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

lalala[1] und sind insofern für komplexe sozialwissenschaftliche Fragestellungen aufgrund des vorhandenen Bias[2]
lalalala[3]

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

[1] Vgl. Athey (2018), S.5
[2] Vgl. Miceli et al. (2022), S.3ff
[3] Vgl. Korab (2021), online

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

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. Für jede der identifizierten APIs existiert eine eigene Dokumentation, welche gegebenenfalls auf GIT Hub oder eine andere Webpräsenz verweist. Innerhalb der Dokumentationen finden sich prägnante Beschreibungen und teilweise auch Verwendungsbeispiele für die jeweilige API.

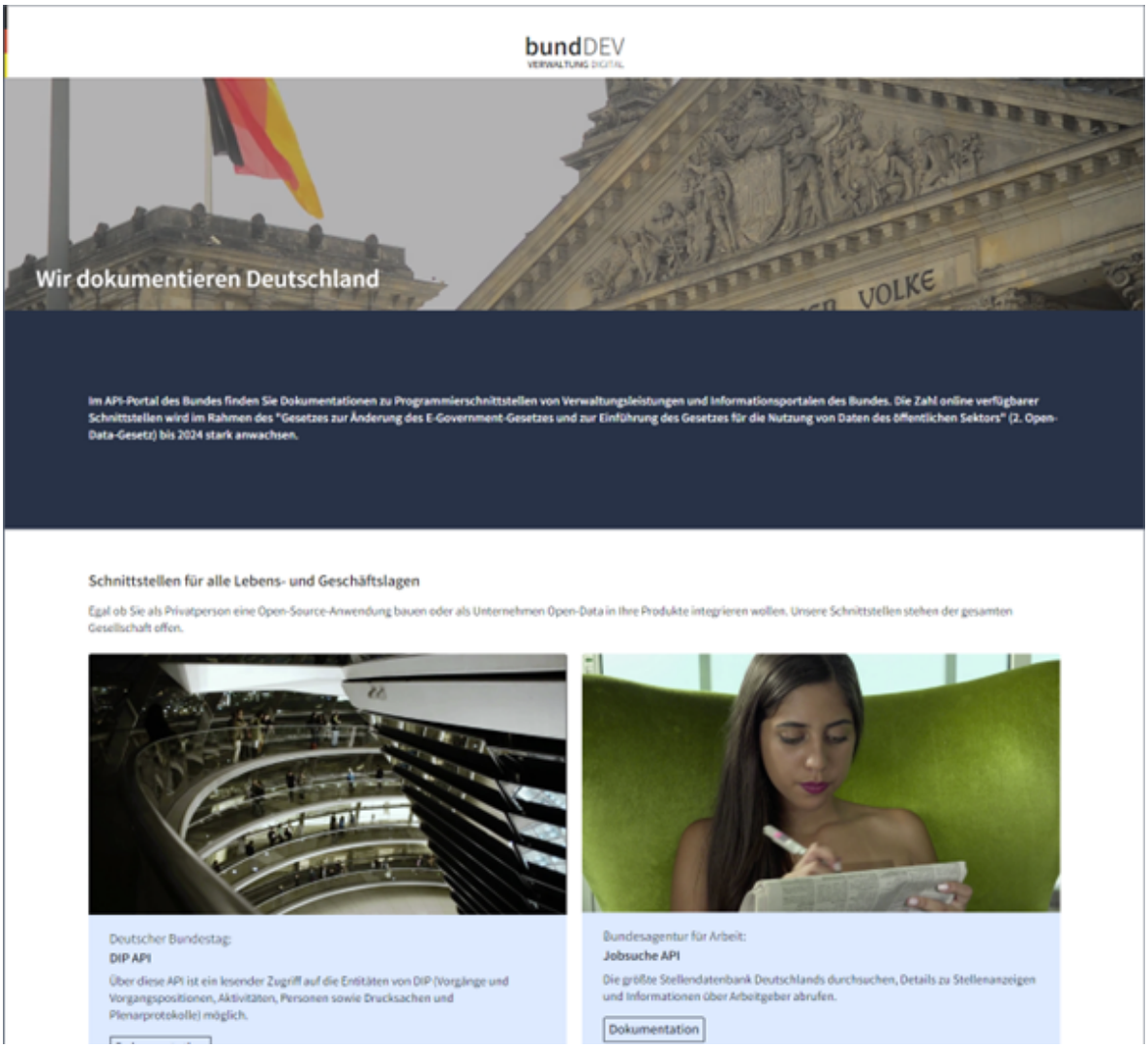

Abbildung 1: Ansicht der Startseite des Verzeichnisdienstes

In diesem Kontext ist die Integration individueller Application Programming Interfaces (APIs) nicht vorgesehen. Ebenso besteht keine Möglichkeit zur Erstellung eigener Code-Zweige sowie der Integration von Eigenprogrammierung. Lediglich die Meldung von Code-Problemen mittels GIT Hub-Funktionalität sowie das durchführen eines Pull Requestes ist gestattet. Bei einigen APIs ist es außerdem möglich, Probleme über alternative Kommunikationskanäle zu melden. Eine standardisierte Vorlage für das Reporting von Schwierigkeiten besteht nicht

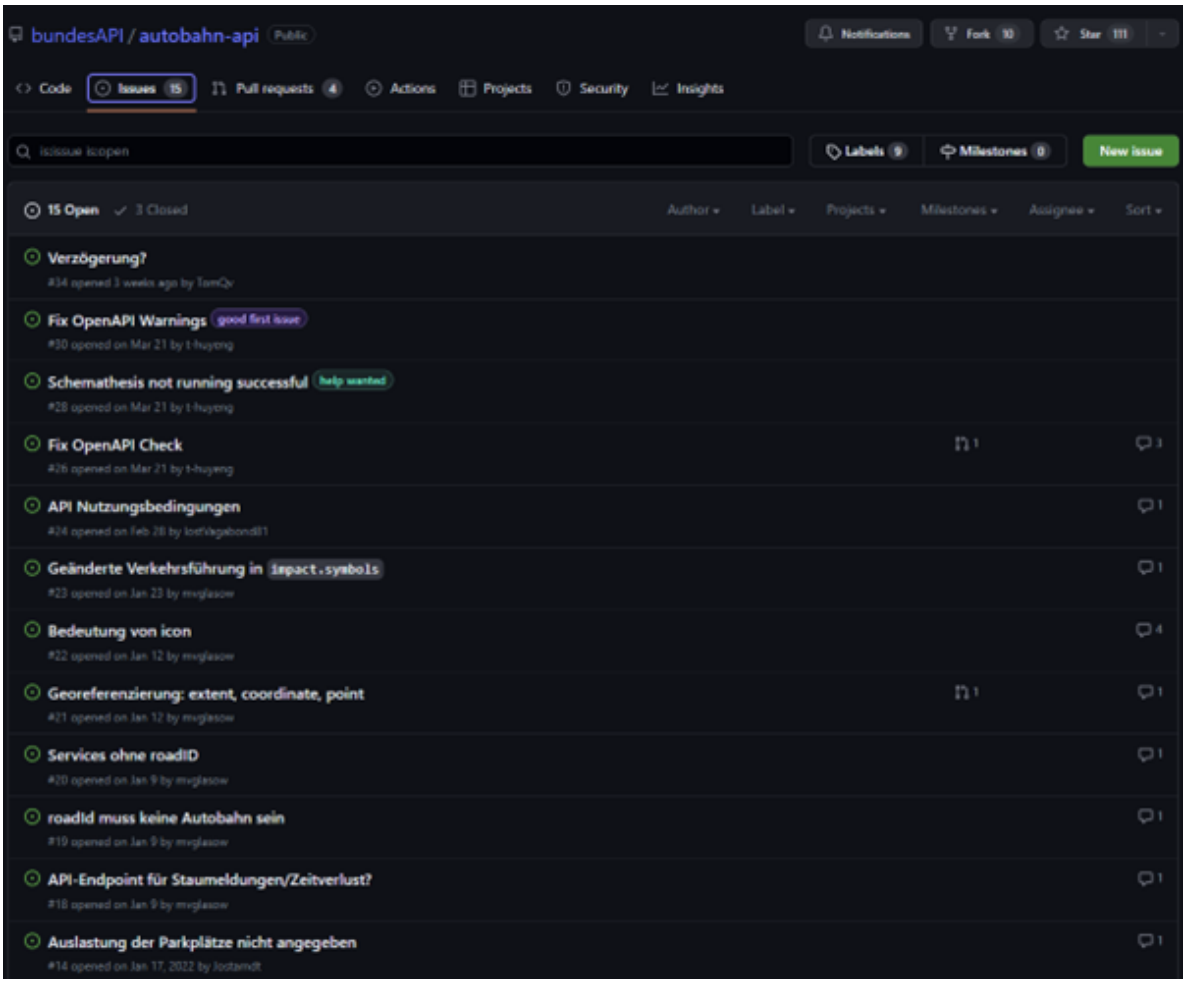

Abbildung 2: Gemeldete Probleme

Wie in Abb.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

Beide Schnittstellen sind Softwareschnittstellen, die sich in einzelnen Punkten und Funktionalitäten unterscheiden:
Eine API (Application Programming Interface) ist eine Art von Schnittstelle, die es Anwendungen ermöglicht, mit anderen Anwendungen, Systemen oder Diensten zu interagieren. Es stellt eine Sammlung von Methoden, Funktionen und Protokollen bereit, die von einer Anwendung aufgerufen werden können, um bestimmte Aktionen auszuführen oder Daten abzurufen. Eine API dient als Abstraktionsschicht, die es einer Anwendung ermöglicht, auf die Funktionalität eines anderen Systems zuzugreifen, ohne dass sie das interne Datenmodell oder die Implementierungsdetails kennen muss. APIs sind in der Regel gut dokumentiert und stellen eine klare Schnittstelle bereit, über die Anwendungen miteinander kommunizieren können.
Eine datenorientierte Schnittstelle ist eine Art von Schnittstelle, die es Anwendungen ermöglicht, auf strukturierte oder unstrukturierte Daten zuzugreifen, sie zu manipulieren und zu speichern. Im Gegensatz zu einer API, die eine Sammlung von Methoden und Funktionen bereitstellt, um bestimmte Aktio-

nen auszuführen oder Daten abzurufen, liegt der Schwerpunkt bei einer datenorientierten Schnittstelle auf dem direkten Zugriff auf die Daten selbst. Datenorientierte Schnittstellen können beispielsweise in Form von Datenbankzugriffen oder Dateisystemzugriffen bereitgestellt werden. Im Allgemeinen erfordern sie spezielle Kenntnisse und Fähigkeiten, um sie zu nutzen, und sind oft weniger benutzerfreundlich als APIs.

Kriterium	API-Schnittstelle	Datenschnittstelle
Flexibilität	Hoch APIs bieten in der Regel mehr Flexibilität, da sie in der Lage sind, unterschiedliche Arten von Anwendungen und Plattformen zu bedienen.	Niedrig Datenorientierte Schnittstellen sind in der Regel weniger flexibel, da sie sich auf spezifische Datenmodelle und -typen konzentrieren.
Sicherheit	Höher APIs bieten in der Regel mehr Sicherheitsfunktionen wie Autorisierung, Authentifizierung und Verschlüsselung.	Niedriger Datenorientierte Schnittstellen bieten in der Regel weniger Sicherheitsfunktionen, da sie sich auf den Zugriff auf Daten konzentrieren.
Kompatibilität	Hoch APIs sind in der Regel kompatibel mit einer Vielzahl von Plattformen, Sprachen und Systemen.	Niedriger Datenorientierte Schnittstellen sind in der Regel auf bestimmte Datenquellen oder Datenbanken beschränkt.
Geschwindigkeit	In der Regel Höher APIs bieten in der Regel schnellere Verarbeitungsgeschwindigkeiten, da sie oft auf leichte und effiziente Übertragung von Daten ausgelegt sind.	In der Regel niedriger Datenorientierte Schnittstellen sind in der Regel langsamer, da sie oft auf komplexe Datenstrukturen und Datenbankabfragen zugreifen müssen.
Verwendung Granularität	Für Entwickler und Programmierer Höhere Granularität APIs bieten in der Regel eine höhere Granularität, da sie in der Lage sind, einzelne Funktionen und Dienste aufzurufen.	Für Endbenutzer Geringere Granularität Datenorientierte Schnittstellen bieten in der Regel eine geringere Granularität, da sie sich auf den Zugriff auf bestimmte Datensätze oder Datenquellen konzentrieren.

Tabelle 1: Vergleich von API-Schnittstellen und Datenschnittstellen

Es ist zu beachten, dass die Unterschiede zwischen den beiden Typen von Schnittstellen nicht immer so klar abgegrenzt sind und dass einige APIs datenorientiert sein können und einige Datenorientierte Schnittstellen APIs-ähnliche Funktionen bieten können.

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. Übersicht: 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 ist ein wichtiger Faktor für die Offenheit einer API, da es Entwicklern ermöglicht, die API zu verstehen, anzupassen und zu erweitern. Wenn der Quellcode einer API offen zugänglich ist, können Entwickler die Funktionalität der API an ihre spezifischen Anforderungen anpassen und Fehler beheben, die in der API auftreten können. Offener Quellcode fördert auch die Zusammenarbeit und Innovation, indem er es anderen Entwicklern ermöglicht, den Code zu überprüfen und Verbesserungen und Erweiterungen vorzuschlagen. Dies kann dazu beitragen, die Qualität und Sicherheit der API zu verbessern.
   - Token/ Registrierung: Eine einfache Registrierung oder Token-Nutzung kann dazu beitragen, die API-Nutzung für die Benutzer zu vereinfachen. Anstatt komplizierte Authentifizierungsmethoden zu verwenden, können Benutzer einfach einen Token generieren oder sich registrieren, um Zugriff auf die API zu erhalten.
   - Zugänglichkeit: Eine offene API sollte für Entwickler leicht zugänglich sein. Die API sollte einfach zu finden und zu registrieren sein, und es sollte keine unnötigen Hürden geben, die die Nutzung der API einschränken.
   - Request Limit: Ein Request-Limit ist eine Einschränkung für die Anzahl der API-Anfragen, die ein Benutzer innerhalb eines bestimmten Zeitraums an den API-Server senden darf. Ein Request-Limit ist wichtig, um die API-Performance, Zuverlässigkeit und Sicherheit zu gewährleisten. Eine API kann nur eine begrenzte Anzahl von Anfragen gleichzeitig bearbeiten. Wenn zu viele Anfragen gleichzeitig eingehen, kann die API-Performance beeinträchtigt werden. Wenn zu viele Anfragen gleichzeitig eingehen und die API überlasten, kann dies zu Systemausfällen oder Verzögerungen bei der Verarbeitung von Anfragen führen. Einige Benutzer können versuchen, die API durch eine hohe Anzahl von Anfragen zu überlasten, um Schwachstellen auszunutzen oder unerwünschte Aktivitäten durchzuführen. Ein Request-Limit stellt sicher, dass alle Benutzer die gleiche Chance haben, die API zu nutzen.
   - Kontakt: Es ist wichtig, dass zu einer API ein Kontakt hinterlegt ist, damit Benutzer der API einen Ansprechpartner haben, falls sie Fragen oder Probleme haben. Wenn Benutzer auf Probleme bei der Nutzung der API stoßen, können sie sich an den hinterlegten Kontakt wenden, um Hilfe und Unterstützung zu erhalten. Ein Kontakt kann Feedback von den Benutzern der API sammeln und an den API-Anbieter weiterleiten. Dies kann dazu beitragen, die API zu verbessern, indem die Benutzeranforderungen und -bedürfnisse berücksichtigt werden. Ein Kontakt kann dazu beitragen, die Sicherheit der API zu erhöhen, indem er Benutzer über Sicherheitsbedrohungen informiert und sicherstellt, dass diese angemessen behandelt werden.

3. Qualität:

- Granularität: Die Granularität einer API bezieht sich auf die Anzahl und Komplexität der verfügbaren API-Endpunkte und der darin enthaltenen Funktionalität. Eine API mit zu hoher Granularität enthält möglicherweise zu viele spezialisierte Endpunkte, die für bestimmte Anwendungsfälle entwickelt wurden, während eine API mit zu niedriger Granularität möglicherweise nicht genügend Endpunkte enthält, um die erforderliche Funktionalität bereitzustellen. Es ist daher wichtig, dass die Granularität der API weder zu hoch noch zu niedrig ist.
- TLS: HTTPS bei einer API eingesetzt wird, um die Sicherheit und Integrität der übertragenen Daten zu gewährleisten. HTTPS ist eine Verschlüsselungsprotokoll, das auf dem HTTP-Protokoll basiert und eine sichere Verbindung zwischen einem Client und einem Server herstellt. TLS ist ein Protokoll, das die Datenverschlüsselung, Authentifizierung und Integritätssicherung während der Übertragung von Daten über ein Netzwerk ermöglicht.
- Statuscode: Es ist wichtig, dass alle Statuscodes für eine API angegeben sind, da sie den Client über den Zustand der Anfrage informieren und ihm ermöglichen, geeignete Maßnahmen zu ergreifen. Statuscodes sind dreistellige Zahlen, die von einem Server zurückgegeben werden, um den Erfolg oder Misserfolg einer Anfrage anzuzeigen.

4. Dokumentation:
   - Routen: Es ist wichtig, dass alle Routen einer API in der Dokumentation erwähnt sind, um den Entwicklern zu helfen, die API vollständig zu verstehen und korrekt zu nutzen.
   - Ressourcen: Wenn alle Ressourcen in der Dokumentation erwähnt werden, können Entwickler sicherstellen, dass sie alle verfügbaren Entitäten und Objekte der API verstehen und nutzen. Wenn eine Ressource in der Dokumentation fehlt, kann dies dazu führen, dass Entwickler fehlerhafte Anfragen senden oder unerwartete Ergebnisse erhalten.
   - Parameter:
5. Verfügbarkeit

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.

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

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

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, 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."[4]

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 Situation 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.
Benchmarks

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 2: Verwendete, externe Abhängigkeiten

Listing 1: 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 2-4 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 2: CLI Command zum Starten des Datenbankcontainers

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

Listing 3: 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 4: 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 5: 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 1) 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 6 zu sehen.

Listing 6: 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 7 installiert und ausgeführt werden.

Listing 7: 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.

4.2.2 Implementierung API

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

4.2.2.2.1 Client

4.2.2.2.2 REST API

4.2.2.3 Deployment

Für beide selbst entwickelten Services wird mit Hilfe von Dockerfiles (Listing 8) 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 8: Dockerfile für die Erstellung des REST-API Images

```
FROM rust:1.60.0-bullseye AS build
WORKDIR /app
COPY . .
RUN cargo build --release
RUN mkdir -p /app/lib
RUN cp -LR $(ldd ./target/release/rust-actix-surreal-rest-api | grep "=>" | cut -d
      ' ' -f 3) /app/lib

FROM scratch AS app
WORKDIR /app
COPY --from=build /app/lib /app/lib
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 9 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 9: 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

Listing 10: CLI Kommandos zur lokalen Installation der Datenbank für Windows Linux und macOS

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

Listing 11: CLI Kommando zur Übertragung der Daten aus der Datei in Listing 12

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

Listing 12: Ausschnitt der sql Setupdatei

```
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');
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');
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.

5.2 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.

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

Albora, Giambattista, Luciano Pietronero, Andrea Tacchella, and Andrea Zaccaria (2023), "Product progression: a machine learning approach to forecasting industrial upgrading." *Scientific Reports*, 13, 1481.

Angione, Claudio, Eric Silverman, and Elisabeth Yaneske (2022), "Using machine learning as a surrogate model for agent-based simulations." *PloS one*, 17, e0263150.

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

Athey, Susan and Guido W Imbens (2019), "Machine learning methods that economists should know about." *Annual Review of Economics*, 11, 685–725.

Basheer, Mohammed, Victor Nechifor, Alvaro Calzadilla, Claudia Ringler, David Hulme, and Julien J Harou (2022), "Balancing national economic policy outcomes for sustainable development." *Nature Communications*, 13, 5041.

Bertoletti, Alice, Jasmina Berbegal-Mirabent, and Tommaso Agasisti (2022), "Higher education systems and regional economic development in europe: A combined approach using econometric and machine learning methods." *Socio-Economic Planning Sciences*, 82, 101231.

Blogger, ML (2022), "Rl in economics." `https://medium.com/@mlblogging.k/rl-in-economics-794b43ffc995`, zuletzt abgerufen: März 2023.

Buckmann, Marcus, Andreas Joseph, and Helena Robertson (2022), "An interpretable machine learning workflow with an application to economic forecasting." Technical report, Bank of England.

Giglio, Stefano, Bryan T Kelly, and Dacheng Xiu (2021), "Factor models, machine learning, and asset pricing." *Machine Learning, and Asset Pricing (October 15, 2021)*.

Jomthanachai, Suriyan, Wai Peng Wong, and Khai Wah Khaw (2023), "An application of machine learning to logistics performance prediction: An economics attribute-based of collective instance." *Computational economics*, 1–52.

Korab, Petr (2021), "Use of machine learning in economic research: What the literature tells us." `https://towardsdatascience.com/use-of-machine-learning-in-economic-research-what-the-literature-tells-us-28b473f26043`, zuletzt abgerufen: März 2023.

Kumbure, Mahinda Mailagaha, Christoph Lohrmann, Pasi Luukka, and Jari Porras (2022), "Machine learning techniques and data for stock market forecasting: A literature review." *Expert Systems with Applications*, 116659.

Merler, Silvia (2018), "Machine learning and economics." `https://www.bruegel.org/blog-post/machine-learning-and-economics`, zuletzt abgerufen: März 2023.

Miceli, Milagros, Julian Posada, and Tianling Yang (2022), "Studying up machine learning data: Why talk about bias when we mean power?" *Proceedings of the ACM on Human-Computer Interaction*, 6, 1–14.