rust ist dafür bekannt, Laufzeitfehler quasi auszuschließen (von daher der Name - einmal ausgerollt kann die Anwendung vor ich hin rosten). Dies wird durch eine Kombination aus verschiedenen Techniken erreicht, darunter die bereits erwähnten Konzepte, den Verzicht auf nulls und einen in vielen Fällen funktionalen Programmierstil. Ausschlaggebend für die hohe Laufzeitstabilität ist zudem der tiefgreifende Compiler, der bereits bei der Übersetzung des Codes umfangreiche Fehlerprüfungen durchführt. Dadurch werden viele potenzielle Fehlerquellen bereits im Vorfeld erkannt und beseitigt.
- Gute Dokumentation: Die Gesamtdokumentation, insbesondere das Rust Book, aber auch die Dokumentation der einzelnen Bibliotheken, bietet sowohl Einsteigern als auch erfahrenen Entwicklern Hilfestellungen, um die Sprache zu erlernen und ihre Fähigkeiten zu verbessern.
- Management von Abhängigkeiten: das Management von Abhängigkeiten durch das Cargo-Build-System garantiert eine Kompatibilität der (transitiven) Abhängigkeiten und ein replizierbares Kompilat/Binary, sowie durch SemVer eine einfache Verwaltung der Abhängigkeiten
- Rust hat eine schnell wachsende Community und wird von immer mehr Unternehmen für die (Re)implementierung kritischer Komponenten eingesetzt (z.B. npm, Cloudflare und AWS Lambda). Teile des Android Kernels, sowie des Linuxkernels und neuerdings auch Systemkomponenten in Windows werden in Rust neu geschrieben. Diese Entwicklung deutet auf eine stabile Zukunft sowohl hinsichtlich technischem Support, also auch wachsender Entwicklerressourcen hin - ein gewichtiges Argument bei der Businessentscheidung für eine Sprache.

4.2.1.3 Komponenten

Aus der Beschreibung in Verbindung mit den nicht funktionalen Anforderungen lässt sich die Entscheidung für Rust für die systemkritischen (REST API) Komponenten ableiten. Sicherheit, Stabilität, eine hohe Flexibilität der Laufzeitumgebungen, Performanz (s. auch Abb.3 sowie voraussichtlich geringe TOC sind bei einer Umsetzung mit Rust wahrscheinlicher als in den meisten anderen Sprachen. Microsoft führt beispielsweise einen großen Teil der Schwachstellen auf fehlerhafte Speicherverwaltung zurück, dieses Risiko wird durch die garantierte Memory-Safety minimiert:

> "Microsoft revealed at a conference in 2019 that from 2006 to 2018 70 percent of their vulnerabilities were due to memory safety issues. Google also found a similar percentage of memory safety vulnerabilities over several years in Chrome."[3]



Responses per second at 20 updates per request, Dell R440 Xeon Gold + 10 GbE (440 tests)

Rnk	Framework	Performance (higher is better)		Errors	Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA
1	xitca-web	24,488	100.0%	0	Plt	rs	Non	xit	Lin	Pg	Lin	Raw	Rea
2	just-js	24,454	99.9%	0	Plt	JS	jus	Non	Lin	Pg	Lin	Raw	Rea
3	axum [postgresql]	24,324	99.3%	0	Ful	rs	rs	hyp	Lin	Pg	Lin	Raw	Rea
4	may-minihttp	24,192	98.8%	0	Mcr	rs	rs	may	Lin	Pg	Lin	Raw	Rea
5	salvo [pg]	23,733	96.9%	0	Mcr	rs	rs	hyp	Lin	Pg	Lin	Raw	Rea
6	officefloor-async	23,691	96.7%	0	Plt	Jav	off	woo	Lin	Pg	Lin	Raw	Rea
7	drogon-core	21,877	89.3%	0	Ful	C++	Non	Non	Lin	Pg	Lin	Raw	Rea
8	redkale	20,770	84.8%	0	Ful	Jav	red	red	Lin	Pg	Lin	Raw	Rea
9	actix-http	20,635	84.3%	0	Plt	rs	Non	act	Lin	Pg	Lin	Raw	Rea
10	aspcore-ado-pg-up	19,644	80.2%	0	Plt	C#	.NE	kes	Lin	Pg	Lin	Raw	Rea
11	redkale-graalvm	19,117	78.1%	0	Ful	Jav	red	red	Lin	Pg	Lin	Raw	Rea
12	aspcore-mw-dap-pg-up	18,775	76.7%	0	Mcr	C#	.NE	kes	Lin	Pg	Lin	Mcr	Rea
13	aspcore-mvc-ado-pg-up	18,687	76.3%	0	Ful	C#	.NE	kes	Lin	Pg	Lin	Raw	Rea
14	aspcore-mw-ado-pg-up	18,633	76.1%	0	Mcr	C#	.NE	kes	Lin	Pg	Lin	Raw	Rea
15	aspcore-mvc-dap-pg-up	18,287	74.7%	0	Ful	C#	.NE	kes	Lin	Pg	Lin	Mcr	Rea
16	appmpower [middleware, pg, ado]	18,078	73.8%	0	Plt	C#	.NE	kes	Lin	Pg	Lin	Raw	Rea
17	vertx-postgres	17,686	72.2%	0	Plt	Jav	ver	Non	Lin	Pg	Lin	Raw	Rea
18	wizzardo-http	17,393	71.0%	0	Mcr	Jav	Non	Non	Lin	Pg	Lin	Raw	Rea
19	officefloor-sqlclient	17,358	70.9%	0	Plt	Jav	off	woo	Lin	Pg	Lin	Raw	Rea
20	easyrpc	17,027	69.5%	0	Ful	C#	.NE	kes	Lin	Pg	Lin	Raw	Rea

Abbildung 3: Benchmark Backend Webframeworks

Die Entscheidung auch das Frontend in Rust zu implementieren war hingegen eher experimenteller Natur und würde - auch aufgrund der teils noch nicht ausgereiften Frameworks - in einer realen Situa-

tion vermutlich anders ausfallen. Dennoch soll die Entscheidung an dieser Stelle kurz begründet werden.

Da Rust problemlos in Maschinencode als auch Webassembly (Entwicklung 2018) kompiliert werden kann, verzichten die meisten Webframeworks, die in Rust geschrieben sind, komplett auf Javascript. Systemnahe Sprachen, typischerweise Assembler, C++ oder Rust, aber auch interpretierte Sprachen wie C# können mit der Laufzeitumgebung Webassembly in bytecode kompiliert werden, welcher plattformunabhängig und extrem schnell im Browser, zunehmend aber auch auf verteilten Systemen ausgeführt wird. Da die Last durch die Ausführung der Anwendungslogik im Browser hier auf Clientseite liegt, impliziert der Ansatz ein Abrücken vom traditionellen Client-Server Paradigma. Das verwendete Framework Dioxus zeichnet sich durch seinen Reactive Ansatz (ähnlich Svelte oder Solid.js), sowie eine sehr hohe Performanz aus (s. auch Abb.4. Zudem ist auch das Deployment für Mobiles und Desktoplattformen möglich.

Name Duration for...	vanillajs	sledge-hammer-v1.0.0	leptos-v0.2.1	dioxus-v0.3.0	vue-v3.2.47	elm-v0.19.1-3	svelte-v3.50.1	angular-v15.0.1	yew-v0.20.0	react-v18.2.0	blazor-wasm-aot-v6.0.1
Implementation notes	772	772 1139	1139	1139		1139			1139		1139
Implementation link	code	code	code	code	code	code	code	code	code	code	code
create rows creating 1,000 rows (5 warmup runs).	39.1 ±0.6 (1.02)	39.3 ±0.4 (1.03)	46.5 ±0.5 (1.21)	40.7 ±0.6 (1.06)	45.4 ±0.7 (1.18)	50.6 ±2.0 (1.32)	49.1 ±0.2 (1.28)	48.3 ±0.4 (1.26)	69.3 ±0.8 (1.81)	52.2 ±0.9 (1.36)	109.1 ±0.3 (2.85)
replace all rows updating all 1,000 rows (5 warmup runs).	41.5 ±0.7 (1.03)	41.2 ±0.6 (1.02)	48.7 ±0.4 (1.20)	45.8 ±0.6 (1.13)	45.6 ±0.6 (1.13)	49.2 ±2.2 (1.22)	52.1 ±0.3 (1.29)	51.9 ±0.5 (1.28)	76.0 ±0.9 (1.88)	54.6 ±1.0 (1.35)	111.8 ±0.7 (2.76)
partial update updating every 10th row for 1,000 rows (3 warmup runs). 16x CPU slowdown.	106.0 ±2.9 (1.09)	106.9 ±2.0 (1.10)	105.4 ±2.9 (1.08)	107.9 ±2.3 (1.11)	120.3 ±3.8 (1.23)	116.6 ±3.9 (1.20)	113.9 ±2.2 (1.17)	110.8 ±1.9 (1.14)	119.1 ±3.0 (1.22)	145.6 ±3.8 (1.49)	400.1 ±3.9 (4.10)
select row highlighting a selected row. (5 warmup runs). 16x CPU slowdown.	12.1 ±0.8 (1.10)	12.3 ±0.8 (1.12)	14.0 ±0.7 (1.28)	17.2 ±0.8 (1.57)	19.9 ±1.0 (1.82)	17.5 ±1.0 (1.59)	19.1 ±0.8 (1.74)	16.6 ±1.4 (1.51)	22.1 ±0.7 (2.02)	44.4 ±1.5 (4.05)	304.0 ±3.2 (27.75)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4x CPU slowdown.	29.0 ±0.9 (1.07)	28.2 ±0.9 (1.04)	29.0 ±0.9 (1.07)	31.0 ±1.1 (1.14)	29.3 ±0.3 (1.08)	41.5 ±4.6 (1.53)	30.0 ±0.4 (1.10)	171.0 ±0.9 (6.30)	31.9 ±1.1 (1.18)	168.6 ±1.4 (6.21)	98.1 ±0.7 (3.61)
remove row removing one row. (5 warmup runs). 4x CPU slowdown.	45.9 ±1.0 (1.01)	46.2 ±0.6 (1.02)	48.0 ±1.1 (1.05)	47.0 ±0.8 (1.03)	51.6 ±0.8 (1.13)	54.3 ±1.8 (1.19)	49.5 ±1.2 (1.09)	47.9 ±1.3 (1.05)	49.3 ±1.1 (1.08)	53.5 ±1.1 (1.17)	118.8 ±0.6 (2.61)
create many rows creating 10,000 rows. (5 warmup runs with 1k rows).	417.0 ±1.5 (1.00)	419.2 ±1.7 (1.01)	511.9 ±3.0 (1.23)	449.5 ±2.1 (1.08)	494.5 ±4.3 (1.19)	495.5 ±2.5 (1.19)	541.6 ±3.3 (1.30)	497.2 ±2.5 (1.19)	2,216.2 ± 6.0 (5.32)	682.9 ±3.5 (1.64)	1,134.9 ± 3.4 (2.73)
append rows to large table appending 1,000 to a table of 10,000 rows. 2x CPU slowdown.	86.9 ±0.3 (1.00)	86.8 ±0.2 (1.00)	102.8 ±0.5 (1.18)	99.8 ±0.9 (1.15)	98.9 ±0.8 (1.14)	99.7 ±2.3 (1.15)	113.5 ±0.6 (1.31)	106.9 ±0.4 (1.23)	153.3 ±1.1 (1.77)	117.7 ±0.6 (1.36)	266.9 ±1.9 (3.07)
clear rows clearing a table with 1,000 rows. 8x CPU slowdown. (5 warmup runs).	30.3 ±1.1 (1.04)	31.0 ±0.9 (1.06)	33.7 ±1.4 (1.16)	40.0 ±0.9 (1.38)	37.1 ±1.4 (1.28)	36.3 ±0.7 (1.25)	42.9 ±1.4 (1.48)	69.4 ±1.6 (2.39)	61.8 ±1.4 (2.13)	40.3 ±1.0 (1.39)	75.5 ±1.6 (2.60)
geometric mean of all factors in the table	1.04	1.04	1.16	1.17	1.23	1.28	1.29	1.59	1.81	1.85	3.85
compare: Green means significantly faster, red significantly slower	compare	compare	compare	compare	compare	compare	compare	compare	compare	compare	compare

Abbildung 4: Benchmark Frontend Webframeworks

DB

4.2.1.4 Eingesetzte Frameworks und Libraries

Service-komponente	Name (Version)	Funktion	Vorteil	Nachteil
Client	Sycamore	Webassembly Webframework		
Client	Perseus	Sycamore Erweiterung		
REST API	Serde	JSON (De)serialisierung		
REST API	Actix	Webserver		
REST API	utoipa	Open API Doc Generation		
Datenbank	Surreal DB	vollständiges DBMS und integrierter Server		

Tabelle 6: Verwendete, externe Abhängigkeiten

Listing 4: cargo.toml Datei zur Organisation der Abhängigkeiten in Rust

```
[package]
name = "rust-actix-surreal-rest-api"
version = "0.1.0"
edition = "2021"
authors = ["Hannes Roever"]

[dependencies]
actix-web = "4"
actix-cors = "*"
serde = {version = "1.0.152", features = ["derive"]}
serde_json = {version = "1.0.93"}
tokio = { version = "1", features = ["full"] }
mini-redis = "0.4"
env_logger = "0.10.0"
log = "0.4"
futures = "0.3"
utoipa = { features = ["actix_extras"] }
utoipa-swagger-ui = { features = ["actix-web"] }
chrono = "*"
reqwest = {features = ["json"]}
```

4.2.1.5 Konfiguration Entwicklungsumgebung

Voraussetzung für die dargestellten Schritte ist, dass Docker bereits installiert ist (Docker Client auf Windows, Docker Engine auf Linux). Da dies, analog zum Vorhandensein einer geeigneten IDE oder eines Editors, zu den Basiswerkzeugen in der Entwicklung gehört, wird der allgemeine Installations- und Konfigurationsprozess nicht weiter ausgeführt (zumal er sich je nach OS auch unterscheidet und bestens dokumentiert ist).

4.2.1.5.1 Datenbank

Die Datenbank kann sehr unkompliziert als Docker-Container gestartet werden. Das entsprechende CLI Kommando bzw. der Inhalt und das Kommando zum Ausführen der docker-compose.yml sind in den Listings 5-7 dargestellt. s sollte nur eine der Optionen genutzt werden. Anschließend läuft die Datenbank mit in-memory Option (weitere sind möglich) unter Port 8000 des localhost.

Listing 5: CLI Command zum Starten des Datenbankcontainers

```
docker run --rm --pull always -p 8000:8000 surrealdb/surrealdb:latest start
```

Listing 6: Alternative mit docker-compose zum Starten des Datenbankcontainers

```
version: '3.8'
services:
  db:
    image: surrealdb/surrealdb:latest
    restart: always
    command: start --user root --pass root memory
    ports:
      - '8000:8000'
    volumes:
      - db:/var/lib/surrealdb/data
volumes:
  db:
    driver: local
```

Listing 7: CLI Command zum Ausführen der docker-compose Datei. Das Kommando muss im Verzeichnis ausgeführt werden in dem die Datei liegt oder der Pfad der Datei über die flag –f spezifiziert werden

```
docker-compose up -d
```

4.2.1.5.2 REST API

Für die Entwicklung in Rust wird die Rust Toolchain benötigt (bestehend aus rustup, rustc und cargo). Die Installation erfolgt über die Kommandozeile oder für Windows mit einem Installer, welcher unter https://www.rust-lang.org/tools/install heruntergeladen werden kann. Ggfs. muss noch die ensprechende Umgebungsvariable gesetzt werden. Die Toolchain umfasst alle notwendigen Commandlinetools für die Kompilierung, Codeformatierung, Abruf von Dokumentation (ähnlich zu MAN Pages), Tests und Deployment.

Listing 8: CLI Command zur Installation von Rust in Linux und macOS

```
curl --proto '=https' --tlsv1.3 https://sh.rustup.rs -sSf | sh
```

Für die Erstellung eines neuen Projekts muss das Kommando cargo new projektname ausgeführt werden. Im entsprechenden Verzeichnis wird ein Ordner mit den Konfigfiles, main und Gitrepository angelegt. Die Bearbeitung des Codes kann mit einem einfachen Editor (z.B. Vim, Neovim, Emacs, Sublime, Nano), einem erweiterten Editor (VS Code) oder einer vollumfänglichen IDE (Intellij IDEA, CLion) vorgenommen werden. Wir nutzen IntelliJ und für die schnelle Bearbeitung, z.B. auf einem über SSH verbundenen Server, Nano.
Weitere Schritte sind nicht notwendig, die Abhängigkeiten können in der cargo.toml (s.a. Listing 12) Datei hinzugefügt werden und werden beim nächsten Build, so noch nicht lokal vorhanden, automatisch gezogen und kompiliert. Mit cargo run (bauen, ausführen) bzw cargo build (bauen), fürs publishing mit –release flag, wird das Programm ausgeführt.

4.2.1.5.3 Client WebApp

Um die Kompilierung in WASM zu ermöglichen sind zwei weitere, global bereitzustellende Abhängigkeiten notwendig, die Installation ist in Listing 9 zu sehen.

Listing 9: CLI Command zur Installation der Laufzeitumgebung webassembly und des WASM-Buildtools Trunk für Rust

```
rustup target add wasm32-unknown-unknown
cargo install --locked trunk
```

Der Start eines bereits erstellten Projektes kann mit trunk –serve durchgeführt werden, durch das Buildtool wird automatisch ein lokaler Webserver bereitgestellt. Perseus baut auf Sycamore auf und kann mit den Commands aus Listing 10 installiert und ausgeführt werden.

Listing 10: CLI Command zur Installation der Perseus CLI und Ausführung eines Projektes

```
cargo install perseus-cli
perseus serve -w
```

4.2.1.6 Deployment

Das Deployment wird, dem Industriestandard folgend, als Containerlösung realisiert. Um den Rahmen nicht zu sprengen, haben wir uns für Docker entschieden und auf einen Orchestrierungslayer, z.B. mit K8, verzichtet. Die physische Bereitstellung erfolgt bei einem IAAS Anbieter, aufgrund der Nutzung von Docker ist die Linux-Distribution zweitrangig - Debian oder Ubuntu als etablierte Serverdistros oder Alpine als Low-Footprint Distro sind naheliegende Optionen. Windows Server oder spezielle oder proprietäre Lösungen sind nicht notwendig und u.E. auch nicht sinnvoll, weil sie eine Abhängigkeit von einer bestimmten Firma bzw. Technologie schaffen. Zudem haben Umgebungen wie Java Application Server einen extrem hohen Overhead, den wir vermeiden wollen um die gesteckten Ziele nicht zu gefährden.

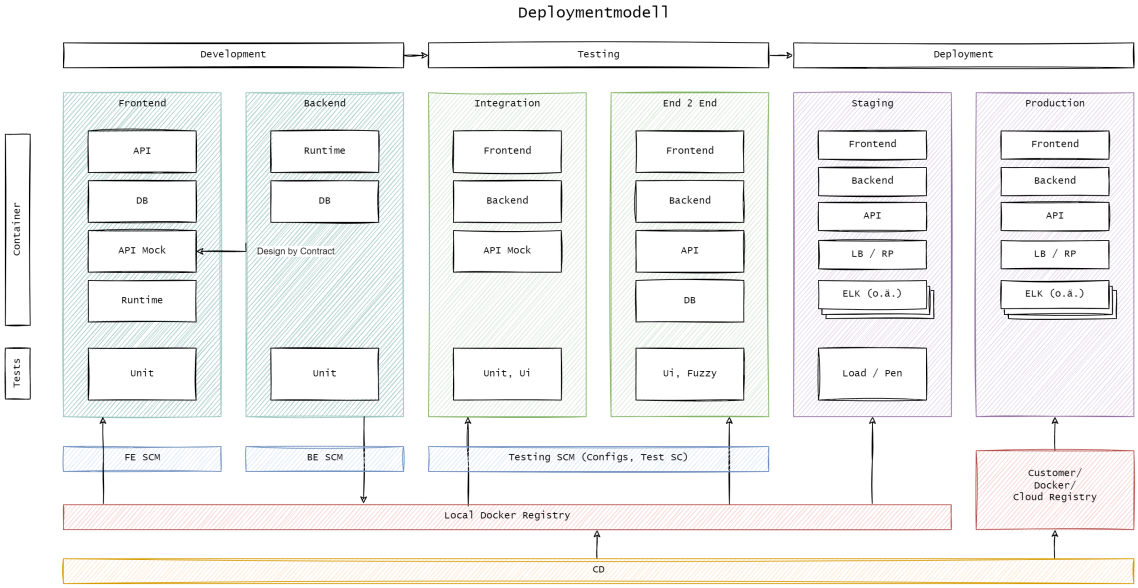


Abbildung 5: Deploymentdiagramm

4.2.2 Umsetzung

Entwicklung einer Web-API (mind. 6 Operationen bzw. Datenressourcen - ggf. CRUD) und eines korrespondierenden Client

- Berücksichtigen Sie in der Doku Analyse, Design, Implementierung und Test
- Deployment (Installation) innerhalb der Laufzeitumgebung

Analyse: nee, Test nee

4.2.2.1 Design

Komponentendiagramm, Deploymentdiagramm,

4.2.2.2 Implementierung

Blablabla

4.2.2.2.1 Client

Blablabla

Listing 11: cargo.toml Datei zur Organisation der Abhängigkeiten in Rust

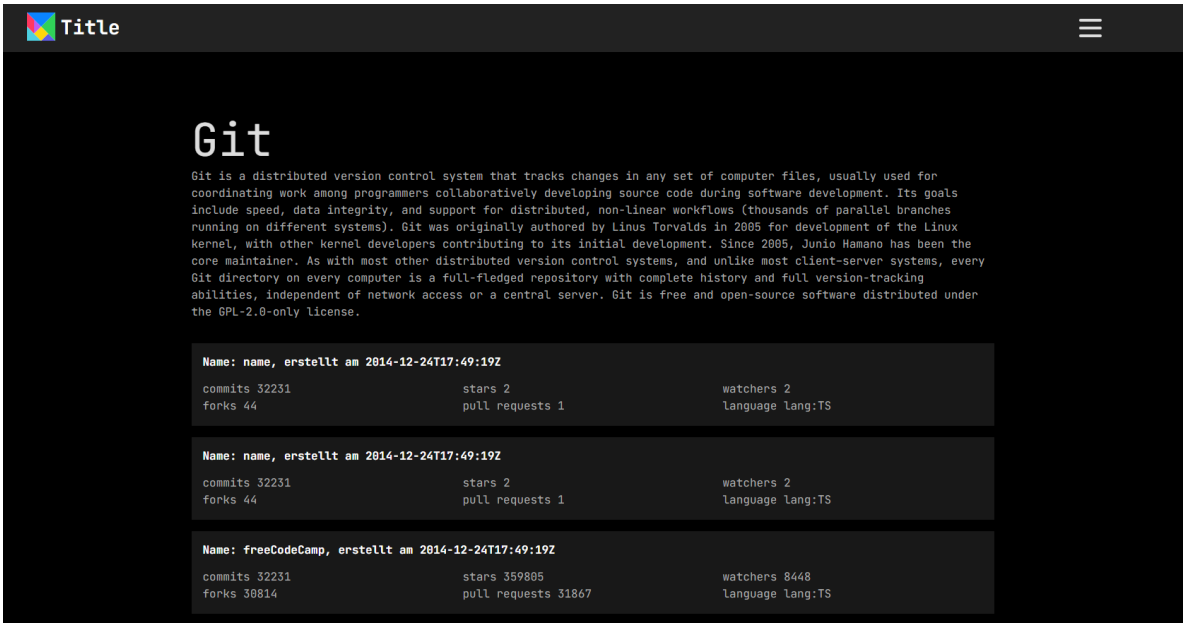


Abbildung 6: Screenshot des Clients

4.2.2.2.2 REST API

Blablabla

Listing 12: cargo.toml Datei zur Organisation der Abhängigkeiten in Rust

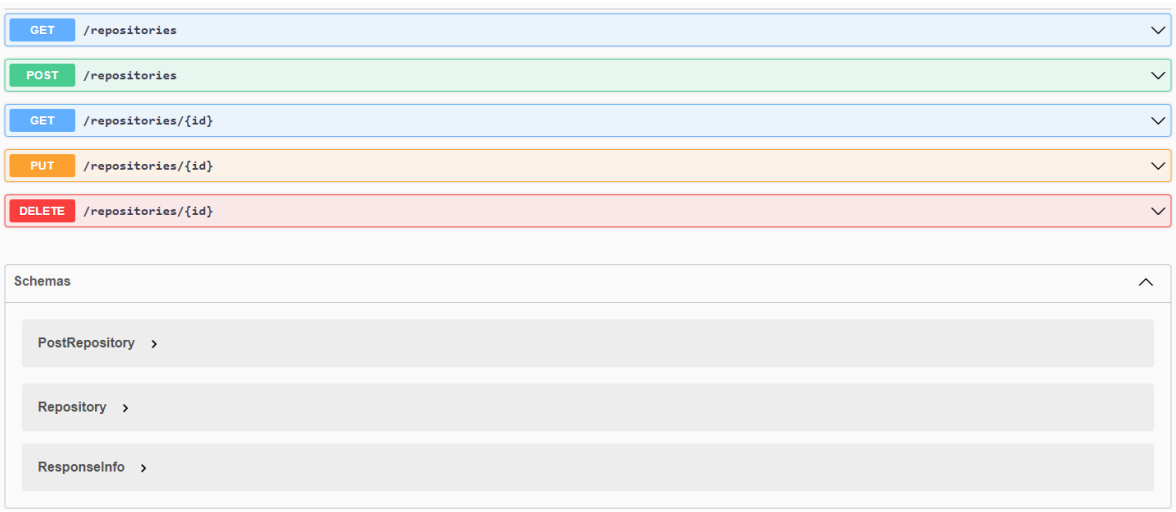


Abbildung 7

4.2.2.3 Tests

Blablabla

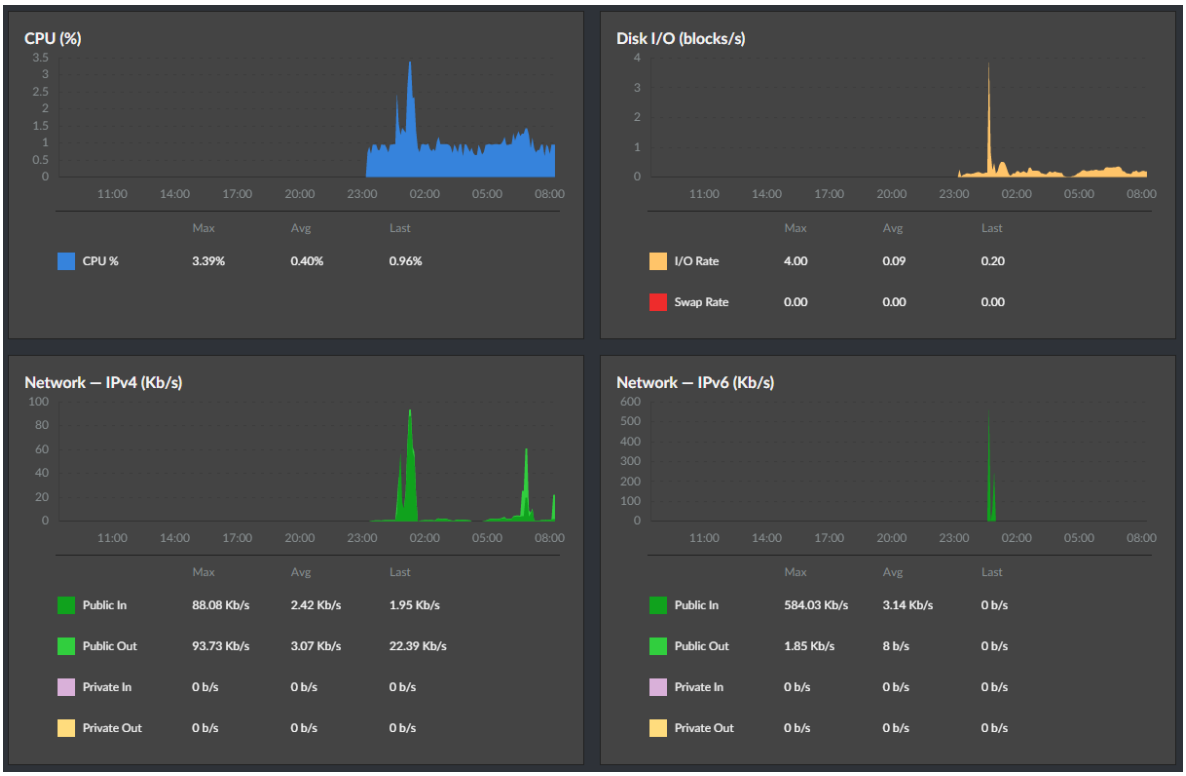


Abbildung 8

```
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS         PORTS                                                NAMES            SIZE
c5729c2b2ecc   hansendockedin/rust-be:latest  "./rust-actix-surrea…"   8 hours ago     Up 8 hours     0.0.0.0:8088->8088/tcp, :::8088->8088/tcp            root_rest-api_1  0B (virtual 30MB)
893699d003ac   surrealdb/surrealdb:latest     "/surreal start --us…"   8 hours ago     Up 8 hours     0.0.0.0:8000->8000/tcp, :::8000->8000/tcp            root_db_1        0B (virtual 50.7MB)
```

Abbildung 9

```
┌──(hannes㉿LT-6FGD373)-[~]
└─$ wrk -t100 -c100 -d5s http://139.144.71.117:8088/repositories
Running 5s test @ http://139.144.71.117:8088/repositories
  100 threads and 100 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   301.64ms  111.74ms   1.11s    84.66%
    Req/Sec     3.38      1.55    10.00     56.26%
  1582 requests in 5.10s, 10.63MB read
Requests/sec:    310.17
Transfer/sec:      2.08MB
```

Abbildung 10

```
┌──(hannes㉿LT-6FGD373)-[~]
└─$ wrk -t100 -c100 -d5s http://localhost:8088/repositories
Running 5s test @ http://localhost:8088/repositories
  100 threads and 100 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    27.99ms   25.25ms 202.10ms   89.88%
    Req/Sec    40.63     31.37   131.00     75.50%
  20204 requests in 5.10s, 4.44MB read
Requests/sec:   3961.09
Transfer/sec:      0.87MB
```

Abbildung 11

4.2.2.3 Deployment

Für beide selbst entwickelten Services wird mit Hilfe von Dockerfiles (Listing 13) ein Image erstellt und

aufs Dockerhub gepusht. In einem produktiven Szenario, insbesondere wenn der Code Closed Source ist, sollte stattdessen eine eigene Docker Registry verwendet werden. Die Schritte zur Öffentlichung sind jedoch bis auf den Host im Command docker push ... dieselben.

Listing 13: Dockerfile für die Erstellung des REST-API Images

```
1  FROM rust:1.60.0-bullseye AS build
2  WORKDIR /app
3  COPY . .
4  RUN cargo build --release
5  RUN mkdir -p /app/lib
6  RUN cp -LR $(ldd ./target/release/rust-actix-surreal-rest-api | grep "=>" | cut -d
       ' ' -f 3) /app/lib
7
8  FROM scratch AS app
9  WORKDIR /app
10 COPY --from=build /app/lib /app/lib
11 COPY --from=build /lib64/ld-linux-x86-64.so.2 /lib64/ld-linux-x86-64.so.2
12 COPY --from=build /app/target/release/rust-actix-surreal-rest-api rust-actix-
       surreal-rest-api
13 ENV LD_LIBRARY_PATH=/app/lib
14 ENTRYPOINT ["./rust-actix-surreal-rest-api"]
```

Auf dem Server, auf welchem die Services laufen sollen, muss ein Pull der Images erfolgen oder der Pfad zur Registry im docker-compose File angegeben sein, dann wird das Image automatisch bezogen. Die Ausführung von docker-compose up -d erstellt dann aus den Images die Container mit der dargestellten Konfiguration. Die virtuell erstellten Netzwerke in Docker (s. Listing 14 in Zeile 11, 18 und 31) ermöglichen eine zusätzliche Kapselung, der einzig nach außen geöffnete Port ist im Beispiel 8080. In einer produktiven Umgebung wäre hier entweder noch ein weiterer Service in Form eines Reverse Proxys (z.B. nginx oder traefik) vorhanden, welcher über Port 443 erreichbar ist und über LetsEncrypt ein Zertifikat bezieht. Alternativ könnte auf dem Server direkt ein nginx Webserver bereitgestellt werden, Hauptsache die über HTTP erreichbaren Services sind in einem gekapselten Netzwerk nur über die Weiterleitung der Anfragen des Reverse Proxys erreichbar.

Listing 14: docker-compose.yml zur Bereitstellung des kompletten Stacks

```
1  version : '3.8'
2  services:
3    db:
4      image: surrealdb/surrealdb:latest
5      restart: always
6      command: start --user root --pass root memory
7      expose:
8        - 8000
9      volumes:
10       - db:/var/lib/surrealdb/data
11     networks:
12       - backend
13
14   rest-api:
15     image: rust-actix-surreal-rest-api
16     expose:
17       - 8088
18     networks:
19       - backend
20       - frontend
21     depends_on:
22       - db
23     environment:
24       - BASE_URL=http://db
25       - CORS_ALLOW=http://localhost:8080
26
27   client:
28     image: rust-client
29     ports:
30       - '8080:8080'
31     networks:
```

```
32        - frontend
33      depends_on:
34        - rest-api
35      environment:
36        - SERVICE_URL=http://rest-api
37
38  networks:
39    backend:
```

Die dargestellte Form des Deployments ermöglicht eine sehr schnelle Aktualisierung der Services. Der Veröffentlichung des neuen Images würde i.d.R. natürlich ein umfangreiches, automatisiertes Testing vorausgehen, das Image selbst ist dann das Artefakt. Die erneute Ausführung von docker-compose up -d würde dann ausschließlich die Container neu starten, für welche Änderungen der Images festgestellt wurden. Dies dauert maximal einige Sekunden. Um auch dies zu vermeiden wäre es mit wenigen Zeilen zusätzlicher Konfiguration möglich, die Container zu replizieren und nach Terminierung der Verbindung im Reverse Proxy dynamisch die Last zu verteilen. Der Reverse Proxy hat dann dementsprechend gleichzeitig die Funktion eines Loadbalancers.
Um den Rahmen nicht zu sprengen, haben wir kein Monitoring und Remote Logging realisiert, auch dies wäre jedoch durch die Nutzung von Docker einfach umzusetzen. Images, z.B. für den oft verwendeten ELK Stack oder alternativ die Kombination von Grafana und Prometheus, sind vorhanden und mit wenigen Anpassungen als weitere Services innerhalb der docker-compose File einsetzbar. Des weiteren würden die Services in einem produktiven Umfeld als Cluster auf physisch getrennten Systemen laufen. Das Deployment kann auf jedem beliebigen Linuxserver erfolgen, auf dem Docker installiert ist. In unserem Fall haben wir Linode (Akamai) als IAAS Anbieter ausgewählt und die Anwendung auf einem Alpine Server mit 1GB RAM bereitgestellt.

4.2.3 Anbindung Datenbank

Originäre Verwendung eines DBMS (auch NoSQL) als Service-Schnittstelle

- Prototypisches Aufsetzen eines konkreten Datenbanksystems (ggf. Cloud)
- Details der Konfiguration und Administration – ggf. Probleme
- Eigene Kapselung mit Hilfe einer WSDL, Swagger oder GraphQL
- Performanter Umgang mit XML/JSON-basierten Datenströmen

[Sharp]C

Listing 15: CLI Kommandos zur lokalen Installation der Datenbank für Windows Linux und macOS

```
1 iwr https://windows.surrealdb.com -useb | iex
2 curl -sSf https://install.surrealdb.com | sh
3 brew install surrealdb/tap/surreal
```

Listing 16: CLI Kommando zur Übertragung der Daten aus der Datei in Listing 17

```
1 cat schemashort.sql | surreal sql --conn http://localhost:8000 --user root --pass
      root --ns base --db base
```

Listing 17: Ausschnitt der sql Setupdatei

```
1 INSERT INTO repository (name, stars_count, forks_count, watchers, pull_requests,
      primary_language, languages_used, commit_count, created_at, licence) VALUES ('
      react', 159266, 30464, 8497, 2911, lang:JavaScript, [lang:JavaScript, lang:
      HTML, lang:CSS], 5562, '2013-05-24T16:15:54Z', 'MIT License');
2 INSERT INTO repository (name, stars_count, forks_count, watchers, pull_requests,
      primary_language, languages_used, commit_count, created_at, licence) VALUES ('
      scikit-learn', 38327, 18225, 4968, 1701, lang:Python, [lang:Python, lang:
      Cython, lang:HTML, lang:CSS], 4085, '2010-01-10T09:58:52Z', 'BSD-3-Clause
      License');
```

```
3 INSERT INTO repository (name, stars_count, forks_count, watchers, pull_requests,
      primary_language, languages_used, commit_count, created_at, licence) VALUES ('
      angular', 68521, 24536, 6779, 2197, lang:TypeScript, [lang:TypeScript, lang:
      JavaScript, lang:HTML, lang:CSS], 4248, '2014-09-18T16:12:01Z', 'MIT License')
      ;
```

5 Übung 4d: Sicherheit von Web APIs

5.1 Sicherheitsrisiken in Verbindung mit dem HTTP Protokoll

- Beschreiben Sie stichpunktartig die Eigenschaften von TLS (SSL) in Verbindung mit HTTPS
- Welche Verfahren zur Authentifizierung können im Rahmen des HTTP Protokolls **direkt** verwendet werden? Gehen Sie auf Stärken und Schwächen ein
- Authentifizierungs- und Benutzerinformationen werden bei Webanwendungen häufig mit Hilfe von Cookies übertragen. Gehen Sie auf die damit einhergehenden Nachteile ein.

5.1.1 HTTPS und TLS

HTTP Eigenschaften:

- ist das Standardprotokoll für die Internetkommunikation
- arbeitet nach dem Client Server Modell
- nutzt TCP
- hat zwei Typen: non-persistent (weniger Overhead, einmalige Verbindung, wird nicht aufrechterhalten) und persistent (Verbindung wird nach Aufbau aufrechterhalten, s. **??**)
- funktioniert über Request und Response Message
- Benutzer- und Serverstatus werden über Cookies aufrechterhalten
- Webcache kann Geschwindigkeit erhöhen (lokal im Browser oder serverseitig auf Proxy), Response ist 304 (conditional GET)
- HTTPS (S steht für secure) ist eine Erweiterung von HTTP

TLS Eigenschaften: (Wiki)

- Nachfolger von SSL, steht für Transport Layer Security
- Verbindung zu HTTPS: kommt im TCP/IP Stack zwischen Transport und Anwendungsebene zum Einsatz und wird i.d.R. zusätzlich zum TCP Protokoll eingesetzt.
- ermöglicht eine Ende zu Ende Verschlüsselung von Data in Transit (und ist deshalb z.B. bei Mails durch zusätzliche Teilnehmer zwischen den "Enden" nur eingeschränkt sicher)
- Im TLS Handshake findet ein sicherer Schlüsselaustausch und eine Authentifizierung statt.
- Für den Schlüsselaustausch sind in den älteren TLS-Versionen verschiedene Algorithmen mit unterschiedlichen Sicherheitsgarantien im Einsatz. Die neueste Version TLS 1.3 verwendet allerdings nur noch das Diffie-Hellman-Schlüsselaustausch Protokoll (DHE oder ECDHE) auf Basis elliptischer Kurven.
- Dabei wird für jede Verbindung ein neuer Sitzungsschlüssel (Session Key) ausgehandelt. Da dies ohne Verwendung eines Langzeitschlüssels geschieht, erreicht TLS 1.3 Perfect Forward Secrecy.
- Alle TLS-Handshakes verwenden eine asymmetrische Kryptographie (öffentlicher und privater Schlüssel), aber nicht alle nutzen den privaten Schlüssel beim Generieren von Sitzungsschlüsseln.

Vorteile TLS (1.3): TLS 1.3 hat die Unterstützung für ältere, weniger sichere kryptografische Features eingestellt und unter anderem TLS-Handshakes schneller gemacht. Die Hauptvorteile von TLS 1.3 gegenüber TLS 1.2 sind schnellere Geschwindigkeiten und verbesserte Sicherheit. TLS und verschlüsselte Verbindungen erzeugen naturgemäß einen Overhead bei der Übertragung. HTTP / 2 hat bei diesem Problem durch die Verringerung der Schritte beim Aufbau der TCP Verbindung geholfen, aber TLS 1.3 beschleunigt verschlüsselte Verbindungen durch Funktionen wie TLS false start und Zero Round Trip Time (0-RTT) noch weiter. Die Einführung elliptischer Kurven verbessert zudem bei gleicher Schlüssellänge die Sicherheit und vermeidet beispielsweise Angriffe wie LogJam, die auf dem Number field sieve Algorithmus basieren (welcher die Tatsache ausnutzt, dass immer dieselbe Primzahl verwendet wird). Auch export-grade Funktionalitäten (welche demselben Angriff zugrunde lagen) sind in TLS 1.3 nicht mehr eingebaut.

5.1.2 Authentifizierungmöglichkeiten HTTP(S)

Es gibt mehrere Möglichkeiten, Benutzer (Clients) im Rahmen des HTTP-Protokolls zu authentifizieren. Verbreitet sind (Wiki, ssl.com):

- Basic Authentication: Die Basic Authentication (Basisauthentifizierung) wird seit 2015 durch RFC 7617 spezifiziert und ist eine häufig verwendete Art der HTTP-Authentifizierung. Der Webserver fordert mit Eingabe von Benutzername und Passwort eine Authentifizierung an. Ein Vorteil der Basic Authentication ist ihre Einfachheit in der Implementierung. Ein Nachteil ist, dass die Anmeldeinformationen im Klartext übertragen werden und daher leicht abgefangen werden können. Deshalb sollte diese Methode nur für den Hobbybereich eingesetzt werden.

- Digest Access Authentication: Die Hashwertauthentifizierung ist ein Verfahren, das die Basic-Authentifizierung ersetzen soll(te). Der Server sendet eine Zeichenfolge zufälliger Daten, auch Nonce genannt, als Challenge an den Client. Der Client reagiert mit einem Hash, der neben anderen Informationen den Benutzernamen, das Kennwort und die Nonce enthält. Die Digest Access Authentication bietet mehr Sicherheit als die Basic Authentication, da sie einen Hash verwendet und somit die Anmeldeinformationen nicht im Klartext übertragen werden.

Weitere Verfahren, die im Zusammenhang mit HTTPS eingesetzt werden, sind nicht den direkten Verfahren zuzurechnen, weil sie alle eine dritte Instanz hinzuziehen müssen (Validierung Zertifikate über PKI, Authorisierungs- bzw. Authentifizierungsprüfung über Drittanbieter bei OAuth2 und OIDC).

5.1.3 Cookies

5.1.3.1 Begriff

Cookies sind kleine Textdateien, die von einer Webseite im Internetbrowser eines Nutzers gespeichert werden können. Sie dienen dazu, Informationen über den Nutzer und seine Interaktionen mit der Webseite zu speichern.
Cookies können nützlich sein, indem sie beispielsweise Einstellungen im Webbrowser abspeichern oder dafür sorgen, dass ein Warenkorb beim Online-Shopping zu einem späteren Zeitpunkt wieder aufgerufen werden kann. Sie können auch dazu verwendet werden, das Surfverhalten von Nutzern im Internet über einen längeren Zeitraum zu verfolgen und detaillierte Nutzerprofile anzulegen (was eher aus Sicht des Betreibers ein Vorteil ist...).
Grundsätzlich werden zwei Arten von Cookies unterschieden: technisch notwendige und technisch nicht notwendige Cookies. Technisch notwendige Cookies sind für das Funktionieren der Webseite notwendig, während technisch nicht notwendige Cookies für Zwecke wie das Verfolgen des Surfverhaltens verwendet werden können.
Es ist zwar möglich, eine Webseite ohne Cookies zu betreiben, allerdings kann dies zu Einschränkungen in der Funktionalität führen. Beispielsweise müssten Nutzer bei jedem Besuch der Webseite erneut ihre Einstellungen vornehmen oder sich erneut anmelden. Sofern jedoch eine Persistierung von Nutzerdaten nicht notwendig ist, z.B. bei statischen, rein informativen Seiten, kann auf Cookies verzichtet werden (was oft nicht passiert, weil durch den Einsatz von Tracking, eingebetteten Webfonts usw. auch auf solchen Seiten, Cookies gesetzt werden). Dabei sollte jedoch zwischen tendenziell unkritischen Sessioncookies, die lokal und verschlüsselt gespeichert werden und Trackingcookies, welche übertragen werden,

unterschieden werden.
Ein großer Nachteil von Cookies ist mit ihnen einhergehende Sicherheitsrisiko. Da sie Informationen über den Nutzer und seine Interaktionen mit der Webseite speichern, können sie von Dritten abgefangen und missbraucht werden.

5.1.3.2 Sicherheitsrisiken

Cookie Poisoning, XSS, Cookiethief

5.2 Möglichkeiten zur Risikominderung

5.2.1 OWASP

Machen Sie sich mit den OWASP Top 10 API Security Risiken vertraut

- Gehen Sie für 5 Sicherheitslücken auf entwicklerseitige oder betriebliche Möglichkeiten zur Verminderung bzw. Abmilderung ein
- Gehen Sie für 2 Sicherheitslücken auf die Möglichkeiten von Tests zur Aufdeckung potentieller Schwachstellen ein.

Dem Ranking liegt das OWASP API Security Threat Model zugrunde. Es ist eine Methodologie zur Identifizierung und Bewertung von Bedrohungen für APIs. Es folgt einem strukturierten Ansatz, der aus vier Schritten besteht:

1. API-Beschreibung: Hier werden die API-Funktionen und Datenobjekte beschrieben.
2. Bedrohungsmodellierung: In diesem Schritt werden potenzielle Bedrohungen identifiziert und priorisiert. Die Bedrohungsmodelle werden auf der Basis der API-Beschreibung entwickelt.
3. Risikobewertung: Die Bedrohungsmodelle werden bewertet, um diejenigen mit dem höchsten Risiko zu identifizieren.
4. Empfehlungen: Basierend auf den identifizierten Bedrohungen und Risiken werden Empfehlungen für Sicherheitsmaßnahmen abgeleitet, um das Risiko zu minimieren.

Die OWASP klassifiziert die Bedrohungen folgendermaßen (mehr Punkte = höhere Gefahr):

Punkte	Exploitability (Ausnutzbarkeit)	Weakness Prevalence (Häufigkeit)	Weakness Detectability (Aufspürbarkeit)	Technical Impact (Auswirkungen)
3	Easy	Widespread	Easy	Severe
2	Average	Common	Average	Moderate
1	Difficult	Difficult	Difficult	Minor

Tabelle 7: OWASP Thread Model, allgemeine Form

Die Methodologie soll helfen, API-Designs zu verbessern und Entwicklern zu helfen, sicherere APIs zu erstellen. Für die ausgewählten Lücken geben wir zunächst die (aufgrund der technischen Fachtermini im englischen belassene) Beschreibung der OWASP Foundation an. Anschließend zeigen wir je anhand eines negativen und positiven Beispiels, wie das Ausnutzen der Lücke eingeschränkt werden kann. Dabei wird im jeweiligen Beispiel nur die entsprechende Schwachstelle beachtet, d.h. andere Teile im Beispiel können hinsichtlich weiterer Schwachstellen durchaus anfällig sein. Zudem vereinfachen die Beispiele die oftmals komplexen Problematiken natürlich.

Vulnerability	Exploitability (Ausnutzbarkeit)	Weakness Prevalence (Häufigkeit)	Weakness Detectability (Aufspürbarkeit)	Technical Impact (Auswirkungen)
Broken Object Level Authorization	3	3	2	3
Excessive Data Exposure	3	2	2	2
Broken Function Level Authorization	3	2	1	2
Injection	3	2	3	3
Improper Assets Management	3	3	2	2

Tabelle 8: OWASP Thread Model für APIs für die 5 ausgewählten Sicherheitslücken

5.2.1.1 Broken Object Level Authorization

> APIs tend to expose endpoints that handle object identifiers, creating a wide attack surface Level Access Control issue. Object level authorization checks should be considered in every function that accesses a data source using an input from the user.

Das erste Beispiel (Listing 18) stellt ein Sicherheitsrisiko dar, weil keine Überprüfung der Zugriffsberechtigungen für den Abruf einer bestimmten Ressource aus der Datenbank vorgenommen wird. Jeder Benutzer, der auf diese Route zugreift, kann alle Elemente abrufen, unabhängig von den Zugriffsberechtigungen.

Listing 18: Negativbeispiel Broken Object Level Authorization

```
[HttpGet("items/{id}")]
public Task<IActionResult<Item>> GetItem(int id)
{
 var item = await _context.Items
        .Where(i => i.Id == id)
        .SingleOrDefaultAsync();
    return Ok(item ?? NotFound());
}
```

Das zweite Beispiel (Listing 19) verbessert die Sicherheit, da überprüft wird, ob der angemeldete Benutzer (current user) das Eigentümer-Attribut des Elements besitzt, bevor das Element zurückgegeben wird. Auf diese Weise wird sichergestellt, dass nur der Eigentümer des Elements darauf zugreifen kann und andere Benutzer keinen Zugriff darauf haben.

Listing 19: Positivbeispiel Broken Object Level Authorization

```

[HttpGet("items/{id}")]
public async Task<IActionResult<Item>> GetItem(int id)
{
    var currentUser = HttpContext.User;
    var item = await _context.Items
        .Where(i => i.Id == id && i.OwnerId == currentUser.FindFirstValue(
            ClaimTypes.NameIdentifier))
        .SingleOrDefaultAsync();

    return Ok(item ?? NotFound());
```

```
11 }
```

5.2.1.2 Excessive Data Exposure

> Looking forward to generic implementations, developers tend to expose all object properties without considering their individual sensitivity, relying on clients to perform the data filtering before displaying it to the user.

In diesem Beispiel (Listing 20) enthält der Response zum Abrufen eines Order-Objekts das Kundenobjekt als Property. Es mag Gründe für ein solches Datenmodell geben, in dem Fall werden aber ohne Einschränkung durch die direkte Rückgabe alle Informationen preisgegeben. Eine Erweiterung des Objektes um weitere Eigenschaften durch einen Entwickler würde über diese Route die neue Eigenschaft ungeprüft mit zurückgeben. Wenn ein Angreifer auf diesen Endpunkt zugreifen würde, könnte er diese sensiblen Informationen abrufen, obwohl sie für die Erfüllung der Anfrage nicht erforderlich sind.

Listing 20: Negativbeispiel Excessive Data Exposure

```
1 [HttpGet("order/{id}")]
2 public IActionResult<Order> GetOrder(int id)
3 {
4     var order = _context.Orders.Include(o => o.Customer).FirstOrDefault(o => o.Id
          == id);
5     return Ok(order);
6 }
7
8 public record Order(int Id, int Total, DateTime Date, int CustomerId)
```

In diesem Beispiel (Listing 21) gibt der API-Endpunkt nur die für die Erfüllung der Anforderung erforderlichen Informationen zurück (d. h. die Auftragskennung, den Gesamtbetrag und das Datum). Das Response Model entspricht nicht dem Rückgabeobjekt von _context.Orders und die notwendigen Properties werden auf den response gemappt. Das o.g. Problem, z.B. bei einer Erweiterung, tritt nicht auf und neue Property müsste explizit auch dem Response Model hinzugefügt werden und zusätzlich gemappt werden.

Listing 21: Positivbeispiel Excessive Data Exposure

```
1
2 [HttpGet("orders/{id}")]
3 public IActionResult<OrderResponse> GetOrder(int id)
4 {
5     var order = _context.Orders.FirstOrDefault(o => o.Id == id);
6
7     if (order == null)
8     {
9         return NotFound();
10    }
11
12    OrderResponse response = new {
13        Id = order.Id,
14        Total = order.Total,
15        Date = order.Date
16    };
17
18    return Ok(response);
19 }
20 public record Order(int Id, int Total, DateTime Date, int CustomerId)
21 public record OrderResponse(int Id, int Total, DateTime Date)
```

5.2.1.3 Broken Function Level Authorization

> Complex access control policies with different hierarchies, groups, and roles, and an unclear separation between administrative and regular functions, tend to lead to authorization flaws. By exploiting these issues, attackers gain access to other users' resources and/or administrative functions.

Die folgenden beiden Beispiele (Listings 22 und 23)unterschieden sich lediglich in der Annotation in der zweiten Methode, welche vorgibt, welche Rollen berechtigt sind, den Endpunkt überhaupt aufzurufen. Die Überprüfung innerhalb der Methode wäre bei Implementierung der Standardlibrary von .NET gar nicht nötig und ist hier nur zur Verdeutlichung eingefügt. Die erste Methode könnte hingegen über die entsprechende Route von jedem Nutzer aufgerufen werden und die entsprechenden Informationen über den zurückgegebenen User unabhängig der Autorisierung des Aufrufers eingesehen und verwendet werden.

Listing 22: Negativbeispiel Broken Function Level Authorization

```
1 [HttpGet("orders/{id}")]
2 public IActionResult<Order> GetOrder(int id)
3 {
4     Order order = _context.Orders.Find(id);
5     return Ok(order ?? NotFound());
6 }
```

Listing 23: Positivbeispiel Broken Function Level Authorization

```
1  [HttpGet("orders/{id}")]
2  [Authorize(Roles = "admin, manager")]
3  public IActionResult<Order> GetOrder(int id)
4  {
5      var currentUser = HttpContext.User;
6      if (!currentUser.IsInRole("admin") || !currentUser.IsInRole("manager"))
7      {
8          return Forbid();
9      }
10     Order order = _context.Orders.Find(id);
11     return Ok(order ?? NotFound());
12 }
```

5.2.1.4 Injection (hier SQL)

> Injection flaws, such as SQL, NoSQL, Command Injection, etc., occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's malicious data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

Das negative Beispiel (Listing 24) weist eine SQL-Injection-Vulnerability auf, da der Wert der Variable ïd"direkt in die SQL-Abfrage eingefügt (injected) wird. Ein Angreifer kann die Eingabe manipulieren, um die Abfrage zu verändern und unautorisierten Zugriff auf die Datenbank zu erlangen oder schädlichen Code einzufügen.

Listing 24: Negativbeispiel Injection

```
1 [HttpGet("users/{id}")]
2 public IActionResult<User> GetUser(string id)
3 {
4     string query = $"SELECT * FROM Users WHERE Id = '{id}'";
5     // ...
6 }
```

Im Gegensatz dazu verwendet das positive Beispiel in Listing **??** parameterisierte Abfragen und schützt damit gegen SQL-Injection-Attacken, da der Wert von ïdäls Parameter an die Abfrage übergeben wird, anstatt direkt in die Abfrage eingefügt zu werden. Als zusätzliche Sicherheitsmaßnahme wird statt einer id eine GUID verwendet, welche im richtigen Format vorliegen muss und es bei der Abfrage erschwert, id's zu "erraten" (was Security by Obscurity wäre - also für sich keinerlei Sicherheit bietet, in Kombination jedoch sinnvoll ist).

Listing 25: Positivbeispiel Injection

```
[HttpGet("users/{id}")]
public IActionResult<User> GetUser(Guid id)
{
    string query = "SELECT * FROM Users WHERE Id = @id";
    var command = new SqlCommand(query, connection);
    command.Parameters.AddWithValue("@id", id);
    // ...
}
```

5.2.1.5 Improper Assets Management

> APIs tend to expose more endpoints than traditional web applications, making proper and updated documentation highly important. Proper hosts and deployed API versions inventory also play an important role to mitigate issues such as deprecated API versions and exposed debug endpoints.

Wir greifen die Problematik der SQL-Injection auf - in diesem Beispiel (Listing 26) wird durch Parameterisierung versucht, einen SQL-Injection-Angriff zu verhindern. Allerdings wird im Falle eines Fehlers beim Ausführen des SQL-Statements nur eine allgemeine Fehlermeldung an den Client zurückgegeben. Es gibt keine weiteren Maßnahmen zur Überwachung oder Protokollierung des Vorfalls.

Listing 26: Negativbeispiel Improper Assets Management

```
[HttpGet("users/{id}")]
public IActionResult GetUser(string id)
{
    try
    {
        string query = "SELECT * FROM Users WHERE Id = @id";
        var command = new SqlCommand(query, connection);
        command.Parameters.AddWithValue("@id", id);
        // ...
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error occurred while getting user with ID {id}", id)
            ;
        return StatusCode(500);
    }
}
```

In positiven Beispiel (Listing 27) wird ebenfalls versucht, einen SQL-Injection-Angriff zu verhindern, indem ein parametrisiertes SQL-Statement verwendet wird. Wenn jedoch ein Fehler beim Ausführen des Statements auftritt, wird der Fehler sowohl protokolliert als auch an einen Remote-Logger gesendet, der in das mit einem Monitoring Tool verbundenen Sink schreibt. Je nach Incidence oder bei wiederholten Vorfällen kann dann entsprechend (automatisch) reagiert werden, um möglicherweise eine weitere Attacke zu verhindern (z.B. durch ein Blacklisting der IP).

Listing 27: Positivbeispiel Improper Assets Management

```
[HttpGet("users/{id}")]
public IActionResult<User> GetUser(string id)
{
```

```
4       try
5       {
6           string query = "SELECT * FROM Users WHERE Id = @id";
7           var command = new SqlCommand(query, connection);
8           command.Parameters.AddWithValue("@id", id);
9           // ...
10      }
11      catch (SqlException ex)
12      {
13          _logger.LogError(ex, "Error occurred while getting user with ID {id}", id)
                ;
14          _monitoringService.LogEvent(new EventLog {
15              EventType = "Security",
16              EventLevel = "Warning",
17              EventMessage = "Failed attempt to get user with ID " + id
18          });
19          return StatusCode(500);
20      }
21      catch (Exception ex)
22      {
23          _logger.LogError(ex, "Unknown error occurred while getting user with ID {
                id}", id);
24          _monitoringService.LogEvent(new EventLog {
25              EventType = "Common",
26              EventLevel = "Error",
27              EventMessage = "Unknown error occurred while getting user with ID " +
                    id
28          });
29          return StatusCode(500);
30      }
31  }
```

5.2.2 OAuth 2 und OIDC

Analysieren Sie die grundlegende Arbeitsweise von OAuth2

- Welche grundlegenden Rollen werden in OAuth2 unterscheiden?
- Welche Zusammenhänge bestehen zwischen OAuth2 und OIDC?

5.3 Praktische Anwendung von OAuth2

5.3.1 Testwerkzeuge

Verwenden Sie ein Testwerkzeug zur Nutzung von Web APIs und beschreiben Sie exakt die Schritte und Konfigurationen von mindestens 3 OAuth2 konformen Funktionsaufrufen (request), sie deren erhaltenen Antworten (response)

5.3.2 Implementierung

Gehen Sie auf die Möglichkeiten einer programmiertechnischen Einbindung von OAuth2 konformen Aufrufen innerhalb der Programmiersprache Java oder ggfs. auch JavaScript ein.

- Voraussetzungen dokumentieren (genutzte APIs)
- prototypische Verwendung mit Hilfe eines lauffähigen Quellcodefragments
- Test des entwickelten Prototypen, d.h. OAuth2 Zugriff auf Web-APIs

Literatur

Athey, Susan (2018), "The impact of machine learning on economics." In *The economics of artificial intelligence: An agenda*, 507–547, University of Chicago Press.

DevInsider (2022), "Was ist eine api." `https://www.dev-insider.de/was-ist-eine-api-a-583923`, zuletzt abgerufen: Mai 2023.