



FACHHOCHSCHULE AACHEN, CAMPUS JÜLICH

FACHBEREICH 09 - MEDIZINTECHNIK UND TECHNOMATHEMATIK  
STUDIENGANG ANGEWANDTE MATHEMATIK UND INFORMATIK

BACHELORARBEIT

---

**Konzeption eines webbasierten  
Datenservices für die Einbindung in eine  
Flugsimulationsumgebung unter  
Berücksichtigung geltender  
Sicherheitsrichtlinien und -risiken**

---

*Autor:*

Sven Bergmann, 3231105

*Betreuer:*

Prof. Dr. rer. nat. Hans Joachim Pflug

Dipl. Ing. Christoph Hennecke

Stolberg, den 8. August 2022



## **Eidesstattliche Erklärung**

Hiermit versichere ich, dass ich die Bachelorarbeit mit dem Thema „Konzeption eines webbasierten Datenservices für die Einbindung in eine Flugsimulationsumgebung unter Berücksichtigung geltender Sicherheitsrichtlinien und -risiken“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Stolberg, den 8. August 2022

.....  
(Sven Bergmann)



# Inhaltsverzeichnis

|   |             |
|---|-------------|
| <b>Abbildungsverzeichnis</b>                    | <b>III</b>  |
| <b>Tabellenverzeichnis</b>                      | <b>V</b>    |
| <b>Verzeichnis der Programmlistings</b>         | <b>VII</b>  |
| <b>Akronyme</b>                                 | <b>IX</b>   |
| <b>Glossar</b>                                  | <b>XIII</b> |
| <b>1 Einleitung</b>                             | <b>3</b>    |
| <b>2 Grundlagen</b>                             | <b>5</b>    |
| 2.1 Internetreferenzmodell . . . . .            | 5           |
| 2.2 HTTP . . . . .                              | 6           |
| 2.3 TLS/SSL . . . . .                           | 7           |
| 2.3.1 TLS Handshake Protocol . . . . .          | 7           |
| 2.3.2 TLS Record Protocol . . . . .             | 8           |
| 2.4 Ausgewählte Absicherungsverfahren . . . . . | 8           |
| 2.4.1 API-Keys . . . . .                        | 9           |
| 2.4.2 OAuth 2.0 . . . . .                       | 9           |
| <b>3 Aufgabenstellung</b>                       | <b>15</b>   |
| 3.1 Obligatorische Anforderungen . . . . .      | 15          |
| 3.2 Optionale Anforderungen . . . . .           | 15          |
| <b>4 Analyse</b>                                | <b>17</b>   |
| 4.1 Datenhaltung . . . . .                      | 17          |
| 4.2 Zielsystem . . . . .                        | 17          |
| 4.2.1 Betriebssystem . . . . .                  | 17          |
| 4.2.2 Möglichkeiten des Log-Ins . . . . .       | 17          |
| 4.2.3 Mögliche Frameworks . . . . .             | 18          |
| 4.3 IT Grundschutz Kompendium . . . . .         | 19          |
| 4.3.1 APP.3.1 . . . . .                         | 20          |
| 4.4 OWASP Top 10 . . . . .                      | 25          |

|          |  |           |
|----------|--|-----------|
| <b>5</b> | <b>Realisierung</b>                    | <b>29</b> |
| 5.1      | Gesamtstruktur . . . . .               | 29        |
| 5.1.1    | Autorisierungsserver . . . . .         | 29        |
| 5.1.2    | Webservice . . . . .                   | 30        |
| 5.1.3    | Tests . . . . .                        | 32        |
| 5.1.4    | Demo Anwendung . . . . .               | 33        |
| 5.2      | Sicherheitsaspekte . . . . .           | 33        |
| 5.3      | Bedienung . . . . .                    | 35        |
| <b>6</b> | <b>Ergebnis und Schlussfolgerungen</b> | <b>37</b> |
| <b>7</b> | <b>Zusammenfassung und Ausblick</b>    | <b>39</b> |
|          | <b>Literaturverzeichnis</b>            | <b>41</b> |
| <b>A</b> | <b>Abbildungen</b>                     | <b>43</b> |
| <b>B</b> | <b>Tabellen</b>                        | <b>51</b> |
| <b>C</b> | <b>Listings</b>                        | <b>55</b> |
| C.1      | Authorization Server . . . . .         | 55        |
| C.2      | Webservice . . . . .                   | 57        |
| C.3      | WebserviceTests . . . . .              | 61        |
| C.4      | Demo Anwendung . . . . .               | 65        |

# Abbildungsverzeichnis

|     |   |    |
|-----|---|----|
| 2.1 | OAuth 2.0 Abstrakter Protokoll Fluss . . . . .            | 11 |
| 2.2 | OAuth 2.0 Refreshing an Expired Access Token . . . . .    | 12 |
|     |   |    |
| A.1 | Schichtenmodell . . . . .                                 | 43 |
| A.2 | Vergleich der Requestzeiten . . . . .                     | 44 |
| A.3 | Sequenzdiagramm einer Anfrage . . . . .                   | 45 |
| A.4 | Pseudodiagramm der relevanten Datenbanktabellen . . . . . | 46 |
| A.5 | HTTP-Methoden S. 1 . . . . .                              | 47 |
| A.6 | HTTP-Methoden S. 2 . . . . .                              | 48 |
| A.7 | HTTP-Methoden S. 3 . . . . .                              | 49 |





# Tabellenverzeichnis

|     |   |    |
|-----|---|----|
| B.1 | Kreuzreferenztable APP.3.1 . . . . .        | 51 |
| B.2 | .NET Core 2.1 vs. .NET Framework . . . . .  | 52 |
| B.3 | IT Sicherheitsanforderungstabelle . . . . . | 53 |



# Verzeichnis der Programmlistings

|      |  |    |
|------|--|----|
| C.1  | Program.cs in Autorisierungsserver . . . . .                 | 55 |
| C.2  | Startup.cs in Autorisierungsserver . . . . .                 | 55 |
| C.3  | Config.cs in Autorisierungsserver . . . . .                  | 56 |
| C.4  | Program.cs in Webservice . . . . .                           | 57 |
| C.5  | ConfigureServices in Webservice . . . . .                    | 57 |
| C.6  | Configure in Webservice . . . . .                            | 58 |
| C.7  | DataNotFoundException in Webservice . . . . .                | 59 |
| C.8  | ApiControllerBase.cs in Webservice . . . . .                 | 60 |
| C.9  | CountriesController.cs in Webservice . . . . .               | 60 |
| C.10 | AuthorizedClientFixture.cs in WebserviceTests . . . . .      | 61 |
| C.11 | CountriesSQLTest.cs in WebserviceTests . . . . .             | 62 |
| C.12 | CountriesWebTestAuthorized.cs in WebserviceTests . . . . .   | 63 |
| C.13 | CountriesWebTestUnauthorized.cs in WebserviceTests . . . . . | 64 |
| C.14 | Application.cs in der Demo Anwendung . . . . .               | 65 |
| C.15 | Token in der Demo Anwendung beantragen . . . . .             | 66 |
| C.16 | Datenabfrage in der Demo Anwendung . . . . .                 | 67 |



# Akronyme

|         |  |
|---------|--|
| API     | Application Programming Interface <i>Glossar:</i> <a href="#">API</a> , <a href="#">XV</a> , <a href="#">8</a> , <a href="#">9</a> , <a href="#">15</a> , <a href="#">30</a> , <a href="#">44</a>  |
| ARINC   | Aeronautical Radio Incorporated <i>Glossar:</i> <a href="#">ARINC</a> , <a href="#">32</a>   |
| BSI     | Bundesamt für Sicherheit in der Informationstechnik <a href="#">15</a> , <a href="#">19</a> , <a href="#">21</a> , <a href="#">24</a>  |
| CRUD    | Create Read Update Delete <a href="#">6</a> , <a href="#">15</a> , <a href="#">16</a> , <a href="#">39</a>   |
| DLL     | Dynamic Link Library <i>Glossar:</i> <a href="#">DLL</a> , <a href="#">17</a> ,  |
| FMS     | Flight Management System <i>Glossar:</i> <a href="#">FMS</a> , <a href="#">3</a> , <a href="#">37</a>  |
| FTP     | File Transfer Protocol <a href="#">6</a> , <a href="#">43</a>  |
| GUI     | Graphical User Interface <a href="#">18</a>  |
| HATEOAS | Hypermedia as the Engine of Application State <a href="#">15</a>   |
| HTML    | Hypertext Markup Language <i>Glossar:</i> <a href="#">HTML</a> ,   |
| HTTP    | Hypertext Transfer Protocol <a href="#">XVI</a> , <a href="#">3</a> , <a href="#">6</a> , <a href="#">7</a> , <a href="#">9</a> , <a href="#">16</a> , <a href="#">20</a> , <a href="#">30</a> , <a href="#">31</a> , <a href="#">32</a> , <a href="#">43</a> , <a href="#">45</a> |
| HTTPS   | Hypertext Transfer Protocol Secure <a href="#">9</a> , <a href="#">20</a> , <a href="#">29</a> , <a href="#">30</a> , <a href="#">43</a>   |
| ICAO    | International Civil Aviation Organization <i>Glossar:</i> <a href="#">ICAO</a> , <a href="#">32</a>  |
| ICMP    | Internet Control Message Protocol <a href="#">5</a>  |
| IEEE    | Institute of Electrical and Electronic Engineers <i>Glossar:</i> <a href="#">IEEE</a> , <a href="#">5</a> ,  |
| IETF    | Internet Engineering Task Force <i>Glossar:</i> <a href="#">IETF</a> , <a href="#">7</a> , <a href="#">9</a>   |

|       |   |
|-------|---|
| IIS   | Internet Information Services <i>Glossar:</i> <a href="#">IIS</a> , 19  |
| IP    | Internet Protocol <a href="#">5</a> , <a href="#">6</a> , <a href="#">28</a> , <a href="#">43</a>   |
| ISO   | International Standards Organisation <a href="#">5</a> , <a href="#">6</a>  |
| JDK   | Java Development Kit <i>Glossar:</i> <a href="#">JDK</a> , 27   |
| JSON  | JavaScript Object Notation <i>Glossar:</i> <a href="#">JSON</a> , <a href="#">31</a> , <a href="#">32</a> , <a href="#">35</a>  |
| JWT   | JSON Web Token <i>Glossar:</i> <a href="#">JWT</a> , <a href="#">25</a> , <a href="#">35</a> , <a href="#">45</a>   |
| LFID  | Luftfahrt Informations Datenbank <a href="#">3</a> , <a href="#">15</a> , <a href="#">17</a> , <a href="#">35</a> , <a href="#">39</a> , <a href="#">45</a>   |
| LINQ  | Language Integrated Query <i>Glossar:</i> <a href="#">LINQ</a> , <a href="#">18</a> , <a href="#">32</a> , <a href="#">33</a> , <a href="#">53</a>  |
| MAC   | Message Authentication Code <i>Glossar:</i> <a href="#">MAC</a> , 8   |
| ORM   | Object Relational Mapping <i>Glossar:</i> <a href="#">ORM</a> , 26  |
| OSI   | Open Systems Interconnection <a href="#">5</a> , <a href="#">6</a>  |
| OWASP | Open Web Application Security Project <a href="#">3</a> , <a href="#">4</a> , <a href="#">24</a> , <a href="#">25</a> , <a href="#">26</a> , <a href="#">39</a>   |
| REST  | Representational State Transfer <a href="#">3</a> , <a href="#">4</a> , <a href="#">9</a> , <a href="#">15</a> , <a href="#">20</a> , <a href="#">52</a>  |
| SID   | Standard Instrument Departure <i>Glossar:</i> <a href="#">SID</a> ,   |
| SMTP  | Simple Mail Transfer Protocol <a href="#">6</a> , <a href="#">43</a>  |
| SQL   | Structured Query Language <i>Glossar:</i> <a href="#">SQL</a> , <a href="#">XV</a> , <a href="#">17</a> , <a href="#">26</a> , <a href="#">31</a> , <a href="#">33</a> , <a href="#">34</a> , <a href="#">35</a> , <a href="#">37</a> , <a href="#">44</a> , <a href="#">45</a> |
| SSL   | Secure Sockets Layer <a href="#">7</a>  |
| SSMS  | SQL Server Management Studio <i>Glossar:</i> <a href="#">SSMS</a> , <a href="#">17</a>  |
| SSRF  | Server-Side-Request-Forgery <a href="#">28</a>  |
| STAR  | Standard Terminal Arrival Routes <i>Glossar:</i> <a href="#">STAR</a> ,   |
| TCP   | Transmission Control Protocol <a href="#">5</a> , <a href="#">6</a> , <a href="#">43</a>  |
| TLS   | Transport Layer Security <a href="#">3</a> , <a href="#">7</a>  |

|         |  |
|---------|--|
| UDP     | User Datagram Protocol <a href="#">5</a>   |
| URI     | Uniform Resource Identifier <i>Glossar:</i> <a href="#">URI</a> , <a href="#">XVI</a> , <a href="#">6</a> , <a href="#">28</a> , <a href="#">33</a>                    |
| URL     | Uniform Resource Locator <i>Glossar:</i> <a href="#">URL</a> , <a href="#">XVI</a> , <a href="#">25</a> , <a href="#">28</a> , <a href="#">32</a> , <a href="#">35</a> |
| URN     | Uniform Resource Name <i>Glossar:</i> <a href="#">URN</a> , <a href="#">XVI</a> ,  |
| WAF     | Web Application Firewall <i>Glossar:</i> <a href="#">WAF</a> , <a href="#">24</a>  |
| WPF     | Windows Presentation Foundation <i>Glossar:</i> <a href="#">WPF</a> , <a href="#">19</a>   |
| ZSimNav | Zentrum für Simulations- und Navigationsunterstützung Fliegende Waffensysteme der Bundeswehr <a href="#">3</a>   |





# Glossar

**Aeronautical Radio Incorporated (ARINC)** Eine – vor allem in der Luftfahrt – bekannte Firma, welche Standards für Protokolle und Datenformate veröffentlicht

**Application Programming Interface (API)** Satz von Befehlen, Protokollen und Funktionen, welche Entwickler für die Erstellung von Software zur Ausführung allgemeiner Operationen verwenden können. Diese Programmierschnittstelle stellt somit die Grundlage für die Kommunikation zwischen Anwendungen bereit.

**Authentifizierung** Stellt die Prüfung der behaupteten [Authentisierung](#) dar (Verifizierung der Identität)

**Authentisierung** Ein Nutzer legt Nachweise vor, welche dessen Identität bestätigen sollen (Behauptung einer Identität)

**Autorisierung** Nach erfolgreicher [Authentifizierung](#) werden spezielle Rechte an den Nutzer vergeben (Vergabe oder Verweigerung von Rechten)

**Brute-Force** Zu deutsch „rohe Gewalt“ was im Zusammenhang mit der Informatik häufig genutzt wird, um eine Problemlösung zu beschreiben, bei der keine effizienten Algorithmen bekannt sind, sondern einfach alle möglichen Lösungen durchprobiert werden

**CI/CD-Pipeline** Steht für „Contiguous Integration/Contiguous Delivery“-Pipeline, welche mehrere Schritte umfasst, die für die Bereitstellung einer neuen Softwareversion ausgeführt werden müssen

**Cipher Suite** Sammlung kryptografischer Algorithmen, welche für die Verschlüsselung von Nachrichten verwendet werden

**Cookie** Kleine Textdateien, welche Nutzerdaten beinhalten, die vom Browser gespeichert werden

**Dynamic Link Library (DLL)** Dynamische Programmbibliothek, welche vor allem unter Windows ausführbar ist

**Flight Management System (FMS)** elektronisches Hilfsmittel für Piloten zur Steuerung und Navigation

**Framework** Zu deutsch „Rahmenwerk“, was eine wiederverwendbare Struktur bereitstellt, welche für die Entwicklung verschiedener Programme genutzt werden kann

**Hypertext Markup Language (HTML)** Sprache zur Darstellung von Inhalten über einen Web Browser

**Institute of Electrical and Electronic Engineers (IEEE)** Weltweiter Berufsverband von Ingenieuren, Wissenschaftlern und Technikern, welcher vor allem für Standardisierungen und andere wissenschaftliche Veröffentlichungen bekannt ist

**International Civil Aviation Organization (ICAO)** Zu deutsch „Internationale Zivilluftfahrtorganisation“ ist eine Organisation, welche das Ziel hat, die internationale Luftfahrt zu fördern

**Internet Information Services (IIS)** Dienstplattform von Microsoft für die Bereitstellung von Daten, Dokumenten und Funktionen über Internetprotokolle

**Internet Engineering Task Force (IETF)** Organisation zur Verbesserung und Weiterentwicklung der Funktionsweise des Internets

**ISO/OSI-Referenzmodell** Modell für die Netzwerkprotokolle als Darstellung über eine Schichtenarchitektur, siehe auch Abbildung [A.1](#) auf Seite [43](#)

**Java Development Kit (JDK)** Zusammensetzung aus Java-Compiler, Java-Debugger und einigen anderen Entwicklungswerkzeuge, welche die Entwicklung von Java Applikationen ermöglichen

**JavaScript Object Notation (JSON)** Maschinenlesbare Sprache zur Darstellung von Objekten. Bei JavaScript kann dies unter anderem für eine Instanziierung neuer Objekte genutzt werden.

**JSON Web Token (JWT)** Token, welches Behauptungen (claims) über den Benutzer enthält. Es stellt sich in einer codierten Form dar und kann durch Applikationen decodiert und validiert werden

**Konstruktor Injektion** Wird als Methodik bezeichnet, die in [Frameworks](#) genutzt wird um Klassen oder Attribute zu kreieren, welche wiederum von der aufrufenden Klasse gebraucht werden

**Language Integrated Query (LINQ)** Sprachkonstrukt in C#, bei dem die Abfragen [SQL](#)-Befehlen ähneln

**Man-in-the-Middle Angriff** Angriffsszenario, bei dem ein Dritter ein System zwischen zwei Kommunikationspartnern platziert, um sensible Daten abfangen zu können

**Message Authentication Code (MAC)** Code zur Integritätsprüfung einer Nachricht

**Object Relational Mapping (ORM)** Technik, um Objekte aus objektorientierten Programmiersprachen in relationalen Datenbanken abzulegen

**OpenAPI Specification** Offene Spezifikation einer [API](#), welche genutzt werden kann um Client Code in verschiedenen Sprachen zu generieren

**Salted-Hash-Verfahren** Wird als zusätzliche Absicherung z. B. bei der Speicherung von Passwörtern genutzt. Bei diesem Verfahren wird dem Klartext vor der Weiterverarbeitung eine zufällig generierte Zeichenfolge hinzugefügt.

**Singleton** Entwurfsmuster, wobei nur ein einziges Objekt einer Klasse existiert, welches als Singleton bezeichnet wird.

**SQL Server Management Studio (SSMS)** Programm von Microsoft zur Verwaltung einer [SQL](#)-Datenbank

**SSL-Zertifikat** Datensatz, welcher von einer Zertifizierungsstelle an Unternehmen und / oder Betreiber von Webseiten ausgestellt wird. Dieser enthält zahlreiche Daten, unter anderem den Namen des Ausstellers, die Gültigkeitsdauer, eine Seriennummer oder auch den Fingerabdruck der Verschlüsselung.

**Standard Instrument Departure (SID)** Festgelegte Strecken, welche ein Flugzeug nach dem Start abfliegen kann, um eine Luftfahrtstrecke erreichen zu können

**Standard Terminal Arrival Routes (STAR)** Festgelegte Strecken, welche ein Flugzeug zum Erreichen eines Ziels abfliegen kann

**Structured Query Language (SQL)** Sprache zur Definition der Struktur einer Datenbank, sowie Abfragen der enthaltenen Daten

**Uniform Resource Name (URN)** Adressierung von Objekten ohne ein Protokoll festzulegen (Eindeutige und gleichbleibende Referenz - Name der Ressource)

**Uniform Resource Identifier (URI)** Eindeutige Adressierung von abstrakten und physikalischen Ressourcen im Internet

URI: [URLs](#)  $\cup$  [URNs](#)

**Uniform Resource Locator (URL)** Adressierung von Informationsobjekten mit Festlegung des Zugangs-Protokolls (Ort der Ressource)

**Web Application Firewall (WAF)** Verfahren um [Web Services](#) vor Angriffen über das [HTTP](#) zu schützen. Hierbei wird eine zusätzliche Ebene geschaffen, welche alle Anfragen auf Schadcode untersucht.

**Web Service** Softwaresystem, welches die Maschine-zu-Maschine Kommunikation über ein Netzwerk realisiert<sup>1</sup>.

**Windows Presentation Foundation (WPF) Framework** zur einfachen Erstellung einer Benutzeroberfläche, welches in .Net Core seit Version 3 zur Verfügung steht und Bestandteil des .NET Frameworks ist.

---

<sup>1</sup>Vgl. David Booth u. a. *Web Services Architecture*. Feb. 2004. URL: <https://www.w3.org/TR/ws-arch/> (besucht am 06. 07. 2022).

## **Gender Erklärung**

Aus Gründen der besseren Lesbarkeit wird in dieser Bachelorarbeit auf die gleichzeitige Verwendung der Sprachformen männlich, weiblich und divers (m/w/d) verzichtet. Sämtliche Formulierungen gelten gleichermaßen für alle Geschlechter.



# 1 Einleitung

Ein Flugsimulator besteht aus verschiedenen Simulationskomponenten, von denen viele mit speziell aufbereiteten Daten versorgt werden müssen. Hervorzuheben sind dabei beispielsweise Simulationen von Kommunikations- und Navigationsgeräten und [Flight Management Systemen \(FMS\)](#). Diese Komponenten müssen mit passenden Daten versorgt werden. Die Logistik dafür übernimmt dabei das [Luftfahrt Informations Datenbank \(LFID\)](#)-System, welches Kommunikations- und Navigationsdaten in einer Datenbank verwaltet. Diese wird zentral beim [Zentrum für Simulations- und Navigationsunterstützung Fliegende Waffensysteme der Bundeswehr \(ZSimNav\)](#) gepflegt und kann von angeschlossenen Simulatoren genutzt werden. Am Simulator selbst werden diese Daten noch in speziellen Formaten genutzt, welche allerdings nicht mehr mit der größeren Datenmenge zurecht kommen und deshalb obsolet sind. Die Idee ist es daher, einen Navigations- und Kommunikationsdatenservice im Simulator bereitzustellen, sodass unterschiedliche Simulatorkomponenten ihre benötigten Daten von dort beziehen. Dieser Service soll web-basiert sein, sich an der [Representational State Transfer \(REST\)](#)-Architektur orientieren und den Ansprüchen der Bundeswehr in Bezug auf IT-Sicherheit gerecht werden. Hierbei spielt auch die [Authentifizierung](#) eine große Rolle sowie die nutzbaren Transportprotokolle.

Das Kapitel [Grundlagen](#) stellt eine etwas allgemeiner gehaltene Beschreibung und Erklärung wichtiger Begriffe dar. So wird zuerst auf das Internetreferenzmodell eingegangen, woraufhin das [Hypertext Transfer Protocol \(HTTP\)](#) und damit einhergehend danach die [Transport Layer Security \(TLS\)](#) folgt. Schließlich werden noch zwei Absicherungsverfahren vorgestellt.

Die [Aufgabenstellung](#) behandelt obligatorische und optionale Anforderungen, welche anfangs an die Arbeit bzw. an das Produkt gestellt wurden.

Es folgt die in vier Abschnitte unterteilte [Analyse](#) der Problemstellung. Die Datenhaltung behandelt grundsätzlich, wie und wo die Daten verfügbar gemacht werden. Das Zielsystem wird vor allem für die kompatiblen [Frameworks](#) analysiert. Um die Sicherheit der Übertragung gewährleisten zu können, wird noch das IT Grundschatz Kompendium<sup>1</sup> näher behandelt, woraufhin auch die Top 10 des [Open Web Application Security Projects \(OWASP\)](#) folgen.

Die [Realisierung](#) beinhaltet Erklärungen darüber, wie die [Web Services](#) implementiert

---

<sup>1</sup>Vgl. Holger Schildt. *IT-Grundschatz-Kompendium*. Bundesamt für Sicherheit in der Informationstechnik, Bonn, Feb. 2022. ISBN: 978-3-8462-0906-6.

wurden und was zur Umsetzung der notwendigen Sicherheitsanforderungen verwendet wurde.

In [Ergebnis und Schlussfolgerungen](#) werden alle Resultate dieser Arbeit aufgezählt und Schlussfolgerungen über die weiteren Vorgehensweisen in Bezug darauf angerissen.

Das schließende Kapitel [Zusammenfassung und Ausblick](#) fasst alles Vorhergehende noch einmal zusammen und stellt einen Ausblick dar.

Das Ziel dieser Arbeit wird es sein, einen [REST](#)-basierten [Web Service](#) zu konzipieren, welcher sowohl den Anforderungen der Bundesregierung bezüglich IT-Sicherheit gerecht wird, als auch den Anforderungen des Zielsystems (Simulator), in dem dieser betrieben werden soll. Zudem werden die [Open Web Application Security Project \(OWASP\)](#) Top 10 angesprochen, analysiert und beachtet. Die Leser dieser Arbeit sollten danach in der Lage sein, eine Entscheidung zu treffen, welches Absicherungsverfahren bei unterschiedlichen Szenarien genutzt werden kann sowie (in Ansätzen) verstehen, wie ein [Web Service](#) gestaltet werden sollte und in [.NET Core 2.1](#) implementiert werden könnte.



## 2 Grundlagen

### 2.1 Internetreferenzmodell

Um die nächsten Kapitel in den Gesamtzusammenhang einordnen zu können, wird folgend kurz auf die „Grundlage des Internets“ eingegangen.

Im Allgemeinen existieren zwei Modelle: Einmal das [ISO/OSI-Referenzmodell](#), welches ab 1977 durch die [International Standards Organisation \(ISO\)](#) für die Kommunikation in offenen Netzwerken ([Open Systems Interconnection \(OSI\)](#)) bereitgestellt wurde und eine (etwas einfachere und) neuere Version, das [TCP/IP-Referenzmodell](#), bei dem der Name vom [Transmission Control Protocol \(TCP\)](#) und vom [Internet Protocol \(IP\)](#) stammt. Diese sogenannten Schichtenmodelle definieren einen Stapel von Ebenen, welche versuchen, die Aufgaben zu abstrahieren, voneinander abzukoppeln und damit zu vereinfachen. In [Abbildung A.1](#) auf Seite [43](#) sind die verschiedenen Lagen der Modelle gegenübergestellt und Beispiele genannt, welche Protokolle in welcher Schicht liegen.

Fängt man nun an, die Schichten von unten nach oben durchzuzählen, befindet sich auf der ersten Schicht im [ISO/OSI-Referenzmodell](#) die Bitübertragungsschicht, welche zusammen mit der zweiten Schicht, der Sicherungsschicht, die „Host-to-Network“ Schicht im [TCP/IP-Referenzmodell](#) ergibt. Diese Ebene ist dafür verantwortlich, die physikalische Übertragung zu gewährleisten. Beispiele hierfür sind vor allem das Ethernet (Kabel) und das WLAN nach [IEEE 802.11](#) Standard, wobei einfache Fehlererkennungsverfahren zusätzlich darin angesiedelt sind.

Die nächste Schicht beinhaltet das [IP](#) und ist damit zuständig für die Adressierung und Übertragung der Pakete. Die Reihenfolge der Pakete muss dabei allerdings nicht zwingend eingehalten werden. Zudem ist es möglich, dass Übertragungsfehler entstehen, welche jedoch durch diese Schicht nicht erkannt werden können. Die Internet-, bzw. Vermittlungsschicht stellt auch noch das [Internet Control Message Protocol \(ICMP\)](#) bereit, welches den Versand von Kontrollinformationen – beispielsweise Änderung von Routing, oder Statusabfragen – möglich macht.

In beiden Modellen folgt darauf die „Transportschicht“. Diese ist für die Kommunikation zwischen zwei Partnern zuständig und beinhaltet zwei Protokolle. Das [TCP](#) ist zuverlässig und verbindungsorientiert, weshalb es für kritische und sicherheitsrelevante Übertragungen genutzt wird. Das [User Datagram Protocol \(UDP\)](#) ist verbindungslos und unzuverlässig, jedoch dadurch schneller, wodurch es zur Übertragung großer Datenströme – wie beispielsweise Videostreams – genutzt werden kann, wobei das Fehlen

einzelner kleinerer Pakete nicht auffällt.

Die drei ursprünglichen Schichten (Sitzung, Darstellung und Anwendung) im ISO/OSI-Referenzmodell wurden im TCP/IP-Referenzmodell zu einer Schicht, der Verarbeitungsschicht, zusammengefasst. Dieser Bereich dient zur Bereitstellung höherwertiger Protokolle und Anwendungssoftware. Hierzu gehören unter anderem das File Transfer Protocol (FTP), das Simple Mail Transfer Protocol (SMTP) und auch das HTTP.

## 2.2 HTTP

1996 wurde die erste Version des HTTP – HTTP/1.0<sup>1</sup> – veröffentlicht. Seither wird das Protokoll vor allem für kleinere Web Services genutzt, da es schnell, leichtgewichtig und einfach zu implementieren ist. Um die Ladezeiten des Protokolls zu verbessern, wurde 1999 „HTTP/1.1“<sup>2</sup> veröffentlicht und standardisiert. Es existierte nun ein Header Feld namens „Connection: keep-alive“, welches vom Clienten dazu genutzt werden kann, die darunterliegende TCP-Verbindung aufrechtzuerhalten. Das „HTTP/2.0“<sup>3</sup> stellte eine erneute Beschleunigung der Übertragung dar. Mit einer Abwärtskompatibilität zur Version 1.1 wurde diese Version 2015 standardisiert und kann nach einer Anfrage des Clienten mit dem Header Feld „Connection: Upgrade“ genutzt werden. Die Neuheit war nun das „Multiplexing“, was eine Aufteilung der Datenübertragung in Streams darstellt. Diese Streams können nun mehrere Anfragen oder Antworten parallel übertragen, wobei nicht auf die Reihenfolge geachtet werden muss, da jeder Stream eine eindeutige ID besitzt. Für jede Aktion werden unterschiedliche Methoden im HTTP genutzt. So existiert unter anderem zu jeder Operation aus dem Akronym Create Read Update Delete (CRUD) eine äquivalente HTTP-Methode. Die Methoden werden im Folgenden aufgeführt und erklärt und – falls möglich – einer CRUD-Operation zugewiesen.

- **GET** fordert eine im Request-URI definierte Ressource an. (**READ**)
- **HEAD** ist ähnlich wie GET, übermittelt aber nur den Header ohne den Body.
- **POST** überträgt Daten an den Server. (**CREATE**)
- **PUT** möchte alle Daten, welche momentan auf dem Server liegen, mit den eigenen Daten überschreiben. (**UPDATE**)

<sup>1</sup>Vgl. Henrik Nielsen, Roy T. Fielding und Tim Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.0*. RFC 1945. Mai 1996. DOI: [10.17487/RFC1945](https://doi.org/10.17487/RFC1945). URL: <https://rfc-editor.org/rfc/rfc1945.txt>.

<sup>2</sup>Vgl. Roy T. Fielding und Julian Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. RFC 7230. Juni 2014. DOI: [10.17487/RFC7230](https://doi.org/10.17487/RFC7230). URL: <https://rfc-editor.org/rfc/rfc7230.txt>.

<sup>3</sup>Vgl. Mike Belshe, Roberto Peon und Martin Thomson. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540. Mai 2015. DOI: [10.17487/RFC7540](https://doi.org/10.17487/RFC7540). URL: <https://rfc-editor.org/rfc/rfc7540.txt>.

- **DELETE** löscht Dokumente auf dem Server (sofern die Rechte dazu vorhanden sind). (**DELETE**)
- **CONNECT** kann einen Tunnel zum Server für die bidirektionale Kommunikation öffnen.
- **OPTIONS** fordert den Zugriffspfad einer Ressource an und die weitere Kommunikation, welche darüber möglich ist.
- **TRACE** ist zum Testen. Die Anfrage wird vom Server zurückgeschickt.

## 2.3 TLS/SSL

Um die Übertragung der Daten beispielsweise durch das [HTTP](#) sicher zu gestalten und vor dem Zugriff Dritter zu schützen, wird das [TLS](#)-Protokoll genutzt. Der noch meistens äquivalent verwendete Name [Secure Sockets Layer \(SSL\)](#) wurde 1999 offiziell durch die Veröffentlichung von „The TLS Protocol Version 1.0“<sup>4</sup> durch die [Internet Engineering Task Force \(IETF\)](#) in [TLS](#) umbenannt. Diese Version gilt allerdings mittlerweile als ungültig und sollte daher nicht mehr genutzt werden. Aktuell werden die Versionen 1.2<sup>5</sup> und 1.3<sup>6</sup> genutzt. Im Kern werden für die Absicherung zwei interne Protokolle genutzt, welche im Folgenden erläutert werden.

### 2.3.1 TLS Handshake Protocol

Alle [TLS](#) Handshakes nutzen ein asymmetrisches Verschlüsselungsverfahren. Das Protokoll durchläuft dafür folgende Schritte:

1. Der Client schickt eine „Hello“ Nachricht an den Server und leitet somit den Handshake ein. Die Nachricht besteht aus der unterstützten [TLS](#)-Version, den unterstützten [Cipher Suites](#) und einer zufälligen Anordnung von Bytes, welche als „Client Random“ bezeichnet werden.
2. Die Antwort vom Server, auch als „Server-Hello“ bezeichnet, enthält das [SSL-Zertifikat](#), die ausgewählte [Cipher Suite](#) und ein „Server Random“, welches ebenfalls eine Byte-Folge ist.
3. Der Client prüft daraufhin das [SSL-Zertifikat](#) bei der Zertifizierungsstelle.

<sup>4</sup>Vgl. Christopher Allen und Tim Dierks. *The TLS Protocol Version 1.0*. RFC 2246. Jan. 1999. DOI: [10.17487/RFC2246](https://doi.org/10.17487/RFC2246). URL: <https://rfc-editor.org/rfc/rfc2246.txt>.

<sup>5</sup>Vgl. Eric Rescorla und Tim Dierks. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. Aug. 2008. DOI: [10.17487/RFC5246](https://doi.org/10.17487/RFC5246). URL: <https://rfc-editor.org/rfc/rfc5246.txt>.

<sup>6</sup>Vgl. Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018. DOI: [10.17487/RFC8446](https://doi.org/10.17487/RFC8446). URL: <https://rfc-editor.org/rfc/rfc8446.txt>.

4. Anschließend wird das „Premaster Secret“, eine zufällig generierte Byte-Folge, mit dem im [SSL-Zertifikat](#) enthaltenen öffentlichen Schlüssel verschlüsselt und an den Server geschickt, welcher dieses nur mit dem privaten Schlüssel wieder entschlüsseln kann.
5. Der Server entschlüsselt nun das „Premaster Secret“, woraufhin beide Partner einen Sitzungsschlüssel aus „Client Random“, „Server Random“ und „Premaster Secret“ generieren.
6. Nun senden Client und Server jeweils eine „Fertig“-Nachricht, welche beide mit dem Sitzungsschlüssel verschlüsselt sind.
7. Der Handshake ist abgeschlossen und die weitere Kommunikation wird mit dem Sitzungsschlüssel fortgesetzt.

### 2.3.2 TLS Record Protocol

Das Record Protokoll wird für die eigentliche Datenübertragung genutzt und ist vollständig abgekoppelt vom [TLS Handshake Protocol](#). Zuerst werden die zu verschickenden Daten in handliche Blöcke aufgeteilt, welche optional auch komprimiert werden können. Jeder Nachricht wird dann ein Nachrichtenauthentifizierungscode (engl. [Message Authentication Code \(MAC\)](#)) zugewiesen, der zur Integritätsprüfung genutzt wird. Final werden noch genau diese Daten mit dem im [TLS Handshake Protocol](#) ausgehandelten Schlüssel chiffriert. Sind die Daten nun vollständig verschlüsselt und verschickt, wird die empfangende Seite das Protokoll noch einmal rückwärts durchlaufen, um die Daten wieder darstellen zu können.

## 2.4 Ausgewählte Absicherungsverfahren

Für die Identifizierung der Kommunikationspartner und damit auch die höherwertige Absicherung von [Application Programming Interface \(API\)s](#), ist es wichtig den Unterschied zwischen den Begrifflichkeiten [Authentisierung](#), [Authentifizierung](#) und [Autorisierung](#) zu verstehen. Mit der [Authentisierung](#) weist der Nutzer seine Identität gegenüber des Services/Systems nach. Diese Nachweise sind in der Regel Ausweisdokumente, oder in der digitalen Welt, Passwörter, Muster, Fingerabdrücke, oder Ähnliches. In dem darauf folgenden Schritt, der [Authentifizierung](#), werden eben diese Nachweise durch eine unabhängige Behörde verifiziert oder falsifiziert, was schlussendlich dazu führt, dass der Service/das System dem Nutzer Rechte einräumt, beziehungsweise Ressourcen freigibt oder verweigert. Diesen Schritt bezeichnet man als [Autorisierung](#).

Nachfolgend werden zwei Verfahren beschrieben, welche zur Absicherung genutzt werden können. Diese werden unter anderem in den Kontext der Begrifflichkeiten eingeordnet.

### 2.4.1 API-Keys

Die **API-Keys** eignen sich zur einfachen **Autorisierung** eines Clienten und werden häufig von Programmen genutzt, bei welchen kein direkter Zugriff auf sensible Daten nötig ist. Hierzu wird bei beiden Partnern ein gemeinsamer Schlüssel hinterlegt, wodurch der Kommunikationspartner identifiziert werden kann. Dieser Schlüssel sollte einmalig vergeben werden und wird im Falle von **REST** und **HTTP** entweder über die Request-Query bei einem POST-Request oder als Header oder **Cookie** bei einem GET-Request übergeben. Der **Web Service** prüft basierend auf dem Schlüssel die genehmigten oder gesperrten Ressourcen und wertet anhand der **Autorisierung** die Anfrage aus. Dieses Verfahren ist stark durch **Man-in-the-Middle Angriffe** bedroht, was unter anderem dazu führt, dass es nur unter Benutzung des **Hypertext Transfer Protocol Secure (HTTPS)** als sicher gilt, da ansonsten der Schlüssel abgefangen und genutzt werden kann.

### 2.4.2 OAuth 2.0

Um einen offenen Standard für die **API-Zugriffsdelegation** zu erstellen, wurde im November 2006 eine Initiative gestartet, welche zum Ziel haben sollte, das OAuth-Protokoll in der Version 1.0 zu entwickeln. Ein offener Standard war gewünscht, da verschiedene Firmen (z. B. Twitter) bereits eigene Verfahren entwickelt hatten und das Beste aus diesen Verfahren zusammengetragen werden sollte, um die Implementierung dieser Delegation zu vereinfachen. Die Delegation der **Authentifizierung** gewinnt zunehmend an Notwendigkeit durch immer mehr verschiedene **Web Services** und Webanwendungen, welche ansonsten alle eine eigene Nutzerverwaltung bräuchten. OAuth schafft hierbei Abhilfe, indem nur gewisse Rechte an die zu nutzende Applikation übertragen werden. Die Version OAuth 1.0 wurde also vier Jahre später durch die **IETF** standardisiert<sup>7</sup>. Da diese Version allerdings schwierig zu implementieren war und manche Implementierungen sogar Sicherheitslücken aufwiesen, wurde im Oktober 2012 die Version 2.0 veröffentlicht<sup>8</sup>. Diese hatte zudem einige andere Vorteile und hat Version 1.0 fast komplett abgelöst, da unter anderem auch keine Kompatibilität gegeben ist. Im Folgenden wird daher nur auf OAuth 2.0 eingegangen.

---

<sup>7</sup>Vgl. Eran Hammer-Lahav. *The OAuth 1.0 Protocol*. RFC 5849. Apr. 2010. DOI: [10.17487/RFC5849](https://doi.org/10.17487/RFC5849). URL: <https://rfc-editor.org/rfc/rfc5849.txt>.

<sup>8</sup>Vgl. Dick Hardt. *The OAuth 2.0 Authorization Framework*. RFC 6749. Okt. 2012. DOI: [10.17487/RFC6749](https://doi.org/10.17487/RFC6749). URL: <https://rfc-editor.org/rfc/rfc6749.txt>.

### 2.4.2.1 Rollen

OAuth 2.0 definiert vier verschiedene Rollen, welche unter anderem die klassische Client-Server-[Authentifizierung](#) in der Art revolutioniert, dass der Client nun in zwei Rollen aufgeteilt wird.

#### Resource Owner

Falls es sich um eine Person handelt wird diese Rolle auch „end-user“ genannt. Diese stellt eine Entität dar, welche die Fähigkeit hat, einem Dritten Zugriff auf ihre geschützten Ressourcen zu gewährleisten.

#### Resource Server

Der Resource Server stellt (geschützte) Daten bereit und hostet diese. Zusätzlich dazu ist dieser ebenfalls in der Lage, [Access Token](#) zu akzeptieren oder zu verweigern.

#### Client

Der Term [Client](#) erfordert keine besonderen Charakteristiken und beschreibt eine (Web)-Applikation, welche Ressourcen beim [Resource Server](#) in Vertretung des „end-users“ anfragen kann.

#### Authorization Server

Um [Token](#) für die [Autorisierung](#) beim [Resource Server](#) zu bekommen, wird der [Authorization Server](#) genutzt. Dieser kann [Token](#) erstellen, verschicken und validieren.

### 2.4.2.2 Token

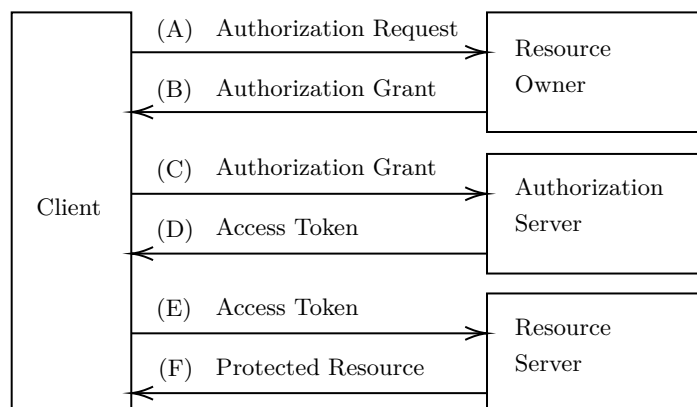
Die Sicherheit im Zugriff auf geschützte Ressourcen kann bei [OAuth 2.0](#) durch „Token“ gewährleistet werden. Am häufigsten kommen sogenannte „Bearer-Token“ zum Einsatz.

#### Access Token

Diese Token werden genutzt, um Zugangsdaten eines [Resource Owners](#), spezifische Bereiche und die Dauer des Zugriffs für einen [Clienten](#) verschlüsselt bereitzustellen. Die Darstellung erfolgt als String und ist üblicherweise undurchsichtig für den [Clienten](#). Zudem können sie je nach Sicherheitsrichtlinien des Servers in der Darstellung und Verschlüsselung variieren.

### Refresh Token

Refresh Token sind optional und – wie der Name sagt – dafür da, die bereits ausgestellte Berechtigung zu erneuern. Der **Client** nutzt dieses Token nur im Austausch mit dem **Authorization Server**, um ein neues **Access Token** zu erhalten. Hiermit kann die Sicherheit dieses Protokolls zusätzlich verstärkt werden, falls die Lebensdauer des **Access Tokens** kurz gewählt wird und der **Client** für Anfragen häufig über das **Refresh Token** ein neues **Access Token** beantragen muss.



**Abbildung 2.1:** OAuth 2.0 Abstrakter Protokoll Fluss

Quelle: Dick Hardt. The OAuth 2.0 Authorization Framework. RFC 6749. Okt. 2012. DOI: [10.17487/RFC6749](https://doi.org/10.17487/RFC6749). URL: <https://rfc-editor.org/rfc/rfc6749.txt>

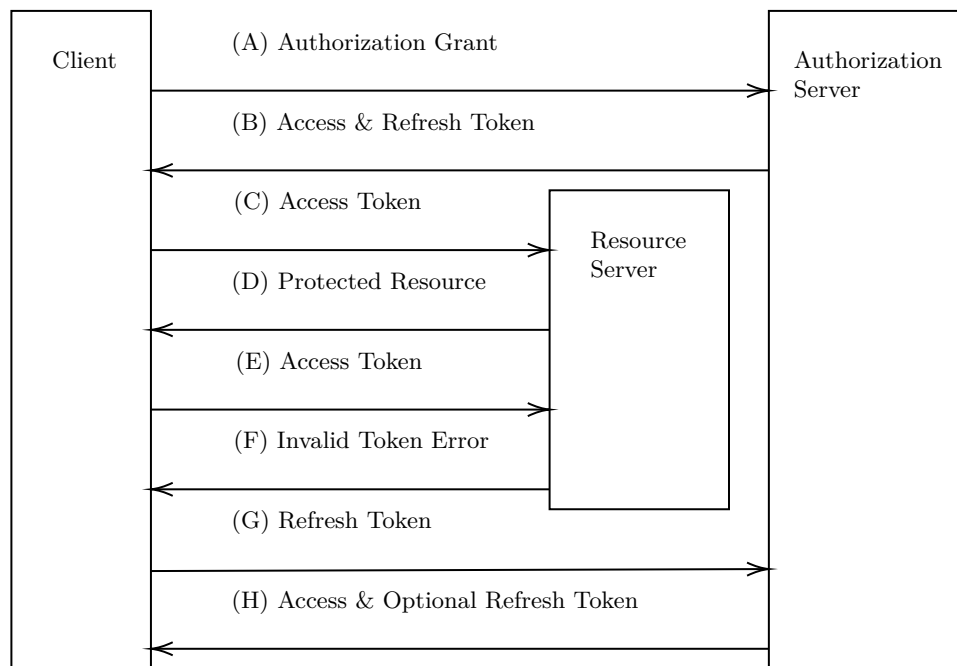
#### 2.4.2.3 Abstrakter Protokollfluss

Abbildung 2.1 zeigt den allgemeinen abstrakten Ablauf einer **Autorisierung** mit OAuth 2.0 und ist wie folgt dargestellt:

- (A) Der **Client** beantragt die **Autorisierung** des **Resource Owners**. Diese Anfrage kann entweder direkt an den **Resource Owner** gestellt werden oder besser indirekt durch den **Authorization Server** als Vermittler.
- (B) Der **Resource Owner** antwortet mit einer **Autorisierungsgenehmigung**, welche eine aus vier zugelassenen Typen sein kann, die jeweils abhängig von den unterstützten Typen des **Authorization Servers** sind.
- (C) Der **Client** fordert ein **Access Token** in Kombination mit der vorher erhaltenen **Autorisierungsgenehmigung** beim **Authorization Server** an.
- (D) Der **Authorization Server** authentisiert den **Client**, validiert die **Autorisierungs-**

genehmigung und gibt ein **Access Token** heraus.

- (E) Der **Client** fragt die Ressource beim **Resource Server** an und **authentifiziert** sich mit dem **Access Token**.
- (F) Der **Resource Server** prüft das **Token** im Zusammenspiel mit dem **Authorization Server** und erteilt Zugriff auf die angefragte Ressource.



**Abbildung 2.2:** OAuth 2.0 Refreshing an Expired Access Token

Quelle: Dick Hardt. The OAuth 2.0 Authorization Framework. RFC 6749. Okt. 2012. DOI: [10.17487/RFC6749](https://doi.org/10.17487/RFC6749). URL: <https://rfc-editor.org/rfc/rfc6749.txt>

#### 2.4.2.4 Beantragung eines Refresh Tokens

Um das vorher genannte **Access Token** neu beantragen zu können, da meistens eine Gültigkeitsdauer gesetzt ist, wird – wie in Abbildung 2.2 gezeigt – zusammen mit dem **Refresh Token** ein neues **Access Token** beantragt. Der Ablauf ist wie folgt:

- (A) Wie bei der ersten Beantragung eines **Access Tokens** schickt der **Client** eine **Authentifizierungsanfrage** an den **Authorization Server**.
- (B) Der **Authorization Server** antwortet daraufhin sowohl mit einem **Access Token** als auch mit einem **Refresh Token**.



- (C) Der **Client** beantragt die geschützten Daten beim **Resource Server** in Kombination mit dem **Access Token**.
- (D) Falls das **Access Token** valide ist, wird der Request mit den angefragten Daten beantwortet.
- (E) Die Schritte (C) und (D) werden so lange wiederholt, bis die Lebensdauer des **Access Tokens** ausgelaufen ist. Wenn der **Client** dies weiß, wird direkt zum Schritt (G) gesprungen, andernfalls wird ein erneuter Request abgesetzt.
- (F) Da das **Access Token** nun ungültig ist, wird vom **Resource Server** ein „Invalid Token Error“ zurückgegeben.
- (G) Der **Client** beantragt ein neues **Access Token** beim **Authorization Server**, indem dieser das **Refresh Token** zeigt und sich autorisiert.
- (H) Wenn die Validierung des **Refresh Tokens** geklappt hat, erteilt der **Authorization Server** dem **Client** ein neues **Access Token**, sowie optional auch ein neues **Refresh Token**.



## 3 Aufgabenstellung

Die Arbeit soll zeigen, dass unter anderem eine abgesicherte Kommunikation zwischen Client und [Web Service](#), zusammen mit einer [Authentifizierung](#), sinnvoll und auch notwendig ist. Die [Obligatorischen Anforderungen](#) sind daher stark begrenzt, anders als die [Optionalen Anforderungen](#), da durch die Komplexität dieses Themas theoretisch noch mehr gezeigt und implementiert werden kann.

### 3.1 Obligatorische Anforderungen

Der [Web Service](#) muss den IT-Richtlinien des [Bundesamts für Sicherheit in der Informationstechnik \(BSI\)](#) gerecht werden und somit den Bestimmungen des IT-Grundschutz-Kompendiums<sup>1</sup> entsprechen. Es müssen über eine [REST](#)-Schnittstelle Methoden für den Zugriff auf die dahinter liegende Datenbank bereitgestellt werden. Das Ganze soll schnell und zuverlässig passieren und mehrere Anfragen pro Sekunde verarbeiten können. Wie aus der Beschreibung eines [REST-Web Services](#) hervorgeht, muss die [API](#) zudem selbst sprechend sein (Stichwort: [Hypermedia as the Engine of Application State \(HATEOAS\)](#)) und – soweit möglich – alle [CRUD](#)-Operationen widerspiegeln können. Um den [Web Service](#) testen und die Resultate grafisch darzustellen, unter anderem dafür, dass die Einbindung in den Simulator gezeigt wird, muss auch noch eine Benutzeroberfläche existieren, die optimalerweise der eines Cockpits ähnelt. Des Weiteren müssen alle Programme in C# geschrieben werden und selbstredend auf dem Zielsystem laufen.

### 3.2 Optionale Anforderungen

Das [LFID](#)-System stellt Daten bereit, welche von verschiedenen Programmen unterschiedlich interpretiert werden können. Diese Anbindung an die Datenbank erfolgt momentan für jedes einzelne Programm noch über einen direkten Datenbankzugriff und wird zudem aktuell noch nicht direkt im Flugsimulator genutzt. Die Lösung soll sein, alle Programme und Flugsimulatorkomponenten, welche diese Daten benötigen, so umzustellen, dass diese sich bei dem [Web Service](#) [autorisieren](#) müssen, um damit auch Daten abfragen zu können. Bei den Programmen, welche nur Leserechte benötigen, kann dann durch die [Autorisierung](#) ein anderes Zugriffslevel zugewiesen werden, als bei Programmen, welche tatsächlich dazu befugt sind, Daten zu ändern. Hier würde dann natürlich

---

<sup>1</sup>Schildt, *IT-Grundschutz-Kompendium*.

auch direkt mitspielen, dass für jede Entität alle [HTTP](#)-Methoden bereitgestellt werden müssen, um alle [CRUD](#)-Operationen abbilden zu können.

Zudem ist der [Autorisierungsserver](#) bisher nur ein Kommandozeilenprogramm, wie auch der [Web Service](#), was eventuell auch noch durch eine grafische Oberfläche – vor allem für die Einrichtung der Rechteverteilung – unterstützt werden könnte.

## 4 Analyse

Bisher wird das [LFID](#)-System und die dadurch bereitgestellten Daten noch nicht direkt in einer Simulatorumgebung genutzt. Der [Web Service](#) kann also, bis auf die Beachtung folgender Punkte, relativ frei gestaltet werden.

### 4.1 Datenhaltung

Wie der Name schon sagt, sind die abzufragenden Daten des [LFID](#)-Systems in einer Datenbank bereitgestellt. Diese stellt sich als [Structured Query Language \(SQL\)](#)-Datenbank dar und wird mit dem [SQL Server Management Studio \(SSMS\)](#) verwaltet. Zur Zeit wird immer ein Administratorzugang eingerichtet, welcher vollen Zugriff auf alle Daten und Operationen bekommt. Die hier behandelten Datenbanktabellen finden sich beispielhaft gezeigt in [Abbildung A.4](#) auf [Seite 46](#) wieder.

### 4.2 Zielsystem

Dieser [Web Service](#) soll in aller erster Linie für einen Flugsimulator gebaut werden, bei welchem eine Simulationskomponente die benötigten Daten beim [Web Service](#) anfragen muss.

#### 4.2.1 Betriebssystem

Als Betriebssystem wird ein abgespecktes Windows 10 genutzt, welches alle nötigen Services und Programme installiert und alle nötigen Ports freigegeben hat. Somit können ohne Weiteres die fertig gebauten [Dynamic Link Libraries \(DLLs\)](#) gestartet und benutzt werden, da vorher unter anderem die Datenbank installiert wurde. Wie schon erwähnt, sind mehrere Programme (oder Services) von den Daten abhängig, wodurch jeder Service einen eigenen Datenbankzugriff und daher auch alle Rechte bekommt. Die Installation könnte vereinfacht werden, indem ein Container mit der Datenbank, dem [Web Service](#) und dem [Autorisierungsserver](#) bereitgestellt wird und die Ports daraus so freigegeben werden, dass alle Methoden erreicht werden können.

#### 4.2.2 Möglichkeiten des Log-Ins

Die Simulatorumgebung wird einmalig hochgefahren, womit auch alle Services und Programme gestartet werden. Auf diese Umgebung haben dann die Piloten, die Fluglehrer

und noch eventuelle Zuschauer Zugriff, wodurch sich ein Login durch einen speziellen Nutzer während der Phase des Hochfahrens unter großem Aufwand realisieren lässt. Die Absicherung des [Web Services](#) wird also durch die Identifizierung der Programme stattfinden müssen, da zudem auch keine Passworteingabe möglich ist.

### 4.2.3 Mögliche Frameworks

Zur Vereinheitlichung der Implementierung sollen wenn möglich [Frameworks](#) genutzt werden, welche den ganzen Prozess vereinfachen, sodass bestenfalls nicht zusätzlich in unterschiedlichen Sprachen programmiert werden muss. Da so wenig andere Pakete wie nötig auf dem Zielsystem installiert werden sollen, bietet sich die Programmiersprache C# an. Aufgrund von Rahmenbedingungen älterer [Betriebssysteme](#) werden im Folgenden die aktuellsten zwei möglichen C# [Frameworks](#) vorgestellt.

#### 4.2.3.1 .NET Framework

Dieses [Framework](#) ist das in Windows meistgenutzte und älteste Konzept, um Anwendungen zu erstellen. Mit dem Nu-Get Paketmanager lässt sich externer Code einbinden, womit unter anderem die Datenbankanbindung leicht und einheitlich implementiert werden kann. Hier werden allerdings keine Microservices unterstützt, wodurch sich .NET Framework eher zur Programmierung von [Graphical User Interfaces \(GUIs\)](#) eignet. Zudem ist ein kompiliertes .NET Framework-Projekt nicht alleine lauffähig, daher wird seit Windows Vista .NET Framework zusammen mit den Windows Updates installiert.

#### 4.2.3.2 .NET Core 2.1

.NET Core 2.1 wurde am 30.05.2018 von Microsoft zur Nutzung in Visual Studio 2017 Version 15.7 herausgebracht. Hiermit wird ein [Framework](#) bereitgestellt, welches als Open Source Projekt für Windows, MacOS und Linux fungiert. Entwickler können damit plattformübergreifende Anwendungen programmieren und mit dem NuGet Paketmanager eine Reihe an bereits existierendem Code einbinden und nutzen, egal ob dieser von Microsoft oder von anderen Entwicklern in dieser Gemeinschaft stammt. So findet sich dort unter anderem Code für die Erstellung von [Web Services](#) und [OAuth 2.0 Authentifizierungsservices](#). Für die Arbeit mit einer SQL-Datenbank bietet sich vor allem das „Entity Framework Core“-Paket von Microsoft an, welches die Anbindung und Nutzung dieser, unter anderem mit [Language Integrated Query \(LINQ\)](#), extrem vereinfacht und standardisiert.

#### 4.2.3.3 Vergleich beider Frameworks

Beide vorangegangenen Frameworks sind also theoretisch möglich und nutzbar. Allerdings gibt es einige zu berücksichtigende Gesichtspunkte, welche die Entscheidung für ein Framework vereinfachen. Für beide Web Services (Daten- und Authentifizierungsservice) wurde nun .NET Core 2.1 genutzt, da sich die Erstellung und auch das Hosten hierin einfacher gestaltet. In .NET Framework können beispielsweise nur schwierig selbsthostende Web Services erstellt werden, da Microsoft die Internet Information Services (IIS) bereitstellt, um derartige Dinge zu lösen. Will man nun allerdings ein selbsthostendes Programm erstellen, wie es in dieser Arbeit der Fall ist, so muss dies in .NET Core 2.1 realisiert werden. Des Weiteren ist es allerdings so, dass in .NET Core 2.1 nur schwierig grafische Anwendungen erstellt werden können. Es gestaltet sich also einfacher und fast schon intuitiver, auf die Konzepte von .NET Framework, wie Windows Presentation Foundation (WPF) oder Windows Forms zurückzugreifen. Das Demoprogramm zum Testen der beiden Web Services wird also in .NET Framework implementiert. Abschließend ist noch zu erwähnen, dass Microsoft den „.NET Standard“ als Bindeglied zwischen beiden Frameworks eingeführt hat, um die Benutzung gemeinsamer Bibliotheken zu ermöglichen. Dies wird auch hier genutzt, da die Elemente der Datenbank offensichtlich in beiden Programmen, dem Web Service und dem Demo-Programm, gleich sind. Weiterhin sind in der Tabelle B.2 auf Seite 52 noch einmal einige Unterschiede aufgeführt, welche zu der vorhergehenden Entscheidung beigetragen haben.

### 4.3 IT Grundschutz Kompendium

Das IT-Grundschutz-Kompendium ist ein durch den Bundesamt für Sicherheit in der Informationstechnik (BSI) herausgegebenes Dokument, welches Sicherheitsrisiken im Bereich der Informationssicherheit aufzeigt und präventive Maßnahmen dagegen vorstellt. Generell definiert dieses Dokument drei Grundwerte der Informationssicherheit<sup>1</sup>, die es zu schützen gilt:

#### Verlust der Verfügbarkeit:

Ein Softwaresystem funktioniert nur in Anwesenheit gewisser Daten. Wenn diese nicht vorhanden, oder nur eingeschränkt verfügbar sind, kann dies zur Beeinträchtigung einfacher Aufgaben bis hin zu völliger Stilllegung von Arbeitsprozessen führen.

---

<sup>1</sup>Vgl. Schildt, *IT-Grundschutz-Kompendium*.

**Verlust der Vertraulichkeit von Informationen:**

Jedes Unternehmen muss mit personenbezogenen Daten von Kunden und Nutzern vertraulich umgehen, sodass keine Schäden an der Privatsphäre entstehen. Dies sind unter anderem Daten zum Umsatz, Marketing oder Forschung.

**Verlust der Integrität (Korrektheit von Informationen):**

Um fehlerhafte Aufträge, o. ä. vermeiden zu können, ist unbedingt sicherzustellen, dass alle gespeicherten Daten richtig und unverfälscht sind. Die sogenannte „digitale Identität“ spielt zudem eine immer größere Rolle, wodurch die Bedrohung falscher Willenserklärungen oder Identitätsfälschungen wachsen.

Das Dokument ist zur Eliminierung dieser und anderer Risiken in Bausteine aufgeteilt, welche jeweils in:

1. Beschreibung
2. Gefährdungslage
3. Anforderungen
4. Weiterführende Informationen
5. Anlage: Kreuzreferenztafel zu elementaren Gefährdungen

unterteilt sind.

**4.3.1 APP.3.1 Webanwendungen und Webservices**

Im Folgenden wird der für diese Arbeit relevante Baustein „APP.3.1 Webanwendungen und Webservices“ nach diesem Muster analysiert.

**4.3.1.1 Beschreibung**

Hier wird ein [Web Service](#) in der Einleitung noch einmal abgegrenzt und definiert. Ein [Web Service](#) ist anhand dieses Bausteins eine Anwendung, welche das [HTTP](#) oder das [HTTPS](#) verwendet um Ressourcen bereitzustellen. In den allermeisten Fällen wird dieser Service nicht von einem Benutzer direkt gesteuert, sondern über den Zugriff anderer Anwendungen angesprochen. In diesem Baustein geht es zudem ausschließlich um [Web Services](#) mit einer [REST](#)-Schnittstelle<sup>2</sup>.

**4.3.1.2 Gefährdungslage**

Die Gefährdungslage dieses Bausteins ist noch einmal unterteilt. Im Folgenden werden relevante Gefährdungen angesprochen<sup>3</sup>.

<sup>2</sup>Vgl. Schildt, *IT-Grundschutz-Kompendium*, APP.3.1 S. 1.

<sup>3</sup>Vgl. *ebd.*, APP.3.2 S. 1.



**Unzureichende Protokollierung von sicherheitsrelevanten Ereignissen**

Die Protokollierung sicherheitsrelevanter Ereignisse ist erforderlich, um später die Ursachen eines bestimmten Ereignisses ermitteln und somit Schwachstellen, kritische Fehler, oder unerlaubte Änderungen nachvollziehen zu können.

**Offenlegung sicherheitsrelevanter Informationen bei Webanwendungen und Webservices**

Bei der Auslieferung von [Web Services](#) ist darauf zu achten, keine sicherheitsrelevanten Daten offenzulegen, sprich Informationen über genutzte Programme, Systeme oder Versionen weiterzugeben.

**Missbrauch einer Webanwendung durch automatisierte Nutzung**

Ein [Web Service](#) kann durch automatisierte Anfragen missbraucht werden und somit eventuell Nutzerdaten preisgeben. Dadurch könnten Angreifer gültige Benutzernamen sammeln und sich dann damit durch mehrfaches Probieren von Passwörtern Zugriff zur Anwendung verschaffen.

**Unzureichende Authentisierung**

Meistens werden Rollenprofile angelegt, um Benutzern die Möglichkeit zu geben, bestimmte Ressourcen erreichen zu können, welche von anderen Profilen nicht erreicht werden können. Falls die [Authentisierung](#) der Nutzer unzureichend ist, könnte ein Angreifer einfachen Zugriff erlangen und im schlimmsten Fall sogar auf die Nutzerdaten zugreifen, falls diese ebenfalls über diese Rolle erreichbar sind.

**4.3.1.3 Anforderungen**

Die Anforderungen an [Web Services](#) sind wiederum in drei Bereiche unterteilt und zudem sollten immer Zuständigkeiten existieren, sodass eine Stelle immer „grundsätzlich zuständig“ ist und optional auch weitere Zuständigkeiten zugeordnet werden können.

**Basis-Anforderungen**

Vor allem die Basis Anforderungen MÜSSEN laut dem [BSI](#) erfüllt werden<sup>4</sup>.

**APP.3.1.A1 Authentisierung**

[Web Services](#) MÜSSEN immer so gestaltet sein, dass der Zugriff auf gesperrte Ressourcen nur über eine [Authentisierung](#) erfolgen kann. Die Methode hierfür SOLLTE protokolliert

---

<sup>4</sup>Vgl. [ebd.](#), APP.3.1 S. 3.

werden, wobei der Betrieb zudem eine Höchstgrenze für fehlgeschlagene Anmeldeversuche festsetzen muss<sup>5</sup>.

#### **APP.3.1.A4 Kontrolliertes Einbinden von Dateien und Inhalten**

Diese Anforderung entfällt für diesen [Web Service](#), da keine Daten hochgeladen werden.

#### **APP.3.1.A7 Schutz vor unerlaubter automatisierter Nutzung**

Es MUSS garantiert sein, dass der [Web Service](#) vor [unautorisierter](#), automatisierter Nutzung geschützt ist, während das Verhalten der Maßnahmen auf [autorisierte](#) Nutzer auch berücksichtigt werden MUSS<sup>6</sup>.

#### **APP.3.1.A14 Schutz vertraulicher Daten**

Vertrauliche Daten, wie beispielsweise Zugangsdaten, MÜSSEN serverseitig mit Hilfe von [Salted-Hash-Verfahren](#) abgesichert sein, um diese vor [unautorisiertem](#) Zugriff schützen zu können.

### **Standard-Anforderungen**

Standard-Anforderungen SOLLTEN grundsätzlich zusammen mit den [APP.3.1](#) erfüllt sein<sup>7</sup>.

#### **APP.3.1.A8 Systemarchitektur [Beschaffungsstelle]**

Bereits während der Planung eines [Web Services](#) SOLLTEN Sicherheitsrichtlinien und -konzepte beachtet werden<sup>8</sup>.

#### **APP.3.1.A9 Beschaffung von Webanwendungen und Webservices**

Als Ergänzung zu den anderen Anforderungen stellt das IT-Grundschutzkompendium eine Liste bereit, welche Eigenschaften bei der Beschaffung eines [Web Services](#) zusätzlich beachtet werden SOLLTEN<sup>9</sup>:

- sichere Eingabvalidierung und Ausgabekodierung
- sicheres Session-Management
- sichere kryptografische Verfahren
- sichere Authentisierungsverfahren

<sup>5</sup>Vgl. Schildt, *IT-Grundschutz-Kompendium*, APP.3.1 S. 3.

<sup>6</sup>Vgl. *ebd.*, APP.3.1 S. 3.

<sup>7</sup>Vgl. *ebd.*, APP.3.2 S. 4.

<sup>8</sup>Vgl. *ebd.*, APP.3.1 S. 4.

<sup>9</sup>Vgl. *ebd.*, APP.3.1 S. 4.

- sichere Verfahren zum serverseitigen Speichern von Zugangsdaten
- geeignetes Berechtigungsmanagement
- ausreichende Protokollierungsmöglichkeiten
- regelmäßige Sicherheitsupdates durch den Entwickler der Software
- Schutzmechanismen vor verbreiteten Angriffen auf Webanwendungen und [Web Services](#)
- Zugriff auf den Quelltext der Webanwendung oder des [Web Services](#)

#### **APP.3.1.A11 Sichere Anbindung von Hintergrundsystemen**

Hintergrundsysteme auf denen Funktionen und Daten ausgelagert sind, SOLLTEN einzig über definierte Schnittstellen ansprechbar sein. Zudem SOLLTE bei netz- und standort-übergreifenden Anwendungen die Kommunikation [authentisiert](#) und verschlüsselt ablaufen<sup>10</sup>.

#### **APP.3.1.A12 Sichere Konfiguration**

Der Zugriff bzw. die Anfragen auf Ressourcen und Methoden eines [Web Services](#) SOLLTEN derart eingeschränkt sein, sodass nur über vorher definierte Pfade kommuniziert werden kann. Alle anderen Methoden und nicht benötigten Ressourcen SOLLTEN deaktiviert werden.

#### **APP.3.1.A21 Sichere HTTP-Konfiguration bei Webanwendungen**

Folgende Response-Header SOLLTEN grundsätzlich immer gesetzt sein<sup>11</sup>:

- Content-Security-Policy
- Strict-Transport-Security
- Content-Type
- X-Content-Type-Options
- Cache-Control

Diese SOLLTEN zudem immer so restriktiv wie möglich gestaltet werden.

#### **APP.3.1.A22 Penetrationstest und Revision**

Zur Prüfung auf Sicherheitsprobleme und -verstöße SOLLTEN regelmäßig Penetrationstests und Revisionen durchgeführt werden, wobei die Ergebnisse protokolliert werden SOLLTEN<sup>12</sup>.

---

<sup>10</sup>Vgl. [ebd.](#), APP.3.1 S. 4.

<sup>11</sup>Vgl. [ebd.](#), APP.3.1 S. 5.

<sup>12</sup>Vgl. [ebd.](#), APP.3.1 S. 5.

### Anforderungen bei erhöhtem Schutzbedarf

Diese Anforderungen sind dann zu erfüllen, wenn der Schutzbedarf über das dem Stand der Technik entsprechende Schutzniveau hinausgeht. Die Bestimmung dieser erfolgt durch eine individuelle Analyse<sup>13</sup>.

#### APP.3.1.A20 Einsatz von Web Application Firewalls

Eine [Web Application Firewall \(WAF\)](#) SOLLTE eingesetzt werden, um das Sicherheitslevel zu erhöhen. Die Konfiguration dieser ist zudem nach jedem Update anzupassen<sup>14</sup>.

#### 4.3.1.4 Weiterführende Informationen

In diesem Kapitel wird auf das [OWASP](#) verwiesen, was auch in Abschnitt 4.4 auf der nächsten Seite angesprochen und erklärt wird. Zudem wird noch das Dokument „Kryptographische Verfahren: Empfehlungen und Schlüssellängen: BSI TR-02102“ vom [BSI](#) vorgeschlagen, welches Hinweise zur Nutzung kryptographischer Verfahren bereitstellt<sup>15</sup>.

#### 4.3.1.5 Anlage: Kreuzreferenztablelle zu elementaren Gefährdungen

Hier wird eine Kreuzreferenztablelle (s. Tabelle [B.1](#) auf Seite 51) dargestellt, die zeigt welche elementaren Gefährdungen<sup>16</sup> durch welche [Anforderungen](#) abgeschwächt oder gar eliminiert werden können. Folgende elementare Gefährdungen werden behandelt:

**G 0.14** Ausspähen von Informationen (Spionage)

**G 0.15** Abhören

**G 0.18** Fehlplanung oder fehlende Anpassung

**G 0.19** Offenlegung schützenswerter Informationen

**G 0.21** Manipulation von Hard- oder Software

**G 0.28** Software-Schwachstellen oder -Fehler

**G 0.30** Unberechtigte Nutzung oder Administration von Geräten und Systemen

**G 0.31** Fehlerhafte Nutzung oder Administration von Geräten und Systemen

**G 0.43** Einspielen von Nachrichten

**G 0.46** Integritätsverlust schützenswerter Informationen

Des Weiteren existiert eine Spalte „CIA“, welche die Grundwerte der Informationssicherheit, wie auf Seite 19 definiert, behandelt. C (Confidentiality) steht dabei für Vertraulichkeit, I (Integrity) für Korrektheit und A (Availability) für Verfügbarkeit.

<sup>13</sup>Vgl. Schildt, *IT-Grundschutz-Kompendium*, APP.3.1 S. 5.

<sup>14</sup>Vgl. *ebd.*, APP.3.1 S. 5.

<sup>15</sup>Vgl. *ebd.*, APP.3.4 S.6.

<sup>16</sup>Vgl. hierfür *ebd.*, Elementare Gefährdungen S. 1ff

## 4.4 OWASP Top 10

Das [OWASP](#) ist eine öffentliche Gemeinschaft, welche in dreijährigen Abständen Top-10 Listen<sup>17</sup> der am meisten ausgenutzten Sicherheitslücken veröffentlicht. Diese Listen sollen Unternehmen dazu animieren, sichere Webanwendungen bereitzustellen und sich gegen eben diese Bedrohungen zu schützen. Die zu diesem Zeitpunkt aktuellste Version ist die Liste aus dem Jahr 2021. Folgend werden die darin erhaltenen Sicherheitslücken aufgezählt, erläutert und erklärt, wie Entwickler Anwendungen dagegen schützen können.

### A01:2021 - Broken Access Control

Die sogenannten Fehler in der Zugriffskontrolle gewinnen gegenüber 2017 sehr an Bedeutung, da diese von Platz 5 auf Platz 1 gesetzt wurden.

Falls Benutzer einer Anwendung mehr Rechte bekommen als eigentlich vorgesehen, führt dies unweigerlich zur Freigabe von Daten, für die diese nicht [autorisiert](#) sind. Das könnten Angreifer über das Umgehen der Zugriffskontrollen erreichen, indem beispielsweise die [Uniform Resource Locator \(URL\)](#) verändert wird, die mitgeschickten Metadaten manipuliert oder das [JSON Web Token \(JWT\)](#) erneut verwendet oder modifiziert wird. Schon bei der Entwicklung müssen daher Rechte, Rollen und Regeln klar definiert sein, wie sich Benutzer [authentifizieren](#) können und welche Rechte an welche Benutzer vergeben werden dürfen. Dies sollte zudem durch Protokollierung der Ereignisse und Invalidierung der [JWTs](#) sichergestellt werden.

---

<sup>17</sup>Vgl. 2021 OWASPTop10Team. *OWASP Top 10:2021*. 2021. URL: <https://owasp.org/Top10/> (besucht am 06. 07. 2022).

**A02:2021 - Cryptographic Failures**

Vorher als „Sensitive Data Exposure“ oder „Verlust der Vertraulichkeit sensibler Daten“ bekannt, rücken die „kryptographischen Fehler“ auf Platz 2 der Liste.

In der Regel werden hierbei [Man-in-the-Middle Angriffe](#) ausgeführt, welche darauf abzielen, nicht die Verschlüsselung selbst zu brechen, sondern die Daten der Übertragung zur Kompromittierung zu nutzen. Des Weiteren kann durch das Verwenden unsicherer Protokolle, älterer Verschlüsselungsverfahren oder keiner verbindlich erzwingener Verschlüsselung, die Verwundbarkeit der Anwendung erhöht werden.

Verhindern kann man diese Bedrohung durch das Vermeiden von unnötigem Speichern von sensiblen Daten, Deaktivieren des Caches für diese Daten, oder durch die Nutzung sicherer Transportprotokolle.

**A03:2021 - Injection**

Diese Bedrohung ist von Platz 1 auf 3 herabgesetzt worden.

Falls die Webanwendung oder der [Web Service](#) eine Datenbankanbindung verwaltet, ist „Injection“ eine beliebte Angriffsstrategie. Angreifer versuchen durch gezielte Ausnutzung von [SQL-Queries](#) Daten offenzulegen, zu kompromittieren oder zu löschen. Mit einer gelungenen Injection kann beispielsweise auch ein Administratorkonto vorgespielt werden, was natürlich fatal wäre.

Durch korrekte Nutzung eines [Object Relational Mapping \(ORM\)-Frameworks](#) kann diese Bedrohung verringert werden. Zudem sollten Eingabedaten immer auf nicht erlaubte Buchstaben, sowie Zeichen und Länge geprüft werden.

**A04:2021 - Insecure Design**

Die neue Kategorie „Insecure Design“ bezieht sich auf Fehler in der Architektur und im Design von Anwendungen.

Hierbei werden Sicherheitslücken externer [Frameworks](#) und Design-Patterns ausgenutzt. Das [OWASP](#) versucht dadurch, Entwickler von Frameworks auf Sicherheitslücken aufmerksam zu machen und sichere Design-Patterns zu fördern.

Verhindert werden kann das durch die Nutzung mehrfach getesteter Frameworks, sowie die Entwicklung sicherer Entwurfsmuster.

**A05:2021 - Security Misconfiguration**

Diese Kategorie ist im Vergleich zum Jahr 2017 einen Platz weiter nach oben gerutscht. Sie beinhaltet Lücken in der Konfiguration von Firewalls, [Web Services](#) und Webanwendungen.

Angreifer könnten hierbei versuchen sich über Standardpasswörter Zugang zum System zu verschaffen, oder über Listings der Ordnerpfade die kompilierten Klassen finden, diese dann dekompilieren, um über „Reverse Engineering“ eine Sicherheitslücke zu finden. Entwickler können eine derartige Sicherheitslücke minimieren bzw. eliminieren, indem sie unnötige Features, [Frameworks](#) oder Komponenten deinstallieren und eine „abgespeckte“ Anwendung ausliefern. Weiterhin können über regelmäßige Updates des genutzten Systems/[Frameworks](#) veraltete Sicherheitslücken geschlossen werden.

### **A06:2021 - Vulnerable and Outdated Components**

Am zweitstärksten gewinnt diese Kategorie an Bedeutung, da ihre Wichtigkeit um 3 Stufen gestiegen ist. Erfasst wird die Nutzung von Komponenten mit bekannten Schwachstellen.

Ein aktuelles Beispiel davon ist die am 10. Dezember 2021 bekanntgewordene Sicherheitslücke des Logging-[Frameworks](#) „Log4j“ von Java. Durch die interne Struktur der Sprache können so Exploits vorgenommen und der Host-Rechner kontrolliert werden.

Unternehmen sollten daher immer darüber informiert sein, welche externen Bibliotheken genutzt werden, wie diese funktionieren und ob diese auf dem neuesten Stand sind. Im Falle eines Logging-[Frameworks](#) oder anderen Basiskomponenten ist es also manchmal auch von Vorteil, den Code entweder selbst zu schreiben oder Klassen (wie z. B. `java.util.logging`) direkt aus dem [Java Development Kit \(JDK\)](#) zu nutzen.

### **A07:2021 - Identification and Authentication Failures**

Sehr an Bedeutung verloren haben die „Fehler in der [Authentifizierung](#)“.

Angriffe auf die [Authentifizierung](#) sind gut erforscht, wodurch es bereits sehr viele Skripte gibt, welche Wörterbücher als Datengrundlage nutzen, um durch [Brute-Force](#)-Angriffe Benutzernamen und Passwörter herauszufinden. Besonders gefährdet sind auch [Web Services](#) mit nicht erlöschenden Session-Tokens, welche dann für die falsche [Authentifizierung](#) genutzt werden können.

Sofern möglich, sollten daher folgende Themen implementiert sein, um der [Authentifizierung](#) einen höheren Sicherheitsstandard mitzugeben. Für die Risikominimierung automatisierter Angriffe sollte eine Mehrfaktor-[Authentisierung](#) implementiert werden und im Zuge dessen sollten auch alle Passwörter für ein neu angelegtes Nutzerkonto gegen eine Liste der 10000 beliebtesten Passwörter geprüft werden. Zusätzlich dazu dürfen im Auslieferungszustand keine Standardbenutzer und/oder administrative Nutzer vorhanden sein, wobei auch bei fehlgeschlagenen Anmeldeversuchen immer die gleiche Fehlermeldung zurückgegeben werden soll, um somit die Nutzernamen nicht erraten zu können.

**A08:2021 - Software and Data Integrity Failures**

Die zweite neue Kategorie bezieht sich auf Sicherheitslücken in Software-Updates, kritischen Daten und [CI/CD-Pipelines](#) ohne Integritätsprüfung. Hierzu zählt unter anderem die veraltete Kategorie aus 2017 „A08:2017 - Insecure Deserialization“.

Beispielsweise kann im Prozess der Deserialisierung angesetzt werden, um mit fehlerhaften Routinen Schadcode auf dem [Web Service](#) ausführen zu können. Des Weiteren kann über eine fehlende Prüfung neuer Pakete, welche automatisiert nachgeladen werden, externer Code auf einem bestehendem System installiert und ausgeführt werden.

Digitale Signaturen oder ähnliche Mechanismen für die Validierung der Pakete sind daher erforderlich, um solche Risiken auszuschließen. Zudem sollte – gerade im Kontext von Webanwendungen – sichergestellt werden, dass Paketmanager (wie z. B. Maven, Gradle oder npm) Code nur aus vertrauenswürdigen Quellen laden.

**A09:2021 - Security Logging and Monitoring Failures**

Die damalige Kategorie „A10:2017 - Insufficient Logging & Monitoring“ beinhaltet vor allem unzureichende Protokollierung von Anmeldeversuchen oder falsche Sichtbarkeit von Logging-Informationen.

Angreifer können also automatisiert und mit großer Anzahl an Tests versuchen, Anmeldedaten zu erraten. Weiterhin könnten Logs abgegriffen und interpretiert werden.

Es ist also dringend erforderlich, Anmeldeversuche zu protokollieren und die richtigen Schlüsse daraus zu ziehen, beispielsweise die [IP](#)-Adresse zu sperren oder ähnliches. Weiterhin muss darauf geachtet werden, wohin die Logs geschrieben werden, da diese Informationen für außenstehende Personen nicht zugänglich sein dürfen.

**A10:2021 - Server-Side Request Forgery (SSRF)**

Die [Server-Side-Request-Forgery \(SSRF\)](#) ist die dritte neue Bedrohung in 2021 und bezieht sich auf die Kontrolle des [Web Services](#) über Requests, welche vom Angreifer so manipuliert werden, dass die Applikation anderen Code als geplant ausführt oder sogar geschützte Daten preisgibt.

Ausgenutzt werden kann dies, indem zum Beispiel [URL](#)-Anforderungen durch 127.0.0.1 oder „localhost“ ersetzt werden und diese dadurch an das eigene Lookup-Interface weitergeleitet werden.

Schutzmechanismen dagegen sind rar, da sich unter anderem nicht einmal Bibliotheken einig sind, wie gewisse [Uniform Resource Identifier \(URI\)s](#) zu interpretieren sind. Es könnte allerdings – je nach Anwendung – entweder der Whitelist, oder der Blacklist Ansatz für die Validierung der [URLs](#) verwendet werden.



## 5 Realisierung

Wie in den vorhergehenden Kapiteln schon angesprochen und erörtert, wurde der [Web Service](#) und der [Autorisierungsserver](#) in [.NET Core 2.1](#) implementiert, während das Demo-Programm und die anderen Applikationen, die den [Web Service](#) nutzen, in [.NET Framework](#) implementiert sind.

### 5.1 Gesamtstruktur

Das entwickelte Softwaresystem besteht mittlerweile aus drei unterschiedlichen Komponenten, welche für einen reibungslosen Betrieb gut zusammenarbeiten müssen. Diese Programme werden dafür benötigt, die [Authentifizierung](#) durchzuführen, Daten abzufragen und diese letztendlich auch auszuwerten. Im Folgenden wird der Code von den einzelnen Bestandteile analysiert und erklärt, wobei nur auf relevante Zeilen/Methoden/Aufrufe eingegangen wird. Die selbsterklärenden Codefragmente bleiben größtenteils unerwähnt um die wichtigen Zeilen/Aufrufe hervorheben zu können, sodass der Code aber trotzdem genau so lauffähig wäre. Zusätzlich ist noch zu erwähnen, dass [.NET Core 2.1](#) per Konstruktion zwei Klassen benötigt um einen [Web Service](#) erstellen zu können. Die Klasse `Program.cs` beinhaltet hierbei die `Main` Methode, welche intern die Klasse `Startup.cs` erstellt.

#### 5.1.1 Autorisierungsserver

Der Autorisierungsserver stellt das Herzstück der implementierten [OAuth 2.0 Autorisierung](#) dar. Wie in Listing [C.1](#) auf Seite [55](#) zu sehen, wird hier ein neuer [Web Service](#) erstellt, welcher auf den Port 5003 (s. Zeile 15) hört und das [HTTPS](#) nutzt. In Zeile 14 wird festgelegt, dass die Klasse `Startup` genutzt werden soll, welche in Listing [C.2](#) auf Seite [55](#) dargestellt ist. Diese wird dann per [Konstruktor Injektion](#) erstellt und bekommt ein Objekt vom Typen `IConfiguration`. Die eigentlich wichtige Methode für den Autorisierungsserver ist `ConfigureServices`. Diese bekommt eine Kollektion vom Typen `IServiceCollection` übergeben und fügt dieser über einige Methodenaufrufe Funktionen und Eigenschaften hinzu, sodass eine [Autorisierung](#) stattfinden kann. In Zeile 12 und 13 werden die vordefinierten Bereiche und die daraus resultierenden Clienten über die Methoden `Config.GetApiResources` und `Config.GetClients` hinzugefügt. In Listing [C.3](#) auf Seite [56](#) sind eben diese Methoden vorhanden. Zeile 5 und 6 zeigen, dass zwei „scopes“ erstellt werden, einer nur mit Leserechten, der andere mit vollen Rechten. Diese werden

dann wiederum in der Methode `getClients` genutzt, in welcher zwei Clienten erstellt werden. Der erste Client (Zeile 12–20) kann hier wieder nur lesen und bekommt daher auch genau diesen „scope“ zugewiesen, während der andere (Zeile 21–29) wieder alles darf. Des Weiteren existiert in `Startup` noch die Methode `Configure`, welche vor allem dafür zuständig ist, dieses Programm als Identitätsserver darzustellen und `HTTPS` zu nutzen. Zusätzlich wird noch entschieden, ob entwicklerspezifische Fehlercodes angezeigt werden sollen oder nicht, was davon abhängt, ob der Aufruf des Programms im „Debug“ oder „Release“ Modus stattfindet.

### 5.1.2 Webservice

Das eigentliche Programm nutzt in Listing C.4 auf Seite 57 fast den gleichen Code wie der `Autorisierungsserver`, da beide Programme einen `Web Service` darstellen. Einzig der Port auf den dieser `Web Service` hört unterscheidet sich (s. Zeile 15), um beide gleichzeitig nutzen zu können. Ansonsten wird hier ebenfalls `Konstruktor Injektion` genutzt und dadurch die Klasse `Startup` erstellt, welche sich in den Methoden `ConfigureServices` und `Configure` unterscheidet. Zur besseren Übersichtlichkeit wird daher `ConfigureServices` in Listing C.5 auf Seite 57 und `Configure` in Listing C.6 auf Seite 58 dargestellt. Beide Methoden konfigurieren den zu erstellenden `Web Service` und werden intern beim Programmstart aufgerufen.

#### 5.1.2.1 ConfigureServices

In Zeile 7 wird ein `Singleton` einer `AutoMapper` Klasse angelegt und den Services hinzugefügt, welches später dafür sorgen wird, dass die Objekte aus der Datenbank zu sogenannten „shallow Objects“ zugeordnet werden. Das ist sinnvoll, da nicht alle Attribute aus der Datenbank zwingend in diesen Objekten präsentiert werden müssen. Zeile 13 fügt ebenfalls `Singletons` hinzu, die sich um die `Autorisierung` und die `API`-Verwaltung, sprich, wo welche Methoden stehen, kümmern. Die Zeilen 15–34 bearbeiten weiter die `Autorisierung` und verlinken unter anderem die Zugangsberechtigungen und den `Autorisierungsserver` mit diesem `Web Service`. Dafür werden noch Attribute definiert, die als Optionen zu den `HTTP`-Methoden zugeordnet werden können, um die Berechtigungen einzuschränken. Für die einfachere Entwicklung und grafische Ansicht der nutzbaren `HTTP`-Methoden wird Swagger<sup>1</sup> genutzt und in den folgenden Zeilen eingerichtet. Ein weiterer Vorteil hiervon ist, dass durch Swagger automatisch eine XML-Datei erstellt wird, welche in der

---

<sup>1</sup>S. SmartBear. *Swagger UI*. 2022. URL: <https://swagger.io/tools/swagger-ui/> (besucht am 06.07.2022).

[OpenAPI Specification 3.0](#)<sup>2</sup> formatiert ist. Zuletzt wird noch die [SQL](#)-Datenbank über einen `DbContext` hinzugefügt und angesprochen, indem die Verbindungszeichenfolge aus einer Konfigurationsdatei ausgelesen wird.

### 5.1.2.2 Configure

Hier wird neben anderen Konfigurationen ein selbst geschriebener `ExceptionHandler` der Applikation hinzugefügt, welcher angesprochen wird, falls bei der Ausführung irgendeiner [HTTP](#)-Methode eine Ausnahme geworfen wird. In dem internen `switch-case` Konstrukt wird zwischen einer `ArgumentOutOfRangeException`, einer `DataNotFoundException`<sup>3</sup> und allen anderen Exceptions unterschieden. Es werden jeweils die Fehlermeldungen in [JavaScript Object Notation \(JSON\)](#) umgewandelt und zusammen mit einem entsprechenden [HTTP](#)-Statuscode zurückgegeben. Die letzten Zeilen setzen dann nur noch einige Attribute des Programms.

### 5.1.2.3 ApiControllerBase

Dies ist die Elternklasse, welche von allen anderen Controller-Klassen implementiert wird und einige nützliche Attribute bereitstellt. Die Annotationen über dem Klassennamen von Zeile 2–6 werden ebenfalls weitervererbt. Über `[ApiController]` wird die Klasse als Controller gekennzeichnet und kann nun [HTTP](#)-Methoden implementieren, welche alle über `[Produces(„application/json“)]` eine in [JSON](#) formatierte Datei zurückgeben. Um diese Methoden danach aufrufen zu können, wird mit `[Route(„api/controller“)]` festgelegt, dass die Adresse mit `api/<Name des Controllers>/` beginnt. Die [Autorisierung](#) wird in den Zeilen 5–6 gesteuert, indem Schemata und Policy gesetzt werden, welche allerdings noch für einzelne Methoden überschrieben werden können. Diese Klasse arbeitet ebenfalls mit [Konstruktor Injektion](#) und bekommt daher einige Objekte, die dafür genutzt werden, die Klassenattribute zu setzen um damit weiterarbeiten zu können. Der übergebene `IActionDescriptorCollectionProvider` enthält eine zuvor generierte Liste aller verfügbaren Methoden und Links. Wie schon zuvor angesprochen, ist es sinnvoll, nicht alle Attribute des Datenbankobjektes preiszugeben, wofür das Objekt des Typen `IMapper` zuständig ist. Des Weiteren kann jeder Controller mit dem `DbContext` Daten aus der Datenbank abfragen und der `logger` ist natürlich zur Protokollierung da.

<sup>2</sup>S. Darrel Miller u. a. *OpenAPI Specification V3.0.0*. Juli 2017. URL: <https://spec.openapis.org/oas/v3.0.0> (besucht am 06.07.2022).

<sup>3</sup>S. Listing [C.7](#) auf Seite [59](#)

#### 5.1.2.4 Controller

In Listing C.9 auf Seite 60 ist beispielhaft ein Controller dargestellt, welcher `ApiControllerBase` implementiert. Es wird hier dargestellt, wie eine `GET` und eine `HEAD` Route erstellt werden kann und wie die Datenbankabfrage funktioniert.

Im Konstruktor passiert nichts, außer dass der Super-Konstruktor wieder durch `Konstruktor Injektion` aufgerufen wird. Die eigentliche Arbeit passiert in der Methode `GetShadowCountries`, wobei diese durch die Annotationen in Zeile 7 und 8 eine `GET` und eine `HEAD` Methode darstellt. Intern wird hier über das `LfiidContext` Objekt eine Menge aller Länder abgefragt. Falls das resultierende Objekt leer ist, wird eine `DataNotFoundException`<sup>4</sup> geworfen und ansonsten wird damit weitergearbeitet. Die Zeilen 12–15 stellen eine `LINQ` Abfrage dar, welche die zurückgegebene Menge zuerst nach „CustareaCode“<sup>5</sup> und dann nach „IcaoCode“<sup>6</sup> sortiert. Die Elemente werden dann über den „IcaoCode“ gruppiert und schließlich in ein `CountryModel` gemappt. Mit dem Aufruf von `return Ok(...)` wird die resultierende Menge als formatiertes `JSON`-Objekt zurückgegeben. Das Rückgabeobjekt ist eine `Task` von `ActionResult` von `IEnumerable` von `CountryModel`, um unter anderem die Methode asynchron gestalten zu können.

#### 5.1.3 Tests

Die hier aufgeführten Tests existieren vor allem, um die Unterschiede der Request-Zeiten, wie in Abbildung A.2 auf Seite 44 dokumentiert, zu zeigen. Die Klasse `AuthorizedClientFixture` zeigt dabei, wie sich ein Client mit dem `Web Service` autorisieren und verbinden kann. Zeile 7–11 in Listing C.10 auf Seite 61 setzt die nötigen Informationen für den `Authentifizierungsprozess`, wie die `URL` des `Authorisierungsservers`, die `clientId`, den `scope` und das `clientSecret`. Diese Parameter werden dann für die Funktion `RequestTokenToAuthorizationServer` genutzt, welche das `Access Token` beantragt und bei Erfolg zurückgibt. Für die Beantragung werden `HTTP`-Header Felder genutzt. Nach der Rückkehr aus dieser Methode wird das `Access Token` aus dem `JSON`-Format geparkt und an den Clienten weitergegeben, welcher nun ein Header-Feld namens „Bearer“ mit dem `Access Token` als Wert setzen kann. Dieser Client wird dann genutzt, um `autorisierte` Anfragen an den `Webservice` zu schicken. In dem in Listing C.12 auf Seite 63 gezeigten Fall werden alle vorhandenen Länder angefragt. Der Request an sich findet in der Methode

<sup>4</sup>S. Listing C.7 auf Seite 59

<sup>5</sup>Der `CustareaCode` stellt einen im `Aeronautical Radio Incorporated (ARINC)`-Format spezifizierten Code für die korrespondierende Region dar.

<sup>6</sup>Der `International Civil Aviation Organization (ICAO)`-Code dient zur eindeutigen Identifizierung von Flugplätzen. Da die ersten zwei Buchstaben das Land beschreiben, werden diese hier auch in der Tabelle der Länder geführt.

`GetCountryModels` in Zeile 28–36 statt, wobei mit einem asynchronen Aufruf der gegebenen [URI](#) die Ressource angefragt wird. Diese Liste der Länder wird dann deserialisiert und als `List` Objekt zurückgegeben. Die gleiche Methode findet sich in Listing [C.13](#) auf Seite [64](#) wieder, mit der einzigen Unterscheidung, dass dieser Client nicht [autorisiert](#) ist. Der letzte Test (s. Listing [C.11](#) auf Seite [62](#)) zeigt, wie sich ein Programm direkt mit der Datenbank verbinden kann und Daten über eine [SQL](#)-Anfrage erhält.

#### 5.1.4 Demo Anwendung

Diese Anwendung wird zur Nachbildung der Simulatorkomponente genutzt, welche später die Daten abfragen soll. Der Code für die GUI ist hier extra nicht erwähnt, da dieser nicht wirklich relevant für die Arbeit ist und die Erklärung unnötig verkompliziert. Die Klasse `MainWindow`<sup>7</sup> stellt im Kern auch nur einen [autorisierten](#) Clienten bereit, wie auch schon in Listing [C.10](#) auf Seite [61](#) erklärt. Wie und über welche Routen die Requests abgeschickt werden, wird in Listing [C.16](#) auf Seite [67](#) dargestellt. Diese sind allerdings fast die gleichen Methoden, die auch in den [Tests](#) genutzt werden.

## 5.2 Sicherheitsaspekte

Für die bessere Übersicht der realisierten und behandelten Sicherheitsaspekte wird alles, was im Folgenden behandelt wird, noch einmal in Tabelle [B.3](#) auf Seite [53](#) kurz zusammengefasst. Zur Beibehaltung der Struktur werden die Risiken der [OWASP Top 10](#) zu einigen Anforderungen des [IT Grundschutz Kompendiums](#) zugeordnet und – wenn möglich – chronologisch abgearbeitet.

Die Bedrohung [A01:2021](#) wird dadurch abgeschwächt, dass es bei diesem [Web Service](#) zwar keine klassischen Nutzer gibt, jedoch jedes [autorisierte](#) Programm eigene Rechte und Bereiche zugewiesen bekommt, welche freigegeben und erreicht werden dürfen. Damit wird vor allem die [Basis Anforderung APP.3.1.A1](#) neben anderen erfüllt.

Durch die Nutzung von [TLS/SSL](#) wird [A02:2021](#) entgegengewirkt, da somit die übertragenen Daten generell verschlüsselt sind und auch gar nicht erst verschickt werden, falls der Kommunikationspartner nicht [autorisiert](#) ist. Dies kann den [Standard-Anforderungen APP.3.1.A8](#), [APP.3.1.A9](#) und [APP.3.1.A11](#) zugeordnet werden.

Da [.NET Core 2.1](#) mit einem [Framework](#) zur Erstellung eines [Web Services](#) genutzt wird, wird bei jeder Route automatisch gefragt, ob der gegebene Parameter dem Übergabewert entspricht. Zudem sind die Implementierungen der Methoden mit [LINQ](#) gestaltet,

---

<sup>7</sup>S. Listing [C.14](#) auf Seite [65](#)

was bedingt, dass intern Microsofts Entity Framework für die Datenbankabfragen genutzt wird, wodurch die übergebenen Werte noch zusätzlich geprüft werden und quasi keine [SQL](#)-Injektion mehr möglich ist. Hiermit wird [APP.3.1.A14](#), [APP.3.1.A9](#) und [APP.3.1.A11](#) erfüllt sowie [A03:2021](#) entgegen gewirkt.

Die vierte Bedrohung, [A04:2021](#), bezieht sich vor allem auf [APP.3.1.A8](#) und [APP.3.1.A9](#), die Beschaffung von externem Material, da vor allem auch Fremdcode auf Sicherheit geprüft werden muss. Dieses Programm benutzt daher auch nur Pakete, welche entweder direkt von Microsoft zur Verfügung gestellt werden oder sehr viele Nutzer und positive Einträge in Bezug auf die Sicherheit hat. Zudem wird bei jedem Paket oder auch [Framework](#) darauf geachtet, dass der aktuellste Patch der jeweiligen Version genutzt wird, keine Versionskonflikte entstehen und notwendige Updates durchgeführt werden. Damit wird auch direkt [A06:2021](#) und [A08:2021](#) entgegen gewirkt. Leider kann dies aber hinsichtlich der Rahmenbedingungen des [Betriebssystems](#) nicht bei .NET Core gewährleistet werden.

Da der [Web Service](#) so offen wie nötig, aber so restriktiv wie möglich gestaltet werden soll und wurde, wird [A05:2021](#) und [A07:2021](#) angegangen. Speziell können [APP.3.1.A1](#), [APP.3.1.A7](#), [APP.3.1.A14](#), [APP.3.1.A9](#) und [APP.3.1.A12](#) dieser Bedrohung zugeordnet werden. Von Anfang an wurden nur [autorisierten](#) Clienten Methoden freigegeben und keine Standardpasswörter oder -benutzerkennungen benutzt, wobei eine zusätzliche Sicherheitsebene durch den implementierten [OAuth 2.0](#)-Standard, sowie dem Auslaufen der somit generierten [Access Tokens](#) geschaffen wurde.

Zur Aufzeichnung der Aktionen auf dem [Web Service](#) sollten während jeder Sitzung Logging-Dateien erstellt werden, was auch von [APP.3.1.A1](#) gefordert ist. Um beispielsweise fehlgeschlagene Anmeldeversuche erkennen und nach Gefährlichkeit einstufen zu können, müssen diese Informationen sicher gespeichert werden und insbesondere dürfen diese nicht an den Nutzer weitergegeben werden. Damit wird [A09:2021](#) vereitelt, falls unter anderem bei einem fehlgeschlagenen [Authentifizierungsversuch](#) dieser gespeichert wird und an den Nutzer nur die Meldung zurückgegeben wird, dass der Versuch nicht erfolgreich war. Zusätzlich wird durch die Protokollierung [APP.3.1.A7](#) erfüllt. Hier werden alle Aktionen des [Web Services](#) in einer Datei gespeichert, welche von außen nicht einsehbar ist, um alles nachvollziehen zu können.

Die letzte zu behandelnde Bedrohung [A10:2021](#) zielt eher auf eine Gefährdung für den Nutzer und nicht für den bereitstellenden Dienst ab. Trotzdem kann damit noch einmal die Anforderung [APP.3.1.A12](#) angesprochen und realisiert werden, da schon mit einer restriktiven Freigabe von Methoden sehr viel erreicht wird, was sich auch in der Implementierung wiederfindet.

## 5.3 Bedienung

Falls ein Client beim [Autorisierungsserver](#) registriert ist und Rechte, Bereiche sowie eine ID zugewiesen bekommen hat, so funktioniert die Bedienung immer genauso wie es in [Abbildung A.3](#) auf Seite 45 dargestellt ist. Zuerst muss der Client ein [JWT](#) beim [Autorisierungsserver](#) beantragen, welcher dieses nach erfolgreicher Validierung als Antwort zurückschickt. Danach folgt erst die eigentliche Anfrage der Daten. Die Abbildungen [HTTP-Methoden S. 1](#), [HTTP-Methoden S. 2](#) und [HTTP-Methoden S. 3](#) zeigen die momentan verfügbaren [URLs](#) des [Webservices](#), welche alle – bis auf [/api/Countries/fullAnonymous](#)<sup>8</sup> – nur über eine vorherige [Autorisierung](#) erreichbar sind. Der Client muss nun also zusammen mit dem [JWT](#) im Header eine Anfrage an eine dieser Methoden losschicken, welche dann vom [Webservice](#) weiter bearbeitet werden kann. Dieser lässt zuerst das Token durch den [Autorisierungsserver](#) validieren, um bei Erfolg die Daten bei der Datenbank des [LFID-Systems](#) über eine [SQL](#)-Anfrage abfragen zu können. Falls die Daten erfolgreich aus der Datenbank gelesen wurden, werden diese wieder als [JSON](#)-Objekt an den Clienten zurückgeschickt.

---

<sup>8</sup>Die Route [/api/Countries/fullAnonymous](#) wurde zu Testzwecken eingeführt, um vor allem die [Abbildung A.2](#) auf Seite 44 erstellen zu können.





## 6 Ergebnis und Schlussfolgerungen

Die Gestaltung der [Authentifizierung](#) war – wie im [Autorisierungsserver](#) zu sehen – erfolgreich und realisierbar, sowie auch die Bereitstellung einiger Methoden zur Abfrage der Daten. Grundlegend konnten also die [obligatorischen Anforderungen](#) erfüllt werden. Das Demoprogramm fragt im Hintergrund genau die Daten strukturiert ab, welche von einem echten [Flight Management System \(FMS\)](#) benötigt werden. Die Benutzeroberfläche ist dabei eher zweitrangig geblieben, da verschiedene [FMSs](#) mit unterschiedlichen Anzeigen existieren. Wichtig hierbei ist nur, dass die angefragten Daten kaskadenartig präsentiert werden.

Laut dem [Vergleich der Requestzeiten](#) braucht die webbasierte Anfrage mit einer [Autorisierung](#) ca.  $5ms \pm 2ms$ . Damit unterscheidet sich die Dauer kaum zur [unautorisierten](#) Anfrage, woraus schlussgefolgert werden kann, dass sich eine [Autorisierung](#) nicht negativ auf die Antwortzeiten auswirkt. Offensichtlich ist die direkte [SQL](#)-Abfrage vor allem nach der ersten Anfrage viel schneller, allerdings ist genau dies nicht gewünscht. Bis auf den [Web Service](#) soll kein anderes Programm direkte [SQL](#)-Statements an die Datenbank absetzen dürfen. Wenn also nur noch der [Web Service](#) direkten Zugriff auf die Datenbank hat, wird das ganze System durch die Einheitlichkeit und der eliminierten Fehlerquelle von mehreren gleichzeitigen Anfragen stabiler und auch schneller.



## 7 Zusammenfassung und Ausblick

Vor allem zur Bereitstellung neuer Daten und Formate des [LFID](#)-Systems sollte ein [Web Service](#) so gestaltet und konzipiert werden, dass dieser sicher mit den Clienten kommunizieren kann und sowohl zuverlässig als auch schnell läuft. Hierfür wurde die Grundlage der webbasierten Kommunikation mit dem [HTTP](#) geschaffen, welches mit [TLS/SSL](#) chiffriert Daten verschicken kann. Allerdings reicht das noch nicht für eine sichere Kommunikation, wofür danach [OAuth 2.0](#) vorgestellt wurde. Dieses Protokoll wird daher zur [Autorisierung](#) der Clienten beim [Web Service](#) genutzt, indem die [Authentifizierung](#) auf einen weiteren [Web Service](#) ausgelagert wird. Um das alles implementieren zu können wurde danach das Zielsystem analysiert, insbesondere welche Sprachen für die Implementierung sinnvoll und realisierbar sind. Die Entscheidung fiel auf C# und [.NET Core 2.1](#) für den [Web Service](#) und auf [.NET Framework](#) für das Demoprogramm. Für die korrekte Gestaltung dieser Anwendung wurde danach das [IT Grundschutz Kompendium](#) angesprochen, wobei aufgrund der schieren Größe dieses Dokuments nur auf den Baustein [APP.3.1](#) eingegangen wurde. Dieser wurde dann aufgespalten und für jeden Unterpunkt analysiert. Wie auch in [Weiterführende Informationen](#) beschrieben, folgten die Top 10 Bedrohungen des [OWASP](#), welche eine etwas allgemeinere Definition der Sicherheitsrisiken im Internet darstellen, aber trotzdem gut im Zusammenspiel mit dem [IT Grundschutz Kompendium](#) funktionieren. Dies sieht man auch in der [IT Sicherheitsanforderungstabelle](#), da dort jeder Bedrohung eine oder mehrere Anforderungen hinzugefügt worden sind. Die Beschreibung des Quellcodes befindet sich im Kapitel [Realisierung](#), wobei unter anderem viel auf die genutzten Attribute des [.NET Core 2.1-Frameworks](#) eingegangen wurde.

Um an diese Arbeit anzuknüpfen und den [Web Service](#) breiter nutzbar zu machen, könnte man – wie auch schon in den [optionalen Anforderungen](#) beschrieben – auf jeden Fall zu allen Datenbankentitäten Routen erstellen. Diese sollten alle [CRUD](#)-Operationen widerspiegeln und auch derart abgesichert sein, dass es einen Unterschied zwischen nur lesenden und bearbeitenden Clienten gibt. Weiterhin sollte eine Benutzeroberfläche für den [Autorisierungsserver](#) implementiert werden, welche den Administrator der Anwendungen dazu befähigt, Rollen und Gruppen zu erstellen und somit auch Rechte an Clienten zu verteilen. Eine Webseite für den [Web Service](#) sollte auch existieren, sodass der Administrator gewisse Einstellungen bei dem Programm vornehmen kann. Dazu zählen zum Beispiel ein Datenbank Backup speichern und laden, neue Daten in die Datenbank einspielen und Logging Dateien auslesen können. Vor allem aber soll der Verwalter des

Systems das Datenbank Passwort setzen können und somit die Verbindung zur Datenbank herstellen. Zudem muss einer der nächsten Schritte sein, die Programme auf die neuste Version von .NET (aktuell .NET 6.0) umzustellen um dadurch Sicherheitslücken und bekannte Fehler beheben zu können.

# Literaturverzeichnis

- Allen, Christopher und Tim Dierks. *The TLS Protocol Version 1.0*. RFC 2246. Jan. 1999. DOI: [10.17487/RFC2246](https://doi.org/10.17487/RFC2246). URL: <https://rfc-editor.org/rfc/rfc2246.txt>.
- Belshe, Mike, Roberto Peon und Martin Thomson. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540. Mai 2015. DOI: [10.17487/RFC7540](https://doi.org/10.17487/RFC7540). URL: <https://rfc-editor.org/rfc/rfc7540.txt>.
- Booth, David, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris und David Orchard. *Web Services Architecture*. Feb. 2004. URL: <https://www.w3.org/TR/ws-arch/> (besucht am 06.07.2022).
- Fielding, Roy T. und Julian Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. RFC 7230. Juni 2014. DOI: [10.17487/RFC7230](https://doi.org/10.17487/RFC7230). URL: <https://rfc-editor.org/rfc/rfc7230.txt>.
- Hammer-Lahav, Eran. *The OAuth 1.0 Protocol*. RFC 5849. Apr. 2010. DOI: [10.17487/RFC5849](https://doi.org/10.17487/RFC5849). URL: <https://rfc-editor.org/rfc/rfc5849.txt>.
- Hardt, Dick. *The OAuth 2.0 Authorization Framework*. RFC 6749. Okt. 2012. DOI: [10.17487/RFC6749](https://doi.org/10.17487/RFC6749). URL: <https://rfc-editor.org/rfc/rfc6749.txt>.
- Miller, Darrel, Jeremy Whitlock, Marsh Gardiner, Mike Ralphson, Ron Ratovsky und Uri Sarid. *OpenAPI Specification V3.0.0*. Juli 2017. URL: <https://spec.openapis.org/oas/v3.0.0> (besucht am 06.07.2022).
- Nielsen, Henrik, Roy T. Fielding und Tim Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.0*. RFC 1945. Mai 1996. DOI: [10.17487/RFC1945](https://doi.org/10.17487/RFC1945). URL: <https://rfc-editor.org/rfc/rfc1945.txt>.
- OWASPTop10Team, 2021. *OWASP Top 10:2021*. 2021. URL: <https://owasp.org/Top10/> (besucht am 06.07.2022).
- Rescorla, Eric. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018. DOI: [10.17487/RFC8446](https://doi.org/10.17487/RFC8446). URL: <https://rfc-editor.org/rfc/rfc8446.txt>.
- Rescorla, Eric und Tim Dierks. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. Aug. 2008. DOI: [10.17487/RFC5246](https://doi.org/10.17487/RFC5246). URL: <https://rfc-editor.org/rfc/rfc5246.txt>.
- Schildt, Holger. *IT-Grundschutz-Kompendium*. Bundesamt für Sicherheit in der Informationstechnik, Bonn, Feb. 2022. ISBN: 978-3-8462-0906-6.
- SmartBear. *Swagger UI*. 2022. URL: <https://swagger.io/tools/swagger-ui/> (besucht am 06.07.2022).

Srivastava, Nimit. *Differences between .NET Core and .NET framework*. Juni 2022.  
URL: <https://www.geeksforgeeks.org/differences-between-net-core-and-net-framework/#:~:text=Application%20Models-,.,windows%20forms%20and%20WPF%20applications>. (besucht am 06.07.2022).

# A Abbildungen

Abbildung A.1: Schichtenmodell

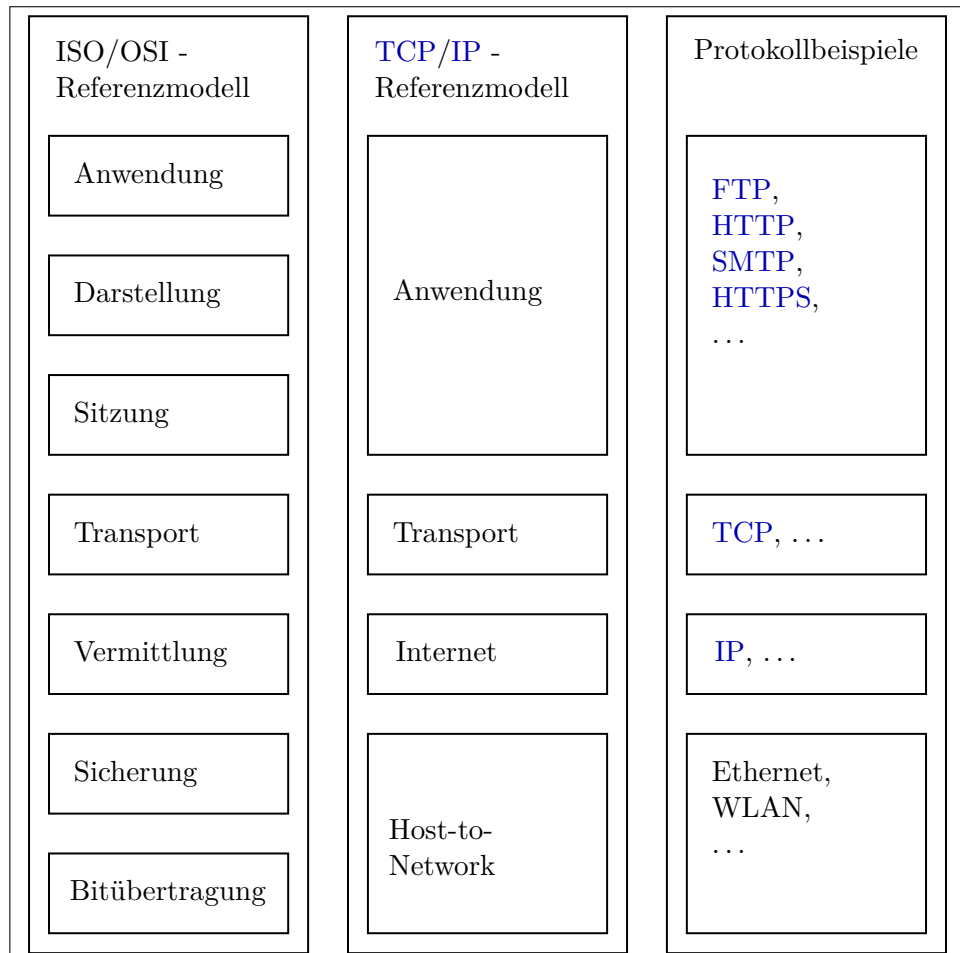


Abbildung A.2: Vergleich der Requestzeiten

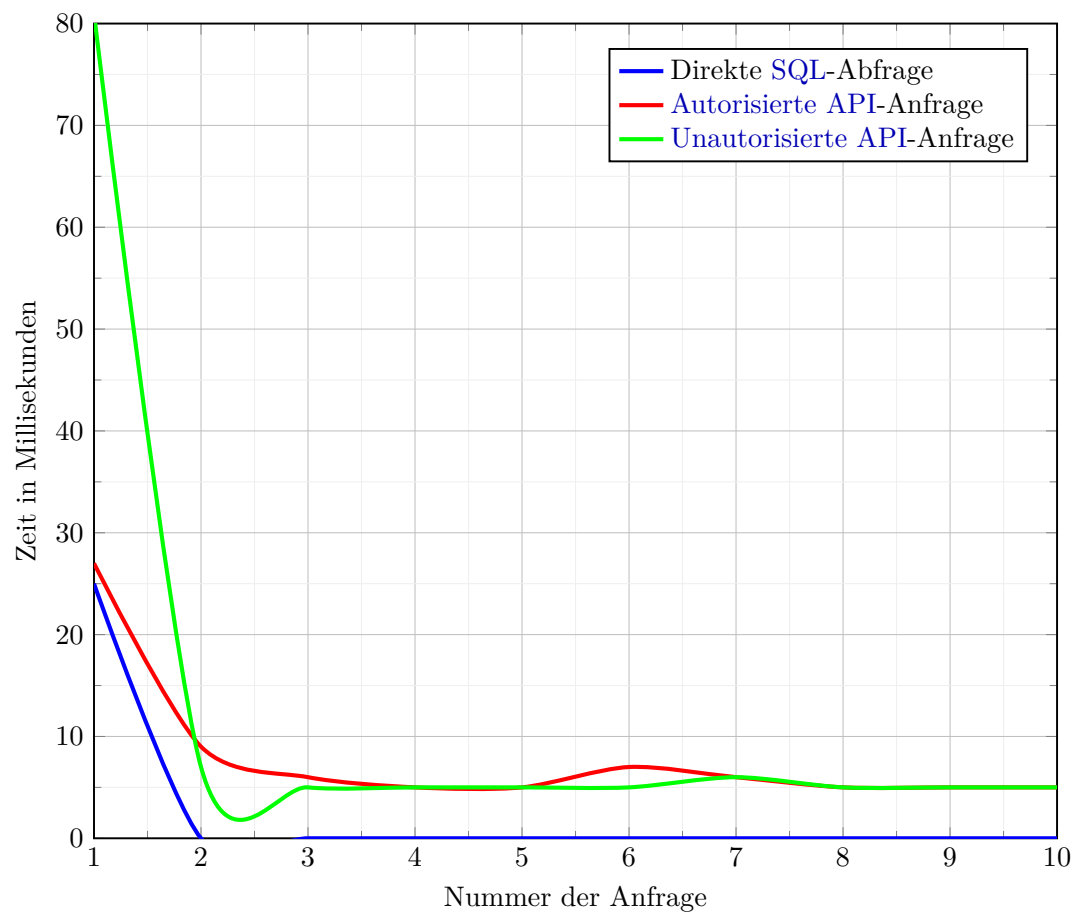
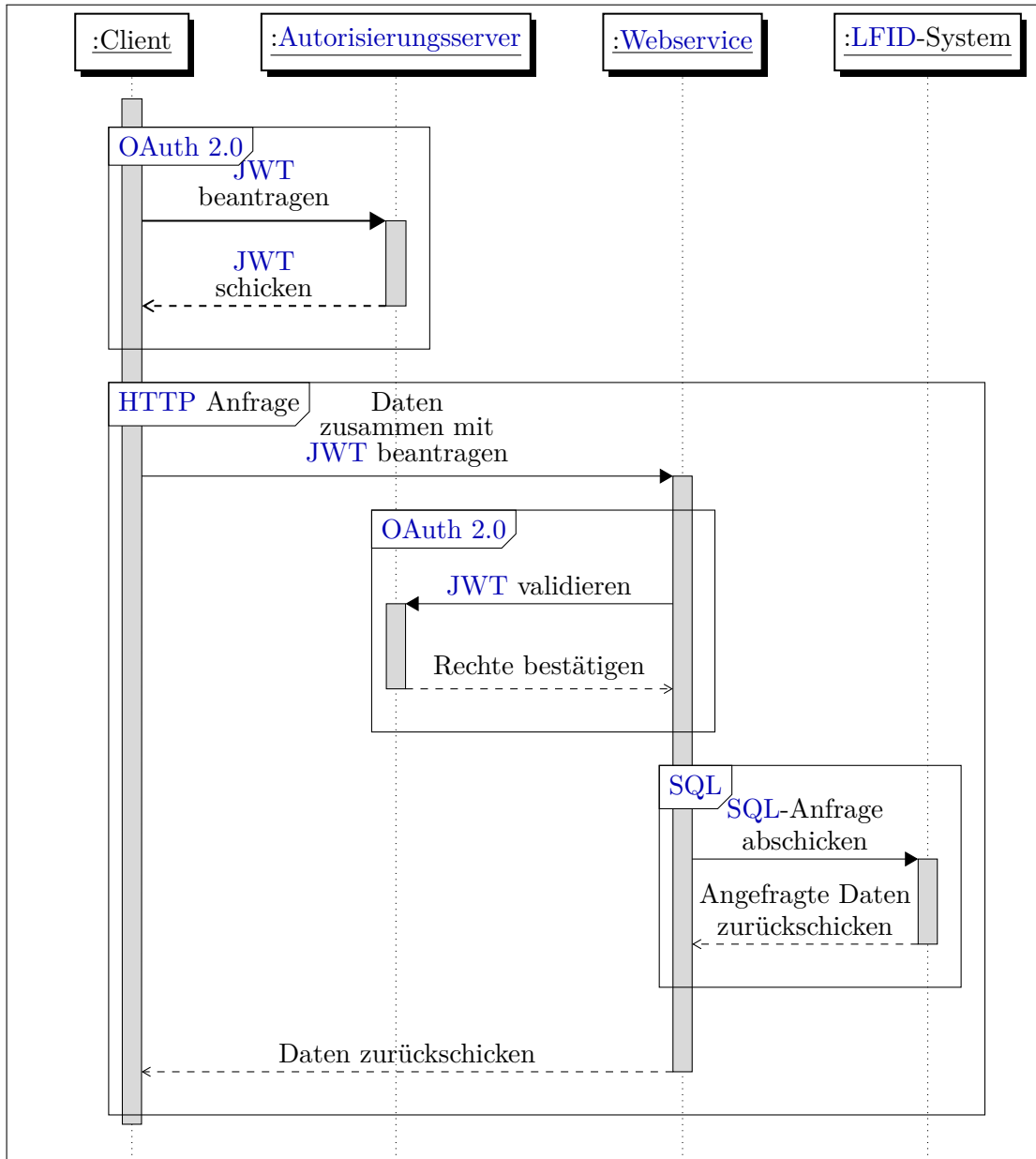




Abbildung A.3: Sequenzdiagramm einer Anfrage



**Abbildung A.4:** Pseudodiagramm der relevanten Datenbanktabellen

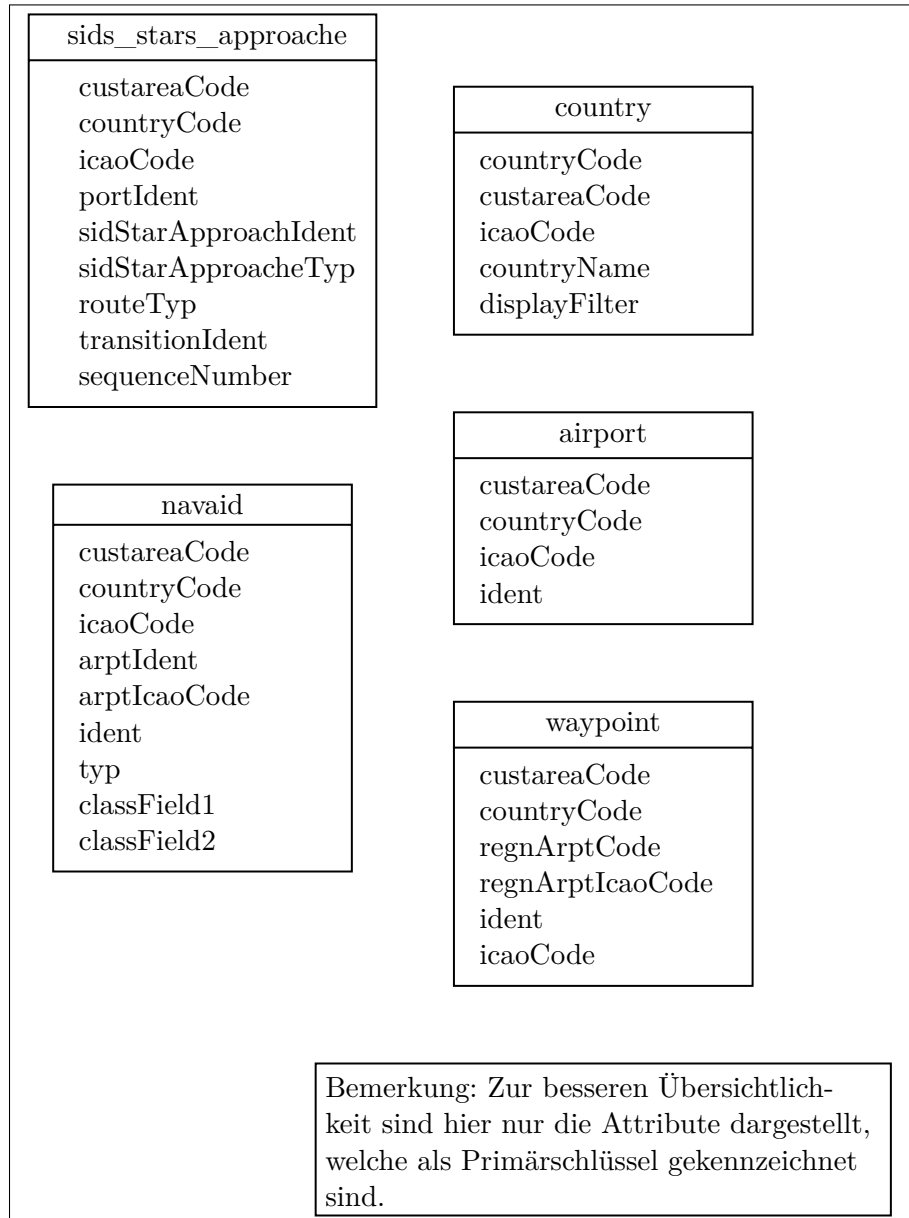



Abbildung A.5: HTTP-Methoden S. 1



Select a definition 

v1


# LFID Web Service

v1 OAS3

/swagger/v1/swagger.json












A .Net Core 2.1 Web API for managing the LFID database.

Contact Sven Bergmann

Authorize 

## Airports

▼

|      |                                |   |   |
|------|--------------------------------|---|---|
| GET  | /api/Airports/shadow/icao      | GET and HEAD request for returning an IEnumerable of shadow airports matching the given icaoCode. API call: Airports/shadow/icao/"icaoCode"       |   |
| HEAD | /api/Airports/shadow/icao      | GET and HEAD request for returning an IEnumerable of shadow airports matching the given icaoCode. API call: Airports/shadow/icao/"icaoCode"       |  |
| GET  | /api/Airports/shadow/custArea  | GET and HEAD request for returning an IEnumerable of shadow airports matching the given custArea. API call: Airports/shadow/custArea/"custArea"   |  |
| HEAD | /api/Airports/shadow/custArea  | GET and HEAD request for returning an IEnumerable of shadow airports matching the given custArea. API call: Airports/shadow/custArea/"custArea"   |  |
| GET  | /api/Airports/hateoas/icao     | GET and HEAD request for returning an IEnumerable of hateoas airports matching the given icaoCode. API call: Airports/hateoas/icao/"icaoCode"     |  |
| HEAD | /api/Airports/hateoas/icao     | GET and HEAD request for returning an IEnumerable of hateoas airports matching the given icaoCode. API call: Airports/hateoas/icao/"icaoCode"     |  |
| GET  | /api/Airports/hateoas/custArea | GET and HEAD request for returning an IEnumerable of hateoas airports matching the given custArea. API call: Airports/hateoas/custArea/"custArea" |  |
| HEAD | /api/Airports/hateoas/custArea | GET and HEAD request for returning an IEnumerable of hateoas airports matching the given custArea. API call: Airports/hateoas/custArea/"custArea" |  |
| GET  | /api/Airports/full/icao        | GET and HEAD request for returning an IEnumerable of full airports matching the given icaoCode. API call: Airports/full/icao/"icaoCode"           |  |
| HEAD | /api/Airports/full/icao        | GET and HEAD request for returning an IEnumerable of full airports matching the given icaoCode. API call: Airports/full/icao/"icaoCode"           |  |
| GET  | /api/Airports/full/custArea    | GET and HEAD request for returning an IEnumerable of full airports matching the given custArea. API call: Airports/full/custArea/"custArea"       |  |

**Abbildung A.6: HTTP-Methoden S. 2**

|                                   |                              |   |   |
|-----------------------------------|------------------------------|---|---|
| GET                               | /api/Airports/full/custArea  | matching the given custArea. API call: Airports/full/custArea/"custArea"  | — |
| HEAD                              | /api/Airports/full/custArea  | GET and HEAD request for returning an IEnumerable of full airports matching the given custArea. API call: Airports/full/custArea/"custArea" | 🔒 |
| <b>Countries</b> <span>▼</span>   |                              |   |   |
| GET                               | /api/Countries/shadow        | GET and HEAD request for all shadow countries. API call: Countries/shadow   | 🔒 |
| HEAD                              | /api/Countries/shadow        | GET and HEAD request for all shadow countries. API call: Countries/shadow   | 🔒 |
| GET                               | /api/Countries/hateoas       | GET and HEAD request for all hateoas countries. API call: Countries/hateoas   | 🔒 |
| HEAD                              | /api/Countries/hateoas       | GET and HEAD request for all hateoas countries. API call: Countries/hateoas   | 🔒 |
| GET                               | /api/Countries/full          | GET and HEAD request for all countries. API call: Countries/full  | 🔒 |
| HEAD                              | /api/Countries/full          | GET and HEAD request for all countries. API call: Countries/full  | 🔒 |
| GET                               | /api/Countries/fullAnonymous |   | 🔒 |
| HEAD                              | /api/Countries/fullAnonymous |   | 🔒 |
| <b>HateoasHome</b> <span>▼</span> |                              |   |   |
| GET                               | /api/HateoasHome             | GET and HEAD request for returning available URIs. API call: "api"  | 🔒 |
| HEAD                              | /api/HateoasHome             | GET and HEAD request for returning available URIs. API call: "api"  | 🔒 |
| <b>Navaid</b> s <span>▼</span>    |                              |   |   |
| GET                               | /api/Navaid/s/full/icao      | GET and HEAD request for returning an IEnumerable of full navaid/s matching the given icaoCode. API call: Navaid/s/full/icao/"icaoCode"     | 🔒 |
| HEAD                              | /api/Navaid/s/full/icao      | GET and HEAD request for returning an IEnumerable of full navaid/s matching the given icaoCode. API call: Navaid/s/full/icao/"icaoCode"     | 🔒 |
| GET                               | /api/Navaid/s/full/custArea  | GET and HEAD request for returning an IEnumerable of full navaid/s matching the given custArea. API call: Navaid/s/full/custArea/"custArea" | 🔒 |
| HEAD                              | /api/Navaid/s/full/custArea  | GET and HEAD request for returning an IEnumerable of full navaid/s matching the given custArea. API call: Navaid/s/full/custArea/"custArea" | 🔒 |
| GET                               | /api/Navaid/s/full           |   | 🔒 |
| HEAD                              | /api/Navaid/s/full           |   | 🔒 |

Abbildung A.7: HTTP-Methoden S. 3

## SidsStarsApproaches



|      |  |   |  |
|------|--|---|--|
| GET  | /api/SidsStarsApproaches/hateoas/portIdent | GET and HEAD request for returning an IEnumerable of hateoas Sids, Stars or Approaches matching the given port ident. API call: "SidsStarsApproaches/hateoas/portIdent"/"portIdent"/"sidStarApproachType" |  |
| HEAD | /api/SidsStarsApproaches/hateoas/portIdent | GET and HEAD request for returning an IEnumerable of hateoas Sids, Stars or Approaches matching the given port ident. API call: "SidsStarsApproaches/hateoas/portIdent"/"portIdent"/"sidStarApproachType" |  |
| GET  | /api/SidsStarsApproaches/full/portIdent    | GET and HEAD request for returning an IEnumerable of full Sids, Stars or Approaches matching the given port ident. API call: "SidsStarsApproaches/full/portIdent"/"portIdent"/"sidStarApproachType"       |  |
| HEAD | /api/SidsStarsApproaches/full/portIdent    | GET and HEAD request for returning an IEnumerable of full Sids, Stars or Approaches matching the given port ident. API call: "SidsStarsApproaches/full/portIdent"/"portIdent"/"sidStarApproachType"       |  |
| GET  | /api/SidsStarsApproaches/shadow/portIdent  | GET and HEAD request for returning an IEnumerable of shadow Sids, Stars or Approaches matching the given port ident. API call: "SidsStarsApproaches/shadow/portIdent"/"portIdent"/"sidStarApproachType"   |  |
| HEAD | /api/SidsStarsApproaches/shadow/portIdent  | GET and HEAD request for returning an IEnumerable of shadow Sids, Stars or Approaches matching the given port ident. API call: "SidsStarsApproaches/shadow/portIdent"/"portIdent"/"sidStarApproachType"   |  |
| GET  | /api/SidsStarsApproaches/shadow/sortedLegs |   |  |
| HEAD | /api/SidsStarsApproaches/shadow/sortedLegs |   |  |

## Waypoints



|      |                     |   |  |
|------|---------------------|---|--|
| GET  | /api/Waypoints/full | GET and HEAD request for all waypoints with the given fixIdentifier. API call: Waypoints/full/ident/"ident" |  |
| HEAD | /api/Waypoints/full | GET and HEAD request for all waypoints with the given fixIdentifier. API call: Waypoints/full/ident/"ident" |  |

## Schemas





## B Tabellen

**Tabelle B.1:** Kreuzreferenztablette zu elementaren Gefährdungen APP.3.1

Quelle: Holger Schildt. IT-Grundschutz-Kompendium. Bundesamt für Sicherheit in der Informationstechnik, Bonn, Feb. 2022. ISBN: 978-3-8462-0906-6

| APP.3.1     | CIA | G<br>0.14 | G<br>0.15 | G<br>0.18 | G<br>0.19 | G<br>0.23 | G<br>0.28 | G<br>0.30 | G<br>0.31 | G<br>0.43 | G<br>0.46 |
|-------------|-----|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| APP.3.1.A1  |     |           |           | X         |           |           |           | X         | X         |           |           |
| APP.3.1.A4  |     |           |           |           | X         | X         |           | X         |           | X         |           |
| APP.3.1.A7  |     |           |           |           |           | X         |           | X         | X         |           |           |
| APP.3.1.A8  |     |           |           | X         |           |           |           |           |           |           |           |
| APP.3.1.A9  |     |           |           | X         |           |           | X         |           |           |           |           |
| APP.3.1.A11 |     |           | X         |           | X         | X         |           |           |           | X         | X         |
| APP.3.1.A12 |     |           |           |           |           | X         |           | X         | X         |           |           |
| APP.3.1.A14 |     |           |           |           | X         |           |           |           |           |           |           |
| APP.3.1.A20 | CIA |           |           |           |           | X         |           |           |           |           |           |
| APP.3.1.A21 |     | X         |           |           |           | X         |           |           |           |           |           |
| APP.3.1.A22 |     |           |           | X         |           |           | X         |           |           |           |           |

**Tabelle B.2:** *.NET Core 2.1* vs. *.NET Framework*

Quelle: Vgl. Nirmal Srivastava. Differences between .NET Core and .NET framework. Juni 2022. URL: <https://www.geeksforgeeks.org/differences-between-net-core-and-net-framework/#:~:text=Application%20Models-,.,windows%20forms%20and%20WPF%20applications.> (besucht am 06.07.2022)

|  | <i>.NET Core 2.1</i>   | <i>.NET Framework</i>  |
|--|--|--|
| Open Source  | Ja   | Nur manche Komponenten   |
| Cross-Platform   | Ja, läuft auf Windows, Linux und MacOS   | Nein, läuft nur auf Windows  |
| Application Models   | Unterstützt keine Entwicklung von Desktopanwendungen, fokussiert sich mehr auf die Entwicklung von Webanwendungen  | Wird für die Entwicklung von Desktopanwendungen genutzt, kann aber auch für Webanwendungen genutzt werden                      |
| Installation   | Alle Abhängigkeiten werden in das auszuliefernde Paket gepackt, damit es plattformübergreifend genutzt werden kann | Hier wird nur das projektspezifische Paket gepackt, alle anderen werden durch die Installation im Betriebssystem nachgeliefert |
| Unterstützung von Micro-Services und <i>REST</i> -Services | Ja, unterstützt beides   | Unterstützt nur die Implementierung von <i>REST</i> -Services  |
| Leistung und Skalierbarkeit                                | Leistungsfähiger und skalierbarer  | Nicht so effektiv im Vergleich zu <i>.NET Core 2.1</i>   |



**Tabelle B.3:** *IT Sicherheitsanforderungstabelle*

| OWASP Top 10                 | IT Grundschatz Kompendium            | Realisierung   |
|------------------------------|--------------------------------------|--|
| A01:2021                     | APP.3.1.A1, APP.3.1.A14, APP.3.1.A9  | Einrichtung verschiedener Profile mit unterschiedlichen Rechten und Freigaben  |
| A02:2021                     | APP.3.1.A8, APP.3.1.A11              | Nutzung von TLS/SSL und einer Authentifizierung  |
| A03:2021                     | APP.3.1.A14, APP.3.1.A11             | Benutzen eines Frameworks zur Gestaltung des Web Services, zusammen mit LINQ zur Abfrage der Korrektheit der übergebenen Parameter     |
| A04:2021, A06:2021, A08:2021 | APP.3.1.A8, APP.3.1.A9               | Externe Pakete nur aus vertrauenswürdigen Quellen, wie bspw. Microsoft und aktuellste Versionen bzw. Sicherheitspatches                |
| A05:2021, A07:2021           | APP.3.1.A1, APP.3.1.A14, APP.3.1.A12 | Deaktivieren von Standardpasswörtern und -benutzerkennungen, Methoden nur für autorisierte Nutzer erreichbar, ablaufende Access Tokens |
| A09:2021                     | APP.3.1.A1, APP.3.1.A7               | Anmeldeversuche loggen, Methodenaufrufe loggen, Logs in einer von außen nicht zugänglichen Datei abspeichern                           |
| A10:2021                     | APP.3.1.A12                          |  |



# C Listings

## C.1 Authorization Server

Listing C.1: *Program.cs* in *Autorisierungsserver*

```
1 namespace AuthorizationServer {
2     public class Program {
3         public static void Main(string[] args) {
4             CreateWebHostBuilder(args).Build().Run();
5         }
6
7         public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
8             WebHost.CreateDefaultBuilder(args)
9                 .ConfigureLogging(config =>
10                     {
11                         config.AddDebug();
12                         config.AddConsole();
13                     })
14                 .UseStartup<Startup>()
15                 .UseUrls("https://*:5003");
16     }
17 }
```

Listing C.2: *Startup.cs* in *Autorisierungsserver*

```
1 namespace AuthorizationServer {
2     public class Startup {
3         public Startup(IConfiguration configuration) {
4             Configuration = configuration;
5         }
6
7         public IConfiguration Configuration { get; }
8
9         public void ConfigureServices(IServiceCollection services) {
10             services.AddIdentityServer()
11                 .AddDeveloperSigningCredential()
12                 .AddInMemoryApiResources(Config.GetApiResources())
13                 .AddInMemoryClients(Config.GetClients());
14             services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
15         }
16
17         public void Configure(IApplicationBuilder app, IHostingEnvironment env) {
18             if (env.IsDevelopment())
19                 app.UseDeveloperExceptionPage();
20             else
21                 app.UseHsts();
22             app.UseIdentityServer();
23         }
24     }
25 }
```

```

23     app.UseHttpsRedirection();
24     app.UseMvc(routes => {
25         routes.MapRoute("default",
26             "{controller=Home}/{action=index}/{id?}");
27     });
28 }
29 }
30 }

```

**Listing C.3:** *Config.cs in Autorisierungsserver*

```

1 namespace AuthorizationServer {
2     public class Config {
3         public static IEnumerable<ApiResource> GetApiResources() {
4             return new List<ApiResource> {
5                 new ApiResource("scope.readaccess", "LFID API"),
6                 new ApiResource("scope.fullaccess", "LFID API")
7             };
8         }
9
10        public static IEnumerable<Client> GetClients() {
11            return new List<Client> {
12                new Client {
13                    ClientId = "ClientIdThatCanOnlyRead",
14                    AllowGrantTypes = GrantTypes.ClientCredentials,
15                    ClientSecrets = { new Secret("secret1".Sha256()) },
16                    AllowedScopes = {"scope.readaccess"}
17                },
18                new Client {
19                    ClientId = "ClientIdWithWriteAccess",
20                    AllowGrantTypes = GrantTypes.ClientCredentials,
21                    ClientSecrets = { new Secret("secret1".Sha256()) },
22                    AllowedScopes = {"scope.writeaccess"}
23                }
24            };
25        }
26    }
27 }

```

## C.2 Webservice

Listing C.4: *Program.cs in Webservice*

```
1 namespace Webservice {
2     public class Program {
3         public static void Main(string[] args) {
4             CreateWebHostBuilder(args).Build().Run();
5         }
6
7         public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
8             WebHost.CreateDefaultBuilder(args).
9                 .ConfigureLogging(config =>
10                     {
11                         config.AddDebug();
12                         config.AddConsole();
13                     })
14                 .UseStartup<Startup>()
15                 .UseUrls("https://*:5001");
16     }
17 }
```

Listing C.5: *ConfigureServices in Webservice*

```
1 public void ConfigureServices(IServiceCollection services) {
2     services.AddLogging(config => {
3         config.AddDebug();
4         config.AddConsole();
5     });
6
7     services.AddSingleton(new MapperConfiguration(mc =>
8         { mc.AddProfile(new AutoMapper()); })
9         .CreateMapper());
10
11     services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
12
13     services.AddSingleton<IAuthorizationHandler, ApiHandler>();
14
15     services.AddAuthentication(options => {
16         options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
17         options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
18     }).AddJwtBearer("SchemeReadAccess", options => {
19         options.Authority = "https://localhost:5003";
20         options.Audience = "scope.readaccess";
21         options.RequireHttpsMetadata = false;
22     }).AddJwtBearer("SchemeWriteAccess", options => {
23         options.Authority = "https://localhost:5003";
24         options.Audience = "scope.writeaccess";
25         options.RequireHttpsMetadata = false;
26     });
27 }
```

```

28 services.AddAuthorization(options => {
29     options.DefaultPolicy = new AuthorizationPolicyBuilder()
30         .RequireAuthenticatedUser()
31         .AddAuthenticationSchemes("SchemeReadAccess", "SchemeWriteAccess")
32         .Build();
33     options.AddPolicy("CustomPolicy", policy => { policy.AddRequirements(new
34         ApiRequirement()); });
35 });
36
37 services.AddSwaggerGen(options => {
38     var securityScheme = new OpenApiSecurityScheme {
39         Name = "JWT Authentication",
40         Description = "Enter JWT Bearer token **_only_**",
41         In = ParameterLocation.Header,
42         Type = SecuritySchemeType.Http,
43         Scheme = "bearer",
44         BearerFormat = "JWT",
45         Reference = new OpenApiReference {
46             Id = JwtBearerDefaults.AuthenticationScheme,
47             Type = ReferenceType.SecurityScheme
48         }
49     };
50     options.AddSecurityDefinition(securityScheme.Reference.Id, securityScheme);
51     options.AddSecurityRequirement(new OpenApiSecurityRequirement {
52         {securityScheme, new string[] { }}});
53
54     options.SwaggerDoc("v1", new OpenApiInfo {
55         Version = "v1",
56         Title = "LFID Web Service",
57         Description = "A .Net Core 2.1 Web API for managing the LFID database.",
58         Contact = new OpenApiContact { Name = "Sven Bergmann", Email = "sven.bergmann@cae
59             .de" }
60     });
61
62     var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
63     var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);
64     options.IncludeXmlComments(xmlPath);
65 });
66
67 services.AddDbContext<LfidContext>(options => {
68     options.UseSqlServer(Configuration["ConnectionStrings:lfidDatabase"]);
69 });

```

**Listing C.6:** *Configure in Webservice*

```

1 public void Configure(IApplicationBuilder app, IHostingEnvironment env) {
2     JwtSecurityTokenHandler.DefaultInboundClaimTypeMap.Clear();
3
4     if (env.IsDevelopment())
5         app.UseDeveloperExceptionPage();

```

```

6     else
7         app.UseHsts();
8
9     app.UseExceptionHandler(c => c.Run(async context => {
10         var exception = context.Features
11             .Get<ExceptionHandlerPathFeature>()
12             .Error;
13         string response;
14
15         switch (exception) {
16             case ArgumentOutOfRangeException _:
17                 response = JsonConvert.SerializeObject(new {error = "Argument/s is/are out of
18                     range!"});
19                 context.Response.StatusCode = StatusCodes.Status404NotFound;
20                 break;
21             case DataNotFoundException dataNotFoundException:
22                 response = JsonConvert.SerializeObject(new {dataNotFoundException.Message});
23                 context.Response.StatusCode = (int) dataNotFoundException.StatusCode;
24                 break;
25             default:
26                 response = JsonConvert.SerializeObject(new {error = "Exception Handling not
27                     defined!"});
28                 context.Response.StatusCode = StatusCodes.Status501NotImplemented;
29                 break;
30         }
31
32         context.Response.ContentType = "application/json";
33         await context.Response.WriteAsync(response);
34     }));
35
36     app.UseSwagger();
37     app.UseSwaggerUI(options => {
38         options.SwaggerEndpoint("/swagger/v1/swagger.json", "v1");
39         options.RoutePrefix = string.Empty;
40     });
41
42     app.UseStaticFiles();
43     app.UseAuthentication();
44     app.UseStatusCodePages();
45     app.UseHttpsRedirection();
46     app.UseMvc();
47 }

```

**Listing C.7:** *DataNotFoundException in Webservice*

```

1 public class DataNotFoundException : Exception {
2     public HttpStatusCode StatusCode = HttpStatusCode.NotFound;
3
4     public DataNotFoundException() : base("Requested data was not found.") { }
5 }

```

Listing C.8: *ApiControllerBase.cs* in *Webservice*

```

1 namespace Webservice.API {
2     [ApiController]
3     [Produces("application/json")]
4     [Route("api/controller")]
5     [Authorize(AuthenticationSchemes = "SchemeReadAccess, SchemeWriteAccess",
6         Policy = "CustomPolicy")]
7     public class ApiControllerBase : Controller {
8         private readonly IReadOnlyList<ActionDescriptor> _routes;
9         protected readonly LfidContext lfidContext;
10        protected readonly ILogger<ApiControllerBase> Logger;
11        protected readonly IMapper Mapper;
12        public ApiControllerBase(IActionDescriptorCollectionProvider
13            actionDescriptorCollectionProvider,
14            IMapper mapper, LfidContext lfidContext, ILogger<ApiControllerBase> logger) {
15            _routes = actionDescriptorCollectionProvider.ActionDescriptors.Items;
16            Mapper = mapper;
17            LfidContext = lfidContext;
18            Logger = logger;
19        }
20    }

```

Listing C.9: *CountriesController.cs* in *Webservice*

```

1 namespace Webservice.API.ModelControllers {
2     public class CountriesController : ApiControllerBase {
3         public CountriesController(IActionDescriptorCollectionProvider
4             actionDescriptorCollectionProvider, IMapper mapper, LfidContext lfidContext,
5             ILogger<ApiControllerBase> logger)
6             : base(actionDescriptorCollectionProvider, mapper, lfidContext, logger)
7         { }
8
9         [HttpGet("shadow", Name = nameof(GetShadowCountries))]
10        [HttpHead("shadow")]
11        public async Task<ActionResult<IEnumerable<CountryModel>>> GetShadowCountries() {
12            var res = await lfidContext.Country.ToListAsync();
13            if (!res.Any()) throw new DataNotFoundException();
14            return Ok(res.OrderBy(c => c.CustareaCode)
15                .ThenBy(c => c.IcaoCode)
16                .GroupBy(c => c.IcaoCode, (key, c) => c.FirstOrDefault())
17                .Select(c => Mapper.Map<CountryModel>(c)));
18        }
19    }

```



## C.3 WebserviceTests

Listing C.10: *AuthorizedClientFixture.cs* in *WebserviceTests*

```
1 namespace Webservice.Tests {
2     public class AuthorizedClientFixture : IDisposable {
3         public AuthorizedClientFixture() {
4             Client = new HttpClient {
5                 BaseAddress = new Uri("https://localhost:5001/api/")
6             };
7             authorizationServerTokenIssuerUri =
8                 new Uri("https://localhost:5003/connect/token");
9             const string clientId = "ClientIdThatCanOnlyRead";
10            const string scope = "scope.readaccess";
11            const string clientSecret = "secret1";
12            var jwtToken = RequestTokenToAuthorizationServer(
13                authorizationServerTokenIssuerUri,
14                clientId, scope, clientSecret).Result;
15            var rawToken = JObject.Parse(jwtToken)["accessToken"];
16            Client.DefaultRequestHeaders.Authorization =
17                new AuthenticationHeaderValue("Bearer", rawToken?.ToString());
18        }
19
20        public HttpClient Client { get; }
21
22        public void Dispose() { Client.Dispose(); }
23
24        private async Task<string> RequestTokenToAuthorizationServer( Uri
25            uriAuthorizationServer, string clientId, string scope, string clientSecret) {
26            HttpResponseMessage responseMessage;
27            using(var client = new HttpClient()) {
28                var tokenRequest = new HttpRequestMessage(HttpMethod.Post,
29                    uriAuthorizationServer);
30                HttpContent httpContent = new FormUrlEncodedContent(new[] {
31                    new KeyValuePair<string, string>("grant_type",
32                        "client_credentials"),
33                    new KeyValuePair<string, string>("clientId", clientId),
34                    new KeyValuePair<string, string>("scope", scope),
35                    new KeyValuePair<string, string>("clientSecret", clientSecret)
36                });
37                tokenRequest.Content = httpContent;
38                responseMessage = client.SendAsync(tokenRequest).Result;
39            }
40            return await responseMessage.Content.ReadAsStringAsync();
41        }
42    }
```

**Listing C.11:** *CountriesSQLTest.cs in WebserviceTests*

```
1 namespace Webservice.Test {
2     public class CountriesSqlTest {
3         private const string CONNECTION_STRING = "...";
4         private const string SQL_QUERY = "SELECT * FROM [lfid].[dbo].[country]";
5         private readonly ITestOutputHelper _testOutputHelper;
6
7         public CountriesSqlTest(ITestOutputHelper testOutputHelper) {
8             _testOutputHelper = testOutputHelper;
9         }
10
11         [Theory]
12         [Repeat(10)]
13         public void Should_return_all_countries_from_Database() {
14             var t0 = DateTime.Now;
15             _testOutputHelper.WriteLine($"Started request with query at: {t0}");
16             using(var con = new SqlConnection(CONNECTION_STRING)) {
17                 var command = new SqlCommand(SQL_QUERY, con);
18                 command.Connection.Open();
19                 var reader = command.ExecuteReader();
20                 reader.Read();
21                 command.Connection.Close();
22             }
23             var t1 = DateTime.Now;
24             _testOutputHelper.WriteLine($"Received countries in:
25                 {(t1-t0).Milliseconds} milliseconds.");
26         }
27     }
28 }
```

**Listing C.12:** *CountriesWebTestAuthorized.cs in WebserviceTests*

```
1 namespace WebService.Test {
2     public class CountriesWebTestAuthorized : IClassFixture<AuthorizedClientFixture> {
3         private readonly AuthorizedClientFixture _authorizedClientFixture;
4         private readonly ITestOutputHelper _testOutputHelper;
5
6         public CountriesWebTestAuthorized(ITestOutputHelper testOutputHelper,
7             AuthorizedClientFixture authorizedClientFixture) {
8             _testOutputHelper = testOutputHelper;
9             _authorizedClientFixture = authorizedClientFixture;
10        }
11
12        [Theory]
13        [Repeat(10)]
14        public void Should_return_all_countries_with_Authentication() {
15            var t0 = DateTime.Now;
16            _testOutputHelper.WriteLine($"Started request with
17                authentication at: {t0}");
18            var countries = GetCountryModels(_authorizedClientFixture.Client,
19                "Countries/full").Result;
20            var t1 = DateTime.Now;
21            if (countries.Count != 0)
22                _testOutputHelper.WriteLine($"Received countries in:
23                    {(t1-t0).Milliseconds} milliseconds.");
24            Assert.NotEmpty(countries);
25        }
26
27        public static async Task<List<CountryModel>> GetCountryModels(
28            HttpClient httpClient, string uri) {
29            var response = await httpClient.GetAsync(uri);
30            if (response.IsSuccessStatusCode)
31                return JsonConvert
32                    .DeserializeObject<List<CountryModel>>(
33                        response.Content.ReadAsStringAsync().Result);
34            return null;
35        }
36    }
37 }
```

**Listing C.13:** *CountriesWebTestUnauthorized.cs* in *WebserviceTests*

```
1 namespace Webservice.Test {
2     public class CountriesWebTestUnauthorized {
3         protected static readonly HttpClient Client = new HttpClient {
4             BaseAddress = new Uri("https://localhost:5001/api/")
5         };
6         private readonly ITestOutputHelper _testOutputHelper;
7
8         public CountriesWebTestUnauthorized(ITestOutputHelper testOutputHelper) {
9             _testOutputHelper = testOutputHelper;
10        }
11
12        [Theory]
13        [Repeat(10)]
14        public void Should_return_all_countries() {
15            var t0 = DateTime.Now;
16            _testOutputHelper.WriteLine($"Started request with
17                authentication at: {t0}");
18            var countries = CountriesWebTestAuthorized
19                .GetCountryModels(_authorizedClientFixture.Client,
20                    "Countries/fullAnonymous").Result;
21            var t1 = DateTime.Now;
22            if (countries.Count != 0)
23                _testOutputHelper.WriteLine($"Received countries in:
24                    {(t1-t0).Milliseconds} milliseconds.");
25            Assert.NotEmpty(countries);
26        }
27    }
28 }
```

## C.4 Demo Anwendung

Listing C.14: *Application.cs* in der Demo Anwendung

```
1 namespace Application__LFID_WebService_Parser {
2     public partial class MainWindow {
3
4         private static readonly HttpClient HttpClient =
5             new HttpClient {
6                 BaseAddress = new Uri("https://localhost:5001/api/")
7             };
8
9         public MainWindow() {
10             var authorizationServerTokenIssuerUri =
11                 new Uri("https://localhost:5003/connect/token");
12             const string clientId = "ClientIdThatCanOnlyRead";
13             const string clientSecret = "secret1";
14             const string scope = "scope.readaccess";
15             var jwtToken = RequestTokenToAuthorizationServer(
16                 authorizationServerTokenIssuerUri,
17                 clientId,
18                 scope,
19                 clientSecret)
20                 .Result;
21             if (jwtToken != null) {
22                 var rawToken = JObject.Parse(jwtToken)["access_token"];
23                 HttpClient.DefaultRequestHeaders.Authorization =
24                     new AuthenticationHeaderValue("Bearer", rawToken?.ToString());
25             } else {
26                 const string message =
27                     "The AuthorizationServer has not been started yet. Make sure the
28                     AuthorizationServer is up and running. Please restart the application
29                     after you started the AuthorizationServer.";
30                 const string caption = "AuthorizationServer down.";
31
32                 var result = MessageBox.Show(message, caption, MessageBoxButton.OK);
33                 if (result.Equals(MessageBoxResult.OK)) Close();
34             }
35         }
36     }
37 }
```

**Listing C.15:** *Token in der Demo Anwendung beantragen*

```
1 private static async Task<string> RequestTokenToAuthorizationServer(Uri
2   uriAuthorizationServer, string clientId, string scope, string clientSecret) {
3   HttpResponseMessage responseMessage;
4   using (var client = new HttpClient()) {
5     var tokenRequest = new HttpRequestMessage(HttpMethod.Post, uriAuthorizationServer);
6     HttpContent httpContent = new FormUrlEncodedContent(new[] {
7       new KeyValuePair<string, string>("grant_type", "client_credentials"),
8       new KeyValuePair<string, string>("client_id", clientId),
9       new KeyValuePair<string, string>("scope", scope),
10      new KeyValuePair<string, string>("client_secret", clientSecret)
11    });
12    tokenRequest.Content = httpContent;
13    try {
14      responseMessage = client.SendAsync(tokenRequest).Result;
15      return await responseMessage.Content.ReadAsStringAsync();
16    } catch (Exception e) {
17      Console.WriteLine(e);
18      return null;
19    }
20 }
```

Listing C.16: Datenabfrage in der Demo Anwendung

```
1 private static async Task<IEnumerable<CountryModel>> GetCountriesAsync(HttpClient client) {
2     var response = await client.GetAsync("Countries/shadow");
3     return response.IsSuccessStatusCode ?
4         JsonConvert.DeserializeObject<List<CountryModel>>(response.Content.ReadAsStringAsync
5             ().Result)
6         : null;
7 }
8 private static async Task<IEnumerable<AirportModel>> GetAirportsAsync(HttpClient client,
9     string icao) {
10     var response = await client.GetAsync($"Airports/shadow/icao?icaoCode={icao}");
11     return response.IsSuccessStatusCode ?
12         JsonConvert.DeserializeObject<List<AirportModel>>(response.Content.ReadAsStringAsync
13             ().Result)
14         : null;
15 }
16 private static async Task<IEnumerable<SidStarApproachModel>> GetSidsStarsApproachesAsync(
17     HttpClient client, string airportId, SidStarApproachEnum sidStarApproachEnum) {
18     var response = await client.GetAsync($"SidsStarsApproaches/shadow/portId?portId={
19         airportId}&sidStarApproachType={(int) sidStarApproachEnum}");
20     return response.IsSuccessStatusCode ?
21         JsonConvert.DeserializeObject<List<SidStarApproachModel>>(response.Content.
22             ReadAsStringAsync().Result)
23         : null;
24 }
25 private static async Task<IEnumerable<SidStarApproachModel>> GetSortedLegsAsync(HttpClient
26     client,
27     string custareaCode, string icaoCode, string ssaId, string transitionId) {
28     var res = await client.GetAsync($"SidsStarsApproaches/shadow/sortedLegs?custareaCode={
29         custareaCode}&icaoCode={icaoCode}&ssaId={ssaId}&transitionId={
30         transitionId}");
31     return res.IsSuccessStatusCode ?
32         JsonConvert.DeserializeObject<List<SidStarApproachModel>>(res.Content.
33             ReadAsStringAsync().Result)
34         : null;
35 }
36 private static async Task<List<Waypoint>> GetWaypointsAsync(HttpClient client, string
37     custareaCode, string icaoCode, string id) {
38     var res = await client.GetAsync($"Waypoints/full?custareaCode={custareaCode}&icaoCode={
39         icaoCode}&id={id}");
40     return res.IsSuccessStatusCode ?
41         JsonConvert.DeserializeObject<List<Waypoint>>(res.Content.ReadAsStringAsync().Result)
42         : null;
43 }
```

