A=SIT Technology
E-Government Innovationszentrum

15. Oktober 2021 Studie · Web & Cloud · E-Government · 41 Min. Lesezeit

Whitepaper APIs in der öff. Verwaltung

Archiviert. Dieser Inhalt wird nicht mehr aktualisiert. Er bleibt zur Information verfügbar.

Quelle https://technology2.a-sit.at/veroeffentlichungen/whitepaper-apis-in-der-oeffentlichen-verwaltung/

Application Programming Interfaces, kurz APIs, ermöglichen die Bereitstellung von Web Services und Daten via maschinenlesbarer Schnittstellen bei gleichzeitiger Beschränkung des Zugriffs auf definierte Usergruppen durch Authentifizierungs- und Autorisierungsmechanismen.

Da in Österreich zum derzeitigen Zeitpunkt keine klaren Vorgaben existieren, welche Standards die von den Behörden zur Verfügung gestellten APIs einhalten müssen, war das Ziel dieses Whitepapers, ein Work Item für eine nationale API-Strategie in Österreich zu definieren. Inhalte sind einerseits Grundsätze und Überlegungen, die für eine Behörde bei der Entwicklung von APIs von Bedeutung sein können sowie andererseits technische Implementierungsdetails für API-Entwickler*innen samt Sicherheitsstandards für APIs.

1. Einleitung

Die Digitalisierung der Gesellschaft schreitet stetig voran und damit steigen auch die Ansprüche der Bürger*innen an die öffentliche Verwaltung, einfach zu nutzende digitale Dienste anzubieten und somit die Kommunikation zwischen der öffentlichen Verwaltung und Bürger*innen (G2C) oder Unternehmen (G2B) zu vereinfachen. Gleichzeitig bietet die fortschreitende Digitalisierung die Möglichkeit, Daten innerhalb staatlicher Strukturen einfach und ohne Duplikate auszutauschen und somit die innerbehördliche Kommunikation zu verbessern (G2G). Dies wird beispielsweise durch den Portalverbund ermöglicht, welcher den Zugriff auf über 1300 teilnehmende Applikationen und Services mittels Single Sign-On bietet. Zu diesem Zweck bietet das Portalverbundsystem ein Authentifizierungs- und Autorisierungsschema, mit dem Behörden untereinander Daten und Informationen austauschen können [1].

Durch die derzeitige monolithische Architektur weisen jedoch viele österreichische Services individuelle Schnittstellen auf, was zu Einschränkungen in Bezug auf Größe und Komplexität der Anwendungen führen kann. Dies erschwert insbesondere Anpassungen der einzelnen Anwendungen, da bei einer Aktualisierung die gesamte Anwendung neu bereitgestellt werden muss. Weiters gibt es zum derzeitigen Zeitpunkt keine Vorgaben, welche Standards die eingebundenen Services erfüllen müssen, was dazu führt, dass manche Services XML-SOAP und andere REST basiert sind. Dies kann den Zugriff von mobilen Geräten bzw. von Clients auf die Services erschweren. Durch die Planung der Einführung einer Microservice-Architektur

sowie der Verbindung von zentralen Registern über standardisierte Schnittstellen, wie dem Register- und Systemverbund (RSV) können einige der zuvor genannten Nachteile behoben werden, dennoch ist ein holistischer Ansatz zum Datenaustausch in Österreich essenziell.

Die Europäische Union (EU) hat im Rahmen der APIs4DGov Studie [2] bereits gezeigt, dass der Einsatz samt Definition von einheitlichen Verfahren und Typen von Application Programming Interfaces (APIs) den Aufbau einer Microservice-Architektur unterstützen und Potential für die Digitalisierung der öffentlichen Verwaltung bieten kann. Um die Vorteile von APIs, wie beispielsweise der Austausch von Informationen von öffentlichem Wert mit Stakeholder*innen wie anderen Verwaltungen, Bürger*innen oder Unternehmen in einfacher und maschinenlesbarer Art und Weise, im behördlichen Kontext nachhaltig nutzen zu können, ist es wichtig, ad-hoc-Lösungen zu vermeiden. Dementsprechend sollte für die einzusetzenden APIs auf nationaler Ebene eine API-Strategie entwickelt werden, um die Verwendung von anerkannten Standards sicherzustellen.

Ziel dieses Whitepapers ist es somit, ein Work Item für eine nationale API-Strategie in Österreich zu definieren, um hier im europäischen Vergleich nicht nachzuhinken und Anforderungen auf EU-Ebene, wie beispielsweise die der Open Data EU Direktive 2019/1024, des „Once-Only“ Prinzipes oder des EU E-Government Aktionsplans, zu erfüllen. Dabei soll insbesondere die Art und Weise, in der APIs angeboten werden, harmonisiert und standardisiert werden. Um daher Behörden bei der Entwicklung von APIs zu unterstützen, werden allgemeine Standards inklusive Design- und Implementierungsanleitungen zusammen mit technischen API Best-Practices bereitgestellt.

Dieses Whitepaper befasst sich zunächst mit dem allgemeinen Rahmen für die Verwendung von APIs in der Verwaltung und beschreibt die Grundsätze und Überlegungen, die für eine Behörde bei der Entwicklung von APIs von Bedeutung sein können. Dabei werden sowohl die Auswirkungen der Verwendung von APIs auf die Behörde als auch auf die gesamte öffentliche Verwaltung betrachtet sowie technische Implementierungsdetails für API-Entwickler*innen, einschließlich Sicherheitsstandards für APIs, definiert.

Zielgruppe dieses Whitepapers sind auf strategischer Ebene österreichische Entscheidungsträger*innen in der Verwaltung in allen Gebietskörperschaften. Dies inkludiert beispielsweise auf Landesebene die Ämter der Landesregierungen, auf Bundesebene das Bundesministerium für Digitalisierung und Wirtschaftsstandort sowie andere Organe, wie die Sozialversicherungen. Auch andere interessierte Parteien, wie Softwarearchitekt*innen oder API Entwickler*innen, die Anwendungen entwickeln, die vom öffentlichen Sektor zur Verfügung gestellte APIs verwenden, werden explizit angesprochen.

2. Hintergrund

Das Ziel des öffentlichen Sektors und somit auch der österreichischen Verwaltung ist es, Public Value zu schaffen, also einen Beitrag zur Erhöhung des Gemeinwohles zu leisten und die Wünsche und Bedürfnisse der Öffentlichkeit zu erfüllen. Da Public Value die Form von Informationen, Programmen und Leistungen, welche der öffentliche Sektor zur Verfügung stellt, annehmen kann, sollten auch zur Verfügung gestellte Services und offene Daten die Public Value erhöhen. [3] Eine Möglichkeit, Web Services bereitzustellen, sind Web APIs, welche heutzutage die Grundlage für viele Webseiten und mobile Anwendungen bilden [4]. Sie werden verwendet, um Entwickler*innen notwendige Bausteine zur Erstellung webbasierter Anwendungen zur Verfügung zu stellen [5] und spielen eine wesentliche Rolle bei der Bereitstellung von Public Value für die Öffentlichkeit, weswegen einige Gründe für deren Einsatz sprechen:

- Sie fördern die Erfüllung der Open Data Ziele durch Wiederverwendung und gemeinsame Nutzung von Daten innerhalb und außerhalb der öffentlichen Verwaltung
- Sie bieten der Privatwirtschaft die Möglichkeit, angebotene Dienste enger mit denen der öffentlichen Verwaltung zu verknüpfen
- Sie steigern die Datenqualität durch die Bereitstellung einer einzigen, zuverlässigen Quelle
- Sie fördern die Einführung neuer und innovativer Technologien

Da dieses Dokument insbesondere auf Web APIs abzielt, wird im weiteren Verlauf, sofern nicht anders angegeben, der Begriff "API" für Web APIs verwendet.

2.1. Was sind APIs?

Gemäß NIST sind APIs Systemzugriffspunkte oder Bibliotheksfunktionen, die eine klar definierte Syntax haben und den aufrufenden Programmen und Services eine definierte Funktionalität zur Verfügung stellen [6]. Das Konzept von APIs ist nicht neu, sondern wird bereits seit Jahrzehnten insbesondere vom privaten Sektor verwendet, um Daten und Funktionen interessierten Parteien zur Verfügung zu stellen [5] [7]. Vereinfacht gesagt ist eine API eine strukturierte, maschinenlesbare und dokumentierte Schnittstelle, die es anderen unterschiedlichen Softwarekomponenten, wie Anwendungen, Services oder Systemen, ermöglicht, miteinander zu kommunizieren und dabei nicht an bestimmte Programmiersprachen gebunden ist [4]. APIs stellen somit eine Reihe von Daten und Funktionen zur Verfügung, um die Interaktion zwischen Computerprogrammen zu erleichtern und ihnen den Austausch von Informationen zu ermöglichen [8]. Dabei bieten APIs normalerweise keinen direkten Zugriff auf alle verfügbaren Daten und Funktionen, sondern ermöglichen vielmehr die Beschränkung des Zugriffs auf definierte Usergruppen durch Authentifizierungs- und Autorisierungsmechanismen [7]. Diese Zugriffsbeschränkung wird dokumentiert, indem in der Dokumentation beschrieben wird, wer auf welche Informationen oder Funktionen Zugriff hat und wie dieser Zugriff gewährt wird [4].

2.2. API-Klassifikationen

Zur Veröffentlichung von APIs hat die öffentliche Verwaltung mehrere Möglichkeiten, je nachdem, ob eine interne, innerbehördliche Interaktion oder eine externe Interaktion mit anderen Behörden, Unternehmen oder Bürger*innen gewünscht ist. Dementsprechend können APIs je nach Art des Zuganges für verschiedene Interessensgruppen eingeteilt werden. Die gängigste Unterscheidung ist in interne und externe APIs:

Interne APIs sind nicht sichtbar für externe Benutzer*innen. Sie werden ausschließlich innerhalb einer Behörde verwendet, um die gemeinsame Nutzung von Daten und Diensten zwischen internen Systemen zu erleichtern und somit beispielsweise eine gemeinsame Ressourcennutzung zu ermöglichen oder Geschäftsabläufe effizienter zu gestalten. Durch den Einsatz von internen APIs können Sicherheits- und Zugriffskontrollanforderungen eingehalten werden, ohne eine komplexe Punkt-zu-Punkt Integration zu erfordern. [2] [4]

Externe APIs, auch bekannt als offene oder öffentliche APIs, können von allen interessierten Parteien sowohl innerhalb als auch außerhalb der Behörde verwendet werden und bieten die Möglichkeiten zur Interaktion mit anderen Behörden oder externen Entwickler*innen. Die Verwendung von externen APIs kann*

*ggf. durch festgelegte Zugriffskontrollanforderungen, wie beispielsweise eine Registrierung oder die Verwendung eines API-Schlüssels, auf autorisierte Benutzer*innen eingeschränkt werden. [2] [4]

2.3. Was sind Web Services?

Web Services sind eine Kombination aus offenen Protokollen und Standards, die im Einklang mit definierten Einschränkungen und Richtlinien [5] [8] Dienste über das Internet anbieten und die Kommunikation zwischen Client und Server über das Internet mittels TCP/IP, HTTP, Java, HTML und XML ermöglichen [9]. Durch Sprachunabhängigkeit gewährleisten sie Interoperabilität zwischen unterschiedlichen Anwendungen [9] und können durch einen universellen Ressourcenbezeichner (URI) identifiziert werden [5].

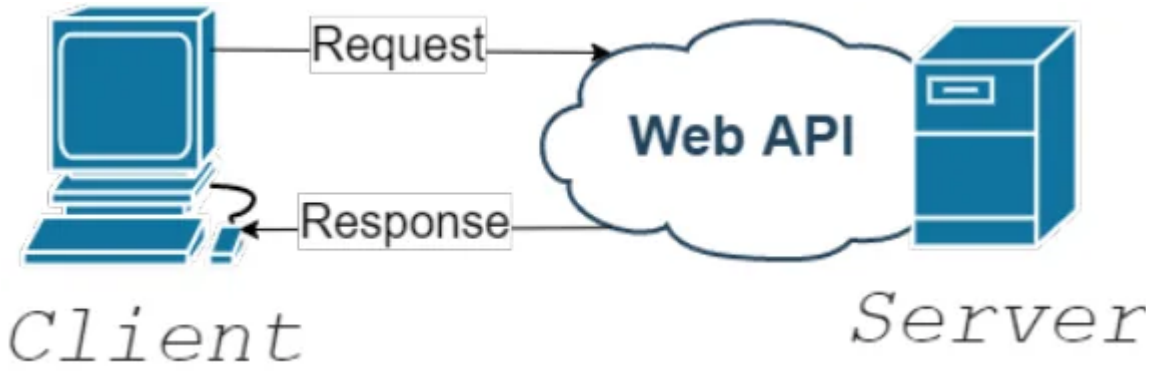
Eine schematische Darstellung der Kommunikation zwischen einem Client und einem Web Service ist zum besseren Verständnis in Abbildung 1 ersichtlich.

Abbildung 1: Kommunikation zwischen Client und Web Service Um Systemen die Interaktion mit einem Web Service ermöglichen zu können, gibt es prinzipiell zwei Möglichkeiten, eine API zu gestalten: entweder mittels des Simple Object Access Protocol (SOAP) oder auf Basis der Representational State Transfer (REST) Architektur [9]. In Österreich sind noch viele Services XML-SOAP basiert, da REST jedoch den de-facto Standard für Web Services darstellt, ist im Kontext dieses Dokuments eine API, sofern nicht anders angegeben, eine REST API.

2.3.1. REST APIs

REST ist der gängige Architekturstil für die Kommunikation mit Web Services über JavaScript und mobile Anwendungen und stellt den Industriestandard für die Entwicklung von APIs für moderne Web Services dar [2], weswegen die Verwendung von REST APIs forciert werden sollte. Dieser Stil eignet sich zudem gut für die Modellierung von Daten und Systemen. REST hat nur wenige formale Anforderungen, die folgenden Einschränkungen sollten jedoch befolgt werden [9]:

- Die REST API sollte für eine Client-Server-Architektur ausgelegt sein, bei der Anfragen per http gestellt werden. Dies ermöglicht die Trennung von Client- und Server-Architekturen, was zu einer höheren Flexibilität bei gleichzeitiger Verbesserung der Skalierbarkeit führt.
- Die Übertragung zwischen Client und Server sollte zustandslos sein, d.h. jede einzelne Client-Anfrage sollte in sich geschlossen sein und darf nicht am Server gespeichert werden, was zu einer Verbesserung der Skalierbarkeit, Sichtbarkeit und Zuverlässigkeit führt.
- Die REST API sollte nach Möglichkeit cachefähig sein, um die Netzwerkleistung durch Reduzierung der Anfragen zu verbessern, da bei einer cachefähigen Antwort diese wiederverwendet werden muss. Dies führt zu einer Verbesserung der Leistung auf der Client-Seite und zu einer Erhöhung der Skalierbarkeit auf der Server-Seite.



- Die REST API sollte ein mehrschichtiges System bestehend aus hierarchischen Schichten sein, was dazu führt, dass eine Komponente nur mit den direkt angrenzenden Komponenten kommunizieren kann. Der Nachteil dieses Systems ist, dass es den Overhead und die Verzögerung bei der Datenverarbeitung erhöht.
- Die REST API sollte optional Code on Demand anbieten, also eine Funktionalität zum Herunterladen und Ausführen von Code. Der Client muss den Code nicht kennen oder verstehen, sondern nur wissen, wie dieser ausgeführt werden kann, was zu einer Verbesserung der Flexibilität, Erweiterbarkeit und Ausbaufähigkeit des Systems führt.
- Eine REST API unterscheidet sich von anderen Netzwerkstilen durch eine einheitliche Schnittstellenbeschränkung. Prinzipien einer einheitlichen Schnittstelle sind:

o die eindeutige Identifizierung von Ressourcen in der Anfrage durch Ressourcenbezeichner o die Manipulation von Ressourcen durch Repräsentationen, was dazu führt, dass ein zukünftiger Aufruf eine andere Darstellung ergeben kann o die Verwendung von selbstbeschreibenden Nachrichten o Hypermedia als Engine des Anwendungsstatus (HATEOAS), d.h. der REST-Client sollte in der Lage sein, über Hyperlinks alle aktuell verfügbaren Maßnahmen zu identifizieren.

2.3.2. SOAP APIs

SOAP (Simple Object Access Protocol) ist ein plattform- und sprachunabhängiges Nachrichtenprotokoll, welches das XML-Nachrichtenformat für die Kommunikation und die Weiterleitung von Daten über das Internet verwendet. Es besteht aus einem Envelope, welcher einen Hinweis auf das SOAP Protokoll enthält und den Beginn und das Ende des Nachrichtenformats kennzeichnet, einem Header, der optionale Daten wie beispielsweise Autorisierungsdetails enthält und einem Body, welches die Nachricht – typischerweise im XML-Format – sowie etwaige Fehlerberichte enthält. [9] Der Hauptunterschied zwischen REST und SOAP besteht darin, dass SOAP ein Protokoll mit integrierten Regeln ist, während REST eine Reihe von Softwarearchitektur-Prinzipien umfasst, die jedoch nur Richtlinien sind und für die Implementierung nicht zwingend eingehalten werden müssen [10]. Weitere Unterschiede sind, dass REST eine geringere Antwortzeit als SOAP hat, einfacher zu verstehen ist und mit jedem Format arbeiten kann, während SOAP auf XML beschränkt ist. SOAP hat den Vorteil, dass es zuverlässiger ist und eine eingebaute WS-Security-Funktion hat, während REST manuell durch Sicherheitsfunktionen erweitert werden muss. [9]

2.4. Service Architektur

Bei der Verwendung von APIs gibt es in der Praxis zwei Architekturansätze, die serviceorientierte Architektur (SOA) und die Microservice-Architektur:

SOA-Architektur : SOA wurde entwickelt, um monolithische Applikationen zu verbessern und einige ihrer Probleme auszumerzen. Sie ist ein Designansatz, bei dem mehrere Services zusammenarbeiten, um eine Reihe von Funktionen bereitzustellen und somit die Wiederverwendbarkeit von Software zu fördern. [11]. Die Kommunikation zwischen den Services erfolgt typischerweise über das SOAP-Protokoll [9].

Microservice-Architektur : Microservices sind kleine, autonome Services, die miteinander über APIs kommunizieren. Jedes Service kann jederzeit erweitert, hinzugefügt oder gelöscht werden, ohne dass dies Auswirkungen auf andere Services in der Architektur. Dies bietet den Vorteil, dass für jedes Service die passende Technologie verwendet werden kann und neue Technologien leichter eingesetzt werden können. [11]

2.5. Status Quo in Österreich

In Österreich werden zur Verfügungstellung von verschiedenen Daten und Services bereits teilweise REST APIs verwendet, ein Teil der aktuellen Service-Infrastruktur ist aber nach wie vor XML-SOAP basiert, was den Zugriff von mobilen Geräten bzw. von Clients auf die Services erschwert. Dieses Unterkapitel zielt darauf ab, einen Überblick über den aktuellen Status Quo in Österreich zu bieten.

2.5.1. Oesterreich.gv.at

Oesterreich.gv.at ist eine behördenübergreifende Plattform, die digitale Services zur Verfügung stellt und somit den Bürger*innen die Möglichkeit bietet, die wichtigsten Behördenwege durchzuführen. Der Zugriff auf die angebotenen Services erfolgt über ein API-Gateway, welches die Authentifizierung über den Identity Provider Shibboleth mittels Handysignatur, EU Login oder ID Austria über das SAML 2 Protokoll bzw. OpenID Connect abwickelt. Um eine flexible Erweiterbarkeit samt Interoperabilität zu ermöglichen, ist geplant, die Plattform auf Basis einer Microservice-Architektur umzusetzen

2.5.2. Data.gv.at

2019 trat die Neufassung der Europäischen Open Data Richtlinie in Kraft, welche darauf abzielt, offene Daten, die im Besitz öffentlicher Stellen oder Unternehmen sind, soweit möglich elektronisch und in maschinenlesbarer Form – im Idealfall via APIs – auffindbar und verfügbar zu machen [14].

Österreich hat bereits 2012 mit der zur Verfügungstellung von offenen Daten für alle interessierten Parteien via data.gv.at begonnen [15] und zählt gemäß dem Open Data Reifegradbericht der Europäischen Kommission zu den Trendsettern, da es bereits einen sehr hohen Reifegrad bei der zur Verfügungstellung offener Daten besitzt [16]. Der Zugriff auf die auf data.gv.at veröffentlichten Daten kann entweder mittels Direktzugriff, verlinktem Zugriff oder über Schnittstellen erfolgen. Bei einem Direktzugriff werden die Daten direkt auf den Webserver data.gv.at hochgeladen, während beim verlinkten Zugriff nur die Metadaten auf data.gv.at erfasst werden. Erfolgt der Zugriff über Schnittstellen, muss die Organisation, die die Daten zur

Verfügung stellt, eine API (im Idealfall CKAN oder XML Metadaten API) anbieten, eine genaue Definition der Anforderungen an APIs existiert nicht. [17]. Für die Verwendung von data.gv.at werden ein Userhandbuch [17], das Vorgehensmodell

[18] sowie Empfehlungen für die Metadatenstruktur [17] zur Verfügung gestellt.

2.5.3. Register- und Systemverbund

Der EU-eGovernment-Aktionsplan 2016-2020 hat für öffentliche Stellen einige Grundsätze definiert, die das Ziel, digitale Dienste einfach und grenzüberschreitend für alle interessierten Personen und Unternehmen anzubieten, unterstützen sollen. Dienste sollen vorzugsweise digital angeboten werden, wobei hierbei die Barrierefreiheit zu beachten ist und deren Interoperabilität nicht nur bundesweit, sondern auch grenzüberschreitend gewährleistet sein muss. Neben dem Grundsatz der Offenheit und Transparenz sowie der Sicherstellung der Vertrauenswürdigkeit und Sicherheit der geteilten Daten ist insbesondere der Grundsatz der einmaligen Erfassung, auch bekannt unter Once-Only-Prinzip, hervorzuheben. Dieser besagt, dass Bürger*innen und Unternehmen Daten nur einmalig an die Behörden übermitteln müssen, um deren Aufwand minimal zu halten. [12] Zur Umsetzung des Once-Only-Prinzips wurde in Österreich der Register- und Systemverbund (RSV) entwickelt, welcher Datenkonsument*innen einen zentralen Zugriff auf die zur Verfügung gestellten Daten der Datenprovider mittels APIs ermöglicht. Ein*e Datenkonsument*in kann nach einer Anmeldung am Portalverbund mittels einer REST API Daten über den RSV abfragen. Der RSV prüft die erforderlichen Berechtigungen und lehnt die Anfrage ab oder leitet diese an den Datenprovider weiter, er selbst speichert keine Daten. Der Datenprovider wird als Datenlieferant vom RSV genutzt und ist ebenfalls mittels REST API an diesen angebunden.

2.5.4. Portalverbund

Durch das Portalverbundsystem können die teilnehmenden Behörden ihre lokale Userverwaltung auch für externe Applikationen verwenden, wodurch ein Single Sign On (SSO) für alle Portalverbundapplikationen ermöglicht wird. Um auf die internen und externen Applikationen zugreifen zu können, müssen sich die Mitarbeiter*innen zunächst am Stammportal entweder mittels Username/Passwort, Bürgerkarte/Handysignatur oder Windows Login über Kerberos authentifizieren. Nach erfolgreicher Authentifizierung am Stammportal erfolgt eine Weiterleitung auf das Anwendungsportal, wo die gewünschten Applikationen verwendet werden können. [13]

3. Vorschlag zur österreichischen API-Strategie

Die zunehmende Digitalisierung von Behördendiensten führt zu immer größeren Herausforderungen wie Konnektivität, Interoperabilität von Diensten und Datensicherheit. Diese Herausforderungen sorgen dafür, dass Entwickler*innen Vorgaben für die Erstellung, Verbesserung und Vereinfachung von APIs benötigen, um Risiken bei der Implementierung zu verringern.

Die Definition einer API-Strategie hilft bei der Standardisierung von APIs, die von Behörden entwickelt werden, wodurch es Dritten erleichtert wird, Services unter Verwendung dieser APIs zu entwickeln. Außerdem verringert sich durch die Umsetzung einer API-Strategie der Entwicklungsaufwand für die Behörden, da gewisse Standards vorgegeben werden. Zum aktuellen Zeitpunkt existiert jedoch weder eine Gesamtübersicht der in Österreich angebotenen APIs noch Vorgaben zum Design ebenjener. Stattdessen

werden APIs und Web Services je nach Use Case von den jeweiligen Verantwortlichen separat und individuell implementiert, was in der Praxis einen unnötigen Aufwand bedeuten kann. Daher soll dieses Kapitel Anforderungen und Prinzipien definieren, die beim Einsatz von APIs beachtet werden sollen.

3.1. Betrachtung der möglichen Auswirkungen

APIs bieten direkten, maschinellen Zugang zu den Ressourcen und Informationen einer Behörde, weswegen es schwerer zu erkennen ist, wenn Informationen fälschlicherweise freigegeben werden.

3.1.1. Risikobetrachtung

Um somit die Risiken, die beim Einsatz von APIs entstehen können, rechtzeitig erkennen und minimieren zu können, sollten vor dem Einsatz von APIs mögliche Auswirkungen betrachtet werden.

Gesetzgebung : Es muss sichergestellt sein, dass APIs die gesetzlichen Anforderungen der jeweiligen Behörde erfüllen. Dies inkludiert die Überprüfung, ob im jeweiligen Fall eine elektronische Erfassung zulässig ist. Weiters ist das Risiko einer falschen Datenübermittlung zu betrachten und die Haftungsfrage in solch einem Fall zu klären.

Datenschutz : Da APIs unter anderem für die Weitergabe von Informationen verwendet werden, muss deren Sicherheit auf einem risikobasierten Ansatz beruhen. Daher ist es wichtig, das jeweilige Datenschutzniveau zu definieren, die Daten angemessen zu schützen und jede API – unabhängig davon, ob diese in einem Produktiv- oder Testsystem ausgeführt wird – gemäß des aktuell gültigen Datenklassifizierungsmodells einzuordnen. Werden durch APIs personenbezogene Daten verarbeitet, muss die zuständige Behörde die Auswirkungen auf den Datenschutz, gegebenenfalls durch die Durchführung einer Datenschutzfolgenabschätzung, berücksichtigen. Es sollte in jeder Entwicklungsphase überprüft werden, ob eine Datenschutzfolgeabschätzung erforderlich ist und welche Sicherheitsbestimmungen einzuhalten sind.

Werden beispielsweise nur offene Daten via data.gv.at zur Verfügung gestellt, sind die Datenschutzanforderungen im Normalfall eher gering und die Sicherheitsanforderungen beschränken sich auf den Einsatz von API-Schlüssel. Werden hingegen mittels API sensible Daten zur Verfügung gestellt, wie beispielsweise der Zugriff auf die Daten des zentralen Melderegisters, müssen strengere Sicherheitsrichtlinien und -kontrollen angewandt werden. Darüber hinaus besteht das Problem, dass mit der vereinfachten Datennutzung auch die Möglichkeit zunimmt, Daten aus verschiedenen Quellen zu kombinieren, was die Risiken für den Schutz der Privatsphäre und das Risiko eines unbeabsichtigten Datenverlusts steigert. Daher sollten regelmäßige Risikobewertungen durchgeführt werden, um sicherzustellen, dass die Datenschutz- und Sicherheitsanforderungen angemessen sind und eingehalten werden.

Neue Dienste : Wenn eine Behörde zum ersten Mal APIs einsetzt, kann es sich um ein völlig neues Dienstleistungsmodell handeln, bei dem möglicherweise neue Arten von Kund*innen (z. B. Softwareentwickler*innen) unterstützt werden müssen, weswegen es wichtig ist, die organisatorischen Auswirkungen zu berücksichtigen.

Neue Prozesse : APIs können zur Einführung neuer Arbeitsabläufe führen, die potenzielle Auswirkungen haben können. Wenn die API zum Beispiel einen manuellen Prozess automatisiert, müssen die Auswirkungen dieses neuen automatisierten Prozesses berücksichtigt werden.

Informationsverarbeitung : Anwendungsentwickler*innen können Daten, die sie über eine API erhalten haben, nach Belieben weiterverwenden, sofern keine Einschränkungen vorgenommen werden. Daher kann es sinnvoll sein, die Verarbeitung von Informationen, die über APIs offengelegt werden, vertraglich einzuschränken.

Identifikation : Es ist zu definieren, ob und wie API-Kund*innen identifiziert werden sollen.

Verfügbarkeit : Die Verfügbarkeit einer API sollte nicht geringer sein als die Verfügbarkeit des entsprechenden Web Services oder der Website der Behörde. Das bedeutet, dass eine API unter Umständen rund um die Uhr verfügbar sein muss, wobei zu definieren ist, ob auch der Support für die API rund um die Uhr verfügbar sein muss.

Nutzungsbedingungen : Die Nutzungsbedingen für APIs sind unter der Berücksichtigung des Zwecks der über die API ausgetauschten Informationen zu definieren. Zudem sollte überlegt werden, wie die Einhaltung der Nutzungsbedingungen durchgesetzt und überwacht werden soll.

Abgrenzung der Zuständigkeiten : Möglicherweise ist es für Behörden sinnvoll, die Zuständigkeiten klar gegenüber den Anwendungsentwickler*innen vertraglich abzugrenzen, um sicherzustellen, dass die Benutzer*innen über ihre rechtlichen Verpflichtungen in Bezug auf die Nutzung der behördlichen APIs informiert werden.

Informationssicherheit : APIs bieten einerseits eine vergrößerte Angriffsfläche durch mehr Zugriffmöglichkeiten und mehr Dienste, die potenziell ausgenutzt werden können und andererseits das Risiko der unbeabsichtigten Offenlegung von Back-End-Daten, Back-End-Architektur und Back-End-Anwendungen. Weitere Risiken, die sich auf den Verlust von Vertraulichkeit, Integrität und Verfügbarkeit der durch APIs zur Verfügung gestellten Daten beziehen und daher betrachtet werden sollten, sind:

- Verfälschung der Daten durch Fehlinformationen, wenn Schreibvorgänge erlaubt sind.
- Verwendung von Schreibvorgängen zur Durchführung eines Denial-of-Service Angriffes durch Überlastung des Servers oder Datenspeichers
- Stilllegung von APIs und Back-End-Anwendungen durch Verwendung von Platzhaltern in Suchfeldern
- Cross-Site-Scripting-Angriffe, die durch Anwendungen ermöglicht werden, bei denen Benutzereingaben nicht überprüft werden
- SQL-Injections in Anwendungen, welche die Datenbank am API-Backend beschädigen
- Parameter-Angriffe wie HTTP Parameter Pollution (HPP)
- Man-in-the-Middle-Angriffe, bei denen API-Anfragen oder -Antworten verändert werden, was zum Abhören von Daten oder Einfügen von Fehlinformationen führt

- Umgehung von Authentifizierungs- oder Autorisierungsmechanismen, um Nachrichten von legitimen Verbraucher*innen zu fälschen
- Diebstahl von Authentifizierungs-Tokens, um sich unerlaubt Informationen zu verschaffen
- Durchsickern von Systeminformationen durch API-Fehlermeldungen, die Details über den Aufbau einer API oder das zugrunde liegende System offenbaren
- Unberechtigter Zugriff durch aufgrund von mangelhaftem Sicherheitsdesign oder Technologiefehlern nicht funktionierenden Authentifizierungsfaktoren, wie fehlerhaften Session-IDs oder kryptographischen Schlüsseln

Sowohl Entscheidungsträger*innen als auch Entwickler*innen und IT-Architekt*innen müssen die Sicherheitsanforderungen an die Verfügbarkeit, Vertraulichkeit und Integrität von APIs verstehen und befolgen. Dementsprechend wichtig ist es, diese Sicherheitsanforderungen auf strategischer Ebene zu definieren und dabei alle internen und externen Benutzer*innen, die mit den APIs interagieren werden, in Betracht zu ziehen.

3.1.2. Risikominimierung

Die Absicherung von REST APIs ist für den Erfolg jeder API-Implementierung von grundlegender Bedeutung, damit die zuvor erwähnten Sicherheitsrisiken minimiert werden können. Die Sicherheitsanforderungen an eine API beschränken sich jedoch nicht nur auf die API und die damit zusammenhängenden Komponenten, sondern sollten aus einer ganzheitlichen Perspektive betrachtet werden. So geht es bei der Absicherung einer API, die für eine mobile Anwendung bestimmt ist, beispielsweise nicht nur um die Anwendung eines OAuth-Profils, sondern auch darum, zu berücksichtigen, wie das mobile Gerät selbst verwaltet und gesichert werden kann.

Da es keine Standardlösung gibt, mit der alle Aspekte der API-Sicherheit abgedeckt werden können, müssen APIs von Grund auf sicher sein und damit das Prinzip Security by Design berücksichtigen. Damit wird sichergestellt, dass:

- die API-konsumierende Anwendung bekannt ist und nur auf API-Ressourcen zugreifen kann, für die sie berechtigt ist.
- der Nachrichteninhalt zwischen Nutzer*in und Anbieter*in nicht manipuliert wurde,
- die Ressourcen zuverlässig von dem*der Anbieter*in stammen, der*die bei der Anfrage der API-konsumierenden Anwendung angegeben wurde.
- die API verfügbar ist, wenn sie benötigt wird, und nicht durch Angriffe von bösartigen Anwendungen beeinträchtigt wird.

Um API-Sicherheitsrisiken zu begegnen, ist somit ein Sicherheitskonzept erforderlich, welches alle oben genannten Sicherheitsaspekte umfasst. Um dies zu gewährleisten, sollte die Verwaltung der Identitäten, also Benutzer*innen, Geräte, Server und Anwendungen, durch ein Identitäts- und Zugriffsmanagement im Mittelpunkt eines jeden API-Sicherheitskonzepts stehen.

3.2. API Referenzarchitektur

Die API Referenzarchitektur, welche in Abbildung 2 veranschaulicht wird, ist eine mehrschichtige Architektur aus verschiedenen, miteinander agierenden Komponenten:

API Clients: API-Clients können zum einen externe Partner*innen oder andere Behörden sein und zum anderen andere Nutzer*innen, die entweder über eine Mobile App oder eine Web App über das API Gateway auf die zur Verfügung stehenden Services zugreifen.

API Management: Das API Gateway ist ein API-Management-Tool, welches die Möglichkeit bietet, die API-Clients von der Backend-Implementierung zu entkoppeln und somit als primärer Zugriffspunkt für exponierte APIs fungiert. Die Aufgaben des API Gateways inkludieren Authentifizierung und Autorisierung über einen Identity Provider, Durchsetzung von Sicherheitsrichtlinien sowie Schutz vor Bedrohungen. Das API Gateway hat zudem Zugriff auf die Berechtigungsspeicher, welche API-Schlüssel und Zertifikate speichern und für Autorisierungs- und Authentifizierungsdienste verwendet werden können.

Das Entwicklerportal bietet Entwickler*innen Zugriff auf die Dokumentation und Spezifikation der APIs und somit die Möglichkeit, diese aufzufinden. Weitere Funktionen sind die Unterstützung bei der Entwicklung, dem Aufbau und dem Test von verbrauchenden Anwendungen.

Der API Manager ist für die zentralisierte API-Verwaltung, das Lebenszyklusmanagement, die Verwaltung und Definition von Sicherheitsrichtlinien sowie die Verwaltung von Registrierungsprozessen für API-Entwickler*innen zuständig.

Die Komponenten API Monitoring, Policy Definition und Analyse helfen Business und Security Analysten bei der Überwachung der Nutzung der APIs, um Veränderungen in der Nutzung bzw. Nachfrage rechtzeitig zu erkennen. Die Überwachung kann dabei auf technischer Ebene erfolgen, sodass sichergestellt ist, dass die Stabilität und Leistung der APIs aufrechterhalten wird, oder auf Leistungsebene, um sicherzustellen, dass die SLA-Anforderungen erfüllt werden.

API: Microservices definieren APIs, die ihre Funktionalität dem API-Gateway und dem Entwicklerportal bereitstellen. Diese APIs können entweder als intern oder extern klassifiziert werden. Um miteinander zu kommunizieren, verwenden die Mikroservices einer Anwendung das Anforderungs-Antwort-Kommunikationsmodell, typischerweise JSON-kodierte REST-API-Aufrufe basierend auf dem HTTP-Protokoll.

Bei der Erstellung einer API können je nach Anwendungsfall folgende Sprachstile gewählt werden:

- Tunnel-Stil: Die API ist eine Sammlung von Funktionen, die remote aufgerufen werden können, z.B. über XML-RPC, SOAP, gRPC, Avro
- Ressourcen-Stil: Die API ist eine Sammlung von Ressourcen, die verschiedene Arten von Interaktionen ermöglichen, z.B. über OpenAPI/Swagger, RAML, API Blueprint
- Hypermedia-Stil: Die API ist eine Sammlung von miteinander verknüpften Ressourcen, z.B. über HAL, Sirene, Atom, HATEOAS
- Abfrage-Stil: Die API stellt ein strukturiertes Datenmodell zur Verfügung, das mit einer generischen Abfragesprache abgefragt und aktualisiert werden kann, z.B. mittels GraphQL, OData, SPARQL

- Ereignisbasierter Stil: Die API ist eine Sammlung von Ereignissen, die von den Anbietern veröffentlicht werden und von den Nutzern abonniert werden können, z.B. über MQ, WebSub, MQTT, XMPP, AMQP, Kafka, AsyncAPI

Diese Komponente sollte für die Pflege der Schnittstellenspezifikationen und deren Veröffentlichung zuständig sein. Wenn möglich sollte die API-Komponente vollständig zustandslos gehalten werden, d.h. dass jeder API-Aufruf als neue Anfrage behandelt wird.

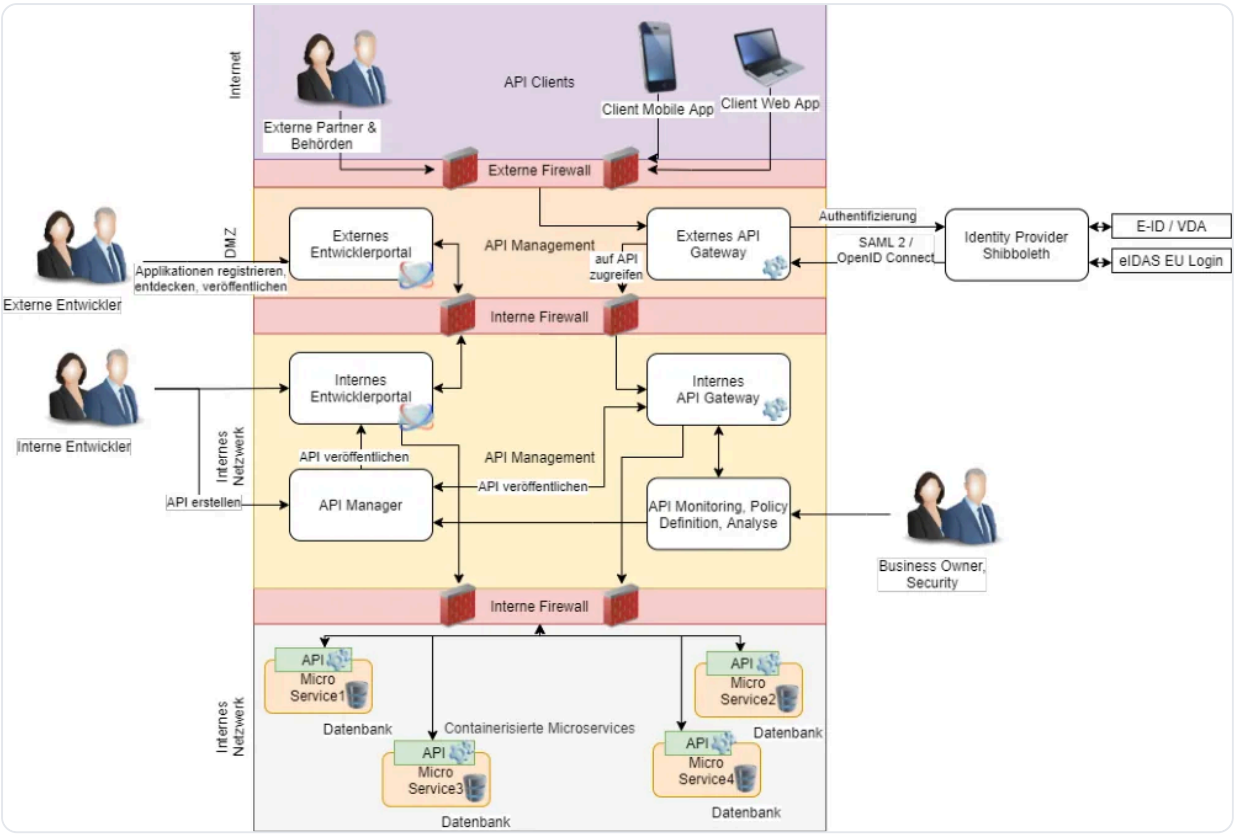


Abbildung 2: API Referenzarchitektur

3.2.1. API Lebenszyklusmanagement

Auch wenn eine API technisch gesehen nur eine Schnittstelle zwischen Maschinen darstellt, sollte diese strategisch als Produkt gesehen werden, da sie den Informationsaustausch sowohl innerhalb als auch außerhalb einer Behörde erleichtert. Dementsprechend müssen APIs in allen Phasen des nachfolgend dargestellten Lebenszyklus verwaltet, gepflegt und unterstützt werden:

Definieren : In Zusammenarbeit mit verschiedenen Interessengruppen sollten die erforderlichen APIs zur Erfüllung der Geschäftsanforderungen ermittelt werden. Das Ergebnis dieser Phase bilden die API-Definition, die API-Spezifikation und das Datenmodell.

Entwickeln : Bei der Entwicklung der API sollten folgende Aspekte berücksichtigt werden: Sicherheit, Leistung, Richtlinienanforderungen, Fehlerbehandlung, Skalierbarkeit und Überwachung. API-Mock-Services sollten parallel entwickelt werden, um den API-Nutzer*innen eine frühzeitige Integration zu ermöglichen. Das Ergebnis dieser Phase bildet eine funktionierende API samt kontinuierlicher Integration.

Testen : Es sollten mehrere Testphasen durchgeführt werden, um sicherzustellen, dass die APIs der erforderlichen Qualität entsprechen. Als Grundlage dafür kann die kontinuierliche Integration mit verschiedenen Testautomatisierungstools dienen. Es wird außerdem empfohlen, Sicherheitstests in den Entwicklungszyklus zu integrieren, die kontinuierliche, wiederholbare und automatisierte Tests ermöglichen, um Sicherheitslücken in APIs und Webanwendungen während der Entwicklung und des Testens zu finden.

Veröffentlichen : Die APIs sollten in einer sicheren Umgebung samt Wartungsplänen und Betriebsanleitungen für die Kunden bereitgestellt werden.

Support : Community-Foren für Benutzer*innen zur Interaktion, Zusammenarbeit und Anforderung von Änderungen/Erweiterungen sollten bereitgestellt werden.

Überwachen : APIs sollten in Echtzeit überwacht werden, wobei Warnungen direkt oder über ein Netzwerkmanagementsystem (NMS) ausgelöst werden können. Aus historischen Daten und aggregierten Transaktionsprotokollen können Trendanalysen abgeleitet werden, um die Nutzung und Akzeptanz zu ermitteln.

Betrieb : Release- und Change-Management-Prozesse sollten befolgt werden, um sicherzustellen, dass nur genehmigte API-Versionen in der Produktion eingesetzt werden.

Außerbetriebnahme : Die Kund*innen sollten über das Einstellen der API benachrichtigt werden. Die Ankündigungsstrategie kann einen Zeitstempel für das Auslaufen im HTTP-Header enthalten. Stillgelegte APIs sollten einen HTTP 410-Statuscode zurückgeben und eine End-of-Life-Antwortmeldung beinhalten.

Bevor eine API in Betrieb genommen wird, sollten Vorkehrungen getroffen werden, um:

- Die interne Entwicklung und das Testen der APIs zu gewährleisten
- Das Release-Management zu organisieren
- Das Dienstleistungsniveau für die Kund*innen zu definieren
- Die Anwendungsentwickler*innen beim Onboarding und der Nutzung der API zu unterstützen, inkl.:

o Lebenszyklus- und Änderungsmanagement o API Versionierung o Incident Management Das API Lebenszyklusmanagement kann durch die ITIL-Standardpraktiken Service Design, Service Transition und Service Operation dargestellt werden.

3.2.2. Service Design

Das Ziel einer API ist es, eine sichere und zuverlässige Kommunikation auf der Grundlage von Standards zu ermöglichen. Daher muss jede von einer österreichischen Behörde erstellte und zur Verfügung gestellte API funktionale und nicht-funktionale Anforderungen an Sicherheit, Benutzerfreundlichkeit, Verlässlichkeit, Testbarkeit und Skalierbarkeit erfüllen. Um sicherzustellen, dass oben genannte Anforderungen erfüllt werden, müssen sorgfältige Designüberlegungen angestellt und den gesamten API-Lebenszyklus über befolgt werden:

- Die meisten Behörden haben eine Vielzahl von Altsystemen, die weiterhin unterstützt und gewartet werden müssen. Die Entwicklung von APIs sollte nicht von den Anforderungen der Altsysteme abhängig gemacht werden, sondern zukunftsorientiert sein und zukünftigen Anforderungen entsprechen.
- Eine API sollte nach Möglichkeit so erstellt werden, dass diese auch behördenintern, beispielsweise in Form eines Web Services, verwendet werden kann. Dies bietet den Vorteil, dass sie vor der Bereitstellung intern getestet werden kann.
- Für identische Informationen sollten einheitliche Datenmodelle verwendet werden, welche vorab definiert werden sollten.
- Es sollte sichergestellt sein, dass eine API nur für genau eine Aufgabe und Anforderung erstellt wird. Dabei sollten eine einfache Erweiterbarkeit und eine ausreichende Leistungsfähigkeit im Vordergrund stehen, um sicherzustellen, dass alle Anforderungen erfüllt werden können.
- Bei der Durchführung der für den Entwurf und die Entwicklung einer API erforderlichen Geschäftsprozessanalyse sollten alle Komponenten der Referenzarchitektur berücksichtigt werden. Dadurch wird sichergestellt, dass die Bedürfnisse aller Beteiligten erfüllt werden.
- Bei der Anforderungsdefinition und Schnittstellengestaltung sollte mit den Benutzer*innen zusammengearbeitet werden, um sicherzustellen, dass die API ihren Bedürfnissen entspricht. Dies kann Innovation fördern und die Interaktion der Entwickler*innen mit der öffentlichen Verwaltung verbessern.
- Die Schnittstellenspezifikation sollte erstellt werden, bevor mit der Entwicklung begonnen wird. Es reicht, wenn diese den Zweck und Inhalt der API beinhaltet und nur wenige Seiten umfasst. Dies birgt den Vorteil, dass potenzielle Anwendungsentwickler*innen anhand der Spezifikation beurteilen können, ob die API ihren Anforderungen entspricht oder ob weitere Informationen erforderlich sind.
- Anwendungsentwickler*innen müssen wissen, wie lange eine API existieren wird, welche Verpflichtungen hinsichtlich ihrer Verfügbarkeit und Leistung bestehen und welche Unterstützung den API-Benutzer*innen angeboten wird. Dementsprechend sollten Service Level Agreements (SLAs) definiert werden, um Anforderungen an Verfügbarkeit und Leistung festzulegen, da ein solides Service-Level-Management helfen kann, die Akzeptanz und das Vertrauen in staatliche APIs zu erhöhen.
- Es ist darauf zu achten, sichere Entwicklungspraktiken anzuwenden, die nach Möglichkeit in Einklang mit den OWASP Secure Coding Prinzipien und anderen OWASP Praktiken und Cheat Sheets, wie dem OWASP API Security Project oder dem OWASP Top Ten Cheat Sheet, stehen.
- Bestehende APIs sollten regelmäßigen Sicherheitsanalysen unterzogen werden, um sicherzustellen, dass diese die Designanforderungen einhalten.

3.2.3. Service Transition

Für die Implementierung der API und deren Bereitstellung für die operative Verwendung sollten folgende Prinzipien beachtet werden:

- Die Versionsnummer einer API sollte sich aus einer Hauptversionsnummer, einer Nebenversionsnummer, einer Revisionsnummer und einer Buildnummer zusammensetzen. Die Hauptversionsnummer bezeichnet hierbei signifikante Änderungen, die oftmals mit einer fehlenden Rückwärtskompatibilität einhergehen, die Nebenversionsnummer kennzeichnet funktionale Erweiterungen, die Revisionsnummer Fehlerbehebungen und die Buildnummer den Fortschritt einer Entwicklung. Es ist sinnvoll, die jeweilige Versionsnummer sowohl in den API-Aufruf als auch in die begleitende Dokumentation einzupflegen.

```
<Hauptversionsnummer>.<Nebenversionsnummer>.<Revisionsnummer>.<Buildnummer>
```

- APIs sollten stabil sein und eine geringe Änderungsgeschwindigkeit aufweisen, da Anwendungsentwickler*innen oft nicht alle neuen Funktionen oder Änderungen sofort implementieren können. Dementsprechend ist es wichtig, dass kleinere Anpassungen immer als vollständig rückwärtskompatible Upgrades bereitgestellt werden. Werden größere Anpassungen gemacht, sollte für einen angemessenen Zeitraum sowohl die neue als auch die alte API-Version bereitgestellt werden, um genug Zeit für die Umstellung zu lassen. Dabei muss kommuniziert werden, ab wann die alte Version endgültig außer Betrieb genommen und der Support eingestellt wird. Anwendungsentwickler*innen sollten zudem rechtzeitig über die Notwendigkeit und den Zeitplan für den Rollout von Notfall-Patches informiert werden.

- APIs sollten iterativ und unter Zuhilfenahme von Agilen Praktiken entwickelt werden, um zu verhindern, dass Fehler zu spät gefunden werden. Sobald die Schnittstellenspezifikation erstellt wurde, sollte daher eine Applikation entwickelt und diese für Feedback veröffentlicht werden, anstatt zu versuchen, eine vollständige API zu entwickeln und veröffentlichen.

- Um die Akzeptanz auf Seiten der Anwendungsentwickler*innen zu fördern, sollte eine API gut dokumentiert und einfach auffindbar sein. Aufgrund der dynamischen Natur von APIs sollte eine API Dokumentation nicht statisch, also als PDF-Dokument, sondern dynamisch bereitgestellt werden. Eine Möglichkeit, eine dynamische Dokumentation zu generieren ist es, API-Spezifikationen auf Basis von OpenAPI 3 zu erstellen. Die OpenAPI 3 Spezifikation bietet jedoch nur die Möglichkeit, API-Endpunkte und deren Struktur zu dokumentieren. Wenn z.B. Use Cases dokumentiert werden sollen, müssen diese aufgrund der eingeschränkten Unterstützung außerhalb der OpenAPI-Spezifikation hinzugefügt werden.

- Zur Veröffentlichung der APIs sollte für Österreich ein API Katalog erstellt werden, welcher einen Überblick über die APIs des öffentlichen Sektors bietet. Zusätzlich können API-Kataloge des privaten Sektors, wie ProgrammableWeb, verwendet werden. Langfristig sollte die Erkennung von APIs automatisch und dynamisch erfolgen, sodass die Ressourcen in Echtzeit integriert und den Kunden präsentiert werden können.

- Die API sollte während ihres gesamten Lebenszyklus verwaltet werden. Dementsprechend ist es wichtig, dass die Leistung der API überwacht wird, um sicherzustellen, dass das angebotene SLA eingehalten wird. Auch die Nutzung der API sollte überwacht werden, um sicherzustellen, dass eine

API-Version erst außer Betrieb genommen wird, wenn sie von der Mehrzahl der Benutzer*innen nicht mehr verwendet wird. Eine Überwachung der API kann auch dazu dienen, nicht genutzte Funktionen zu eruieren und in weiterer Folge zu entfernen.

- Das Testen der API sollte Teil des Entwicklungsprozesses sein und regelmäßig durch automatische und manuelle Tests, wie beispielsweise Usability-Tests, durchgeführt werden.

3.2.4. Service Operation

Bei der Verwaltung der APIs und deren zugrunde liegenden Anwendungen ist folgendes zu beachten:

- Anwendungsentwickler*innen sollten zunächst eine Registrierung, beispielsweise an einem Entwicklerportal, vornehmen müssen, bevor sie die API nutzen können:

o Wenn die API Zugriff auf öffentliche Daten bietet, sollte die Registrierung durch Anwendungsentwickler*innen automatisch genehmigt und Zugangsdaten zur Verfügung gestellt werden.

o Wenn die API Zugriff auf sensible Daten bietet, sollte die Registrierung für Anwendungsentwickler*innen nicht automatisch genehmigt werden, sondern einen von der Behörde definierten Due-Diligence-Workflow in Gang setzen und zu einer für die Anwendung relevanten Zugriffsberechtigung führen.

o Ist die API nur für Entwicklungs-/Testzwecke bestimmt, sollte die Registrierung als Self-Service erfolgen und dem*der Anwendungsentwickler*in die Möglichkeit gegeben werden, die API in einer Sandbox auszuprobieren. Hierbei ist zu klären, ob alle Entwickler*innen mit denselben Testdaten arbeiten können oder ob individuelle Testdaten benötigt werden. In jedem Fall darf die API natürlich keine personenbezogenen Daten bereitstellen o Bietet die API die Möglichkeit zur Datenänderung, sollte die Registrierung eines*einer Kund*in die Ausstellung geeigneter Anmeldeinformationen sowie die Festlegung von Zugangskontrollen (Autorisierung) seitens des*der API Anbieter*in umfassen.

- Die API sollte einen guten Support bieten, sodass die Akzeptanz der APIs gewährleistet und der Zugang zu öffentlichen Informationen gefördert wird. Dies sollte die zeitnahe Reaktion auf Feedback und Fehlerberichte, rasche Bearbeitung von Änderungswünschen und Anfragen für neue APIs, telefonische Unterstützung durch einen technischen Support, die Bereitstellung von Prototyping-Tools und Entwicklungsunterstützung, die Bereitstellung von Sandbox-APIs, die Erstellung einer SDK zur Unterstützung der APIs einer Behörde und die Bereitstellung automatisierter Onboarding-Prozesse, da manuelle Prozesse die Akzeptanz einschränken können, beinhalten.

- Um die Auswirkungen von Ereignissen und ungeplanten Vorfällen so gering wie möglich zu halten, ist es wichtig, diese zeitnah zu erkennen und Abhilfe zu schaffen. Dementsprechend sollte das gesamte System überwacht werden und, wenn nötig, Gegenmaßnahmen ergriffen werden. Weitere Maßnahmen im Bereich des Incident und Event Management umfassen die Erfassung von Berichten über eingetretene Vorfälle samt zeitnaher Information der Anwendungsentwickler*innen, die Bereitstellung von Quick Fixes sowie die Unterrichtung der Anwendungsentwickler*innen über die Behebung der Störungen.

- Alle von einer Behörde veröffentlichten APIs sollten in einer einheitlichen Art und Weise verwaltet werden. Daher sollten regelmäßige Analysen durchgeführt werden, ob eine API ihre Verfügbarkeitsanforderungen erfüllt oder ob Maßnahmen, wie eine Drosselung oder eine Reduzierung der erlaubten Aufrufe pro Zeiteinheit, erforderlich sind, um für alle Benutzer den Zugriff im Rahmen des SLA zu ermöglichen. Überlastete APIs oder Engpässe sollten erkannt und die Verfügbarkeit dynamisch entsprechend der Nachfrage erhöht werden können.

- Während des Betriebs einer API sollten laufend analytische Daten, wie Nutzungszahlen, Leistungskennzahlen – wie beispielsweise Fehlerrate, Antwortzeit, Cache-Leistung oder Transaktionsgeschwindigkeit –, Verhalten bei einem Vorfall, Diagnosedaten oder Informationen über die verwendeten Gerätetypen erhoben werden, um Informationen zu gewinnen, die für die Reaktion auf Veränderungen der Nachfrage nützlich sein können.

3.3. (Sicherheits-)technische Vorgaben zum REST-API Design

Die folgenden Grundsätze sollten beim Entwurf von Sicherheitskonzepten für APIs beachtet werden:

- Beim Entwurf einer API sollte davon ausgegangen werden, dass diese irgendwann über das öffentliche Internet zugänglich sein wird, auch wenn dies im Moment nicht geplant ist.

- Least Privilege - Zugang und Autorisierung sollten den API-Nutzer*innen auf Basis des geringstmöglichen Umfangs zugewiesen werden, den sie zur Ausführung der erforderlichen Funktionen benötigen.

- Es sollte davon ausgegangen werden, dass Angreifer die API potenziell böswillig ausnutzen werden, insbesondere durch die Verwendung von:

o PUTs und POSTs - die interne Daten ändern und für Angriffe oder Fehlinformationen verwendet werden könnten o DELETEs - die dazu verwendet werden könnten, den Inhalt eines internen Ressourcenspeichers zu entfernen

3.3.1. Namenskonvention

Eine REST-API stellt verschiedene Ressourcen, wie beispielsweise Daten oder Dateien, zur Verfügung. Jede Ressource wird dabei eindeutig durch einen Uniform Resource Identifier (URI) identifiziert, der einem bestimmten Format entsprechen sollte, um Einheitlichkeit zu gewährleisten:

```
URI = https://domäne/apiName/v{majorVersion}/ressourcen?query
```

Jede URI sollte die folgenden Regeln befolgen:

- Die URI-Notation sollte lowerCamelCase verwenden, um die Lesbarkeit zu verbessern und zusammengesetzte Namen zu trennen.

- Nachgestellte Schrägstriche dürfen nicht verwendet werden.

- Die URI sollte keine Dateierweiterungen enthalten.

- Wenn ein einzelner Abfrageparameter mehrere Werte haben kann, sollte der Parameter für jeden Wert wiederholt werden.
- Alle APIs MÜSSEN über HTTPS verfügbar gemacht werden.

domäne Die Domäne, in der der Dienst bereitstellt wird. Im Idealfall enthält der Domänenname bereits einen Hinweis darauf, dass es sich beim angebotenen Dienst um eine API handelt.

apiName Name der API, geschrieben in Kleinbuchstaben.

v{majorVersion} Die Version der API, auf die die Verbraucher zugreifen möchten. Die Version sollte in der URI als vN angegeben werden, mit der Hauptversion (N) als Ganzzahl und v als Präfix. Nebenversionsnummern sollten nicht angeben werden.

ressourcen Pfad zu den von der API zur Verfügung gestellten Ressourcen. Ressourcen sollten in Kleinbuchstaben geschrieben, kurz, einfach und klar verständlich sein und auf Nomen basieren. Unterressourcen müssen nach der Ressource angezeigt werden, auf die sie sich beziehen.

query Die Query einer URI kann dazu verwendet werden, eine Antwort zu filtern.

3.3.2. JSON

Nach Möglichkeit sollte JSON als Darstellungsform verwendet werden.

3.3.3. HTTP Methoden

Der Zugriff auf REST-APIs sollte über die folgenden, im Einklang mit dem W3C Standard stehenden Standard-HTTP-Methoden erfolgen. Anfragen über andere HTTP-Methoden sind abzulehnen:

GET wird verwendet, um die Darstellung einer einzelnen Ressource oder einer Sammlung von Ressourcen (z.B. Kunden) zu lesen oder abzurufen.

GET ist sicher und kann ohne das Risiko einer Datenänderung oder -beschädigung aufgerufen werden, da es sich um eine strikt schreibgeschützte Methode handelt, die den Zustand der Ressource niemals ändern sollte. Zudem ist GET idempotent, was bedeutet, dass mehrere identische Anfragen das gleiche Ergebnis haben wie eine einzelne Anfrage.

POST wird am häufigsten verwendet, um eine neue Ressource zu erstellen. War die Erstellung der Ressource erfolgreich, sollte POST einen HTTP-Status 201 als Antwort zurückgeben, der im Location-HTTP-Header einen Link zur neu erstellten Ressource enthält.

POST ist weder sicher noch idempotent, da im Falle von zwei identischen POST-Anfragen höchstwahrscheinlich zwei unabhängige Ressourcen mit denselben Informationen erstellt werden.

PUT wird meistens dazu verwendet, eine gespeicherte Ressource unter der bereitgestellten URI zu aktualisieren. Bei einer erfolgreichen Aktualisierung gibt PUT den HTTP-Status 200 zurück. Ein weiterer Anwendungsfall für PUT ist die Erstellung einer Ressource, sofern die Ressourcen-ID vom Client und nicht

von der API selbst gewählt wird. In diesem Fall prüft der Server, ob die URI auf eine bereits vorhandene Ressource verweist und erstellt, wenn dies nicht der Fall ist, eine neue Ressource unter dieser URI. War die Erstellung erfolgreich, wird der HTTP-Status 201 zurückgegeben.

PUT ist nicht sicher, da diese Operation den Zustand einer Ressource auf dem Server ändert. PUT ist jedoch idempotent, da eine Ressource, die mit PUT erstellt oder aktualisiert wird, nach jedem PUT-Aufruf den gleichen Zustand hat. Es empfiehlt sich, GET für eine Ressource auszuführen, bevor PUT ausgeführt wird, um sicherzustellen, dass die konsumierende Anwendung über die neueste Darstellung dieser Ressource verfügt.

DELETE wird verwendet, um eine durch eine URI identifizierbare Ressource zu löschen. Wenn das Löschen erfolgreich ist, wird der HTTP-Status 200 oder 204 zurückgeben.

DELETE ist nicht sicher, da durch diese Operation die Ressource gelöscht wird. Dementsprechend kann nach einem erfolgreichen DELETE die Ressource von Clients nicht mehr gefunden werden, weswegen jeder zukünftige Versuch, die Zustandsdarstellung der gelöschten Ressource mithilfe von GET oder HEAD abzurufen, zu einem 404-Status führen muss, der von der API zurückgegeben wird.

DELETE ist jedoch idempotent, da ein wiederholtes Aufrufen von DELETE für eine bestimmte Ressource zum gleichen Ergebnis führt: Die Ressource ist weg.

OPTIONS wird verwendet, um Metadaten abzurufen, die die verfügbaren Interaktionen mit einer Ressource beschreiben.

HEAD wird verwendet, um die HTTP-Antwortheader abzurufen, welche die Metadaten zu einer Ressource enthalten. HEAD ist vergleichbar mit GET und somit sicher und idempotent.

PATCH wird verwendet, um die Darstellung einer Ressource teilweise zu aktualisieren, wird jedoch möglicherweise nicht von allen Komponenten unterstützt.

PATCH ist nicht sicher, da durch die Anwendung eines PATCH neue Ressourcen erstellt oder vorhandene geändert werden können. PATCH-Operationen müssen zudem nicht idempotent sein, werden jedoch häufig in der Praxis also solche eingesetzt.

3.3.4. HTTP Header

HTTP Header sollten im Einklang mit RFC2616 der W3C erstellt werden. In benutzerdefinierten HTTP-Headern sollte der Name der jeweiligen Organisation oder Behörde vorgestellt werden, die Verwendung ist jedoch nicht empfehlenswert. Es wird dringend empfohlen, dass alle APIs einen Autorisierungsheader mit einem der folgenden Werten unterstützen:

- API Key
- Basic Auth (APIKey + Secret)
- Benutzername + Passwort
- Inhaber {token}

3.3.5. Fehlerbehandlung

Um sicherzustellen, dass API-Clients auf Fehler angemessen reagieren können, ist es wichtig, Vorgaben zur Fehlerbehandlung zu machen. Informationen zu aufgetretenen Fehlern sollten informativ und sowohl für Menschen als auch Maschinen lesbar sein. Im Falle eines Fehlers sollte der Antworttext Folgendes enthalten:

- Den passenden HTTP-Statuscode
- Einen API-spezifischen Fehlercode
- Eine für Menschen lesbare Fehlermeldung

HTTP-Statuscodes Die Verwendung der im RFC2616 definierten HTTP Status Codes wird empfohlen, um die Fehlerbehandlung zu vereinfachen. Eigens definierte Status-Codes sollten nicht verwendet werden.

- **1xx Informationen:** Die Anfrage wird noch bearbeitet.
- **2xx Erfolgreiche Operation:** Die Anfrage des Clients wurde erfolgreich empfangen, verstanden und akzeptiert.
- **3xx-Weiterleitung:** Um den Erfolg der Anfrage sicherzustellen, müssen weitere Maßnahmen auf Seiten des Clients ergriffen werden.
- **4xx Client-Fehler:** Die Anfrage enthält eine fehlerhafte Syntax oder kann nicht erfüllt werden, der Fehler liegt meist im Verantwortungsbereich des Clients.
- **5xx Serverfehler:** Der Server konnte eine scheinbar gültige Anfrage nicht erfüllen.

API-spezifische Fehlercodes Fehlercodes sollten ausreichend informativ sein, sodass die*der API-Nuzer*in angemessen auf den Fehler reagieren kann, aber nicht zu viel preisgeben. Eine Möglichkeit, dies zu erreichen, besteht darin, einen API-spezifischen Fehlercode in der Antwort an den Consumer zurückzugeben. Im Falle eines Fehlers kann das Support-Team dann diesen Code nachschlagen und genau feststellen, was schiefgelaufen ist und wer es beheben muss.

Für Menschen lesbare Fehlermeldung Die für den Menschen lesbare Fehlermeldung sollte ausreichend informativ sein, ohne zu viele technische Details anzugeben (z.B. „Ein Konto mit dieser ID existiert bereits“). In der Fehlermeldung dürfen keine Systeminformationen preisgegeben werden, um zu verhindern, dass Schwachstellen ausgenutzt werden. Zudem sollten sensible Informationen, wie der Username, weder bestätigt noch abgelehnt werden, da dies potenzielle Angreifer*innen darüber informiert, welche Kriterien richtig eingegeben wurden.

3.3.6. Auditlogs

Vor und nach sicherheitsrelevanten Ereignissen sollten Audit-Logs erstellt werden. Die Logs sollten vor unbefugtem Zugriff und unerlaubter Veränderung geschützt werden.

3.3.7. Transportsicherheit

Folgende Best-Practice-Standards sollten in Bezug auf die Sicherheit der Datenübertragungen beachtet werden.

- Für die Datenübertragung sollte HTTPS mit TLS 1.2 verwendet werden. Die Verwendung von TLS 1.0 oder 1.1 wird nicht empfohlen.
- Alle öffentlich zugänglichen Endgeräte sollten ein digitales Zertifikat verwenden, das von einer zugelassenen Zertifizierungsstelle signiert wurde. Intern zugängliche Endgeräte können auch selbstsignierte digitale Zertifikate verwenden.
- HTTP-Datenverkehr sollte nicht auf HTTPS umgeleitet werden, sondern ist abzulehnen.
- Nicht verwendete HTTP-Methoden sollten deaktiviert werden und Errror 405 zurückgeben.
- Alle Anfragen sollten validiert werden.
- Eine wechselseitige Authentifizierung sollte sowohl zwischen dem Verbraucher und dem API-Gateway als auch zwischen dem API-Gateway und der Back-End-API durchgeführt werden.
- Der Payload sollte während der Übertragung verschlüsselt werden.

3.3.8. Authentifizierung und Autorisierung

Für die Absicherung von APIs ist eine Authentifizierung erforderlich, um die Verbraucher*innen und/oder die Anwendungen zu identifizieren, die auf eine API zugreifen oder sie nutzen wollen. Die Authentifizierung ermöglicht es den API-Anbieter*innen, alle Nutzer*innen einer API zu identifizieren und zu bestätigen, dass die Nutzer*innen, die den Zugriff beantragen, die sind, für die sie sich ausgeben. Dies sollte die Nutzer*innen jedoch nicht automatisch zum Zugriff auf die APIs oder die zugrunde liegenden Ressourcen berechtige, sondern die Möglichkeit bieten, unterschiedliche Service-Levels für verschiedene Kund*innen zu implementieren. So kann z. B. für gewerbliche Kund*innen ein höheres Abfragelimit pro Tag gelten als für Kund*innen, die nicht für den Dienst zahlen. Auch die Registrierung von Anwendungsentwickler*innen bietet Vorteile, wie beispielsweise, dass diese Bedingungen unterschreiben müssen, in denen festgelegt ist, wie sie die von der API erhaltenen Daten nutzen dürfen.

Daher sollte jede Behörde eines oder mehrere der folgenden Authentifizierungsverfahren verwenden, unabhängig davon, ob die API von einem Menschen oder einem System genutzt wird.

Username und Passwort: Der Einsatz einer Authentifizierungsmethodik, die rein auf Username und Passwort basiert, ist insbesondere für produktive APIs nicht empfehlenswert und sollte auf Testund Entwicklungszwecke beschränkt oder durch den Einsatz von API-Schlüssel abgelöst werden, da dieses Modell, auch wenn es einfach zu implementieren ist, viele nicht zu vernachlässigbare Nachteile hat:

- Ein vollständiger Registrierungsprozess für alle Usertypen (Anwendungen, Entwickler*innen) sowie ein Identitätsspeicher, wie z.B. LDAP ist erforderlich.

- Die Nutzung eines föderierten Authentifizierungsmodells, wie beispielsweise Single-Sign-On ist nicht möglich, weswegen bei jeder Nutzung die erneute Eingabe von Username und Passwort erforderlich ist.
- Es besteht die Gefahr, dass das Passwort im Klartext gespeichert wird.
- Die Anfälligkeit für Brute-Force Angriffe ist hoch
- Passwörter haben eine geringe Entropie, müssen zurückgesetzt und verwaltet werden und sind nur schwer auf einer einzelnen Ebene zu widerrufen.

API-Schlüssel: API-Schlüssel sollten überall dort verwendet werden, wo eine Authentifizierung zwischen Systemen und einer produktiven API notwendig ist. Sie sollten für jede Anfrage gefordert und wiederrufen werden, wenn der Client gegen die Nutzungsvereinbarung verstößt. API-Schlüssel sind in der Regel eindeutig und können vom API-Anbieter einer Anwendung, einem Entwickler oder einem Benutzer zugewiesen werden. Ein Vorteil von API-Schlüsseln ist, dass sie die Auswirkungen von Denial-of-Service-Angriffen reduzieren können, dennoch ist es ratsam, sich nicht ausschließlich auf API-Schlüssel zu verlassen, um sensible oder kritische Ressourcen zu schützen.

Sie haben folgende Bestandteile:

- API-Schlüssel: öffentlicher eindeutiger Bezeichner, bestehend aus einer Zeichenkette mit mehr als 40 zufälligen Zeichen zur Authentifizierung der Anwendung gegenüber der API
- API-Secret: private eindeutige Kennung, die nur dem API-Gateway bekannt ist und zur Validierung des API-Schlüssels verwendet wird.

Möchte eine Anwendung eine API-Ressource verwenden, übergibt diese den API-Schlüssel automatisch an die API. Das API-Gateway validiert anschließend den API-Schlüssel anhand des API-Schlüsselspeichers (der Teil der API-Gateway-Funktionalität sein oder von einem anderen sicheren Gerät bereitgestellt werden kann), bevor es der Anwendung den Zugriff auf die angeforderte API samt Backend-Ressourcen gestattet.

Der Vorteil gegenüber Autorisierungsmodellen, die auf Username und Passwort basieren, ist, dass API-Schlüssel nicht mit Usern verknüpft sind und keine kryptografischen Funktionen erfordern. Zudem sind sie aufgrund der höheren Entropie sicherer als Passwörter und schneller, da sie kein Hashing-Verfahren erfordern.

Ein Risiko beim Einsatz von API-Schlüssel besteht darin, dass jede Person, die eine Kopie des API-Schlüssels besitzt, diesen so verwenden kann, als wäre sie dafür befugt. Daher sollte die gesamte Kommunikation über TLS erfolgen, um den Schlüssel während der Übertragung zu schützen. Es liegt in der Verantwortung der Anwendungsentwickler*innen, ihre Kopie des API-Schlüssels angemessen zu schützen; wird der API-Schlüssel in die Anwendung eingebettet ist, kann dieser dekompiliert und extrahiert werden, wird er in reinen Textdateien gespeichert, kann er gestohlen und für böswillige Zwecke wiederverwendet werden.

Single-Sign-On : Standardmäßig sollte die Benutzerauthentifizierung durch einen föderierten Identitätsanbieter (IdP) zentralisiert werden, der Zugriffstoken ausstellt. Der Vorteile dieser Authentifizierungsmethode ist, dass Nutzerinformationen innerhalb eines sicheren Bereiches für andere

Organisationen zur Verfügung gestellt werden können. Damit wird ein Domainübergreifendes Single-Sign-On möglich, und Behörden werden von der Notwendigkeit entbunden, Anmeldedaten von Nutzern zu verwalten. Die gesamte Kommunikation muss über TLS erfolgen, um Vertraulichkeits- und Integritätskontrollen bereitzustellen.

Eine Möglichkeit, Single-Sign-On zu bewerkstelligen, ist die Verwendung von **OAuth2** . Dabei werden die Benutzer*innen zur Anmeldung auf einen Authorization Server weitergeleitet, welcher Zugriff auf zentrale Benutzerkonten besitzt. Hat sich der/die Benutzer*in dort angemeldet, erhält der Client einen sogenannten Access-Token, der ihm im Namen des/der Benutzer*in Zugriff auf die Services im Backend – sogenannte Ressource Server – gibt. Der Access-Token informiert den Ressource Server unter anderem über den entsprechenden Benutzer*innen sowie über die Rechte, die der Client wahrnehmen darf. Zusätzlich finden sich im Token meist auch Metadaten, wie der Aussteller, das Ausstellungsdatum oder die Gültigkeitsdauer.

OpenID Connect kann ergänzend zu OAuth2 verwendet werden und definiert unter anderem, die der Client Informationen über die Benutzer*innen erhalten kann. Dazu stellt es im Wesentlichen Identitätsinformationen in Form eines ID-Tokens bereit, welchen der Client zusätzlich zum Access Token erhalten kann. Während der Access-Token zum Zugriff auf das Backend bestimmt ist, kann der Client aus dem ID-Token direkt Informationen über den/die Benutzer*in entnehmen. Im Gegensatz zu Access-Tokens bei OAuth 2 ist der Aufbau von ID-Tokens vorgegeben. Es handelt sich dabei immer um einen JSON Web Token, welcher signiert und/oder verschlüsselt sein kann. Security Assertion Markup Language (**SAML**) ist ein offener XML-Standard der zur Authentifizierung und Autorisierung von Daten zwischen einem Identitätsanbieter und einem Dienstanbieter verwendet wird. Die Übertragung von Identitätsinformationen wird durch das SAML-Protokoll ermöglicht. **SAML** funktioniert ähnlich wie OpenID Connect, hat aber den Nachteil, dass keine Aktualisierungstoken bereitgestellt werden.

Referenzen

[1] Stadt Wien, „Meldung von Anwendungen im Portalverbund. E-Government - Schnittstellen und Basisfunktionen," [Online]. Available: https://neu.ref.wien.gv.at/at.gv.wien.ref-live/ag-izoeffentliche-informationen#PAI. [Zugriff am 05 Juli 2021].

[2] L. Vaccari, M. Posada Sanchez, M. Boyd, D. Gattwinkel, D. Mavridis, R. Smith, M. Santoro, S.

Nativi, M. Medjaoui, I. Reusa, S. Switzer und A. Friis-Christensen, Application Programming Interfaces in Governments: Why, what and how, Luxembourg: Publications Office of the European Union, 2020.

[3] T. M. Harrison, S. Guerrero, B. G. Burke, M. Cook, A. Cresswell, N. Helbig, J. Hrdinová und T.

Pardo, „Open Government and E-Government: Democratic Challenges from a Public Value Perspective," *Information Polity,* Bd. 17, Nr. 2, pp. 83-97, 2011.

[4] New Zealand Government, „API Standard and Guidelines. Part A - Business," Oktober 2016.

[Online]. Available: https://snapshot.ict.govt.nz/resources/digital-ictarchive/static/localhost_8000/assets/Standards-and-Compliance/API-Standard-and-Guidelines/API-Standard-and-Guidelines-Part-A-Business-v1.0-October-2016.pdf. [Zugriff am 08 07 2021].

[5] M. Santoro, L. Vaccari, D. Mavridis, R. Smith, M. Posada Sanchez und D. Gattwinkel, „Web Application Programming Interfaces (APIs): general-purpose standards, terms and European Commission initiatives," Publications Office of the European Union, Luxemburg, 2019.

[6] NIST, „Application Programming Interface (API) - Glossary | CSRC," [Online]. Available:

https://csrc.nist.gov/glossary/term/Application_Programming_Interface. [Zugriff am 09 07 2021].

[7] K. Goetsch, APIs for Modern Commerce, O'Reilly Media, 2017.

[8] M. Massé, REST API Design Rulebook, Sebastopol: O'Reilly Media, 2012.

[9] A. Soni und V. Ranga, „API Features Individualizing of Web Services: REST and SOAP,"

International Journal of Innovative Technology and Exploring Engineering (IJITEE), Bd. 8, Nr. 9S, pp. 664-671, 2019.

[10] Red Hat, „REST oder SOAP?," [Online]. Available:

https://www.redhat.com/de/topics/integration/whats-the-difference-between-soap-rest. [Zugriff am 07 07 2021].

[11] S. Newman, Building Microservices, Sebastopol,: O'Reilly Media, 2015:.

[12] Amtsblatt der Europäischen Union, „Richtlinie (EU) 2019/1024 des Europäischen Parlaments und des Rates vom 20. Juni 2019 über offene Daten und die Weiterverwendung von Informationen des öffentlichen Sektors," 20 Juni 2019. [Online]. Available: https://eur-lex.europa.eu/legalcontent/DE/TXT/?qid=1561563110433&uri=CELEX:32019L1024.

[13] Bundesministerium Digitalisierung und Wirtschaftsstandort, „Open Government Data (OGD),"

[Online]. Available: https://www.bmdw.gv.at/Themen/Digitalisierung/Verwaltung/Open-Government-Data-(OGD).html. [Zugriff am 02 August 2021].

[14] Capgemini Invent, „Open Data Maturity Report 2020," Publications Office of the European Union, Luxembourg, 2020.

[15] Bundesministerium für Digitalisierung und Wirtschaftsstandort, „Handbuch data.cockpit V 1.8," 01 März 2019. [Online]. Available: https://www.data.gv.at/wp-content/uploads/Benutzerhandbuchdatagvat-Cockpit.pdf. [Zugriff am 02 August 2021].

[16] B. Krabina und B. Lutz, „Open-GovernmentVorgehensmodell. Version 3.0," 22 Juli 2016. [Online].

Available: https://www.kdz.eu/de/wissen/studien/open-government-vorgehensmodell. [Zugriff am 02 August 2021].

[17] G. Eibl, G. Hartmann, B. Krabina, B. Lutz, M. Mittlböck und G. Ramler, „Metadaten data.gv.at 2.5,"

01 März 2019. [Online]. Available: https://neu.ref.wien.gv.at/at.gv.wien.reflive/documents/20189/68315/Metadaten_data.gv.at_2.5_fin.pdf. [Zugriff am 02 August 2021].

[18] Europäische Kommission, „EU-eGovernment-Aktionsplan 2016-2020. COM(2016) 179 final,“ 19 April 2016. [Online]. Available: https://eur-lex.europa.eu/legalcontent/DE/TXT/?uri=CELEX:52016DC0179.

[19] ehealth, „PVP Standardportal - Stammportal - Login-Seite,“ [Online]. Available:

https://estp.ehealth.gv.at/force_login?selectedTab=help. [Zugriff am 28 06 2021].