**Ein Bild, das Text, Schrift, Logo, Grafiken enthält.

KI-generierte Inhalte können fehlerhaft sein.**

**Fakultät Wirtschaft**

**Studiengang Wirtschaftsinformatik**

**Evaluation für Tools zur Service-Testung der**

**cortility gmbh im Rahmen der**

**Softwareentwicklung**

**Projektarbeit 1**

Im Rahmen der Prüfung zum Bachelor of Science (B.Sc.)

**Verfasser: Christopher El Alaoui  
Kurs: WWI22B1  
Partnerunternehmen: cortility gmbh  
Wissenschaftlicher Betreuer: Philipp Pohl  
Abgabedatum: XX.09.2025**

**Eidesstattliche Erklärung**

Ich versichere hiermit, dass ich die vorliegende Projektarbeit mit dem Thema: „Evaluation für Tools zur Service-Testung der cortility gmbh im Rahmen der Softwareentwicklung“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Diese Arbeit wurde bisher in gleicher oder ähnlicher Form oder auszugsweise noch keiner Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

|  |  |
| --- | --- |
| Karlsruhe, XY.05.2025 | Untersuch.\_\_Rift  (Unterschrift) |

Inhaltsverzeichnis

[Inhaltsverzeichnis ii](#_Toc202259505)

[Abkürzungsverzeichnis iv](#_Toc202259506)

[Abbildungsverzeichnis iv](#_Toc202259507)

[1 Einleitung 1](#_Toc202259508)

[1.1 Problemstellung und Motivation 1](#_Toc202259509)

[1.2 Ziel der Arbeit 1](#_Toc202259510)

[1.3 Abgrenzung der Arbeit 2](#_Toc202259511)

[1.4 Aufbau der Arbeit 2](#_Toc202259512)

[2 Theoretische Grundlagen 3](#_Toc202259513)

[2.1 Begriffserklärung: API, Webservices und Testautomatisierung 3](#_Toc202259514)

[2.2 Testarten in der API-Entwicklung (z. B. Integrationstests, Funktionstests, Regressionstests) 3](#_Toc202259515)

[2.3 Anforderungen an moderne API-Testtools 4](#_Toc202259516)

[2.4 Marktübersicht gängiger Tools 5](#_Toc202259517)

[2.5 Ist-Analyse: Einsatz von Postman bei cortility 7](#_Toc202259518)

[2.5.1 Architektur und Workflow 7](#_Toc202259519)

[2.6 Analyse der Anforderungen des Unternehmens 7](#_Toc202259520)

[2.6.1 Funktionale Anforderungen 7](#_Toc202259521)

[2.6.2 Nicht-funktionale Anforderungen 8](#_Toc202259522)

[3 Methodik 9](#_Toc202259523)

[3.1 Vorgehensweise der Toolauswahl 9](#_Toc202259524)

[3.2 Kriterienkatalog zur Bewertung 9](#_Toc202259525)

[3.2.1 Lizenzmodell 10](#_Toc202259526)

[3.2.2 Funktionsumfang 10](#_Toc202259527)

[3.2.3 Erweiterbarkeit 11](#_Toc202259528)

[3.2.4 Benutzerfreundlichkeit 12](#_Toc202259529)

[3.2.5 Teamfähigkeit 12](#_Toc202259530)

[3.2.6 Integration in Entwicklungsumgebungen (CI/CD) 13](#_Toc202259531)

[3.2.7 Support & Dokumentation 14](#_Toc202259532)

[3.3 SWOT-Analyse 14](#_Toc202259533)

[3.4 Nutzwertanalyse 16](#_Toc202259534)

[4 Evaluierung 17](#_Toc202259535)

[4.1 Vorstellung und Vergleich ausgewählter Tools 17](#_Toc202259536)

[4.1.1 Tool Rest Assured (Golmohammadi et al., 2024) 17](#_Toc202259537)

[4.1.2 Tool Postman (AG, 2022) 18](#_Toc202259538)

[4.1.3 Bruno 19](#_Toc202259539)

[4.2 Bewertung und Priorisierung der Tools 20](#_Toc202259540)

[4.2.1 Gewichtung der Kriterien 20](#_Toc202259541)

[4.2.2 Scoring der Shortlist 20](#_Toc202259542)

[4.3 Entscheidungsempfehlung 21](#_Toc202259543)

[5 Implementierung (optional) 23](#_Toc202259544)

[5.1 Pilotierung des empfohlenen Tools 23](#_Toc202259545)

[5.2 Integration in bestehende Entwicklungsprozesse 24](#_Toc202259546)

[5.3 Erste Ergebnisse und Rückmeldungen 24](#_Toc202259547)

[6 Fazit und Ausblick 24](#_Toc202259548)

[6.1 Zusammenfassung der Ergebnisse 24](#_Toc202259549)

[6.2 Reflexion der Vorgehensweise 24](#_Toc202259550)

[6.3 Ausblick: Weiterführende Maßnahmen und strategische Bedeutung 25](#_Toc202259551)

[Literaturverzeichnis 26](#_Toc202259552)

[Anhang 28](#_Toc202259553)

# Abkürzungsverzeichnis

ABAP = Advanced Business Application Programming

API = Application Programming Interface

CI/CD = Continuous Integration / Continuous Delivery

GUI = Graphical User Interface

IDE = Integrated Development Environment

JSON = JavaScript Object Notation

NWA = Nutzwertanalyse

RAP = RESTful Application Programming

SOAP = Simple Object Access Protocol

SWOT‐Analyse = Stärken, Schwächen, Chancen und Risiken

S/4HANA = SAP Business Suite 4 SAP HANA

UI = User Interface

XML = eXtensible Markup Language

# Abbildungsverzeichnis

# Einleitung

## Problemstellung und Motivation

Die cortility GmbH hat in den vergangenen Jahren verstärkt SAP-Lösungen im Umfeld des S/4 HANA-Einsatzes (SAP SE, 2025) implementiert. Mit dieser Systemumstellung geht ein Wandel in der Anwendungsentwicklung einher: ABAP-Entwickler publizieren und konsumieren Dienste zunehmend im RESTful ABAP Programming Model (SAP SE, 2025) (RAP). Traditionell wurden SOAP-basierte Webservices (Haase, 2021) genutzt, doch S/4 HANA begünstigt den Einsatz von REST-Services. In diesem Kontext gewinnt das Testen dieser Services an Bedeutung (Ehsan et al., 2022a), insbesondere um Funktionsfähigkeit, Performance und Regression zu prüfen. Bislang setzt cortility das Tool Postman (AG, 2022) zur API-Testung (AG, 2022) ein.

Aus diesen Gründen ist eine fundierte Neubewertung von Postman sowie eine systematische Marktanalyse alternativer Tools zur API-Testautomatisierung erforderlich. Ziel ist, ein Tool zu finden, das die speziellen Anforderungen von cortility im S/4 HANA/RAP-Umfeld bestmöglich abdeckt und langfristig Wartbarkeit, Verlässlichkeit und Effizienz im QA-Prozess steigert.

## Ziel der Arbeit

Diese Projektarbeit verfolgt das Ziel, den aktuellen Einsatz von Postman bei cortility zu analysieren, die Anforderungen von cortility an ein modernes API-Testtool im Kontext von S/4 HANA und RAP zu ermitteln und darauf aufbauend alternative Lösungen systematisch zu bewerten. Zudem wird eine Entscheidungsempfehlung ausgesprochen.

## Abgrenzung der Arbeit

Die vorliegende Arbeit befasst sich ausschließlich mit bestehenden Werkzeugen zur manuellen sowie teil- oder vollautomatisierten Testung von RESTful APIs und Webservices im Kontext von S/4 HANA und dem RAP-Modell. Dabei werden bestimmte Themenbereiche bewusst ausgeklammert. Nicht berücksichtigt werden detaillierte Analysen von Performance- und Lasttests, etwa mit Tools wie JMeter oder Gatling, sowie das Testen von SOAP-Webservices oder älteren IDoc-Schnittstellen. Die Entwicklung eigener Testwerkzeuge wird ebenso wenig behandelt wie die vollständige Implementierung von Testskripten oder -frameworks; stattdessen werden lediglich konzeptionelle Ansätze oder exemplarische Auszüge dargestellt. Auch rechtliche Aspekte wie detaillierte Lizenzprüfungen bleiben außen vor, wobei grundlegende Lizenzmodelle kurz skizziert werden. Ziel ist es, marktreife Lösungen zu identifizieren, die die Effizienz und Qualität von API-Tests nachhaltig verbessern können.

## Aufbau der Arbeit

Die Arbeit ist in mehrere thematisch aufeinander aufbauenden Kapitel gegliedert. Kapitel 2 vermittelt die theoretischen Grundlagen zu APIs, Webservices und Aspekten der Testautomatisierung. Kapitel 3 beschreibt die Methodik der Toolauswahl, wobei insbesondere der entwickelte Kriterienkatalog, die SWOT-Analyse (Schawel & Billing, 2011) sowie die Nutzwertanalyse (Dittmer, 1995) im Mittelpunkt stehen. Kapitel 4 bildet den Kern der Arbeit und umfasst die Evaluierung des aktuellen Postman-Einsatzes bei cortility, die Anforderungsanalyse, die Vorstellung und den Vergleich ausgewählter Testwerkzeuge sowie die abschließende Bewertung mit einer Entscheidungsempfehlung. Optional gibt Kapitel 5 einen Überblick über die Pilotierung und Integration des empfohlenen Tools in die bestehenden Entwicklungsprozesse. Kapitel 6 schließt die Arbeit mit einer Zusammenfassung der Ergebnisse und einem Ausblick auf mögliche weiterführende Maßnahmen ab.

# Theoretische Grundlagen

## Begriffserklärung: API, Webservices und Testautomatisierung

Ein Application Programming Interface (API) ist eine Schnittstelle, die es erlaubt Programmen, miteinander zu kommunizieren. Die Verbindung passiert auf Ebene des Quellcodes, nicht auf Binärebene.

Zwei häufig genutzte Ansätze für Webservices sind SOAP und REST. SOAP steht für „Simple Object Access Protocol“. Es basiert auf XML und legt Regeln für den Nachrichtenaustausch fest. Außerdem unterstützt es Sicherheitsfunktionen und Transaktionen. REST bedeutet „Representational State Transfer“. Es ist kein Protokoll, sondern ein Architekturstil. REST arbeitet mit Ressourcen und nutzt meist JSON oder XML. REST ist einfach, flexibel und gut für Microservices geeignet. Beim Testen von APIs wird oft auf Automatisierung gesetzt. Dabei führen Programme die Tests aus, nicht Menschen. Es gibt verschiedene Arten von Tests. Funktionstests prüfen, ob die Schnittstelle richtig reagiert – zum Beispiel mit dem passenden Statuscode oder dem korrekten Datenformat. Regressionstests sorgen dafür, dass neue Änderungen keine alten Funktionen zerstören. Integrationstests überprüfen, ob mehrere Systeme gut zusammenarbeiten.

## Testarten in der API-Entwicklung (z. B. Integrationstests, Funktionstests, Regressionstests)

**Unit-Tests:**

Fokus auf einzelnen Funktionen oder Methoden, meist in der Programmiersprache (z. B. ABAP Unit Tests).

**Komponententests:**

Testen einer abgeschlossenen Komponente (z. B. Business Object Service in ABAP) isoliert vom Gesamtsystem.

**Integrations- und Systemtests:**

Prüfung der Kommunikation zwischen Client und API-Service, inklusive Datenbankzugriffen und anderen Services

**End-to-End-Tests:**

Validierung von Geschäftsprozessen über Frontend- und Backend-Systeme hinweg (z. B. Fiori-App → OData Service → Datenbank).

**Last- und Performance-Tests:**

Messen von Antwortzeiten und Systemauslastung unter simuliertem Lastaufkommen (z. B. über JMeter), um Engpässe zu identifizieren. **Sicherheitstests:**

Validierung von Authentifizierung (OAuth 2.0, SAML), Autorisierung, Verschlüsselung und Schutz gegen typische Angriffsvektoren (z. B. Injection, Cross-Site Scripting).

## Anforderungen an moderne API-Testtools

Für die Evaluierung von Tools zur API-Testing sind mehrere Anforderungen zu berücksichtigen. Ein Kriterium ist die Unterstützung verschiedener Protokolle und Datenformat, sowie gängiger Authentifizierungsmechanismen. Im Hinblick auf den Funktionsumfang sollte das Tool die Erstellung, Verwaltung und Ausführung von Testfällen und -skripten ermöglichen. Wichtige Funktionen sind dabei Assertions zur Überprüfung von Statuscodes, Headern, JSON-Schemas oder Antwortzeiten sowie die Unterstützung von Data-Driven Testing mit parametrisierten Eingaben aus externen Quellen wie Datenbanken. Die Erweiterbarkeit und Integrationsfähigkeit ist ein weiterer Aspekt. Tools sollten Skriptsprachen unterstützen, um sich nahtlos in CI/CD-Pipelines sowie Versionskontrollsysteme einbinden lassen. Auch die Benutzerfreundlichkeit spielt eine Rolle. Eine intuitive Benutzeroberfläche oder eine gut dokumentierte Kommandozeilenschnittstelle, kollaborative Funktionen wie Teamworkspaces, Rechteverwaltung und Reporting, sowie Möglichkeiten zur automatisierten Dokumentation sind hier von Bedeutung. Zur Automatisierung gehören Funktionen zur Zeit- oder ereignisgesteuerten Testausführung, etwa bei nächtlichen Builds oder bei jeder Codeänderung. Ergänzend sind E-Mail-Benachrichtigungen, Dashboards und eine strukturierte Ergebnisarchivierung wünschenswert. Ebenso relevant ist das Lizenzmodell: Die Unterscheidung zwischen Open-Source- und kommerziellen Lösungen, nutzungsabhängigen Abrechnungsmodellen (z. B. pro Nutzer, pro Server oder Pay-per-Use) sowie der Umfang der im Standard enthaltenen Funktionen im Vergleich zu kostenpflichtigen Erweiterungen sollte transparent sein. Abschließend ist auch die Verfügbarkeit von Support und Dokumentation entscheidend. Hierzu zählen Trainingsmaterialien, Tutorials, aktive Nutzerforen oder kommerzieller Support sowie regelmäßige Updates, planbare Release-Zyklen und Sicherheitsaktualisierungen.

## Marktübersicht gängiger Tools

Am Markt existiert eine Vielzahl von Tools zur API-Testautomatisierung. Nachfolgend einige exemplarische Tools (nicht abschließend):

**Postman**

* Ursprünglich als reiner REST-Client konzipiert, inzwischen mit umfangreicher Test- und Automatisierungsfunktionen.

**SoapUI / ReadyAPI (SmartBear)**

* Schwerpunkt SOAP, inzwischen auch REST; umfangreiche Funktionalitäten, kommerzielles Lizenzmodell.

**REST Assured**

* Java-basiertes Testframework; ideal für Entwickler, die in Java schreiben und Tests als Code pflegen möchten.

**Katalon Studio**

* Kombination aus Web-UI- und API-Testautomatisierung; bietet GUI-Editor und Testskript-Schnittstelle.

**Insomnia**

* Open-Source REST-Client mit Plugin-Ökosystem, zielt auf Entwickler ab; Tests lassen sich über YAML/JSON definieren.

**Hoppscotch**

* Leichter, webbasierter REST-Client; weniger umfangreich in der Automatisierung, eher für schnelle Explorations-Tests.

**Parasoft SOAtest**

* Enterprise-Lösung mit Fokus auf komplette Testautomatisierung, einschließlich Sicherheits- und Lasttests.

**Newman**

* CLI-Runner für Postman Collections, ideal in CI/CD-Umgebungen.

**Tavern (Python)**

* Framework für API-Testing in YAML, integriert gut in Python-basierte Projekte.

**Karate (Java/Gherkin)**

* Bietet BDD-Stil (Behavior-Driven Development) für API-Tests; integriert Testdatenmanagement und Assertions.

**Bruno**

* Bruno ist ein API-Testtool, das vollständig offline arbeitet und eine vollautomatisierte Testausführung lokal ermöglicht.

Die Auswahl beschränkt sich auf Tools, die RESTful Webservices vollumfänglich unterstützen, Skript- oder Code-basierte Automatisierung erlauben und über eine gewisse Teamfunktionalität verfügen.

## Ist-Analyse: Einsatz von Postman bei cortility

### Architektur und Workflow

Aktuell nutzen die DMS-Abteilung bei cortility die Free Version von Postman nach mehreren Gesprächen mit deren Entwickler Team hat sich heraus kristallisiert das es nur für vereinzelte calls genutzt wird und zudem direkt in der Entwicklungsumgebung HTTP-Requests gemacht werden.

## Analyse der Anforderungen des Unternehmens

Auf Basis der Ist-Analyse und der Gespräche mit ABAP-Entwicklern sowie QA-Mitarbeitern ergeben sich folgende Anforderungen:

### Funktionale Anforderungen

Die Lösung bietet umfassende Unterstützung für RESTful- und RAP-Services, einschließlich vollständiger Kompatibilität mit OData V4 (JSON), der Nutzung von Query Options sowie der Verarbeitung von Batch-Requests. Sicherheitsmechanismen wie Keycloak oder der OAuth 2.0-Flow lassen sich automatisieren, inklusive der Generierung und Erneuerung von Tokens. Für das Testskripting wird auf Skriptsprachen gesetzt, die Entwicklern bereits vertraut sind – darunter JavaScript, Python und Groovy. Testfälle können als Code verwaltet und in Git-Repositories versioniert werden. Dies sollte eine nahtlose Integration in bestehende Entwicklungsprozesse ermöglicht. Data-Driven Testing wird durch die einfache Einbindung externer Datenquellen wie CSV-Dateien, Excel-Tabellen, JSON-Daten oder Datenbankabfragen unterstützt. Mocking und Service-Virtualisierung erlauben die Erzeugung simulierter Service-Antworten, insbesondere zur Entkopplung von abhängigen Altsystemen. Darüber hinaus kann das Verhalten von Endpunkten temporär angepasst werden, um verschiedene Testszenarien abzubilden. Automatisierte Testreports in Formaten wie HTML, JUnit/XML oder Allure liefern detaillierte Auswertungen und Trendanalysen. Ein integriertes Dashboard ermöglicht das Tracking von Fehlerraten sowie die Analyse von Performance-Metriken. Für die CI/CD-Integration stehen native Plugins oder Docker-Support zur Verfügung, um Tests direkt in Azure DevOps, GitLab CI/CD oder Jenkins einzubinden. Zusätzlich können Tests über CLI-Befehle in Build-Pipelines ausgeführt werden. Abgerundet wird das System durch die Möglichkeit, API-Dokumentationen automatisch zu generieren – entweder auf Basis von Tests oder vorhandenen Schemas. Änderungen an Services führen automatisch zur Aktualisierung der Dokumentation, was die Wartung und Transparenz deutlich verbessert.

### Nicht-funktionale Anforderungen

Ein zentrales Auswahlkriterium für geeignete Tools ist die Benutzerfreundlichkeit und der geringe Einarbeitungsaufwand. Eine einsteigerfreundliche grafische Benutzeroberfläche ermöglicht es auch Testern mit begrenzter Programmiererfahrung, effizient zu arbeiten. Ergänzt wird dies durch umfangreiche Tutorials und Schulungsunterlagen, die den Einstieg erleichtern und die Selbstlernkurve verkürzen.

Auch die Teamfähigkeit und Kollaborationsmöglichkeiten spielen eine wichtige Rolle. Funktionen wie ein integriertes Benutzer- und Rechtemanagement sowie definierte Freigabeprozesse fördern die Zusammenarbeit im Team. Die Möglichkeit zur Versionierung auf Projektebene – inklusive Branching und Rollback – sorgt für Nachvollziehbarkeit und Flexibilität bei der Testentwicklung.

Im Bereich Sicherheit und Compliance ist die Unterstützung moderner Sicherheitsstandards wie TLS/SSL-Verschlüsselung sowie die Kompatibilität mit Unternehmens-Proxys essenziell. Darüber hinaus ist die Einhaltung geltender Datenschutzvorgaben, insbesondere der DSGVO, ein Muss für den produktiven Einsatz in Unternehmensumgebungen.

Auch wirtschaftliche Aspekte fließen in die Toolauswahl ein. Das Lizenzmodell sollte sich an einem Budgetrahmen von maximal 10.000 EUR pro Jahr orientieren. Idealerweise erlaubt das Modell einen Einstieg mit einer minimalen Anzahl an Lizenzen und bietet die Möglichkeit, bei wachsendem Bedarf schrittweise zu skalieren.

# Methodik

## Vorgehensweise der Toolauswahl

Die Auswahl geeigneter Testtools erfolgt in mehreren Phasen (vgl. Abbildung 3.1):

Definition des Kriterienkatalogs, basierend auf den Anforderungen aus Kapitel 2.3 werden Bewertungskriterien formuliert und gewichtet. Darauf folgt eine Erstellung einer Identifikation aller am Markt verfügbaren Tools, die für die Testautomatisierung von REST-APIs infrage kommen. Die Erste Vorauswahl (Shortlist) aus der Longlist werden 3–5 Tools ausgewählt, die am besten zu den grundsätzlichen Anforderungen passen (Budget, Infrastrukturanforderungen, Technologie-Stack). Darauf folgt eine SWOT-Analyse (Schawel & Billing, 2011) der Shortlist, für jedes ausgewählte Tool werden Stärken, Schwächen, Chancen und Risiken erarbeitet.

Anhand des Kriterienkatalogs und der Gewichtung erfolgt eine numerische Bewertung der Shortlist-Tools in Form von einer Nutzwertanalyse (Dittmer, 1995) (NWA).

Basierend auf der NWA wird eine Entscheidungsempfehlung abgegeben.

## Kriterienkatalog zur Bewertung

Der entwickelte Kriterienkatalog berücksichtigt sowohl funktionale als auch nicht-funktionale Aspekte bei der Bewertung von API-Testing-Tools. Für die durchzuführende Nutzwertanalyse werden alle Kriterien mit entsprechenden Gewichtungen und Punkteskalen versehen, um eine objektive und nachvollziehbare Bewertung zu ermöglichen. Die systematische Strukturierung erfolgt in verschiedene Kriterienblöcke, die jeweils spezifische Anforderungen und Eigenschaften der Tools adressieren.

### Lizenzmodell

**Ko** Das Lizenzmodell stellt einen entscheidenden Faktor bei der Tool-Auswahl dar, da es sowohl die kurzfristigen als auch langfristigen Kosten maßgeblich beeinflusst. Die Kostenstruktur unterscheidet sich fundamental zwischen verschiedenen Ansätzen: Während einige Anbieter auf Einmalzahlungen setzen, bevorzugen andere Abo-Modelle oder nutzungsbasierte Preisstrukturen. Diese Unterschiede haben direkten Einfluss auf die Budgetplanung und die Total Cost of Ownership.

Ein weiterer wichtiger Aspekt ist die Unterscheidung zwischen Freemium- und Enterprise-Angeboten. Viele Tools bieten Community-Editionen mit grundlegenden Funktionen kostenlos an, während erweiterte Features und professioneller Support nur in kostenpflichtigen Enterprise-Lizenzen verfügbar sind. Diese Strategie ermöglicht es Unternehmen, Tools zunächst zu evaluieren, bevor sie sich für eine kommerzielle Lizenz entscheiden.

Die Skalierbarkeit des Lizenzmodells spielt besonders in wachsenden Organisationen eine zentrale Rolle. Hierbei muss die Preisentwicklung bei steigender Nutzeranzahl oder bei der Durchführung paralleler Testläufe berücksichtigt werden. Transparenz im Lizenzmanagement und die Klarheit der Vertragsbedingungen sind ebenfalls entscheidende Faktoren, da komplexe Lizenzstrukturen zu unerwarteten Kosten und administrativem Aufwand führen können

### Funktionsumfang

Der Funktionsumfang bildet das Herzstück jeder Tool-Bewertung, da er direkt bestimmt, welche Testszenarien abgedeckt werden können. Der Protokoll-Support ist dabei von grundlegender Bedeutung und sollte moderne Standards wie RESTful APIs mit JSON- und XML-Unterstützung, SOAP-Services mit WSDL-Integration sowie GraphQL-APIs umfassen, um verschiedene Architekturansätze zu unterstützen.

Die Verfügbarkeit verschiedener Testskripting-Möglichkeiten erweitert die Flexibilität erheblich. Tools, die Skriptsprachen wie JavaScript, Groovy oder Python unterstützen, ermöglichen es, komplexe Testlogik zu implementieren und dynamische Testszenarios zu erstellen. Dies ist besonders wichtig für die Automatisierung anspruchsvoller Testfälle.

Umfangreiche Assertions- und Validierungsmöglichkeiten sind unerlässlich für die Qualitätssicherung. Hierzu gehören Bibliotheken zur Überprüfung von HTTP-Statuscodes, Header-Informationen und JSON-Schema-Validierungen. Data-Driven Testing erweitert die Testabdeckung durch die Möglichkeit zur Parametrisierung mit externen Datenquellen wie CSV-Dateien, JSON-Strukturen oder Datenbankverbindungen.

Mocking- und Stubbing-Funktionalitäten sind besonders in komplexen Systemlandschaften von Bedeutung, da sie es ermöglichen, Abhängigkeiten zu simulieren und Tests auch dann durchzuführen, wenn externe Services nicht verfügbar sind. Schließlich sollten umfangreiche und anpassbare Test-Reporting-Funktionen mit Dashboards und Trendanalysen zur Verfügung stehen, um Testergebnisse effektiv zu kommunizieren und langfristige Qualitätstrends zu verfolgen.

### Erweiterbarkeit

Die Erweiterbarkeit eines Tools bestimmt maßgeblich dessen Zukunftsfähigkeit und Anpassungsmöglichkeiten an spezifische Anforderungen. Ein reichhaltiges Plugin-Ökosystem mit Community- oder kommerziellen Erweiterungen zeigt die Lebendigkeit der Tool-Community und bietet Zugang zu spezialisierten Funktionen, die nicht im Kern-Tool enthalten sind.

Die Verfügbarkeit von APIs für das Testtool selbst, beispielsweise durch RESTful-APIs oder CLI-Befehle, ermöglicht die programmatische Steuerung und Integration in bestehende Workflows. Dies ist besonders wichtig für die Automatisierung und die Anbindung an übergeordnete Systeme.

Integrationspunkte zu Testmanagement-Systemen wie Jira oder TestRail schaffen nahtlose Workflows zwischen Testplanung, -durchführung und -dokumentation. Skripting-Hooks für Custom Pre- und Post-Request-Scripts sowie Event-Handler bieten zusätzliche Flexibilität für komplexe Testszenarien und ermöglichen es, spezifische Geschäftslogik in den Testablauf zu integrieren.

### Benutzerfreundlichkeit

Die Benutzerfreundlichkeit hat direkten Einfluss auf die Produktivität und Akzeptanz des Tools im Team. Eine intuitive Benutzeroberfläche mitdie Ent einfacher Navigation und Drag-&-Drop-Funktionen reduziert die Einarbeitungszeit und minimiert Bedienungsfehler.

Die Lernkurve und die Qualität der verfügbaren Dokumentation bestimmen, wie schnell neue Teammitglieder produktiv werden können. Hochwertige Tutorials, Beispiele und eine umfassende Dokumentation sind daher essenziell für den Projekterfolg.

Barrierefreiheit umfasst sowohl die Unterstützung verschiedener Betriebssysteme wie Windows, macOS und Linux als auch die Entscheidung zwischen webbasierten und Desktop-Anwendungen. Plattformunabhängigkeit gewährleistet, dass alle Teammitglieder unabhängig von ihrer technischen Umgebung effektiv arbeiten können.

### Teamfähigkeit

Die Teamfähigkeit eines API-Testing-Tools manifestiert sich in verschiedenen technischen und organisatorischen Dimensionen, die für erfolgreiche Kollaboration unerlässlich sind. Zentrale Bedeutung kommt dabei der technischen Infrastruktur für die Zusammenarbeit zu: Gemeinsame Arbeitsbereiche bilden die Grundlage für koordinierte Teamarbeit, während Versionierungssysteme mit Branching-Mechanismen und Rollback-Funktionen gewährleisten, dass parallel durchgeführte Änderungen kontrolliert zusammengeführt werden können. Eine granulare Rechteverwaltung ergänzt diese technischen Möglichkeiten durch die Definition spezifischer Zugriffsberechtigungen für verschiedene Teammitglieder.

Auf organisatorischer Ebene unterstützen strukturierte Arbeitsabläufe die Projektorganisation erheblich. Die gezielte Aufgabenzuweisung einzelner Testfälle an spezifische Nutzer schafft klare Verantwortlichkeiten und verhindert Doppelarbeit. Gleichzeitig ermöglichen individualisierbare Reporting- und Benachrichtigungssysteme eine bedarfsgerechte Informationsverteilung: Verschiedene Rechteebenen für die Ergebnisdarstellung stellen sicher, dass jeder Projektbeteiligte Zugang zu den für ihn relevanten Informationen erhält, während Alarmfunktionen über E-Mail oder Chat-Integrationen zeitkritische Ereignisse proaktiv kommunizieren.

Die Nachvollziehbarkeit aller Aktivitäten durch umfassende Audit-Logs bildet schließlich das Fundament für Qualitätssicherung und Compliance. Diese Dokumentation von Änderungen und Testergebnissen erweist sich besonders in regulierten Umgebungen als unverzichtbar und unterstützt zudem effektive Fehleranalysen durch die lückenlose Rekonstruktion von Ereignisfolgen.

### Integration in Entwicklungsumgebungen (CI/CD)

. Die nahtlose Integration in bestehende Entwicklungs- und Deployment-Pipelines ist für moderne Softwareentwicklung unerlässlich. CLI-Support ermöglicht die Ausführung von Testläufen über die Kommandozeile, wie beispielsweise Newman für Postman, was die Automatisierung in Skripten und Build-Prozessen ermöglicht.

Verfügbare Plugins für gängige CI/CD-Systeme wie Jenkins, GitLab CI/CD, Azure DevOps, Bamboo oder TeamCity reduzieren den Integrationsaufwand erheblich und gewährleisten eine reibungslose Einbindung in bestehende Toolchains.

Die Containerisierung durch verfügbare Docker-Images oder Kubernetes-Charts unterstützt moderne, cloudnative Deployment-Strategien und ermöglicht eine konsistente Ausführungsumgebung über verschiedene Entwicklungsstadien hinweg. Die automatische Testausführung als integraler Bestandteil von Build-Pipelines und Bereitstellungsprozessen gewährleistet kontinuierliche Qualitätskontrolle und frühzeitige Fehlererkennung.

### Support & Dokumentation

Die Qualität des Supports und der Dokumentation kann entscheidend für den Projekterfolg sein, besonders bei der Lösung komplexer Probleme oder der Implementierung spezieller Anforderungen. Community-Support durch Online-Foren, Stack Overflow-Diskussionen und GitHub-Issues bietet oft schnelle Hilfe bei häufigen Problemen und zeigt die Aktivität der Nutzergemeinschaft.

Kommerzieller Support mit SLA-Garantien, direkten Ansprechpartnern und professionellen Ticket-Systemen ist besonders für unternehmenskritische Anwendungen wichtig, da er planbare Reaktionszeiten und professionelle Problemlösung gewährleistet.

Die Dokumentationsqualität, einschließlich vollständiger Referenzdokumente, praktischer Tutorials und einer umfassenden Knowledge Base, beeinflusst direkt die Effizienz bei der Tool-Nutzung und Problemlösung. Eine transparente Release-Strategie mit regelmäßigen Updates, klaren Bugfix-Zyklen und einer öffentlichen Roadmap schafft Vertrauen in die Zukunftsfähigkeit des Tools und ermöglicht eine bessere Projektplanung.

## SWOT-Analyse

Die SWOT-Analyse (Schawel & Billing, 2011) stellt ein bewährtes Instrument zur strukturierten Bewertung dar, indem sie sowohl interne als auch externe Faktoren für jedes untersuchte Tool systematisch erfasst und bewertet. Diese Methodik ermöglicht eine ganzheitliche Betrachtung, die über reine Funktionsvergleiche hinausgeht.

Stärken (Strengths) repräsentieren die internen positiven Faktoren eines Tools. Hierzu zählen beispielsweise ein umfassender Funktionsumfang, der verschiedene Testszenarien abdeckt, eine intuitive Bedienung, die die Produktivität steigert, oder ein großes und aktives Ökosystem, das kontinuierliche Weiterentwicklung und Community-Support gewährleistet.

Schwächen (Weaknesses) identifizieren interne Nachteile, die die Effektivität oder Eignung eines Tools einschränken können. Dies können Defizite in der Benutzerfreundlichkeit sein, die die Einarbeitungszeit verlängern, oder fehlende Funktionsumfänge, die zusätzliche Tools oder Workarounds erforderlich machen.

Chancen (Opportunities) beschreiben externe Möglichkeiten, die sich positiv auf die Tool-Nutzung auswirken können. Beispiele hierfür sind eine steigende Akzeptanz im Enterprise-Umfeld, die zu besserer Unterstützung und Weiterentwicklung führt, oder Integrationsmöglichkeiten in bestehende System-Stacks, die Synergieeffekte schaffen.

Risiken (Threats) umfassen externe Bedrohungen, die den langfristigen Nutzen eines Tools gefährden könnten. Dazu gehören technologische Veralterung durch mangelnde Innovation, potenzielle Sicherheitslücken, die das Vertrauen untergraben, oder eine fehlende Weiterentwicklung aufgrund mangelnder Ressourcen oder strategischer Neuausrichtung des Anbieters.

Die Ergebnisse der SWOT-Analyse werden für jedes Tool in einem übersichtlichen Kurzprofil zusammengefasst, das eine schnelle Orientierung über die wesentlichen Vor- und Nachteile sowie Chancen und Risiken ermöglicht.

|  |  |  |  |
| --- | --- | --- | --- |
| Kategorie | Postman | Bruno | Rest Assured |
| Stärken | Intuittive Oberfläche große Comunity | Open Source, Git-integrierbar, offlinefähig | Java-Integration, CI/CD |
| Schwächen | Eingeschränkte Automatisierung, teils kostenpflichtig | Wenige Enterprise-Features, junge Community | Keine GUI, hohe Einstiegshürde |
| Chancen | Erweiterbar durch Plugins, große Nutzerbasis | Entwicklerfokus, zunehmende Community-Unterstützung | DevOps-orientiert, in Java-Teams beliebt |
| Risiken | Kommerzielle Features limitieren freie Nutzung | Abhängigkeit von Community, neue Tools entstehen schnell | Wartungslast bei komplexen Setups |

## Nutzwertanalyse

Die Nutzwertanalyse (Dittmer, 1995) bildet den quantitativen Kern der Bewertungsmethodik, indem sie die zuvor definierten Kriterien in messbare Werte überführt und eine objektive Vergleichsbasis schafft. Das systematische Vorgehen gliedert sich in zwei wesentliche Schritte.

Die Gewichtung der Kriterien erfolgt durch die Zuweisung von Relevanzgewichten, beispielsweise auf einer Skala von 1 bis 5, an jedes in Kapitel 3.2 definierte Kriterium. Diese Gewichtung spiegelt die spezifische Wichtigkeit des Kriteriums im S/4 HANA-Umfeld wider und berücksichtigt dabei sowohl technische Anforderungen als auch geschäftliche Prioritäten. So können beispielsweise Integration und Skalierbarkeit höher gewichtet werden als reine Feature-Vielfalt, wenn diese für den Projektkontext kritischer sind.

Das Scoring-Verfahren weist jedem Tool für jedes Kriterium eine Punktzahl zu, typischerweise auf einer Skala von 1 bis 10. Diese Bewertung basiert auf objektiven Betrachtungen wie konkreten Lizenzpreismodellen, messbarer Feature-Abdeckung und dokumentiertem Nutzerfeedback aus verschiedenen Quellen. Die Kombination aus Gewichtung und Scoring ermöglicht es, einen Gesamtscore für jedes Tool zu berechnen, der als fundierte Entscheidungsgrundlage für die Tool-Auswahl dient.

**Berechnung des Nutzwerts**

= Gewichtung des Kriteriums iii (wie wichtig ist das Kriterium?)

= Bewertung der Alternative in Bezug auf Kriterium iii (z. B. Skala von 1–10)

= Anzahl der Kriterien

**Rangfolge**

* Auf Basis der summierten Nutzwerte werden die Tools in eine Rangfolge gebracht.

Die NWA liefert eine transparente, nachvollziehbare Entscheidungsgrundlage, die qualitative Einschätzungen quantifiziert und eine datenbasierte Auswahl ermöglicht.

# Evaluierung

## Vorstellung und Vergleich ausgewählter Tools

### Tool Rest Assured (Golmohammadi et al., 2024)

Rest Assured ist ein Java-basiertes Framework zur Testautomatisierung von REST-APIs. Es ist besonders leistungsstark in CI/CD-Pipelines und bietet hohe Flexibilität für Entwickler im Bereich der Java Entwicklung bei cortlility. Allerdings fehlt eine grafische Benutzeroberfläche, was es für nicht-technische Tester ungeeignet macht. Es eignet sich vor allem für Organisationen mit starkem Java-Fokus und Automatisierungsbedarf.

|  |  |
| --- | --- |
| **Kategorie** | **Beschreibung** |
| Stärken | Sehr gute Integration in Java-Testumgebungen, ideal für automatisierte Tests im CI/CD-Kontext. |
| Schwächen | Hohe Einstiegshürde, nur für REST, keine GUI. |
| Chancen | Flexible Automatisierungsmöglichkeiten im DevOps-Umfeld. |
| Risiken | Nicht geeignet für nicht-technische Nutzer. |

### Tool Postman (AG, 2022)

Postman ist ein etabliertes Tool für API-Tests. Es bietet eine grafische Benutzeroberfläche und viele Funktionen. Die Bedienung ist intuitiv. APIs lassen

sich umfassend dokumentieren. Testskripte können mit JavaScript erstellt werden. Teams können gemeinsam an Projekten arbeiten. Postman unterstützt REST-APIs vollständig. SOAP-APIs werden nur eingeschränkt unterstützt. Das Tool lässt sich in CI/CD-Pipelines integrieren. Mit Postman Collections können Tests organisiert und wiederverwendet werden. Nachteile sind die begrenzte Erweiterbarkeit und das Fehlen nativer SOAP-Unterstützung.

|  |  |
| --- | --- |
| **Kategorie** | **Beschreibung** |
| Stärken | Sehr benutzerfreundliche Oberfläche, große Community, gute Dokumentation, native Unterstützung für REST-Services. |
| Schwächen | Keine native Unterstützung für SOAP, eingeschränkte Automatisierungsmöglichkeiten ohne Zusatztools. |
| Chancen | Erweiterung durch Plugins, Integration in CI/CD-Pipelines möglich. |
| Risiken | Kostenpflichtige Features in der Teamarbeit (z. B. Workspaces). |

### Bruno

Bruno ist ein innovativer Open-Source-API-Client, der sich als moderne Alternative zu etablierten Tools wie Postman und Insomnia positioniert. Seine herausragende Eigenschaft ist die Git-freundliche Architektur, die API-Collections direkt im Dateisystem speichert und dabei eine einfache Textsprache namens "Bru" verwendet. Bruno funktioniert vollständig offline, wodurch keine Cloud-Abhängigkeiten entstehen und sensible Testdaten lokal bleiben. Das Tool unterstützt den Import von Postman-Collections, was die Migration vereinfacht, und ermöglicht die Automatisierung von API-Tests durch JavaScript-basierte Testskripte. Die Benutzeroberfläche ist modern und responsiv, jedoch fehlen noch einige erweiterte Features etablierter Konkurrenten. Bruno eignet sich besonders für Teams, die Wert auf Versionskontrolle, Datenschutz und eine schlanke, entwicklerfreundliche Arbeitsweise legen.

|  |  |
| --- | --- |
| **Kategorie** | **Beschreibung** |
| Stärken | Open-Source, keine Lizenzkosten und Leichtgewichtig |
| Schwächen | Relativ neue Community und Einschränkung bei komplexen Testfällen |
| Chancen | Wachsendes Interesse der Entwickler Anpassungsfähigkeit durch open source |
| Risiken | Abhängigkeit von Comunity-Support trotz neuer kommerzieller Struktur. |

## Bewertung und Priorisierung der Tools

### Gewichtung der Kriterien

Die Gewichtung der Kriterien erfolgt auf Basis der Anforderungen von cortility (Skala 1 = gering bis 5 = ausschlaggebend):

|  |  |
| --- | --- |
| **Kriterium** | **Gewicht** |
| Lizenzmodell | 10 % |
| Funktionsumfang | 20 % |
| Unterstützung von REST & SOAP | 10 % |
| Benutzerfreundlichkeit | 20 % |
| Testautomatisierung | 15 % |
| Integration CI/CD | 15 % |
| Support & Dokumentation | 10 % |
| Summe | 100% |

### Scoring der Shortlist

Für jedes Tool wurde anhand der Kriterien ein Score von 1 bis 10 vergeben(1 = schlecht 10 = hervorragend). Die Bewertungen sind auf Basis des Durchschnittes einer Tooltestung und einer Bewertung von Internen Entwicklern entstanden

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Kriterium** | **Gewichtung (%)** | **Postman** | **Bruno** | **Rest Assured** |
| Lizenzmodell | 30 % | 8 | 10 | 10 |
| Funktionsumfang | 20 % | 8 | 9 | 6 |
| Benutzerfreundlichkeit | 15 % | 10 | 7 | 4 |
| Testautomatisierung | 15 % | 7 | 8 | 9 |
| Unterstützung von REST & SOAP | 10 % | 8 | 10 | 5 |
| Integration CI/CD- | 5 % | 7 | 7 | 10 |
| Support & Dokumentation | 5% | 9 | 8 | 6 |
| Gesamtnutzwert | 100% | 8,15 | 8,8 |  |

## Entscheidungsempfehlung

Die Auf Grundlage der durchgeführten systematischen Evaluierung, der SWOT-Analyse sowie der Nutzwertanalyse ergibt sich eine klare Tendenz hinsichtlich des für cortility am besten geeigneten Tools zur API-Testung. Untersucht wurden die drei Tools Postman, Rest Assured und Bruno. Während Postman mit seiner benutzerfreundlichen Oberfläche und einer starken Community überzeugt und Rest Assured sich vor allem durch seine tiefe Integration in Java-basierte Entwicklungsumgebungen sowie seine Eignung für CI/CD-Szenarien auszeichnet, bietet Bruno einen besonders ausgewogenen Gesamtansatz, der sowohl Entwickler- als auch Teamanforderungen abdeckt.

Bruno kombiniert die Vorteile von CLI-basierter Flexibilität mit einer modernen, leichtgewichtigen Benutzeroberfläche und bietet eine vollständige Offline-Nutzung, Versionierung über Git sowie eine transparente Datenstruktur auf Dateibasis. Besonders hervorzuheben ist, dass Bruno vollständig Open Source ist und somit keine Lizenzkosten verursacht – ein signifikanter Vorteil gegenüber anderen Werkzeugen, insbesondere bei wachsender Teamgröße. Die klare Trennung von Testdaten und Testlogik sowie die einfache Integration in bestehende DevOps-Pipelines machen Bruno zu einem technisch fortschrittlichen und gleichzeitig praxistauglichen Tool.

Die Nutzwertanalyse bestätigt dieses Ergebnis: Bruno erreicht den höchsten Gesamtnutzwert und punktet in nahezu allen bewerteten Kategorien – insbesondere in den Bereichen Kosten/Nutzungsverfügbarkeit, Integration, Erweiterbarkeit und Teamfähigkeit. Auch wenn Postman in der Benutzerfreundlichkeit weiterhin führend ist und Rest Assured für sehr spezifische Automatisierungsanforderungen eine gute Ergänzung bleibt, ist Bruno als zukunftssichere Lösung für cortility zu empfehlen.

Daher lautet die Entscheidungsempfehlung, Bruno als zentrales API-Testtool bei cortility einzuführen, Postman optional weiterhin unterstützend einzusetzen und Rest Assured gezielt in automatisierten Back-End-Testpipelines zu verwenden. Diese Kombination stellt sicher, dass sowohl Entwickleranforderungen als auch teamübergreifende Testprozesse effizient und modern unterstützt werden.

# Implementierung (optional)

## Pilotierung des empfohlenen Tools

* Zur Validierung der Entscheidung wird eine Pilotphase (Knödler, 2019) mit Bruno in den folgenden Schritten durchgeführt:

**Vorbereitungsphase**

* Installation von Bruno über ein Skript auf einem Unternehmens Laufwerk
* Anbindung an ein bestehendes Git-Repository für gemeinsamen Zugriff.
* **Interne Doku zur Nutzung von Bruno**

**Einrichtungsphase**

* Migration von zwei bestehenden Postman-Collections auf Bruno-Testprojekte. Dabei werden Testschritte, Assertions und Testdaten übernommen oder umgeschrieben.
* Einrichtung von Mock-Services für abhängige Altsysteme (z. B. RFC-Stubs, IDoc-Emulation).

**Testfallimplementierung**

* Entwicklung von 10 repräsentativen End-to-End-Testfällen, die typisch für SAP S/4HANA RAP-Services sind (z. B. Anlegen/Korrigieren eines Geschäftsvorfalls, Suche mit Filterparametern, Batch-Request an OData).
* Parametrisierung mittels Excel-Dateien für Data-Driven-Tests.

**Pilotlauffrist und Review**

* Pilotlauffrist von vier Wochen, in denen täglich Regressionstests automatisiert ausgeführt werden.
* Wöchentliche Reviews mit den Stakeholdern (QA, Entwicklung, Projektleitung).

## Integration in bestehende Entwicklungsprozesse

Da API-Tests bisher nur vereinzelt mit Postman durchgeführt wurden, bestehen aktuell keine tiefgreifenden Integrationen in CI/CD-Pipelines. Dies bietet die Chance, mit einem neuen Tool wie Bruno direkt moderne Standards zu etablieren.

## Erste Ergebnisse und Rückmeldungen

Nach der Implementierung sind die ersten Rückmeldungen wie folgt:

# Fazit und Ausblick

## Zusammenfassung der Ergebnisse

Die vorliegende Projektarbeit hat gezeigt, dass der bestehende Einsatz von Postman zukünftig für die Anforderungen von cortility nicht ausreichend ist. Zudem zeigt die Analyse, dass mit der Einführung von Bruno keine technischen Hürden beim Umstieg auf das RESTful Application Programming Model zu erwarten sind. Zusätzlich wurde aufgezeigt welche neuen Anforderungen es in der Zukunft geben wird, und welche Tools für welche Szenarien am besten geeignet sind.

## Reflexion der Vorgehensweise

Die Kombination aus qualitativer SWOT-Analyse und quantitativer Nutzwertanalyse erwies sich als geeignet, um sowohl subjektive Einschätzungen (z. B. Benutzerfreundlichkeit) als auch objektive Kriterien (z. B. Funktionsumfang, Lizenzkosten) in Einklang zu bringen. Die Pilotphase (Knödler, 2019) bestätigte die getroffene Auswahl in der Praxis.

Ein möglicher Schwachpunkt liegt in der begrenzten Zahl der evaluierten Tools: Weitere Lösungsansätze (z. B. Open-Source Lösungen wie PostgREST oder spezialisierte SAP-Testwerkzeuge) wurden nicht vertieft betrachtet. Aufgrund der klaren Priorisierung war dies jedoch akzeptabel.

## Ausblick: Weiterführende Maßnahmen und strategische Bedeutung

Ausbau von Performance und Sicherheitstest

Integration von SAP-Spezifika (ABAP Metadaten)

Kontinuierliche Weiterentwicklung (Updates + Onboarding)

Der Ausblick behandelt primär den Ausbau von Performance und Sicherheit sowie die Integration von ABAP-Metadaten und Kontinuierliche weiter Entwicklung. Der Ausbau von Performance und Sicherheit ist die Grundlage um weitere Maßnahmen und strategien zu verfolgen denn ohne die Sicherheit das dasss Programm fehlerfrei läuft kann man nicht sicher gehen das alles so funktioniert darauf aufbauend folgt die Integration der ANAP-Metadaten da diese in dem S/$ Hana Kontext essenziel sind um ABAP spezifische Sicherheit und Performance Test weiter auszubauen und die Kontinuierliche Entwicklung

# Literaturverzeichnis

Anwar, M. (2024, Juli 10). Was ist API-Testautomatisierung? Eine vollständige Anleitung. *Astera*. <https://www.astera.com/de/type/blog/api-test-automation/>

Bruno Open Source Community. (2024). *Bruno – API Client (Alternative zu Postman)*. <https://github.com/usebruno/bruno>

Chen, T. (2024). Challenges and Opportunities in Integrating LLMs into Continuous Integration/Continuous Deployment (CI/CD) Pipelines. *2024 5th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*, 364–367. <https://doi.org/10.1109/AINIT61980.2024.10581784>

Dittmer, G. (1995). Nutzwertanalyse. In *Managen mit Methode: Instrumente für individuelle Lösungen* (S. 43–56). Gabler Verlag. <https://doi.org/10.1007/978-3-663-05929-5_5>

Dizdarevic, N. (2025, April 24). Testautomatisierung: Tool-Auswahl & Kategorien. *Vision11 GmbH*. <https://www.visioneleven.com/blog/effiziente-testautomatisierung/>

Ehsan, A., Abuhaliqa, M. A. M. E., Catal, C., & Mishra, D. (2022a). RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions. *Applied Sciences*, *12*(9), Article 9. <https://doi.org/10.3390/app12094369>

Ehsan, A., Abuhaliqa, M. A. M. E., Catal, C., & Mishra, D. (2022b). RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions. *Applied Sciences*, *12*(9), 4369. <https://doi.org/10.3390/app12094369>

GmbH, V. (2022). *API-Testing-Tools im Vergleich – Welches Tool passt zu Ihrem Projekt?* <https://vision11.de/api-testing-tools-vergleich>

Golmohammadi, A., Zhang, M., & Arcuri, A. (2024). Testing RESTful APIs: A Survey. *ACM Transactions on Software Engineering and Methodology*, *33*(1), 1–41. <https://doi.org/10.1145/3617175>

Haase, O. (2021). *Serviceorientierte Entwicklung mit RAP – OData Services im S/4HANA*. SAP PRESS.

Hocke, M. (2021). API-Testing – Ein Vergleich der bekanntesten Tools. *iX – Magazin für professionelle Informationstechnik*, *3*.

Hönig, O. (o. J.). *Die besten API Testing Tools in 2025*. Abgerufen 4. Juni 2025, von <https://www.qytera.de/blog/api-testing-tools-rest-soap-services>

Hungenberg, H. (2014). *Strategisches Management in Unternehmen: Ziele – Prozesse – Verfahren*. Springer Gabler.

Inc, A. (2025). *Insomnia vs Postman vs SoapUI: Which API Tool is Best?* <https://abstracta.us/blog/testing-tools/insomnia-vs-postman/>

Kajavalta, L. (o. J.). *REST API Security: Testing and Analysis*. Abgerufen 13. Juni 2025, von <https://core.ac.uk/reader/542974100>

Kossmann, D., Leymann, F., & Taubner, D. (2004). Web Services. *Informatik-Spektrum*, *27”, number =*, 113–114.

Lordieck, C. (o. J.). ABAP RESTful Application Programming Model. *Erlebe Software*. Abgerufen 6. Juni 2025, von <https://erlebe-software.de/knowhow/abap-restful-programming-model/>

Lordieck, C. (2025, Mai 6). Kurz erklärt: Das ABAP RESTful Application Programming Model. *Erlebe Software*. <https://erlebe-software.de/sap-hana-entwicklung/kurz-erklaert-das-abap-restful-programming-model/>

Niklas, C. (o. J.). *Nutzwertanalyse | Vorlage für Excel & Beispiel*. Abgerufen 4. Juni 2025, von <https://www.projektmagazin.de/tool/nutzwertanalyse-vorlage-excel>

Pape, U., & Schmidt, R. (2019). Entscheidungsvorbereitung mit Nutzwertanalyse – Ein praxisorientierter Leitfaden. *HMD – Praxis der Wirtschaftsinformatik*, *56*(3).

Richardson, L. (with Ruby, S.). (2007). *RESTful web services*. O’Reilly.

Russ, T. (2021, Januar 6). *Katalon doesn’t work with VPN’s At all—Feedback & Reviews / Bugs Report*. Katalon Community. <https://forum.katalon.com/t/katalon-doesnt-work-with-vpns-at-all/50812>

SAP Community. (2025). *API Automation using SAP S/4 HANA Test Automation Tool*. <https://community.sap.com/t5/enterprise-resource-planning-blogs-by-sap/api-automation-using-sap-s-4-hana-test-automation-tool/ba-p/13494662>

SAP SE. (2020). *ABAP RESTful Application Programming Model (RAP) – Einführung und Architektur*. <https://help.sap.com/viewer/product/ABAP_PLATFORM>

SAP SE. (2025). *ABAP RESTful Application Programming Model (RAP)*. <https://pages.community.sap.com/topics/abap/rap>

Schawel, C., & Billing, F. (2011). SWOT-Analyse. In *Top 100 Management Tools: Das wichtigste Buch eines Managers* (S. 182–183). Gabler. <https://doi.org/10.1007/978-3-8349-6605-6_82>

Suda, B. (o. J.). SOAPWebServices. *2003*.

TestGrid. (2025). *The Best Postman Alternatives to Streamline API Testing*. <https://testgrid.io/blog/postman-alternatives/>

Vahs, D., & Schäfer-Kunz, J. (2020). *Einführung in die Betriebswirtschaftslehre* (9. Aufl.). Schäffer-Poeschel.

Volker, R. (GS-L. S. S. (2020, Oktober 30). Einführung in das ABAP RESTful Application Programming Model (RAP). *Cpro Industry Projects & Solutions GmbH*. <https://cpro-ips.com/blog/einfuehrung-in-das-abap-restful-application-programming-model-rap/>

Weihrich, H. (1982). The TOWS matrix—A tool for situational analysis. *Long Range Planning*, *15*(2), 54–66. <https://doi.org/10.1016/0024-6301(82)90120-0>

Westerveld, D. (2021). *API testing and development with Postman: A practical guide to creating, testing, and managing APIs for automated software testing*. Packt Publishing.

Zühlke, E. (2022). *Postman – API-Testing leicht gemacht*. <https://www.zuehlke.com/de/insights/postman-api-testing>

# Anhang